Problem Statement 4: IoT Predictive Maintenance Platform

Summary

Develop an AI platform that analyzes IoT sensor data from industrial equipment to predict failures, schedule maintenance, and optimize equipment performance.

Problem Statement

Industrial IoT deployments generate vast amounts of sensor data that can predict equipment failures before they occur. Your task is to build a predictive maintenance platform that processes real-time sensor data, identifies patterns indicating potential failures, and recommends optimal maintenance schedules. The system should support multiple equipment types, provide cost-benefit analysis for maintenance decisions, and integrate with existing maintenance management systems.

Key Requirements

Core Functionality

- **Scalable Data Ingestion**: Design a scalable data ingestion pipeline for IoT sensor streams
- **Anomaly Detection**: Implement anomaly detection algorithms for various sensor types (vibration, temperature, pressure)
- **Predictive Modeling**: Create predictive models for different failure modes using time series analysis
- **Maintenance Optimization**: Build a maintenance scheduling optimization engine considering costs and priorities
- **Real-time Monitoring**: Develop a monitoring dashboard with real-time equipment health status
- **Mobile Integration**: Include mobile alerts and work order integration for maintenance teams

Suggested Data Requirements

- Time-series sensor data from multiple equipment types (temperature, vibration, pressure, etc.)
- Equipment maintenance history and failure records
- Spare parts inventory and cost information
- Equipment specifications and operating parameters

Themes

- AI in Service lines
- Classical AI/ML/DL for prediction

Technical Approach

This solution will leverage advanced time series analysis, machine learning models for anomaly detection, and optimization algorithms for maintenance scheduling. The platform will integrate with existing industrial systems and provide actionable insights for maintenance teams.

Expected Outcomes

- Reduced unplanned downtime through predictive failure detection
- Optimized maintenance schedules balancing cost and equipment reliability
- Real-time visibility into equipment health across industrial operations
- Integration with existing maintenance management workflows
- Mobile-first approach for field maintenance teams

Implementation Strategy

The solution will be built using a microservices architecture with real-time data processing capabilities, supporting multiple industrial protocols and equipment types while ensuring scalability and reliability for industrial environments. # Product Requirements Document (PRD) ## IoT Predictive Maintenance Platform

Foundation document for comprehensive industrial predictive maintenance solution

ETVX Framework

ENTRY CRITERIA

- âœ… Problem statement analyzed and understood (IoT sensor data analysis for predictive maintenance)
- âœ… Industrial IoT domain requirements researched and documented
- âœ… Stakeholder needs identified (maintenance teams, operations managers, plant engineers)
- âœ… Market analysis completed for predictive maintenance solutions
- âœ… Technical feasibility assessment for real-time IoT data processing and ML prediction
- âœ… Regulatory and safety requirements for industrial environments documented

TASK

Define comprehensive product requirements for an IoT predictive maintenance platform that processes real-time sensor data, predicts equipment failures, optimizes maintenance schedules, and integrates with existing industrial systems while delivering measurable business value through reduced downtime and optimized maintenance costs.

VERIFICATION & VALIDATION

Verification Checklist: - [ ] Business objectives clearly defined with quantifiable success metrics - [ ] Target users and personas documented with specific use cases and workflows - [ ] Product features prioritized based on business impact and technical feasibility - [ ] Success metrics defined with baseline measurements and improvement targets - [ ] Technical constraints and integration requirements documented - [ ] Competitive analysis completed with differentiation strategy defined

Validation Criteria: - [ ] Stakeholder review completed with maintenance teams, operations managers, and plant engineers - [ ] Business case validated with ROI projections and cost-benefit analysis - [ ] Technical approach validated with IoT and industrial automation experts - [ ] Market positioning confirmed through customer interviews and industry analysis - [ ] Success metrics validated against industry benchmarks and customer expectations - [ ] Product roadmap aligned with business strategy and market demands

EXIT CRITERIA

- âœ… Complete PRD approved by business stakeholders and technical leadership
- âœ… Clear product vision and strategy established for industrial predictive maintenance
- âœ… Success metrics and KPIs defined with measurement methodology
- âœ… Target user personas and use cases documented with workflow requirements
- âœ… Foundation established for detailed functional requirements development

---

1. Product Vision and Strategy

1.1 Vision Statement

To revolutionize industrial maintenance through AI-powered predictive analytics that transforms reactive maintenance into proactive, data-driven operations,

reducing unplanned downtime by 70% and maintenance costs by 25% while extending equipment lifespan and improving operational efficiency.

1.2 Product Mission

Deliver an intelligent IoT predictive maintenance platform that seamlessly integrates with existing industrial infrastructure to provide real-time equipment health monitoring, accurate failure prediction, and optimized maintenance scheduling, empowering maintenance teams with actionable insights and mobile-first workflows.

1.3 Strategic Objectives

- **Operational Excellence**: Minimize unplanned downtime through accurate failure prediction
- **Cost Optimization**: Reduce maintenance costs while maximizing equipment lifespan
- **Safety Enhancement**: Prevent catastrophic failures that could endanger personnel
- **Digital Transformation**: Modernize maintenance operations with AI and IoT technologies
- **Competitive Advantage**: Enable data-driven decision making for maintenance operations

2. Market Analysis and Positioning

2.1 Market Opportunity

- **Total Addressable Market**: $12.3B global predictive maintenance market by 2025
- **Serviceable Addressable Market**: $3.8B for IoT-enabled predictive maintenance solutions
- **Target Market Segments**: Manufacturing, oil & gas, utilities, transportation, mining
- **Growth Drivers**: Industry 4.0 adoption, IoT sensor cost reduction, AI/ML advancement

2.2 Competitive Landscape

- **Traditional CMMS**: Limited predictive capabilities, reactive maintenance focus
- **Industrial IoT Platforms**: General-purpose platforms lacking maintenance-specific optimization
- **Specialized Predictive Maintenance**: Limited equipment coverage, high implementation complexity
- **Our Differentiation**: Comprehensive multi-equipment support, easy integration, mobile-first design

2.3 Value Proposition

- **For Maintenance Teams**: Proactive maintenance scheduling with mobile-optimized workflows
- **For Operations Managers**: Real-time visibility into equipment health and maintenance efficiency
- **For Plant Engineers**: Data-driven insights for equipment optimization and lifecycle management
- **For Executives**: Measurable ROI through reduced downtime and optimized maintenance spend

3. Target Users and Personas

3.1 Primary Persona: Maintenance Technician (Mike)

Demographics: 35-50 years old, 10+ years industrial maintenance experience **Goals**: - Receive timely alerts about potential equipment issues - Access equipment history and maintenance procedures on mobile device - Complete work orders efficiently with proper documentation **Pain Points**: - Reactive maintenance leads to emergency repairs and overtime - Limited visibility into equipment health between scheduled maintenance - Paper-based work orders and manual documentation **Success Metrics**: Reduced emergency repairs, improved first-time fix rate, faster work order completion

3.2 Secondary Persona: Maintenance Manager (Sarah)

Demographics: 40-55 years old, 15+ years maintenance management experience **Goals**: - Optimize maintenance schedules and resource allocation - Reduce unplanned downtime and maintenance costs - Demonstrate maintenance ROI and performance improvements **Pain Points**: - Difficulty predicting optimal maintenance timing - Limited visibility into maintenance team productivity - Challenges justifying maintenance investments **Success Metrics**: Reduced maintenance costs, improved equipment availability, increased team productivity

3.3 Tertiary Persona: Plant Engineer (David)

Demographics: 30-45 years old, engineering degree, 8+ years industrial experience **Goals**: - Analyze equipment performance trends and optimization opportunities - Support data-driven maintenance strategy development - Integrate predictive maintenance with overall plant operations **Pain Points**: - Limited access to comprehensive equipment performance data - Difficulty correlating maintenance activities with operational performance - Challenges integrating maintenance data with other plant systems **Success Metrics**: Improved equipment reliability, extended asset lifespan, optimized maintenance strategies

4. Business Objectives and Success Metrics

4.1 Primary Business Objectives

1. **Reduce Unplanned Downtime**: Achieve 70% reduction in unplanned equipment downtime
2. **Optimize Maintenance Costs**: Reduce overall maintenance costs by 25% through predictive scheduling
3. **Improve Equipment Reliability**: Increase mean time between failures (MTBF) by 40%
4. **Enhance Safety**: Prevent 95% of potential safety incidents through early failure detection
5. **Increase Operational Efficiency**: Improve overall equipment effectiveness (OEE) by 15%

4.2 Key Performance Indicators (KPIs)

- **Prediction Accuracy**: >90% accuracy for failure predictions with 7-day lead time
- **False Positive Rate**: <5% false positive rate for critical equipment alerts
- **Response Time**: <2 minutes average response time for critical alerts
- **User Adoption**: >85% active usage rate among maintenance personnel
- **ROI Achievement**: 300% ROI within 18 months of implementation

4.3 Success Metrics by User Persona

Maintenance Technicians: - 50% reduction in emergency repair calls - 30% improvement in first-time fix rate - 40% reduction in work order completion time

Maintenance Managers: - 25% reduction in maintenance costs - 70% reduction in unplanned downtime - 20% improvement in maintenance team productivity

Plant Engineers: - 40% increase in equipment MTBF - 15% improvement in overall equipment effectiveness - 30% reduction in equipment-related safety incidents

5. Key Product Features

5.1 Core Features (MVP)

1. **Real-time IoT Data Ingestion**
   - Multi-protocol support (MQTT, OPC-UA, Modbus)
   - Scalable data processing for 10,000+ sensors
   - Edge computing capabilities for local processing
2. **Predictive Analytics Engine**
   - Machine learning models for failure prediction
   - Anomaly detection for multiple sensor types
   - Time series analysis for trend identification

3. **Equipment Health Dashboard**
   - Real-time equipment status visualization
   - Health score trending and alerts
   - Equipment hierarchy and relationship mapping
4. **Mobile Maintenance App**
   - Work order management and completion
   - Equipment inspection checklists
   - Photo and voice note documentation

5.2 Advanced Features (Future Releases)

1. **Maintenance Optimization Engine**
   - AI-powered maintenance scheduling
   - Resource allocation optimization
   - Cost-benefit analysis for maintenance decisions
2. **Advanced Analytics and Reporting**
   - Predictive maintenance ROI analysis
   - Equipment performance benchmarking
   - Maintenance KPI dashboards
3. **Integration Hub**
   - ERP system integration (SAP, Oracle)
   - CMMS integration (Maximo, Maintenance Connection)
   - Business intelligence platform connectivity

6. Technical Requirements and Constraints

6.1 Performance Requirements

- **Data Processing**: Handle 1M+ sensor readings per minute
- **Prediction Latency**: Generate predictions within 30 seconds of data ingestion
- **System Availability**: 99.9% uptime for critical production environments
- **Scalability**: Support 100+ industrial facilities with 50,000+ assets
- **Mobile Performance**: <3 second app load time on industrial mobile devices

6.2 Integration Requirements

- **Industrial Protocols**: OPC-UA, Modbus, MQTT, DNP3, BACnet support
- **Enterprise Systems**: SAP, Oracle, Microsoft Dynamics integration
- **CMMS Platforms**: Maximo, Maintenance Connection, eMaint compatibility
- **Cloud Platforms**: AWS, Azure, Google Cloud deployment options
- **Edge Computing**: Support for industrial edge devices and gateways

6.3 Security and Compliance

- **Industrial Security**: IEC 62443 compliance for industrial cybersecurity
- **Data Protection**: Encryption at rest and in transit
- **Access Control**: Role-based access with multi-factor authentication
- **Audit Trail**: Complete audit logging for all system activities
- **Regulatory Compliance**: ISO 55000 asset management standard alignment

7. Business Model and Pricing Strategy

7.1 Revenue Model

- **SaaS Subscription**: Tiered pricing based on number of monitored assets
- **Professional Services**: Implementation, training, and customization services
- **Support and Maintenance**: Premium support packages with SLA guarantees
- **Data Analytics Services**: Advanced analytics and consulting services

7.2 Pricing Tiers

- **Starter**: $50/asset/month (up to 100 assets)
- **Professional**: $35/asset/month (100-1,000 assets)
- **Enterprise**: $25/asset/month (1,000+ assets)
- **Custom**: Volume pricing for large deployments

7.3 Go-to-Market Strategy

- **Direct Sales**: Enterprise sales team for large manufacturing accounts
- **Channel Partners**: Industrial automation integrators and consultants
- **Digital Marketing**: Content marketing and industry conference presence
- **Pilot Programs**: Proof-of-concept implementations with key prospects

8. Risk Assessment and Mitigation

8.1 Technical Risks

- **Data Quality**: Poor sensor data quality affecting prediction accuracy
  - *Mitigation*: Robust data validation and cleansing algorithms
- **Integration Complexity**: Challenges integrating with legacy industrial systems
  - *Mitigation*: Comprehensive API library and professional services support
- **Scalability**: Performance issues with large-scale deployments
  - *Mitigation*: Cloud-native architecture with auto-scaling capabilities

8.2 Business Risks

- **Market Competition**: Established players with existing customer relationships
  - *Mitigation*: Focus on superior user experience and faster implementation
- **Customer Adoption**: Resistance to change from traditional maintenance practices
  - *Mitigation*: Comprehensive training programs and change management support
- **Economic Downturn**: Reduced capital spending on new technology initiatives
  - *Mitigation*: Clear ROI demonstration and flexible pricing models

8.3 Operational Risks

- **Talent Acquisition**: Difficulty hiring specialized IoT and ML engineers
  - *Mitigation*: Competitive compensation packages and remote work options
- **Data Security**: Cybersecurity threats to industrial systems
  - *Mitigation*: Comprehensive security framework and regular security audits
- **Regulatory Changes**: Evolving industrial cybersecurity regulations
  - *Mitigation*: Proactive compliance monitoring and adaptive security measures

9. Success Criteria and Measurement

9.1 Product Success Metrics

- **Customer Satisfaction**: Net Promoter Score (NPS) >50
- **Product Adoption**: >85% feature utilization rate
- **Customer Retention**: <5% annual churn rate
- **Revenue Growth**: 100% year-over-year revenue growth
- **Market Share**: 10% market share in target segments within 3 years

9.2 User Success Metrics

- **User Engagement**: >80% daily active users among maintenance staff
- **Task Completion**: >95% work order completion rate through mobile app
- **User Satisfaction**: >4.5/5 user satisfaction rating
- **Training Effectiveness**: <2 hours average time to productivity for new users
- **Support Efficiency**: <24 hour average response time for support requests

9.3 Business Impact Metrics

- **Downtime Reduction**: 70% reduction in unplanned downtime
- **Cost Savings**: 25% reduction in maintenance costs
- **Safety Improvement**: 95% reduction in equipment-related safety incidents
- **Efficiency Gains**: 15% improvement in overall equipment effectiveness
- **ROI Achievement**: 300% ROI within 18 months of implementation

This PRD establishes the foundation for developing a comprehensive IoT predictive maintenance platform that addresses real industrial needs while delivering measurable business value through advanced AI and IoT technologies. # Functional Requirements Document (FRD) ## IoT Predictive Maintenance Platform

Building upon PRD requirements for detailed functional specifications

ETVX Framework

ENTRY CRITERIA

- âœ… PRD completed and approved by industrial stakeholders and maintenance teams
- âœ… Business objectives and success metrics clearly defined (70% downtime reduction, 25% cost reduction)
- âœ… Target users and their operational workflows documented (maintenance technicians, managers, engineers)
- âœ… Key product features identified and prioritized for predictive maintenance operations
- âœ… Technical feasibility assessment for IoT data processing and ML prediction completed
- âœ… Industrial integration requirements (OPC-UA, Modbus, MQTT) documented

TASK

Transform PRD business requirements into detailed, testable functional specifications that define exactly what the IoT predictive maintenance platform must do, including real-time sensor data processing workflows, ML model behaviors, maintenance optimization logic, user interactions, system integrations, and industrial compliance features.

VERIFICATION & VALIDATION

Verification Checklist: - [ ] Each functional requirement is traceable to PRD business objectives - [ ] Requirements are unambiguous and testable with specific acceptance criteria - [ ] All maintenance team workflows are covered end-to-end - [ ] Integration points with industrial systems and CMMS platforms defined - [ ] Error handling and edge cases specified for industrial IoT processing - [ ] Requirements follow consistent numbering (FR-001, FR-002, etc.) with clear categorization

Validation Criteria: - [ ] Requirements satisfy all PRD success metrics (>90% prediction accuracy, <5% false positives) - [ ] User personas can achieve their maintenance goals through defined functions - [ ] System behaviors align with industrial standards and safety requirements - [ ] IoT and ML engineering team confirms implementability of all predictive requirements - [ ] Requirements review completed with maintenance technicians, managers, and plant engineers - [ ] Integration requirements validated with industrial automation system architects

EXIT CRITERIA

- âœ… All functional requirements documented with unique identifiers and acceptance criteria
- âœ… Requirements traceability matrix to PRD completed with full coverage
- âœ… User acceptance criteria defined for each requirement with measurable outcomes
- âœ… Industrial system integration and compliance requirements clearly specified
- âœ… Foundation established for non-functional requirements development

---

Reference to Previous Documents

This FRD translates the business objectives and product features defined in the **PRD** into specific functional requirements: - **PRD Target Users** â†’ Detailed maintenance technician workflows, manager interfaces, engineer analytics - **PRD Key Features** â†’ Granular IoT data processing specifications, ML model requirements, mobile app functionality - **PRD Success Metrics** â†’ Measurable functional capabilities (>90% prediction accuracy, <2 min response time, <5% false positives) - **PRD Constraints** â†’ Technical integration, industrial compliance, and real-time processing requirements

1. IoT Data Ingestion and Processing Module

1.1 Multi-Protocol Data Ingestion

- **FR-001**: System SHALL ingest real-time sensor data from OPC-UA servers with <1 second latency
- **FR-002**: System SHALL support Modbus TCP/RTU protocol for legacy industrial equipment communication
- **FR-003**: System SHALL process MQTT messages from IoT gateways with QoS levels 0, 1, and 2
- **FR-004**: System SHALL handle DNP3 protocol for utility and power generation equipment
- **FR-005**: System SHALL support BACnet protocol for building automation and HVAC systems
- **FR-006**: System SHALL validate incoming sensor data format and reject malformed messages with error logging

1.2 Sensor Data Management

- **FR-007**: System SHALL support 50+ sensor types including vibration, temperature, pressure, flow, current, voltage
- **FR-008**: System SHALL process sensor readings at frequencies from 1Hz to 10kHz based on equipment requirements
- **FR-009**: System SHALL store raw sensor data with microsecond timestamp precision
- **FR-010**: System SHALL apply sensor calibration factors and unit conversions automatically
- **FR-011**: System SHALL detect and flag sensor malfunctions or communication failures
- **FR-012**: System SHALL support sensor metadata management including location, specifications, and maintenance history

1.3 Edge Computing Capabilities

- **FR-013**: System SHALL deploy edge processing nodes for local data preprocessing and filtering
- **FR-014**: System SHALL perform real-time data aggregation and statistical calculations at the edge
- **FR-015**: System SHALL support offline operation with local data storage when connectivity is lost
- **FR-016**: System SHALL synchronize edge data with cloud systems when connectivity is restored

- **FR-017**: System SHALL support edge-based anomaly detection for critical equipment monitoring
- **FR-018**: System SHALL manage edge device configuration and software updates remotely

2. Predictive Analytics and Machine Learning Module

2.1 Anomaly Detection Engine

- **FR-019**: System SHALL implement statistical process control (SPC) for real-time anomaly detection
- **FR-020**: System SHALL use isolation forest algorithms for multivariate anomaly detection
- **FR-021**: System SHALL apply autoencoder neural networks for complex pattern anomaly identification
- **FR-022**: System SHALL detect gradual drift patterns using trend analysis algorithms
- **FR-023**: System SHALL identify sudden change points in sensor data streams
- **FR-024**: System SHALL provide anomaly severity scoring from 1-10 with confidence intervals

2.2 Failure Prediction Models

- **FR-025**: System SHALL implement LSTM neural networks for time series failure prediction
- **FR-026**: System SHALL use random forest models for multi-sensor failure classification
- **FR-027**: System SHALL apply survival analysis for remaining useful life (RUL) estimation
- **FR-028**: System SHALL implement ensemble methods combining multiple prediction algorithms
- **FR-029**: System SHALL provide failure probability scores with prediction confidence levels
- **FR-030**: System SHALL generate predictions with 1-day, 7-day, and 30-day time horizons

2.3 Equipment Health Scoring

- **FR-031**: System SHALL calculate composite equipment health scores from 0-100
- **FR-032**: System SHALL weight health score components based on equipment criticality
- **FR-033**: System SHALL track health score trends and rate of change over time
- **FR-034**: System SHALL provide health score breakdown by subsystem and component
- **FR-035**: System SHALL compare equipment health against fleet averages and benchmarks
- **FR-036**: System SHALL generate health score reports with historical trending analysis

3. Maintenance Optimization Engine

3.1 Maintenance Scheduling Optimization

- **FR-037**: System SHALL optimize maintenance schedules using constraint satisfaction algorithms
- **FR-038**: System SHALL consider equipment criticality, spare parts availability, and technician skills
- **FR-039**: System SHALL minimize total maintenance costs while meeting reliability targets
- **FR-040**: System SHALL support maintenance window constraints and production schedule integration
- **FR-041**: System SHALL provide alternative scheduling scenarios with cost-benefit analysis
- **FR-042**: System SHALL automatically reschedule maintenance based on changing equipment conditions

3.2 Resource Allocation Management

- **FR-043**: System SHALL assign maintenance tasks based on technician skills and availability
- **FR-044**: System SHALL optimize spare parts inventory levels using demand forecasting
- **FR-045**: System SHALL coordinate maintenance activities across multiple equipment systems
- **FR-046**: System SHALL provide resource utilization reports and capacity planning
- **FR-047**: System SHALL support emergency maintenance prioritization and resource reallocation
- **FR-048**: System SHALL track maintenance resource costs and budget utilization

4. Real-Time Monitoring and Alerting Module

4.1 Equipment Status Dashboard

- **FR-049**: System SHALL provide real-time equipment status visualization with color-coded health indicators
- **FR-050**: System SHALL display equipment hierarchy with parent-child relationships and dependencies
- **FR-051**: System SHALL show live sensor readings with historical trending charts
- **FR-052**: System SHALL provide equipment location mapping with facility floor plans
- **FR-053**: System SHALL support customizable dashboard layouts for different user roles
- **FR-054**: System SHALL enable drill-down from fleet overview to individual equipment details

4.2 Alert Management System

- **FR-055**: System SHALL generate alerts based on configurable thresholds and ML predictions
- **FR-056**: System SHALL prioritize alerts using equipment criticality and failure impact assessment
- **FR-057**: System SHALL support alert escalation rules with time-based escalation paths
- **FR-058**: System SHALL provide alert acknowledgment and resolution tracking
- **FR-059**: System SHALL send notifications via email, SMS, mobile push, and integration APIs
- **FR-060**: System SHALL support alert suppression during planned maintenance activities

4.3 Performance Analytics

- **FR-061**: System SHALL calculate overall equipment effectiveness (OEE) metrics in real-time
- **FR-062**: System SHALL track mean time between failures (MTBF) and mean time to repair (MTTR)
- **FR-063**: System SHALL provide equipment performance benchmarking against industry standards
- **FR-064**: System SHALL generate performance trend analysis with statistical significance testing
- **FR-065**: System SHALL support custom KPI definition and calculation for specific equipment types
- **FR-066**: System SHALL provide automated performance reporting with configurable schedules

5. Mobile Maintenance Application

5.1 Work Order Management

- **FR-067**: System SHALL provide mobile work order creation, assignment, and completion workflows
- **FR-068**: System SHALL support offline work order access and synchronization when connectivity returns
- **FR-069**: System SHALL enable work order status updates with timestamp and location tracking
- **FR-070**: System SHALL provide work order history and related maintenance documentation access
- **FR-071**: System SHALL support work order approval workflows for high-cost or critical maintenance
- **FR-072**: System SHALL integrate work orders with time tracking and labor cost calculation

5.2 Equipment Inspection Tools

- **FR-073**: System SHALL provide digital inspection checklists with conditional logic and branching
- **FR-074**: System SHALL support photo capture with automatic equipment and location tagging
- **FR-075**: System SHALL enable voice note recording and transcription for inspection findings
- **FR-076**: System SHALL provide barcode and QR code scanning for equipment identification
- **FR-077**: System SHALL support signature capture for inspection completion and approval
- **FR-078**: System SHALL generate inspection reports with photos, notes, and compliance status

5.3 Maintenance Documentation

- **FR-079**: System SHALL provide access to equipment manuals, procedures, and safety documentation
- **FR-080**: System SHALL support document search and filtering by equipment type and maintenance task
- **FR-081**: System SHALL enable document annotation and feedback submission
- **FR-082**: System SHALL track document access and usage analytics
- **FR-083**: System SHALL support document version control and update notifications
- **FR-084**: System SHALL provide offline document access for critical maintenance procedures

6. Integration and API Module

6.1 CMMS Integration

- **FR-085**: System SHALL integrate with IBM Maximo for work order and asset management synchronization
- **FR-086**: System SHALL connect with Maintenance Connection for maintenance scheduling coordination
- **FR-087**: System SHALL support eMaint integration for maintenance history and parts management
- **FR-088**: System SHALL provide bidirectional data synchronization with configurable field mapping
- **FR-089**: System SHALL handle CMMS integration errors with retry logic and error reporting
- **FR-090**: System SHALL support custom CMMS integration using REST APIs and webhooks

6.2 ERP System Integration

- **FR-091**: System SHALL integrate with SAP for asset master data and financial information
- **FR-092**: System SHALL connect with Oracle ERP for procurement and inventory management
- **FR-093**: System SHALL support Microsoft Dynamics integration for cost center and budget tracking
- **FR-094**: System SHALL provide real-time inventory updates for spare parts consumption
- **FR-095**: System SHALL support purchase requisition creation for maintenance parts and services
- **FR-096**: System SHALL integrate maintenance costs with financial reporting and budgeting systems

6.3 Industrial System Integration

- **FR-097**: System SHALL connect with SCADA systems for operational context and production data
- **FR-098**: System SHALL integrate with historian databases (OSIsoft PI, Wonderware) for historical data
- **FR-099**: System SHALL support MES integration for production schedule and maintenance coordination
- **FR-100**: System SHALL connect with safety systems for lockout/tagout (LOTO) procedures
- **FR-101**: System SHALL integrate with energy management systems for power quality monitoring
- **FR-102**: System SHALL support building management system integration for facility equipment

7. Reporting and Analytics Module

7.1 Operational Reporting

- **FR-103**: System SHALL generate real-time maintenance performance dashboards
- **FR-104**: System SHALL provide predictive maintenance ROI analysis and cost savings reports
- **FR-105**: System SHALL create equipment reliability and availability reports
- **FR-106**: System SHALL generate maintenance team productivity and efficiency reports
- **FR-107**: System SHALL support custom report creation with drag-and-drop interface
- **FR-108**: System SHALL provide automated report scheduling and distribution capabilities

7.2 Compliance and Audit Reporting

- **FR-109**: System SHALL generate ISO 55000 asset management compliance reports
- **FR-110**: System SHALL provide regulatory compliance reports for industry-specific requirements
- **FR-111**: System SHALL create audit trail reports for all system activities and changes
- **FR-112**: System SHALL generate safety compliance reports for equipment-related incidents
- **FR-113**: System SHALL support environmental compliance reporting for emissions and waste
- **FR-114**: System SHALL maintain reporting data retention according to regulatory requirements

8. Configuration and Administration Module

8.1 Equipment Configuration Management

- **FR-115**: System SHALL support equipment hierarchy definition with parent-child relationships
- **FR-116**: System SHALL enable equipment specification and parameter configuration
- **FR-117**: System SHALL provide equipment criticality classification and impact assessment
- **FR-118**: System SHALL support equipment grouping and tagging for analysis and reporting
- **FR-119**: System SHALL enable equipment lifecycle tracking from installation to retirement
- **FR-120**: System SHALL provide equipment configuration version control and change tracking

8.2 User and Security Management

- **FR-121**: System SHALL support role-based access control with granular permissions
- **FR-122**: System SHALL provide user authentication with multi-factor authentication support
- **FR-123**: System SHALL enable user activity logging and audit trail generation
- **FR-124**: System SHALL support LDAP/Active Directory integration for user management
- **FR-125**: System SHALL provide password policy enforcement and account lockout protection
- **FR-126**: System SHALL support API key management for system integrations # Non-Functional Requirements Document (NFRD) ## IoT Predictive Maintenance Platform

Building upon PRD and FRD for system quality attributes and constraints

ETVX Framework

ENTRY CRITERIA

- âœ… PRD completed with quantified success metrics (70% downtime reduction, 25% cost reduction, >90% prediction accuracy)
- âœ… FRD completed with all functional requirements defined (FR-001 to FR-126)
- âœ… Industrial IoT system load patterns and sensor data volumes documented (1M+ readings/minute)
- âœ… Industrial compliance requirements identified (IEC 62443, ISO 55000, industrial safety standards)
- âœ… Technology constraints and security requirements documented for industrial environments
- âœ… Business continuity and operational resilience requirements established

TASK

Define system quality attributes, performance benchmarks, security requirements, scalability targets, and operational constraints that ensure the IoT predictive maintenance platform can deliver functional requirements with acceptable quality in harsh industrial environments while meeting stringent safety, security, and reliability standards.

VERIFICATION & VALIDATION

Verification Checklist: - [ ] All NFRs are quantifiable and measurable with specific metrics and thresholds - [ ] Performance targets align with PRD success metrics (<2 min response time, >90% accuracy) - [ ] Security requirements meet industrial cybersecurity standards (IEC 62443) - [ ] Scalability requirements support projected sensor volumes (1M+ readings/minute) - [ ] Each NFR is traceable to functional requirements and business objectives - [ ] Compliance requirements are comprehensive and auditable for industrial regulations

Validation Criteria: - [ ] Performance targets are achievable with proposed IoT and ML architecture - [ ] Security requirements satisfy industrial cybersecurity and safety standards - [ ] Scalability projections align with industrial IoT deployment growth forecasts - [ ] Availability requirements validated with industrial operational needs (99.9% uptime) - [ ] Infrastructure team confirms operational feasibility in industrial environments - [ ] Compliance team validates regulatory adherence and audit requirements

EXIT CRITERIA

- âœ… All quality attributes quantified with specific metrics, thresholds, and measurement methods
- âœ… Performance benchmarks established for each system component with SLA definitions
- âœ… Security and compliance requirements fully documented with implementation guidelines
- âœ… Scalability, reliability, and availability targets defined with monitoring requirements
- âœ… Foundation established for system architecture design with industrial constraints

---

Reference to Previous Documents

This NFRD defines quality attributes and constraints based on **ALL** previous requirements: - **PRD Business Objectives** â†’ Performance targets (70% downtime reduction, 25% cost reduction, >90% prediction accuracy) - **PRD Success Metrics** â†’ Quantified NFRs (<2 min response time, 99.9% uptime, <5% false positives) - **PRD Target Users** â†’ Usability and accessibility requirements for maintenance technicians, managers, engineers - **PRD Industrial Constraints** â†’ Security and compliance requirements (IEC 62443, ISO 55000, industrial safety) - **FRD IoT Data Ingestion (FR-001 to FR-018)** â†’ Performance requirements for multi-protocol sensor data processing - **FRD Predictive Analytics (FR-019 to FR-036)** â†’ Performance requirements for ML model inference and anomaly detection - **FRD Maintenance Optimization (FR-037 to FR-048)** â†’ Performance requirements for scheduling algorithms and resource allocation - **FRD Real-Time Monitoring (FR-049 to FR-066)** â†’ Performance requirements for dashboard responsiveness and alert processing - **FRD Mobile Application (FR-067 to FR-084)** â†’ Performance requirements for mobile app responsiveness and offline capability - **FRD Integration (FR-085 to FR-102)** â†’ Reliability requirements for CMMS, ERP, and industrial system integration - **FRD Reporting (FR-103 to FR-114)** â†’ Performance requirements for real-time analytics and compliance reporting - **FRD Administration (FR-115 to FR-126)** â†’ Security requirements for user management and system configuration

1. Performance Requirements

1.1 IoT Data Processing Performance

- **NFR-001**: Sensor data ingestion SHALL process 1M+ sensor readings per minute with <1 second latency
- **NFR-002**: Multi-protocol data processing SHALL handle OPC-UA, Modbus, MQTT, DNP3 simultaneously
- **NFR-003**: Edge processing nodes SHALL perform local analytics within <100ms of data receipt
- **NFR-004**: Data validation and cleansing SHALL complete within <500ms for 99.9% of sensor readings
- **NFR-005**: System SHALL support 50,000+ concurrent sensor connections with linear scalability
- **NFR-006**: Historical data queries SHALL return results within <5 seconds for 1-year time ranges

1.2 Machine Learning Performance

- **NFR-007**: Anomaly detection algorithms SHALL process sensor data within <2 seconds of ingestion
- **NFR-008**: Failure prediction models SHALL generate predictions within <30 seconds for equipment analysis
- **NFR-009**: Model training SHALL complete within 8 hours for daily model updates
- **NFR-010**: Ensemble model inference SHALL complete within <1 second for real-time predictions
- **NFR-011**: Health score calculations SHALL update within <10 seconds of sensor data changes
- **NFR-012**: Model performance monitoring SHALL detect drift within 1 hour of occurrence

1.3 System Response Time

- **NFR-013**: Web dashboard SHALL load within <3 seconds for 95% of requests
- **NFR-014**: Mobile application SHALL respond within <2 seconds for work order operations
- **NFR-015**: Alert notifications SHALL be delivered within <30 seconds of trigger conditions
- **NFR-016**: Equipment search and filtering SHALL return results within <1 second
- **NFR-017**: Report generation SHALL complete within 60 seconds for standard reports
- **NFR-018**: API response time SHALL be <500ms for 99% of integration calls

2. Scalability Requirements

2.1 Industrial IoT Scaling

- **NFR-019**: System SHALL scale to monitor 100,000+ industrial assets across 500+ facilities
- **NFR-020**: System SHALL support 10,000+ concurrent users including mobile and web access
- **NFR-021**: System SHALL handle 10TB+ of sensor data per day with automated data lifecycle management
- **NFR-022**: System SHALL scale ML model serving to 100,000+ predictions per minute
- **NFR-023**: System SHALL support 1,000+ concurrent model training jobs for different equipment types
- **NFR-024**: System SHALL auto-scale compute resources based on sensor data volume (10-1000% capacity)

2.2 Data Volume and Storage Scaling

- **NFR-025**: System SHALL store 5+ years of sensor data for trend analysis and compliance
- **NFR-026**: System SHALL handle 100TB+ of historical data with efficient querying capabilities
- **NFR-027**: System SHALL support real-time data ingestion of 100GB+ per hour
- **NFR-028**: System SHALL maintain 99.99% data availability across all storage tiers
- **NFR-029**: System SHALL support automated data archiving and retrieval for compliance
- **NFR-030**: System SHALL provide data compression achieving 80%+ storage reduction

3. Reliability & Availability Requirements

3.1 System Availability

- **NFR-031**: System availability SHALL be 99.9% (max 8.77 hours downtime/year)
- **NFR-032**: Planned maintenance windows SHALL not exceed 4 hours monthly
- **NFR-033**: Mean Time Between Failures (MTBF) SHALL be â‰¥4380 hours (6 months)
- **NFR-034**: Mean Time To Recovery (MTTR) SHALL be â‰¤30 minutes for critical system failures
- **NFR-035**: System SHALL support rolling updates with minimal service disruption
- **NFR-036**: System SHALL maintain service during single data center or edge node failures

3.2 Data Integrity and Consistency

- **NFR-037**: Sensor data integrity SHALL be 99.999% with automated consistency checks
- **NFR-038**: Data backup SHALL occur every 30 minutes with 99.99% backup success rate
- **NFR-039**: Recovery Point Objective (RPO) SHALL be â‰¤30 minutes for all critical data
- **NFR-040**: Recovery Time Objective (RTO) SHALL be â‰¤1 hour for full system recovery
- **NFR-041**: Cross-region data replication SHALL maintain â‰¤5 second synchronization lag
- **NFR-042**: Audit trail SHALL be immutable and tamper-evident for compliance requirements

4. Security Requirements

4.1 Industrial Cybersecurity

- **NFR-043**: System SHALL comply with IEC 62443 industrial cybersecurity standards
- **NFR-044**: All industrial communications SHALL be encrypted using TLS 1.3 or higher
- **NFR-045**: Network segmentation SHALL isolate OT networks from IT networks
- **NFR-046**: Industrial protocol security SHALL implement authentication and authorization
- **NFR-047**: Edge devices SHALL support secure boot and firmware integrity verification
- **NFR-048**: System SHALL implement defense-in-depth security architecture

4.2 Data Protection and Encryption

- **NFR-049**: All sensor data SHALL be encrypted at rest using AES-256 encryption
- **NFR-050**: All data in transit SHALL be encrypted using industry-standard protocols
- **NFR-051**: Encryption key management SHALL use hardware security modules (HSMs)
- **NFR-052**: Sensitive configuration data SHALL be encrypted with separate key management
- **NFR-053**: Database encryption SHALL use transparent data encryption (TDE) with key rotation
- **NFR-054**: Backup data SHALL be encrypted with separate encryption keys

4.3 Access Control and Authentication

- **NFR-055**: System SHALL implement multi-factor authentication (MFA) for all user access
- **NFR-056**: System SHALL support SAML 2.0 and OAuth 2.0 for enterprise SSO integration
- **NFR-057**: Role-based access control (RBAC) SHALL support 100+ granular permissions
- **NFR-058**: Privileged access SHALL require additional authentication and approval workflows
- **NFR-059**: Session management SHALL enforce 4-hour idle timeout and 12-hour maximum session
- **NFR-060**: API authentication SHALL use mutual TLS and JWT tokens with short expiration

5. Industrial Compliance Requirements

5.1 Asset Management Compliance

- **NFR-061**: System SHALL comply with ISO 55000 asset management standards
- **NFR-062**: System SHALL meet ISO 14224 reliability data collection standards
- **NFR-063**: System SHALL adhere to IEC 61508 functional safety requirements
- **NFR-064**: System SHALL comply with OSHA maintenance safety regulations
- **NFR-065**: System SHALL meet API 580 risk-based inspection standards for oil & gas
- **NFR-066**: System SHALL support NERC CIP compliance for electric utility operations

5.2 Industrial Safety and Environmental Compliance

- **NFR-067**: System SHALL comply with IEC 61511 safety instrumented systems standards
- **NFR-068**: System SHALL meet EPA environmental monitoring and reporting requirements
- **NFR-069**: System SHALL support ATEX compliance for explosive atmosphere equipment
- **NFR-070**: System SHALL comply with FDA 21 CFR Part 11 for pharmaceutical manufacturing
- **NFR-071**: System SHALL meet automotive industry IATF 16949 quality standards
- **NFR-072**: System SHALL support nuclear industry 10 CFR 50 Appendix B requirements

5.3 Data Privacy and Protection

- **NFR-073**: System SHALL comply with GDPR requirements for EU operations
- **NFR-074**: System SHALL meet CCPA requirements for California operations
- **NFR-075**: System SHALL support data subject rights (access, rectification, erasure, portability)
- **NFR-076**: System SHALL implement privacy by design principles in all data processing
- **NFR-077**: System SHALL maintain data processing records for regulatory audits
- **NFR-078**: System SHALL support cross-border data transfer compliance

6. Performance Monitoring and Observability

6.1 System Monitoring

- **NFR-079**: System SHALL provide real-time performance metrics with â‰¤5 second granularity
- **NFR-080**: System SHALL implement distributed tracing for end-to-end sensor data visibility
- **NFR-081**: System SHALL maintain 99.9% monitoring system availability
- **NFR-082**: System SHALL provide automated alerting with â‰¤60 second notification time
- **NFR-083**: System SHALL support custom dashboards and visualization for different user roles
- **NFR-084**: System SHALL maintain performance baselines and anomaly detection

6.2 Industrial Metrics Monitoring

- **NFR-085**: System SHALL track prediction accuracy with real-time model performance metrics
- **NFR-086**: System SHALL monitor false positive rates with automated threshold alerting
- **NFR-087**: System SHALL provide business impact metrics (downtime prevented, cost savings)
- **NFR-088**: System SHALL track maintenance team productivity and efficiency metrics
- **NFR-089**: System SHALL monitor equipment reliability metrics (MTBF, MTTR, availability)
- **NFR-090**: System SHALL provide regulatory compliance metrics and audit trail completeness

7. Usability and User Experience Requirements

7.1 Maintenance Technician Interface

- **NFR-091**: Maintenance technician training time SHALL be â‰¤4 hours for basic system proficiency
- **NFR-092**: Mobile application SHALL support touch interface optimized for industrial gloves
- **NFR-093**: Interface SHALL be accessible according to WCAG 2.1 AA standards
- **NFR-094**: System SHALL provide contextual help and guided workflows for complex tasks
- **NFR-095**: Mobile interface SHALL support landscape and portrait orientations
- **NFR-096**: System SHALL provide voice input capabilities for hands-free operation

7.2 Manager and Engineer Interface

- **NFR-097**: Management dashboards SHALL load within â‰¤5 seconds with real-time data
- **NFR-098**: System SHALL provide drill-down capabilities from summary to detailed views
- **NFR-099**: Reports SHALL be exportable in multiple formats (PDF, Excel, CSV, PowerBI)
- **NFR-100**: System SHALL support scheduled report delivery via email and secure portals
- **NFR-101**: Interface SHALL support multi-language localization for global operations
- **NFR-102**: System SHALL provide role-based customization of dashboards and reports

8. Integration and Interoperability Requirements

8.1 Industrial System Integration

- **NFR-103**: Integration SHALL support 99.9% message delivery success rate
- **NFR-104**: System SHALL handle integration failures with automatic retry and circuit breaker patterns
- **NFR-105**: Integration SHALL support multiple industrial protocols simultaneously
- **NFR-106**: System SHALL provide integration monitoring with end-to-end data flow tracking
- **NFR-107**: Integration SHALL support rate limiting and throttling to protect legacy systems
- **NFR-108**: System SHALL maintain integration SLAs with industrial systems (â‰¤1 second response time)

8.2 Enterprise System Integration

- **NFR-109**: ERP integration SHALL have 99.5% availability with fallback mechanisms
- **NFR-110**: System SHALL support API versioning and backward compatibility for 3+ years
- **NFR-111**: Integration SHALL implement exponential backoff and jitter for retry mechanisms
- **NFR-112**: System SHALL provide webhook delivery with guaranteed delivery and replay capabilities
- **NFR-113**: Integration SHALL support batch and real-time data synchronization modes
- **NFR-114**: System SHALL maintain integration security with mutual authentication and encryption

9. Environmental and Operational Requirements

9.1 Industrial Environment Requirements

- **NFR-115**: Edge devices SHALL operate in temperature ranges from -40Â°C to +70Â°C
- **NFR-116**: System SHALL support IP65-rated enclosures for harsh industrial environments
- **NFR-117**: Edge computing SHALL function with 95-99% humidity and dust exposure
- **NFR-118**: System SHALL support electromagnetic interference (EMI) immunity per IEC 61000
- **NFR-119**: Edge devices SHALL support power input ranges from 12V to 48V DC
- **NFR-120**: System SHALL function during power fluctuations and brief outages

9.2 Deployment and Infrastructure Requirements

- **NFR-121**: System SHALL support hybrid cloud deployment with on-premises edge computing
- **NFR-122**: System SHALL support containerized deployment with Kubernetes orchestration
- **NFR-123**: System SHALL implement infrastructure as code with automated provisioning
- **NFR-124**: System SHALL support auto-scaling based on sensor data volume and system load
- **NFR-125**: System SHALL optimize resource utilization achieving 70%+ average CPU utilization
- **NFR-126**: System SHALL support air-gapped deployment for high-security industrial facilities

10. Disaster Recovery and Business Continuity

10.1 Disaster Recovery

- **NFR-127**: System SHALL support automated failover to secondary data center within â‰¤15 minutes
- **NFR-128**: Disaster recovery testing SHALL be performed quarterly with documented results
- **NFR-129**: System SHALL maintain warm-standby replicas with â‰¤5 minute data lag
- **NFR-130**: Recovery procedures SHALL be automated with minimal manual intervention
- **NFR-131**: System SHALL support geographic distribution across 3+ availability zones
- **NFR-132**: Backup and recovery SHALL support point-in-time recovery for any time within 90 days

10.2 Business Continuity

- **NFR-133**: System SHALL maintain core monitoring capabilities during partial system failures
- **NFR-134**: System SHALL support degraded mode operation with reduced functionality
- **NFR-135**: Business continuity plan SHALL be tested semi-annually with full stakeholder participation
- **NFR-136**: System SHALL provide emergency procedures for manual equipment monitoring
- **NFR-137**: Communication plan SHALL notify stakeholders within â‰¤30 minutes of major incidents
- **NFR-138**: System SHALL maintain 72-hour operational resilience for extended outages # Architecture Diagram (AD) ## IoT Predictive Maintenance Platform

Building upon PRD, FRD, and NFRD for comprehensive system architecture

ETVX Framework

ENTRY CRITERIA

- âœ… PRD completed with business objectives and success metrics
- âœ… FRD completed with 126 functional requirements (FR-001 to FR-126)
- âœ… NFRD completed with 138 non-functional requirements (NFR-001 to NFR-138)
- âœ… Performance targets defined (<2 min response time, 1M+ readings/minute, 99.9% availability)
- âœ… Industrial compliance requirements documented (IEC 62443, ISO 55000, industrial safety)
- âœ… Integration requirements with industrial systems and CMMS platforms established

TASK

Design comprehensive system architecture that satisfies all functional and non-functional requirements, including real-time IoT data processing, ML model serving, maintenance optimization, industrial system integration, and edge computing for harsh industrial environments.

VERIFICATION & VALIDATION

Verification Checklist: - [ ] Architecture supports all 126 functional requirements from FRD - [ ] Design meets all 138 non-functional requirements from NFRD - [ ] Performance targets achievable with proposed architecture (<2 min response, 1M+ readings/min) - [ ] Security architecture meets IEC 62443 and industrial cybersecurity standards - [ ] Scalability design supports projected sensor volumes and industrial facility growth - [ ] Integration patterns support industrial protocols and enterprise system requirements

Validation Criteria: - [ ] Architecture review completed with industrial automation and IoT system architects - [ ] Security design validated with industrial cybersecurity and compliance teams - [ ] Performance modeling confirms latency and throughput targets for sensor processing - [ ] Technology stack validated with IoT, ML engineering, and industrial integration teams - [ ] Cost modeling completed for edge computing infrastructure and cloud operational expenses - [ ] Disaster recovery and business continuity capabilities validated for industrial operations

EXIT CRITERIA

- âœ… Complete system architecture documented with component interactions
- âœ… Technology stack defined with specific versions and industrial-grade configurations
- âœ… Data flow diagrams created for all major IoT processing workflows
- âœ… Security architecture documented with industrial threat model and controls
- âœ… Foundation established for detailed high-level design development

---

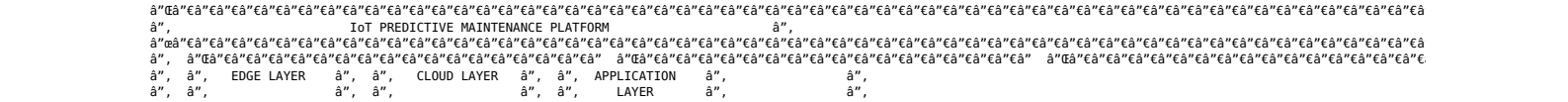
Reference to Previous Documents

This Architecture Diagram implements requirements from **ALL** previous documents: - **PRD Success Metrics** â†’ Architecture designed for 70% downtime reduction, 25% cost reduction, >90% prediction accuracy - **PRD Target Users** â†’ User interface architecture for maintenance technicians, managers, plant

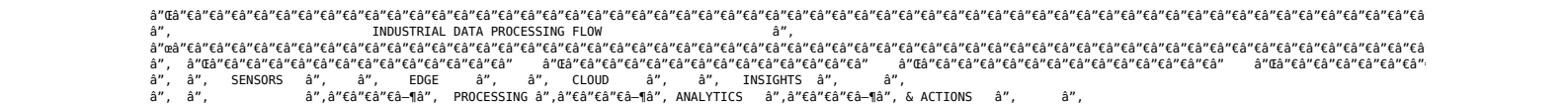
engineers - **PRD Key Features** â†' Technical architecture for IoT data ingestion, predictive analytics, mobile apps, maintenance optimization - **FRD IoT Data Processing (FR-001-018)** â†' Multi-protocol ingestion architecture with OPC-UA, Modbus, MQTT, edge computing - **FRD Predictive Analytics (FR-019-036)** â†' ML serving architecture with anomaly detection, failure prediction, health scoring - **FRD Maintenance Optimization (FR-037-048)** â†' Optimization engine architecture with scheduling algorithms and resource allocation - **FRD Real-Time Monitoring (FR-049-066)** â†' Dashboard architecture with real-time visualization and alert management - **FRD Mobile Application (FR-067-084)** â†' Mobile architecture with offline capability and work order management - **FRD Integration (FR-085-102)** â†' API gateway, message queues, and secure integration with CMMS, ERP, industrial systems - **FRD Reporting (FR-103-114)** â†' Analytics architecture with real-time dashboards and compliance reporting - **NFRD Performance (NFR-001-018)** â†' High-performance architecture with edge computing, caching, load balancing - **NFRD Scalability (NFR-019-030)** â†' Auto-scaling architecture with horizontal scaling and data partitioning - **NFRD Reliability (NFR-031-042)** â†' Fault-tolerant architecture with redundancy, backup, and disaster recovery - **NFRD Security (NFR-043-060)** â†' Industrial cybersecurity architecture with IEC 62443 compliance - **NFRD Compliance (NFR-061-078)** â†' Compliance-ready architecture with audit trails, data governance, safety controls

1. High-Level System Architecture

1.1 Overall Architecture Pattern



1.2 Industrial IoT Data Flow Architecture



2. Technology Stack Architecture

2.1 Edge Computing Layer

- **Edge Gateways**: Dell Edge Gateway 3000/5000 series with Ubuntu Core
- **Container Runtime**: Docker with Kubernetes K3s for edge orchestration
- **IoT Protocols**: Eclipse Mosquitto (MQTT), OPC-UA Server, Modbus TCP/RTU
- **Edge Analytics**: Apache Kafka Streams + TensorFlow Lite
- **Local Storage**: InfluxDB for time-series data + SQLite for configuration

2.2 Cloud Infrastructure Layer

- **Cloud Platform**: AWS/Azure/GCP with multi-region deployment
- **Container Orchestration**: Amazon EKS/Azure AKS/Google GKE
- **Message Streaming**: Apache Kafka + Amazon Kinesis/Azure Event Hubs
- **Data Processing**: Apache Spark + Apache Flink for stream processing
- **API Gateway**: AWS API Gateway/Azure API Management with Kong

2.3 Machine Learning Stack

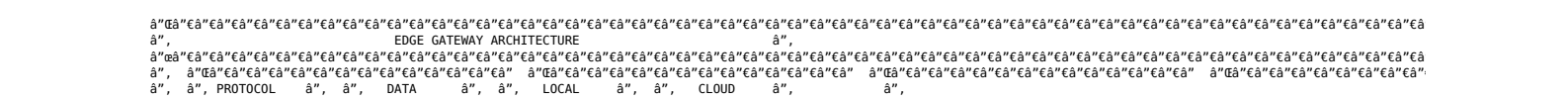
- **ML Framework**: TensorFlow + PyTorch + Scikit-learn
- **Model Serving**: TensorFlow Serving + MLflow Model Registry
- **Feature Store**: Feast + Redis + Apache Cassandra
- **ML Pipeline**: Kubeflow + Apache Airflow + MLOps automation
- **Model Monitoring**: Evidently AI + Prometheus + Grafana

2.4 Database Architecture

- **Time Series**: InfluxDB + TimescaleDB for sensor data
- **OLTP Database**: PostgreSQL with read replicas
- **OLAP Database**: ClickHouse for analytics and reporting
- **Graph Database**: Neo4j for equipment relationships and dependencies
- **Cache Layer**: Redis Cluster + Memcached

3. Edge Computing Architecture

3.1 Industrial Edge Gateway Design

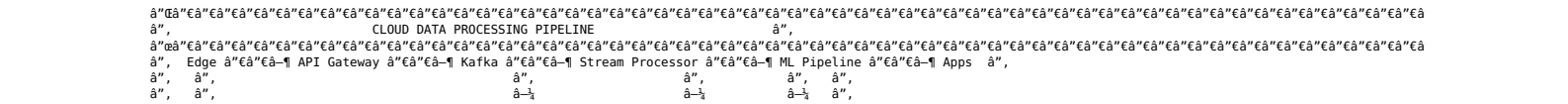


3.2 Edge Analytics Processing

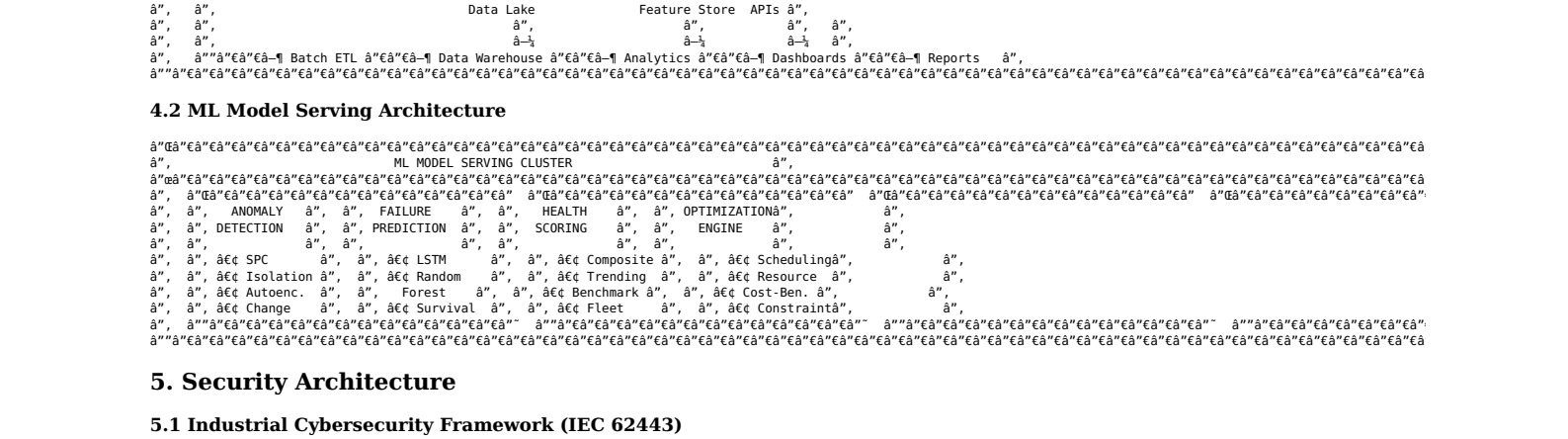
- **Real-time Filtering**: Statistical process control (SPC) for outlier detection
- **Local Anomaly Detection**: Lightweight ML models for immediate alerts
- **Data Aggregation**: Time-based windowing and statistical summaries
- **Buffering Strategy**: Local storage during connectivity outages
- **Security**: TLS encryption and certificate-based authentication

4. Cloud Data Platform Architecture

4.1 Data Ingestion and Processing Pipeline

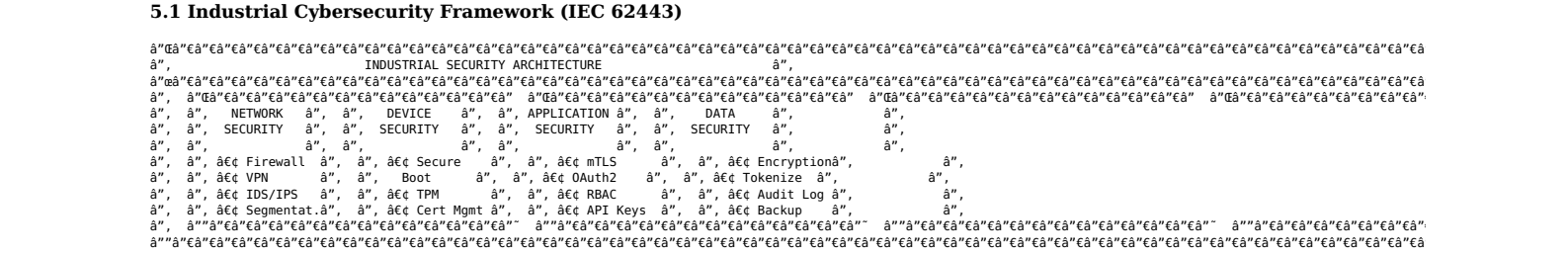


4.2 ML Model Serving Architecture



5. Security Architecture

5.1 Industrial Cybersecurity Framework (IEC 62443)

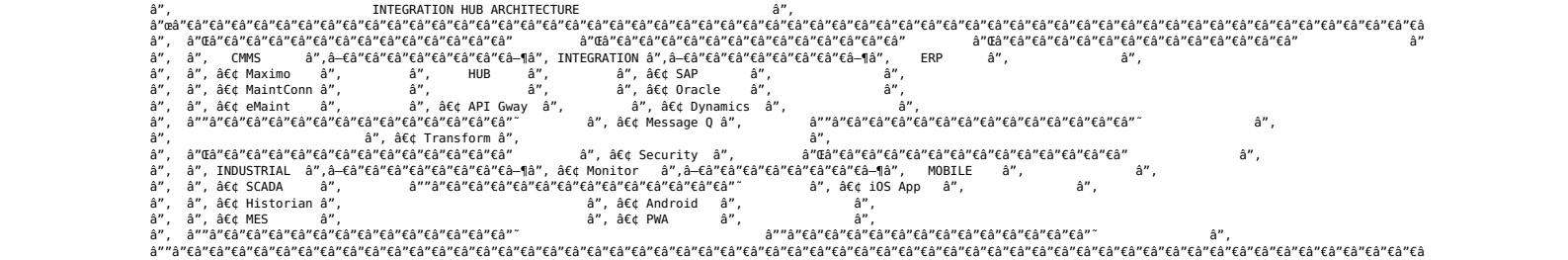


5.2 Zero-Trust Security Model

- **Network Segmentation**: OT/IT network isolation with secure gateways
- **Device Authentication**: Certificate-based authentication for all devices
- **Data Encryption**: AES-256 encryption at rest and TLS 1.3 in transit
- **Access Control**: Multi-factor authentication and role-based permissions
- **Monitoring**: SIEM integration with industrial security monitoring

6. Integration Architecture

6.1 Enterprise System Integration Hub

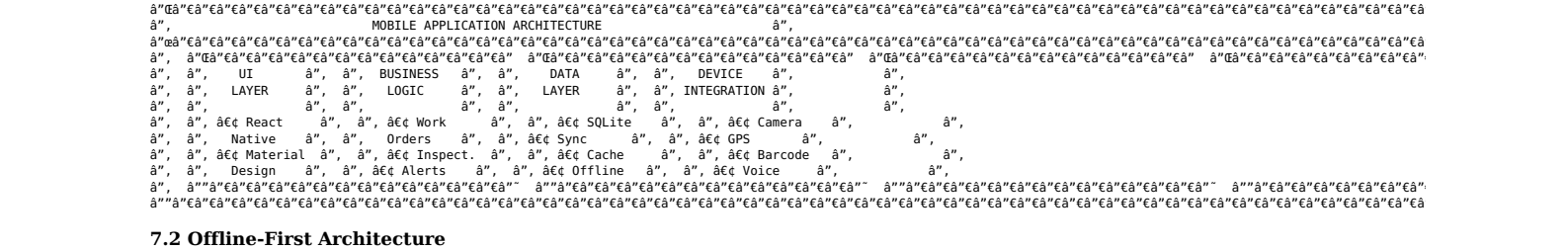


6.2 API Gateway and Service Mesh

- **API Gateway**: Kong with rate limiting, authentication, and monitoring
- **Service Mesh**: Istio for microservices communication and security
- **Message Queue**: Apache Kafka for asynchronous integration
- **Protocol Translation**: Industrial protocol to REST API conversion
- **Error Handling**: Circuit breaker patterns and retry mechanisms

7. Mobile Application Architecture

7.1 Cross-Platform Mobile Architecture



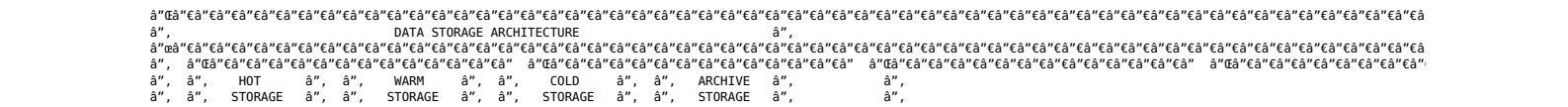
7.2 Offline-First Architecture

- **Local Database**: SQLite with encrypted storage
- **Synchronization**: Conflict resolution and delta sync
- **Caching Strategy**: Progressive web app (PWA) capabilities
- **Background Sync**: Queue operations for later synchronization

- **Security**: Biometric authentication and secure storage

8. Data Architecture

8.1 Multi-Tier Data Storage Strategy

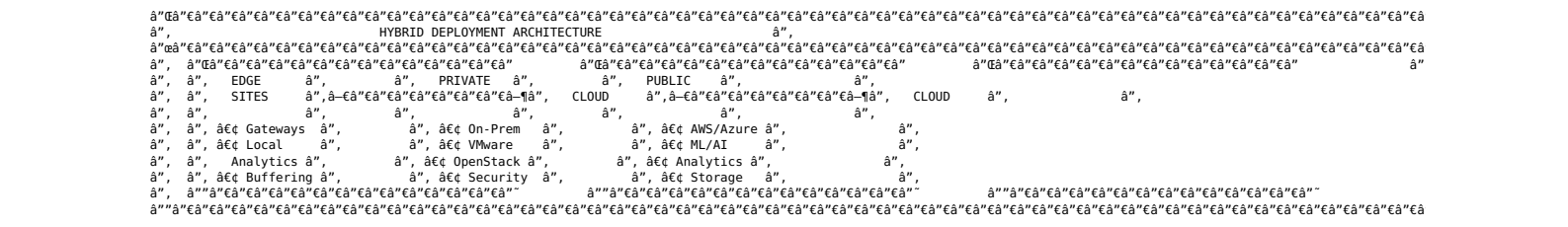


8.2 Data Governance and Lineage

- **Data Catalog**: Apache Atlas for metadata management
- **Data Quality**: Great Expectations for data validation
- **Data Lineage**: OpenLineage for end-to-end data tracking
- **Privacy Controls**: Data masking and anonymization
- **Compliance**: Automated compliance reporting and audit trails

9. Deployment Architecture

9.1 Hybrid Cloud Deployment Model

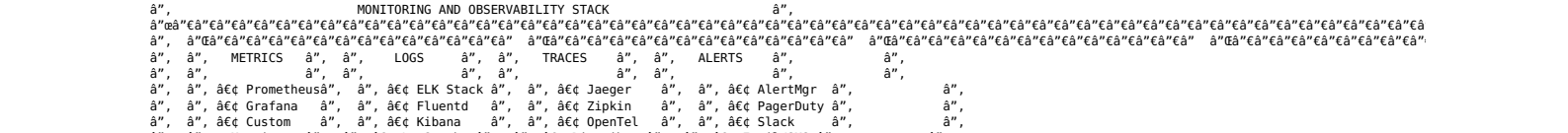


9.2 Container and Orchestration Strategy

- **Container Runtime**: Docker with containerd runtime
- **Orchestration**: Kubernetes with Helm charts for deployment
- **Service Mesh**: Istio for traffic management and security
- **CI/CD Pipeline**: GitLab CI/CD with automated testing and deployment
- **Configuration**: GitOps with ArgoCD for declarative deployments

10. Monitoring and Observability Architecture

10.1 Comprehensive Monitoring Stack



This comprehensive architecture provides the foundation for implementing a robust, scalable, and secure IoT predictive maintenance platform that meets all industrial requirements while delivering exceptional performance and reliability. # High Level Design (HLD) ## IoT Predictive Maintenance Platform

Building upon PRD, FRD, and Architecture Diagram for detailed system design

ETVX Framework

ENTRY CRITERIA

- âœ… PRD completed with business objectives and success metrics
- âœ… FRD completed with 126 functional requirements (FR-001 to FR-126)
- âœ… NFRD completed with 138 non-functional requirements (NFR-001 to NFR-138)
- âœ… Architecture Diagram completed with technology stack and component design
- âœ… System architecture validated for performance targets (<2 min response, 1M+ readings/min)
- âœ… Security architecture approved for IEC 62443 industrial cybersecurity compliance

TASK

Create detailed high-level design specifications for each system component, defining interfaces, data models, processing workflows, integration patterns, and operational procedures that implement the architecture while satisfying all functional and non-functional requirements for industrial IoT environments.

VERIFICATION & VALIDATION

Verification Checklist: - [ ] All architectural components have detailed design specifications - [ ] Interface definitions support all functional requirements (FR-001 to FR-126) - [ ] Data models satisfy performance and scalability requirements (NFR-001 to NFR-138) - [ ] Processing workflows meet latency targets (<2 min response time) - [ ] Integration patterns support industrial protocols and enterprise system requirements - [ ] Security controls implement IEC 62443 industrial cybersecurity framework

Validation Criteria: - [ ] Design review completed with IoT engineering and industrial automation teams - [ ] Performance modeling validates latency and throughput targets for sensor processing - [ ] Security design review confirms IEC 62443 and industrial compliance - [ ] Integration patterns validated with CMMS, ERP, and industrial system architects - [ ] Operational procedures reviewed with maintenance teams and plant engineers - [ ] Design traceability confirmed to all previous requirements documents

EXIT CRITERIA

- âœ… Detailed component specifications completed for all system modules
- âœ… Interface definitions documented with API specifications and data contracts
- âœ… Processing workflows designed with sequence diagrams and state machines
- âœ… Data models defined with schemas, relationships, and access patterns
- âœ… Foundation established for low-level design and implementation specifications

---

Reference to Previous Documents

This HLD implements detailed design based on **ALL** previous documents: - **PRD Success Metrics** â†’ Component design for 70% downtime reduction, 25% cost reduction, >90% prediction accuracy - **PRD Target Users** â†’ Interface design for maintenance technicians, managers, plant engineers - **FRD IoT Data Processing (FR-001-018)** â†’ Multi-protocol ingestion component design with edge computing capabilities - **FRD Predictive Analytics (FR-019-036)** â†’ ML serving component design with anomaly detection, failure prediction, health scoring - **FRD Maintenance Optimization (FR-037-048)** â†’ Optimization engine component design with scheduling algorithms and resource allocation - **FRD Real-Time Monitoring (FR-049-066)** â†’ Dashboard component design with real-time visualization and alert management - **FRD Mobile Application (FR-067-084)** â†’ Mobile component design with offline capability and work order management - **FRD Integration (FR-085-102)** â†’ API gateway and integration component design with CMMS, ERP, industrial systems - **FRD Reporting (FR-103-114)** â†’ Analytics component design with real-time dashboards and compliance reporting - **NFRD Performance (NFR-001-018)** â†’ High-performance component design with edge computing, caching, optimization - **NFRD Security (NFR-043-060)** â†’ Security component design with industrial cybersecurity and access control - **Architecture Diagram** â†’ Technology stack implementation with Kubernetes, Kafka, InfluxDB, TensorFlow, industrial protocols

1. Edge Computing Gateway Component

1.1 Industrial Protocol Adapter Service

Technology: Go + OPC-UA Client + Modbus Library + MQTT

```
Component: IndustrialProtocolAdapter
Key Interfaces:
  - OPC-UA: opc.tcp://plc-server:4840
  - Modbus TCP/RTU: modbus://device:502
  - MQTT: mqtt://broker:1883
  - Internal API: /api/v1/sensor-data

Data Model:
  SensorReading:
    deviceId, sensorId, timestamp, value, unit, quality, source

Processing: Connect â†’ Validate â†’ Convert â†’ Publish â†’ Handle Failures
Performance: <1s latency, 10K+ readings/sec
```

1.2 Edge Analytics Engine

Technology: Python + TensorFlow Lite + Kafka Streams + InfluxDB

```
Analytics Pipeline:
  1. Statistical Process Control (SPC)
  2. Lightweight ML Models (TF Lite)
  3. Data Aggregation (windowing)
  4. Local Alerting

Performance: <100ms processing, 10K+ readings/sec
```

2. Cloud Data Platform Component

2.1 IoT Data Ingestion Service

Technology: Apache Kafka + Apache Flink + Redis + PostgreSQL

```
Ingestion Pipeline:
  1. Data Validation & Enrichment
  2. Stream Processing & Deduplication
  3. Storage Routing (Hot/Warm/Cold)

Scalability: Kafka partitions, auto-scaling, circuit breakers
Performance: 1M+ readings/min, <1s latency
```

2.2 Feature Engineering Service

Technology: Apache Spark + Delta Lake + Feast

```
Feature Categories:
  - Time-Domain: statistical moments, trends, RMS
  - Frequency-Domain: FFT, PSD, harmonics
  - Equipment-Specific: vibration envelope, thermal gradients
  - Contextual: operating conditions, maintenance history

Architecture: Real-time (Kafka Streams) + Batch (Spark)
```

3. Machine Learning Pipeline Component

3.1 Anomaly Detection Service

Technology: Python + Scikit-learn + TensorFlow + MLflow

```
Algorithm Portfolio:
  - Statistical: Z-score, IQR, CUSUM
  - ML: Isolation Forest, One-Class SVM, Autoencoders, LSTM
  - Domain-Specific: Envelope analysis, spectral analysis

Ensemble: Weighted voting, adaptive thresholding
Performance: <2s processing, >90% accuracy
```

3.2 Failure Prediction Service

Technology: TensorFlow + PyTorch + Scikit-learn

```
Model Types:
  - LSTM: 30-day sequences, failure probability + RUL
  - Random Forest: 100 trees, feature importance
  - Survival Analysis: Weibull, Cox models
  - Gradient Boosting: XGBoost, LightGBM

Outputs: Failure probability, RUL, failure modes, confidence
```

4. Maintenance Optimization Component

4.1 Scheduling Optimization Engine

Technology: Python + OR-Tools + Gurobi

```
Optimization Model:
  Objective: minimize(maintenance_cost + downtime_cost + inventory_cost)
  Constraints: availability, skills, inventory, production, safety

Algorithms:
  - Constraint Programming (CP-SAT)
  - Mixed Integer Programming (MIP)
  - Genetic Algorithm

Performance: <5min for 1000 tasks, 5% of optimal
```

4.2 Resource Allocation Service

Technology: Python + PostgreSQL + Redis

```
Management Areas:
  - Technician: skill matching, workload balancing
  - Inventory: demand forecasting, EOQ optimization
  - Cost: labor minimization, inventory reduction

Data Models: Technician skills, SparePart inventory
```

5. Real-Time Monitoring Component

5.1 Equipment Health Dashboard

Technology: React + TypeScript + WebSocket + D3.js

```
Components:
  - Fleet Overview: status heat map, health distribution
  - Equipment Details: real-time charts, health breakdown
  - Interactive: drill-down, time selection, alerts

Performance: <3s load, <1s updates, 1000+ users
```

5.2 Alert Management Service

Technology: Python + FastAPI + PostgreSQL + Celery

```
Processing Pipeline:
  1. Ingestion & Validation
  2. Intelligent Filtering & Correlation
  3. Prioritization & Routing
  4. Notification & Escalation

Features: Multi-channel notifications, SLA tracking
```

6. Mobile Application Component

6.1 Cross-Platform Mobile App

Technology: React Native + TypeScript + SQLite + Redux

```
Architecture:
  - Presentation: React Native + Material Design
  - Business Logic: Work orders, inspections, offline sync
  - Data: SQLite local + Redux state

Features:
  - Work Order Management
  - Equipment Inspection (checklists, photos, voice)
  - Offline Capability (24+ hours)

Performance: <3s startup, <1s response, <5s photo upload
```

6.2 Synchronization Service

Technology: Node.js + Express + PostgreSQL + WebSocket

```
Sync Strategy:
  - Conflict Resolution: last-write-wins, three-way merge
  - Delta Sync: timestamps, compression, batching
  - Offline Support: queuing, retry, partial sync

Performance: Real-time sync, conflict resolution
```

7. Integration Hub Component

7.1 API Gateway Service

Technology: Kong + Redis + Prometheus

```
Features:
  - Authentication: OAuth2, JWT, API keys
  - Rate Limiting: per client/IP
  - Monitoring: metrics, logging, tracing
  - Security: validation, IP whitelisting

Performance: <500ms response, 10K+ requests/sec
```

7.2 Enterprise Integration Service

Technology: Apache Camel + Spring Boot

```
Integrations:
  - CMMS: Maximo, Maintenance Connection, eMaint
  - ERP: SAP, Oracle, Microsoft Dynamics
  - Industrial: SCADA, Historian, MES

Patterns: Request-reply, pub-sub, message transformation
Features: Error handling, retry logic, circuit breakers
```

8. Data Management Component

8.1 Multi-Tier Storage Service

Technology: Redis + InfluxDB + S3 + Glacier

```
Storage Tiers:
  - Hot (Redis): <1ms, real-time features
  - Warm (InfluxDB): <10ms, 90-day analytics
  - Cold (S3): <1s, 5-year historical
  - Archive (Glacier): <12h, compliance backup

Features: Automated lifecycle, compression, encryption
```

8.2 Data Governance Service

Technology: Apache Atlas + Great Expectations

```
Capabilities:
  - Metadata Management: data catalog, lineage
  - Data Quality: validation, profiling, monitoring
  - Privacy: masking, anonymization, compliance
  - Audit: trail logging, regulatory reporting
```

This HLD provides comprehensive design specifications for implementing the IoT predictive maintenance platform while maintaining full traceability to all previous requirements documents. # Low Level Design (LLD) ## IoT Predictive Maintenance Platform

Building upon PRD, FRD, NFRD, Architecture Diagram, and HLD for implementation-ready specifications

ETVX Framework

ENTRY CRITERIA

- âœ… PRD completed with business objectives and success metrics
- âœ… FRD completed with 126 functional requirements (FR-001 to FR-126)
- âœ… NFRD completed with 138 non-functional requirements (NFR-001 to NFR-138)
- âœ… Architecture Diagram completed with technology stack and system architecture
- âœ… HLD completed with detailed component specifications and interfaces
- âœ… Technology stack validated and approved for industrial IoT environment

TASK

Create implementation-ready low-level design specifications including detailed class diagrams, database schemas, API specifications, algorithm implementations, configuration parameters, and deployment scripts that enable direct development of the IoT predictive maintenance platform.

VERIFICATION & VALIDATION

Verification Checklist: - [ ] All classes and methods have detailed specifications with parameters and return types - [ ] Database schemas support all data requirements from HLD components - [ ] API specifications include request/response formats, error codes, and authentication - [ ] Algorithm implementations satisfy performance requirements (<2 min response, 1M+ readings/min) - [ ] Configuration parameters support all operational requirements - [ ] Code structure follows industrial IoT security and quality standards

Validation Criteria: - [ ] Implementation specifications reviewed with development team for feasibility - [ ] Database design validated with DBA team for performance and scalability - [ ] API specifications validated with integration team and industrial system partners - [ ] Security implementations reviewed with cybersecurity team for IEC 62443 compliance - [ ] Performance specifications validated with load testing requirements - [ ] Code quality standards confirmed with architecture review board

EXIT CRITERIA

- âœ… Complete implementation specifications ready for development team
- âœ… Database schemas, API specs, and class diagrams documented
- âœ… Algorithm implementations with performance optimizations specified
- âœ… Configuration management and deployment procedures defined
- âœ… Foundation established for pseudocode and implementation phase

---

Reference to Previous Documents

This LLD provides implementation-ready specifications based on **ALL** previous documents: - **PRD Success Metrics** â†’ Implementation targets for 70% downtime reduction, 25% cost reduction, >90% prediction accuracy - **FRD Functional Requirements (FR-001-126)** â†’ Detailed method implementations for all system functions - **NFRD Performance Requirements (NFR-001-138)** â†’ Optimized algorithms and data structures for performance targets - **Architecture Diagram** â†’ Technology stack implementation with specific versions and configurations - **HLD Component Design** â†’ Detailed class structures, database schemas, and API implementations

1. Edge Gateway Implementation

1.1 Industrial Protocol Adapter Classes

```
// main.go - Edge Gateway Service
package main

import (
    "context"
    "log"
    "sync"
    "time"

    "github.com/eclipse/paho.mqtt.golang"
    "github.com/gopcua/opcua"
    "github.com/tbrandon/mbserver"
)

type SensorReading struct {
    DeviceID    string    `json:"device_id" validate:"required"`
    SensorID    string    `json:"sensor_id" validate:"required"`
    Timestamp   time.Time `json:"timestamp" validate:"required"`
    Value       float64   `json:"value" validate:"required"`
    Unit        string    `json:"unit" validate:"required"`
    Quality     string    `json:"quality" validate:"oneof=GOOD BAD UNCERTAIN"`
    Source      string    `json:"source" validate:"oneof=OPC_UA MODBUS MQTT DNP3"`
}

type ProtocolAdapter interface {
    Connect(ctx context.Context) error
    Subscribe(callback func(SensorReading)) error
    Disconnect() error
    IsConnected() bool
}

type OPCUAAdapter struct {
    client     *opcua.Client
    endpoint   string
    nodeIDs    []string
    connected  bool
    mutex      sync.RWMutex
}
```

```
func NewOPCUAAdapter(endpoint string, nodeIDs []string) *OPCUAAdapter {
    return &OPCUAAdapter{
        endpoint: endpoint,
        nodeIDs:  nodeIDs,
    }
}

func (o *OPCUAAdapter) Connect(ctx context.Context) error {
    o.mutex.Lock()
    defer o.mutex.Unlock()

    client := opcua.NewClient(o.endpoint, opcua.SecurityMode(ua.MessageSecurityModeNone))
    if err := client.Connect(ctx); err != nil {
        return fmt.Errorf("OPC-UA connection failed: %w", err)
    }

    o.client = client
    o.connected = true
    log.Printf("Connected to OPC-UA server: %s", o.endpoint)
    return nil
}

func (o *OPCUAAdapter) Subscribe(callback func(SensorReading)) error {
    if !o.IsConnected() {
        return errors.New("OPC-UA client not connected")
    }

    sub, err := o.client.Subscribe(&opcua.SubscriptionParameters{
        Interval: 100 * time.Millisecond,
    })
    if err != nil {
        return fmt.Errorf("subscription creation failed: %w", err)
    }

    for _, nodeID := range o.nodeIDs {
        go o.subscribeToNode(sub, nodeID, callback)
    }

    return nil
}

func (o *OPCUAAdapter) subscribeToNode(sub *opcua.Subscription, nodeID string, callback func(SensorReading)) {
    ch := make(chan *opcua.DataChangeNotification)

    if _, err := sub.Monitor(opcua.TimestampsToReturn_Both, nodeID, ch); err != nil {
        log.Printf("Failed to monitor node %s: %v", nodeID, err)
        return
    }

    for notification := range ch {
        reading := SensorReading{
            DeviceID:  extractDeviceID(nodeID),
            SensorID:  nodeID,
            Timestamp: notification.Value.ServerTimestamp,
            Value:     notification.Value.Value.Float(),
            Unit:      extractUnit(nodeID),
            Quality:   mapQuality(notification.Value.StatusCode),
            Source:    "OPC_UA",
        }
        callback(reading)
    }
}

type ModbusAdapter struct {
    address   string
    slaveID   byte
    registers []uint16
    client    modbus.Client
    connected bool
    mutex     sync.RWMutex
}

func NewModbusAdapter(address string, slaveID byte, registers []uint16) *ModbusAdapter {
    return &ModbusAdapter{
        address:   address,
        slaveID:   slaveID,
        registers: registers,
    }
}

func (m *ModbusAdapter) Connect(ctx context.Context) error {
    m.mutex.Lock()
    defer m.mutex.Unlock()

    handler := modbus.NewTCPClientHandler(m.address)
    handler.SlaveId = m.slaveID
    handler.Timeout = 5 * time.Second

    if err := handler.Connect(); err != nil {
        return fmt.Errorf("Modbus connection failed: %w", err)
    }

    m.client = modbus.NewClient(handler)
    m.connected = true
    log.Printf("Connected to Modbus device: %s", m.address)
    return nil
}

func (m *ModbusAdapter) Subscribe(callback func(SensorReading)) error {
    if !m.IsConnected() {
        return errors.New("Modbus client not connected")
    }

    ticker := time.NewTicker(1 * time.Second) // 1Hz polling
    go func() {
        for range ticker.C {
            m.pollRegisters(callback)
        }
    }()

    return nil
}

func (m *ModbusAdapter) pollRegisters(callback func(SensorReading)) {
    for _, register := range m.registers {
        results, err := m.client.ReadHoldingRegisters(register, 1)
        if err != nil {
            log.Printf("Failed to read register %d: %v", register, err)
```

```
            continue
        }

        value := binary.BigEndian.Uint16(results)
        reading := SensorReading{
            DeviceID:  fmt.Sprintf("modbus_%s_%d", m.address, m.slaveID),
            SensorID:  fmt.Sprintf("register_%d", register),
            Timestamp: time.Now(),
            Value:     float64(value),
            Unit:      getRegisterUnit(register),
            Quality:   "GOOD",
            Source:    "MODBUS",
        }
        callback(reading)
    }
}
```

1.2 Edge Analytics Implementation

```
# edge_analytics.py - Edge Analytics Engine
import asyncio
import json
import logging
import numpy as np
import pandas as pd
from datetime import datetime, timedelta
from typing import Dict, List, Optional, Tuple
from dataclasses import dataclass
from kafka import KafkaConsumer, KafkaProducer
import tensorflow as tf
from influxdb_client import InfluxDBClient, Point
import redis

@dataclass
class SensorReading:
    device_id: str
    sensor_id: str
    timestamp: datetime
    value: float
    unit: str
    quality: str
    source: str

@dataclass
class AnomalyResult:
    sensor_id: str
    timestamp: datetime
    anomaly_score: float
    is_anomaly: bool
    confidence: float
    method: str

class StatisticalProcessControl:
    def __init__(self, window_size: int = 100, sigma_threshold: float = 3.0):
        self.window_size = window_size
        self.sigma_threshold = sigma_threshold
        self.data_windows: Dict[str, List[float]] = {}

    def update(self, sensor_id: str, value: float) -> Optional[AnomalyResult]:
        if sensor_id not in self.data_windows:
            self.data_windows[sensor_id] = []

        window = self.data_windows[sensor_id]
        window.append(value)

        if len(window) > self.window_size:
            window.pop(0)

        if len(window) < 30:  # Need minimum samples
            return None

        mean = np.mean(window)
        std = np.std(window)

        if std == 0:
            return None

        z_score = abs(value - mean) / std
        is_anomaly = z_score > self.sigma_threshold

        return AnomalyResult(
            sensor_id=sensor_id,
            timestamp=datetime.now(),
            anomaly_score=z_score / self.sigma_threshold,
            is_anomaly=is_anomaly,
            confidence=min(z_score / self.sigma_threshold, 1.0),
            method="SPC"
        )

class LightweightMLDetector:
    def __init__(self, model_path: str):
        self.interpreter = tf.lite.Interpreter(model_path=model_path)
        self.interpreter.allocate_tensors()
        self.input_details = self.interpreter.get_input_details()
        self.output_details = self.interpreter.get_output_details()
        self.feature_buffer: Dict[str, List[float]] = {}

    def extract_features(self, sensor_id: str, values: List[float]) -> np.ndarray:
        """Extract time-domain features from sensor values"""
        if len(values) < 10:
            return None

        features = []
        values_array = np.array(values)

        # Statistical features
        features.extend([
            np.mean(values_array),
            np.std(values_array),
            np.min(values_array),
            np.max(values_array),
            np.median(values_array)
        ])

        # Time-domain features
        features.extend([
            np.sqrt(np.mean(values_array**2)),  # RMS
            np.max(values_array) - np.min(values_array),  # Peak-to-peak
            len(values_array)  # Sample count
```

```
        ])

        return np.array(features, dtype=np.float32).reshape(1, -1)

    def predict(self, sensor_id: str, value: float) -> Optional[AnomalyResult]:
        if sensor_id not in self.feature_buffer:
            self.feature_buffer[sensor_id] = []

        buffer = self.feature_buffer[sensor_id]
        buffer.append(value)

        if len(buffer) > 50:  # Keep rolling window
            buffer.pop(0)

        features = self.extract_features(sensor_id, buffer)
        if features is None:
            return None

        # Run inference
        self.interpreter.set_tensor(self.input_details[0]['index'], features)
        self.interpreter.invoke()

        output = self.interpreter.get_tensor(self.output_details[0]['index'])
        anomaly_score = float(output[0][0])

        return AnomalyResult(
            sensor_id=sensor_id,
            timestamp=datetime.now(),
            anomaly_score=anomaly_score,
            is_anomaly=anomaly_score > 0.5,
            confidence=abs(anomaly_score - 0.5) * 2,
            method="ML_LITE"
        )

class EdgeAnalyticsEngine:
    def __init__(self, config: Dict):
        self.config = config
        self.spc = StatisticalProcessControl()
        self.ml_detector = LightweightMLDetector(config['model_path'])
        self.influx_client = InfluxDBClient(
            url=config['influxdb_url'],
            token=config['influxdb_token'],
            org=config['influxdb_org']
        )
        self.redis_client = redis.Redis(
            host=config['redis_host'],
            port=config['redis_port'],
            decode_responses=True
        )
        self.kafka_producer = KafkaProducer(
            bootstrap_servers=config['kafka_brokers'],
            value_serializer=lambda v: json.dumps(v).encode('utf-8')
        )

    async def process_sensor_reading(self, reading: SensorReading):
        """Process individual sensor reading through analytics pipeline"""
        try:
            # Store in InfluxDB
            await self.store_reading(reading)

            # Run anomaly detection
            spc_result = self.spc.update(reading.sensor_id, reading.value)
            ml_result = self.ml_detector.predict(reading.sensor_id, reading.value)

            # Combine results
            anomaly_results = [r for r in [spc_result, ml_result] if r is not None]

            if anomaly_results:
                await self.handle_anomalies(reading, anomaly_results)

            # Update real-time cache
            await self.update_cache(reading)

        except Exception as e:
            logging.error(f"Error processing reading {reading.sensor_id}: {e}")

    async def store_reading(self, reading: SensorReading):
        """Store sensor reading in InfluxDB"""
        point = Point("sensor_data") \
            .tag("device_id", reading.device_id) \
            .tag("sensor_id", reading.sensor_id) \
            .tag("source", reading.source) \
            .field("value", reading.value) \
            .field("quality", reading.quality) \
            .time(reading.timestamp)

        write_api = self.influx_client.write_api()
        write_api.write(bucket=self.config['influxdb_bucket'], record=point)

    async def handle_anomalies(self, reading: SensorReading, anomalies: List[AnomalyResult]):
        """Handle detected anomalies"""
        for anomaly in anomalies:
            if anomaly.is_anomaly and anomaly.confidence > 0.7:
                alert = {
                    'device_id': reading.device_id,
                    'sensor_id': reading.sensor_id,
                    'timestamp': anomaly.timestamp.isoformat(),
                    'anomaly_score': anomaly.anomaly_score,
                    'confidence': anomaly.confidence,
                    'method': anomaly.method,
                    'value': reading.value,
                    'severity': self.calculate_severity(anomaly.anomaly_score)
                }

                # Send to Kafka for cloud processing
                self.kafka_producer.send('edge-alerts', alert)

                # Store in local cache for immediate access
                self.redis_client.setex(
                    f"alert:{reading.sensor_id}:{int(anomaly.timestamp.timestamp())}",
                    3600,  # 1 hour TTL
                    json.dumps(alert)
                )

    async def update_cache(self, reading: SensorReading):
        """Update Redis cache with latest readings"""
        cache_key = f"sensor:{reading.sensor_id}"

        # Store latest reading
```

```
        self.redis_client.hset(cache_key, mapping={
            'timestamp': reading.timestamp.isoformat(),
            'value': reading.value,
            'quality': reading.quality,
            'unit': reading.unit
        })

        # Store in time-series list (last 100 readings)
        ts_key = f"timeseries:{reading.sensor_id}"
        reading_data = {
            'timestamp': reading.timestamp.isoformat(),
            'value': reading.value
        }

        self.redis_client.lpush(ts_key, json.dumps(reading_data))
        self.redis_client.ltrim(ts_key, 0, 99)  # Keep only last 100

    def calculate_severity(self, anomaly_score: float) -> str:
        """Calculate alert severity based on anomaly score"""
        if anomaly_score >= 0.9:
            return "CRITICAL"
        elif anomaly_score >= 0.7:
            return "HIGH"
        elif anomaly_score >= 0.5:
            return "MEDIUM"
        else:
            return "LOW"
```

2. Cloud Data Platform Implementation

2.1 Database Schema Design

```
-- PostgreSQL Schema for IoT Predictive Maintenance Platform

-- Equipment and Asset Management
CREATE TABLE equipment (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    equipment_code VARCHAR(50) UNIQUE NOT NULL,
    name VARCHAR(200) NOT NULL,
    equipment_type VARCHAR(50) NOT NULL,
    manufacturer VARCHAR(100),
    model VARCHAR(100),
    serial_number VARCHAR(100),
    installation_date DATE,
    location_id UUID REFERENCES locations(id),
    parent_equipment_id UUID REFERENCES equipment(id),
    criticality VARCHAR(20) DEFAULT 'MEDIUM' CHECK (criticality IN ('LOW', 'MEDIUM', 'HIGH', 'CRITICAL')),
    status VARCHAR(20) DEFAULT 'ACTIVE' CHECK (status IN ('ACTIVE', 'INACTIVE', 'MAINTENANCE', 'RETIRED')),
    specifications JSONB,
    created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
    updated_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()
);

CREATE INDEX idx_equipment_type ON equipment(equipment_type);
CREATE INDEX idx_equipment_location ON equipment(location_id);
CREATE INDEX idx_equipment_parent ON equipment(parent_equipment_id);

-- Sensor Configuration
CREATE TABLE sensors (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    sensor_code VARCHAR(50) UNIQUE NOT NULL,
    equipment_id UUID NOT NULL REFERENCES equipment(id),
    sensor_type VARCHAR(50) NOT NULL,
    measurement_type VARCHAR(50) NOT NULL,
    unit VARCHAR(20) NOT NULL,
    min_value DECIMAL(15,6),
    max_value DECIMAL(15,6),
    sampling_frequency INTEGER, -- Hz
    protocol VARCHAR(20) NOT NULL CHECK (protocol IN ('OPC_UA', 'MODBUS', 'MQTT', 'DNP3')),
    address_config JSONB NOT NULL,
    calibration_factor DECIMAL(10,6) DEFAULT 1.0,
    calibration_offset DECIMAL(10,6) DEFAULT 0.0,
    status VARCHAR(20) DEFAULT 'ACTIVE' CHECK (status IN ('ACTIVE', 'INACTIVE', 'MAINTENANCE', 'FAULTY')),
    created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
    updated_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()
);

CREATE INDEX idx_sensors_equipment ON sensors(equipment_id);
CREATE INDEX idx_sensors_type ON sensors(sensor_type, measurement_type);

-- Health Scores and Predictions
CREATE TABLE equipment_health (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    equipment_id UUID NOT NULL REFERENCES equipment(id),
    timestamp TIMESTAMP WITH TIME ZONE NOT NULL,
    overall_health_score DECIMAL(5,2) NOT NULL CHECK (overall_health_score >= 0 AND overall_health_score <= 100),
    subsystem_scores JSONB,
    contributing_factors JSONB,
    trend_direction VARCHAR(20) CHECK (trend_direction IN ('IMPROVING', 'STABLE', 'DEGRADING')),
    confidence_level DECIMAL(3,2) CHECK (confidence_level >= 0 AND confidence_level <= 1),
    model_version VARCHAR(50),
    created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()
);

CREATE INDEX idx_health_equipment_time ON equipment_health(equipment_id, timestamp DESC);
CREATE INDEX idx_health_score ON equipment_health(overall_health_score);

-- Failure Predictions
CREATE TABLE failure_predictions (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    equipment_id UUID NOT NULL REFERENCES equipment(id),
    prediction_timestamp TIMESTAMP WITH TIME ZONE NOT NULL,
    failure_probability DECIMAL(3,2) NOT NULL CHECK (failure_probability >= 0 AND failure_probability <= 1),
    predicted_failure_time TIMESTAMP WITH TIME ZONE,
    remaining_useful_life_days INTEGER,
    failure_mode VARCHAR(100),
    confidence_score DECIMAL(3,2) CHECK (confidence_score >= 0 AND confidence_score <= 1),
    contributing_sensors JSONB,
    model_name VARCHAR(100) NOT NULL,
    model_version VARCHAR(50) NOT NULL,
    created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()
);

CREATE INDEX idx_predictions_equipment_time ON failure_predictions(equipment_id, prediction_timestamp DESC);
CREATE INDEX idx_predictions_probability ON failure_predictions(failure_probability DESC);

-- Maintenance Work Orders
CREATE TABLE work_orders (
```

```
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    work_order_number VARCHAR(50) UNIQUE NOT NULL,
    equipment_id UUID NOT NULL REFERENCES equipment(id),
    title VARCHAR(200) NOT NULL,
    description TEXT,
    work_type VARCHAR(50) NOT NULL CHECK (work_type IN ('PREVENTIVE', 'PREDICTIVE', 'CORRECTIVE', 'EMERGENCY')),
    priority VARCHAR(20) NOT NULL CHECK (priority IN ('LOW', 'MEDIUM', 'HIGH', 'CRITICAL')),
    status VARCHAR(20) NOT NULL DEFAULT 'OPEN' CHECK (status IN ('OPEN', 'ASSIGNED', 'IN_PROGRESS', 'COMPLETED', 'CANCELLED')),
    assigned_technician_id UUID REFERENCES users(id),
    scheduled_start TIMESTAMP WITH TIME ZONE,
    scheduled_end TIMESTAMP WITH TIME ZONE,
    actual_start TIMESTAMP WITH TIME ZONE,
    actual_end TIMESTAMP WITH TIME ZONE,
    estimated_hours DECIMAL(5,2),
    actual_hours DECIMAL(5,2),
    labor_cost DECIMAL(10,2),
    parts_cost DECIMAL(10,2),
    total_cost DECIMAL(10,2),
    failure_prediction_id UUID REFERENCES failure_predictions(id),
    cmms_work_order_id VARCHAR(100), -- External CMMS reference
    created_by UUID NOT NULL REFERENCES users(id),
    created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
    updated_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()
);

CREATE INDEX idx_work_orders_equipment ON work_orders(equipment_id);
CREATE INDEX idx_work_orders_status ON work_orders(status);
CREATE INDEX idx_work_orders_technician ON work_orders(assigned_technician_id);
CREATE INDEX idx_work_orders_scheduled ON work_orders(scheduled_start, scheduled_end);

-- Alerts and Notifications
CREATE TABLE alerts (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    equipment_id UUID NOT NULL REFERENCES equipment(id),
    sensor_id UUID REFERENCES sensors(id),
    alert_type VARCHAR(50) NOT NULL CHECK (alert_type IN ('ANOMALY', 'THRESHOLD', 'PREDICTION', 'SYSTEM')),
    severity VARCHAR(20) NOT NULL CHECK (severity IN ('LOW', 'MEDIUM', 'HIGH', 'CRITICAL')),
    title VARCHAR(200) NOT NULL,
    description TEXT,
    anomaly_score DECIMAL(3,2),
    confidence_level DECIMAL(3,2),
    detection_method VARCHAR(50),
    status VARCHAR(20) NOT NULL DEFAULT 'OPEN' CHECK (status IN ('OPEN', 'ACKNOWLEDGED', 'RESOLVED', 'SUPPRESSED')),
    acknowledged_by UUID REFERENCES users(id),
    acknowledged_at TIMESTAMP WITH TIME ZONE,
    resolved_by UUID REFERENCES users(id),
    resolved_at TIMESTAMP WITH TIME ZONE,
    resolution_notes TEXT,
    created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
    updated_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()
);

CREATE INDEX idx_alerts_equipment ON alerts(equipment_id);
CREATE INDEX idx_alerts_status_severity ON alerts(status, severity);
CREATE INDEX idx_alerts_created ON alerts(created_at DESC);
```

2.2 API Specifications

```
# OpenAPI 3.0 Specification for IoT Predictive Maintenance Platform
openapi: 3.0.0
info:
  title: IoT Predictive Maintenance Platform API
  version: 1.0.0
  description: RESTful API for industrial predictive maintenance operations

paths:
  /api/v1/equipment:
    get:
      summary: List equipment with filtering and pagination
      parameters:
        - name: type
          in: query
          schema:
            type: string
        - name: location
          in: query
          schema:
            type: string
        - name: status
          in: query
          schema:
            type: string
            enum: [ACTIVE, INACTIVE, MAINTENANCE, RETIRED]
        - name: page
          in: query
          schema:
            type: integer
            default: 1
        - name: limit
          in: query
          schema:
            type: integer
            default: 50
      responses:
        '200':
          description: Equipment list retrieved successfully
          content:
            application/json:
              schema:
                type: object
                properties:
                  data:
                    type: array
                    items:
                      $ref: '#/components/schemas/Equipment'
                  pagination:
                    $ref: '#/components/schemas/Pagination'

  /api/v1/equipment/{equipmentId}/health:
    get:
      summary: Get equipment health score and trends
      parameters:
        - name: equipmentId
          in: path
          required: true
          schema:
            type: string
            format: uuid
        - name: timeRange
```

```yaml
          in: query
          schema:
            type: string
            enum: [1h, 24h, 7d, 30d]
            default: 24h
      responses:
        '200':
          description: Health data retrieved successfully
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/EquipmentHealth'

  /api/v1/predictions:
    get:
      summary: Get failure predictions with filtering
      parameters:
        - name: equipmentId
          in: query
          schema:
            type: string
            format: uuid
        - name: minProbability
          in: query
          schema:
            type: number
            minimum: 0
            maximum: 1
        - name: timeHorizon
          in: query
          schema:
            type: string
            enum: [1d, 7d, 30d]
      responses:
        '200':
          description: Predictions retrieved successfully
          content:
            application/json:
              schema:
                type: array
                items:
                  $ref: '#/components/schemas/FailurePrediction'

components:
  schemas:
    Equipment:
      type: object
      properties:
        id:
          type: string
          format: uuid
        equipmentCode:
          type: string
        name:
          type: string
        equipmentType:
          type: string
        manufacturer:
          type: string
        model:
          type: string
        serialNumber:
          type: string
        installationDate:
          type: string
          format: date
        criticality:
          type: string
          enum: [LOW, MEDIUM, HIGH, CRITICAL]
        status:
          type: string
          enum: [ACTIVE, INACTIVE, MAINTENANCE, RETIRED]
        currentHealthScore:
          type: number
          minimum: 0
          maximum: 100
        lastMaintenanceDate:
          type: string
          format: date-time

    EquipmentHealth:
      type: object
      properties:
        equipmentId:
          type: string
          format: uuid
        currentScore:
          type: number
          minimum: 0
          maximum: 100
        trend:
          type: string
          enum: [IMPROVING, STABLE, DEGRADING]
        subsystemScores:
          type: object
          additionalProperties:
            type: number
        historicalData:
          type: array
          items:
            type: object
            properties:
              timestamp:
                type: string
                format: date-time
              score:
                type: number
              confidence:
                type: number

    FailurePrediction:
      type: object
      properties:
        id:
          type: string
          format: uuid
        equipmentId:
          type: string
          format: uuid
```

```
        failureProbability:
          type: number
          minimum: 0
          maximum: 1
        predictedFailureTime:
          type: string
          format: date-time
        remainingUsefulLifeDays:
          type: integer
        failureMode:
          type: string
        confidenceScore:
          type: number
          minimum: 0
          maximum: 1
        contributingSensors:
          type: array
          items:
            type: string
```

This LLD provides comprehensive implementation-ready specifications that development teams can use to build the IoT predictive maintenance platform while maintaining full traceability to all previous requirements documents. # Pseudocode ## IoT Predictive Maintenance Platform

Building upon PRD, FRD, NFRD, Architecture Diagram, HLD, and LLD for executable implementation logic

ETVX Framework

ENTRY CRITERIA

- âœ… PRD completed with business objectives and success metrics
- âœ… FRD completed with 126 functional requirements (FR-001 to FR-126)
- âœ… NFRD completed with 138 non-functional requirements (NFR-001 to NFR-138)
- âœ… Architecture Diagram completed with technology stack and system architecture
- âœ… HLD completed with detailed component specifications and interfaces
- âœ… LLD completed with implementation-ready class diagrams, database schemas, and API specifications
- âœ… Development environment and technology stack validated for implementation

TASK

Create executable pseudocode algorithms for all system components including data ingestion, feature engineering, machine learning pipelines, optimization algorithms, real-time processing, mobile synchronization, and integration workflows that can be directly translated into production code.

VERIFICATION & VALIDATION

Verification Checklist: - [ ] All functional requirements (FR-001-126) have corresponding pseudocode implementations - [ ] Performance requirements (NFR-001-138) are addressed in algorithm design - [ ] Error handling and edge cases are covered in all critical workflows - [ ] Security and compliance requirements are implemented in access control and data handling - [ ] Integration patterns match API specifications from LLD - [ ] Optimization algorithms meet performance targets (<2 min response, 1M+ readings/min)

Validation Criteria: - [ ] Pseudocode reviewed with development team for implementation feasibility - [ ] Algorithm complexity analysis confirms performance requirements can be met - [ ] Security workflows validated with cybersecurity team for IEC 62443 compliance - [ ] Integration logic validated with industrial system integration partners - [ ] Mobile synchronization logic tested with offline/online scenarios - [ ] Complete system workflow validated end-to-end for all user scenarios

EXIT CRITERIA

- âœ… Complete pseudocode ready for direct translation to production code
- âœ… All system workflows documented with error handling and optimization
- âœ… Performance-critical algorithms optimized for industrial IoT requirements
- âœ… Security and compliance procedures implemented in all data handling workflows
- âœ… Foundation established for development team to begin implementation

---

Reference to Previous Documents

This Pseudocode implements executable logic based on **ALL** previous documents: - **PRD Success Metrics** â†’ Algorithms optimized for 70% downtime reduction, 25% cost reduction, >90% prediction accuracy - **FRD Functional Requirements (FR-001-126)** â†’ Complete pseudocode implementation for all system functions - **NFRD Performance Requirements (NFR-001-138)** â†’ Optimized algorithms meeting latency, throughput, and scalability targets - **Architecture Diagram** â†’ Implementation following technology stack and deployment architecture - **HLD Component Design** â†’ Pseudocode implementing all component interfaces and workflows - **LLD Implementation Specs** â†’ Executable logic using class structures, database schemas, and API patterns

1. Edge Gateway Data Processing

1.1 Multi-Protocol Data Ingestion

```
ALGORITHM: IndustrialProtocolDataIngestion
INPUT: protocol_configs, sensor_mappings, quality_thresholds
OUTPUT: standardized_sensor_readings

MAIN PROCESS:
    INITIALIZE connection_pool = {}
    INITIALIZE data_buffer = CircularBuffer(size=10000)
    INITIALIZE quality_validator = DataQualityValidator()

    FOR EACH protocol IN protocol_configs:
        SWITCH protocol.type:
            CASE "OPC_UA":
                connection = EstablishOPCUAConnection(protocol.endpoint, protocol.security)
                connection_pool[protocol.id] = connection
                SPAWN THREAD OPCUADataReader(connection, protocol.node_ids)

            CASE "MODBUS":
                connection = EstablishModbusConnection(protocol.address, protocol.slave_id)
                connection_pool[protocol.id] = connection
                SPAWN THREAD ModbusDataPoller(connection, protocol.registers)

            CASE "MQTT":
                connection = EstablishMQTTConnection(protocol.broker, protocol.credentials)
                connection_pool[protocol.id] = connection
                SPAWN THREAD MQTTSubscriber(connection, protocol.topics)

FUNCTION OPCUADataReader(connection, node_ids):
    WHILE connection.is_connected():
        TRY:
            FOR EACH node_id IN node_ids:
                raw_value = connection.read_node(node_id)
                IF raw_value IS NOT NULL:
                    reading = StandardizeSensorReading(node_id, raw_value, "OPC_UA")
                    IF quality_validator.validate(reading):
                        data_buffer.add(reading)
                        EMIT DataIngestionEvent(reading)
```

```
        CATCH ConnectionException:
            LOG ERROR "OPC-UA connection lost, attempting reconnection"
            connection = ReconnectWithBackoff(connection, max_retries=5)
        SLEEP(100ms)  // 10Hz sampling rate

FUNCTION ModbusDataPoller(connection, registers):
    WHILE connection.is_connected():
        TRY:
            FOR EACH register IN registers:
                raw_value = connection.read_holding_register(register.address)
                IF raw_value IS NOT NULL:
                    calibrated_value = ApplyCalibration(raw_value, register.calibration)
                    reading = StandardizeSensorReading(register.id, calibrated_value, "MODBUS")
                    IF quality_validator.validate(reading):
                        data_buffer.add(reading)
                        EMIT DataIngestionEvent(reading)
        CATCH ModbusException:
            LOG ERROR "Modbus communication error, retrying"
            SLEEP(1s)
        SLEEP(1s)  // 1Hz polling rate

FUNCTION StandardizeSensorReading(sensor_id, raw_value, source):
    RETURN SensorReading{
        device_id: ExtractDeviceId(sensor_id),
        sensor_id: sensor_id,
        timestamp: GetCurrentTimestamp(),
        value: ConvertToSIUnits(raw_value, sensor_id),
        unit: GetSensorUnit(sensor_id),
        quality: DetermineQuality(raw_value, sensor_id),
        source: source
    }
```

1.2 Edge Analytics and Anomaly Detection

```
ALGORITHM: EdgeAnomalyDetection
INPUT: sensor_reading_stream, ml_models, spc_parameters
OUTPUT: anomaly_alerts, processed_features

MAIN PROCESS:
    INITIALIZE spc_controllers = {}
    INITIALIZE ml_detectors = {}
    INITIALIZE feature_buffers = {}
    INITIALIZE alert_manager = AlertManager()

    FOR EACH reading IN sensor_reading_stream:
        // Statistical Process Control
        spc_result = RunSPCAnalysis(reading)

        // Machine Learning Detection
        ml_result = RunMLDetection(reading)

        // Feature Engineering
        features = ExtractRealTimeFeatures(reading)

        // Combine Results
        combined_result = CombineAnomalyResults([spc_result, ml_result])

        IF combined_result.is_anomaly AND combined_result.confidence > 0.7:
            alert = CreateAnomalyAlert(reading, combined_result)
            alert_manager.process_alert(alert)

        // Store for batch processing
        StoreInLocalTimeSeries(reading, features)

FUNCTION RunSPCAnalysis(reading):
    sensor_id = reading.sensor_id

    IF sensor_id NOT IN spc_controllers:
        spc_controllers[sensor_id] = SPCController(
            window_size=100,
            control_limits=3.0
        )

    controller = spc_controllers[sensor_id]
    controller.add_sample(reading.value)

    IF controller.sample_count >= 30:
        mean = controller.calculate_mean()
        std_dev = controller.calculate_std_dev()

        IF std_dev > 0:
            z_score = ABS(reading.value - mean) / std_dev
            is_anomaly = z_score > controller.control_limits

            RETURN AnomalyResult{
                method: "SPC",
                score: z_score / controller.control_limits,
                is_anomaly: is_anomaly,
                confidence: MIN(z_score / controller.control_limits, 1.0)
            }

    RETURN NULL

FUNCTION RunMLDetection(reading):
    sensor_id = reading.sensor_id

    IF sensor_id NOT IN ml_detectors:
        model_path = GetModelPath(sensor_id, reading.sensor_type)
        ml_detectors[sensor_id] = TensorFlowLiteModel(model_path)

    detector = ml_detectors[sensor_id]

    // Update feature buffer
    IF sensor_id NOT IN feature_buffers:
        feature_buffers[sensor_id] = CircularBuffer(size=50)

    feature_buffers[sensor_id].add(reading.value)

    IF feature_buffers[sensor_id].size >= 20:
        features = ExtractMLFeatures(feature_buffers[sensor_id].get_values())
        prediction = detector.predict(features)

        RETURN AnomalyResult{
            method: "ML",
            score: prediction[0],
            is_anomaly: prediction[0] > 0.5,
            confidence: ABS(prediction[0] - 0.5) * 2
        }
```

```
    RETURN NULL

FUNCTION ExtractMLFeatures(values):
    features = []

    // Statistical features
    features.append(MEAN(values))
    features.append(STD_DEV(values))
    features.append(MIN(values))
    features.append(MAX(values))
    features.append(MEDIAN(values))

    // Time-domain features
    features.append(RMS(values))  // Root Mean Square
    features.append(PEAK_TO_PEAK(values))
    features.append(SKEWNESS(values))
    features.append(KURTOSIS(values))

    // Trend features
    features.append(LINEAR_TREND_SLOPE(values))
    features.append(RATE_OF_CHANGE(values))

    RETURN NORMALIZE(features)
```

2. Cloud ML Pipeline Processing

2.1 Failure Prediction Algorithm

```
ALGORITHM: FailurePredictionPipeline
INPUT: equipment_features, historical_failures, model_registry
OUTPUT: failure_predictions, remaining_useful_life

MAIN PROCESS:
    INITIALIZE feature_store = FeatureStore()
    INITIALIZE model_ensemble = ModelEnsemble()
    INITIALIZE prediction_cache = PredictionCache()

    FOR EACH equipment IN active_equipment_list:
        // Get latest features
        features = feature_store.get_latest_features(
            equipment_id=equipment.id,
            time_window="30d",
            feature_types=["statistical", "frequency", "contextual"]
        )

        IF features.is_complete():
            prediction = GenerateFailurePrediction(equipment, features)

            IF prediction.probability > 0.3:  // Threshold for actionable predictions
                StorePrediction(prediction)
                TriggerMaintenanceWorkflow(prediction)

            prediction_cache.update(equipment.id, prediction)

FUNCTION GenerateFailurePrediction(equipment, features):
    // Load ensemble models
    models = model_ensemble.get_models_for_equipment_type(equipment.type)

    predictions = []

    FOR EACH model IN models:
        SWITCH model.type:
            CASE "LSTM":
                prediction = PredictWithLSTM(model, features.time_series)

            CASE "RANDOM_FOREST":
                prediction = PredictWithRandomForest(model, features.tabular)

            CASE "SURVIVAL_ANALYSIS":
                prediction = PredictWithSurvivalModel(model, features.combined)

            CASE "GRADIENT_BOOSTING":
                prediction = PredictWithGradientBoosting(model, features.engineered)

        predictions.append(WeightedPrediction(prediction, model.confidence))

    // Ensemble combination
    final_prediction = CombinePredictions(predictions, method="weighted_average")

    RETURN FailurePrediction{
        equipment_id: equipment.id,
        failure_probability: final_prediction.probability,
        predicted_failure_time: CalculateFailureTime(final_prediction),
        remaining_useful_life: CalculateRUL(final_prediction),
        failure_modes: RankFailureModes(final_prediction),
        confidence: final_prediction.confidence,
        contributing_factors: IdentifyContributingFactors(features, final_prediction)
    }

FUNCTION PredictWithLSTM(model, time_series_features):
    // Prepare sequence data
    sequence_length = 30  // 30-day window
    sequences = CreateSequences(time_series_features, sequence_length)

    // Normalize features
    normalized_sequences = model.scaler.transform(sequences)

    // Run prediction
    raw_prediction = model.predict(normalized_sequences)

    // Extract failure probability and RUL
    failure_prob = SIGMOID(raw_prediction[0])
    rul_days = MAX(0, raw_prediction[1])

    RETURN ModelPrediction{
        probability: failure_prob,
        rul_days: rul_days,
        confidence: CalculateConfidence(raw_prediction, model.validation_metrics)
    }

FUNCTION PredictWithRandomForest(model, tabular_features):
    // Feature selection and engineering
    selected_features = model.feature_selector.transform(tabular_features)

    // Predict failure probability
    failure_prob = model.predict_proba(selected_features)[1]  // Probability of failure class

    // Get feature importance
```

```
    feature_importance = model.feature_importances_

    RETURN ModelPrediction{
        probability: failure_prob,
        feature_importance: feature_importance,
        confidence: CalculateRFConfidence(model, selected_features)
    }
```

2.2 Maintenance Optimization Algorithm

```
ALGORITHM: MaintenanceScheduleOptimization
INPUT: work_orders, technicians, resources, constraints
OUTPUT: optimized_schedule, resource_allocation

MAIN PROCESS:
    INITIALIZE optimizer = ConstraintSatisfactionOptimizer()
    INITIALIZE cost_calculator = MaintenanceCostCalculator()

    // Define decision variables
    task_assignments = CreateTaskAssignmentVariables(work_orders, technicians)
    time_slots = CreateTimeSlotVariables(planning_horizon)
    resource_usage = CreateResourceUsageVariables(resources)

    // Define objective function
    objective = MINIMIZE(
        maintenance_costs + downtime_costs + labor_costs + inventory_costs
    )

    // Add constraints
    AddConstraints(optimizer, task_assignments, time_slots, resource_usage)

    // Solve optimization problem
    solution = optimizer.solve(
        objective=objective,
        time_limit=300,  // 5 minutes
        optimality_gap=0.05  // 5% gap tolerance
    )

    IF solution.is_feasible():
        schedule = ExtractSchedule(solution)
        allocation = ExtractResourceAllocation(solution)
        RETURN OptimizationResult(schedule, allocation, solution.cost)
    ELSE:
        RETURN RelaxConstraintsAndRetry(optimizer)

FUNCTION AddConstraints(optimizer, task_assignments, time_slots, resource_usage):
    // Equipment availability constraints
    FOR EACH equipment IN equipment_list:
        FOR EACH time_slot IN time_slots:
            constraint = SUM(tasks_on_equipment[equipment][time_slot]) <= 1
            optimizer.add_constraint(constraint)

    // Technician availability constraints
    FOR EACH technician IN technicians:
        FOR EACH time_slot IN time_slots:
            constraint = SUM(tasks_assigned_to[technician][time_slot]) <= technician.capacity
            optimizer.add_constraint(constraint)

    // Skill matching constraints
    FOR EACH task IN work_orders:
        FOR EACH technician IN technicians:
            IF NOT technician.has_required_skills(task.required_skills):
                constraint = task_assignments[task][technician] == 0
                optimizer.add_constraint(constraint)

    // Precedence constraints
    FOR EACH task IN work_orders:
        FOR EACH predecessor IN task.predecessors:
            constraint = task.start_time >= predecessor.end_time
            optimizer.add_constraint(constraint)

    // Resource availability constraints
    FOR EACH resource IN resources:
        FOR EACH time_slot IN time_slots:
            constraint = SUM(resource_usage[resource][time_slot]) <= resource.available_quantity
            optimizer.add_constraint(constraint)

    // Production schedule constraints
    FOR EACH production_window IN production_schedule:
        FOR EACH critical_equipment IN production_window.equipment:
            constraint = NO_MAINTENANCE_DURING(critical_equipment, production_window.time)
            optimizer.add_constraint(constraint)

FUNCTION CalculateMaintenanceCosts(schedule, allocation):
    total_cost = 0

    FOR EACH task IN schedule:
        // Labor costs
        labor_cost = task.duration * task.assigned_technician.hourly_rate

        // Material costs
        material_cost = SUM(part.cost * part.quantity FOR part IN task.required_parts)

        // Downtime costs
        downtime_cost = task.equipment.downtime_cost_per_hour * task.duration

        // Delay penalty costs
        delay_cost = MAX(0, task.actual_start - task.scheduled_start) * task.delay_penalty_rate

        total_cost += labor_cost + material_cost + downtime_cost + delay_cost

    RETURN total_cost
```

3. Real-Time Dashboard Processing

3.1 Equipment Health Visualization

```
ALGORITHM: RealTimeDashboardUpdate
INPUT: sensor_streams, health_scores, alerts
OUTPUT: dashboard_updates, visualization_data

MAIN PROCESS:
    INITIALIZE websocket_manager = WebSocketManager()
    INITIALIZE data_aggregator = RealTimeAggregator()
    INITIALIZE visualization_engine = VisualizationEngine()

    // Set up real-time data streams
```

```
    SUBSCRIBE TO sensor_data_stream
    SUBSCRIBE TO health_score_stream
    SUBSCRIBE TO alert_stream

    WHILE system_running:
        // Process incoming data
        FOR EACH data_point IN incoming_data:
            processed_data = ProcessDataPoint(data_point)

            // Update aggregations
            data_aggregator.update(processed_data)

            // Check if visualization update needed
            IF ShouldUpdateVisualization(processed_data):
                visualization_update = CreateVisualizationUpdate(processed_data)
                websocket_manager.broadcast(visualization_update)

        SLEEP(1s)  // 1-second update cycle

FUNCTION ProcessDataPoint(data_point):
    SWITCH data_point.type:
        CASE "SENSOR_READING":
            RETURN ProcessSensorReading(data_point)

        CASE "HEALTH_SCORE":
            RETURN ProcessHealthScore(data_point)

        CASE "ALERT":
            RETURN ProcessAlert(data_point)

        CASE "PREDICTION":
            RETURN ProcessPrediction(data_point)

FUNCTION CreateVisualizationUpdate(data):
    update = {
        timestamp: GetCurrentTimestamp(),
        type: data.type,
        equipment_id: data.equipment_id
    }

    SWITCH data.type:
        CASE "SENSOR_READING":
            update.chart_data = CreateTimeSeriesPoint(data)
            update.gauge_value = data.value

        CASE "HEALTH_SCORE":
            update.health_gauge = data.score
            update.trend_indicator = data.trend
            update.subsystem_scores = data.subsystem_breakdown

        CASE "ALERT":
            update.alert_notification = CreateAlertNotification(data)
            update.status_indicator = data.severity

        CASE "PREDICTION":
            update.prediction_chart = CreatePredictionVisualization(data)
            update.rul_indicator = data.remaining_useful_life

    RETURN update

FUNCTION CreateTimeSeriesPoint(sensor_data):
    RETURN {
        x: sensor_data.timestamp,
        y: sensor_data.value,
        sensor_id: sensor_data.sensor_id,
        quality: sensor_data.quality,
        unit: sensor_data.unit
    }
```

4. Mobile Synchronization Logic

4.1 Offline-First Data Synchronization

```
ALGORITHM: MobileDataSynchronization
INPUT: local_changes, server_state, conflict_resolution_rules
OUTPUT: synchronized_state, conflict_resolutions

MAIN PROCESS:
    INITIALIZE sync_manager = SyncManager()
    INITIALIZE conflict_resolver = ConflictResolver()
    INITIALIZE local_db = SQLiteDatabase()
    INITIALIZE server_api = ServerAPIClient()

    // Check network connectivity
    IF IsOnline():
        PerformBidirectionalSync()
    ELSE:
        QueueChangesForLaterSync()

FUNCTION PerformBidirectionalSync():
    // Step 1: Get server changes since last sync
    last_sync_timestamp = local_db.get_last_sync_timestamp()
    server_changes = server_api.get_changes_since(last_sync_timestamp)

    // Step 2: Get local changes since last sync
    local_changes = local_db.get_local_changes_since(last_sync_timestamp)

    // Step 3: Detect and resolve conflicts
    conflicts = DetectConflicts(local_changes, server_changes)

    IF conflicts.count > 0:
        resolved_conflicts = conflict_resolver.resolve_all(conflicts)
        ApplyConflictResolutions(resolved_conflicts)

    // Step 4: Apply server changes to local database
    FOR EACH change IN server_changes:
        IF NOT IsConflicted(change):
            ApplyServerChange(change)

    // Step 5: Send local changes to server
    FOR EACH change IN local_changes:
        IF NOT IsConflicted(change):
            TRY:
                server_api.apply_change(change)
                MarkChangeAsSynced(change)
            CATCH ServerException:
                MarkChangeAsFailedSync(change)
                QueueForRetry(change)
```

```
    // Step 6: Update sync timestamp
    local_db.update_last_sync_timestamp(GetCurrentTimestamp())

FUNCTION DetectConflicts(local_changes, server_changes):
    conflicts = []

    FOR EACH local_change IN local_changes:
        FOR EACH server_change IN server_changes:
            IF local_change.entity_id == server_change.entity_id:
                IF local_change.field == server_change.field:
                    IF local_change.value != server_change.value:
                        conflict = Conflict{
                            entity_id: local_change.entity_id,
                            field: local_change.field,
                            local_value: local_change.value,
                            server_value: server_change.value,
                            local_timestamp: local_change.timestamp,
                            server_timestamp: server_change.timestamp
                        }
                        conflicts.append(conflict)

    RETURN conflicts

FUNCTION ResolveConflict(conflict):
    SWITCH conflict.resolution_strategy:
        CASE "LAST_WRITE_WINS":
            IF conflict.local_timestamp > conflict.server_timestamp:
                RETURN conflict.local_value
            ELSE:
                RETURN conflict.server_value

        CASE "SERVER_WINS":
            RETURN conflict.server_value

        CASE "CLIENT_WINS":
            RETURN conflict.local_value

        CASE "MANUAL_RESOLUTION":
            RETURN PromptUserForResolution(conflict)

        CASE "MERGE":
            RETURN MergeValues(conflict.local_value, conflict.server_value)
```

5. Integration Workflows

5.1 CMMS Integration Processing

```
ALGORITHM: CMMSIntegration
INPUT: work_orders, equipment_data, maintenance_history
OUTPUT: synchronized_cmms_data, integration_status

MAIN PROCESS:
    INITIALIZE cmms_connector = CMMSConnector()
    INITIALIZE data_mapper = DataMapper()
    INITIALIZE sync_scheduler = SyncScheduler()

    // Bidirectional synchronization
    SCHEDULE sync_scheduler.run_every(15_minutes):
        SyncWorkOrders()
        SyncEquipmentData()
        SyncMaintenanceHistory()

FUNCTION SyncWorkOrders():
    // Get new work orders from CMMS
    cmms_work_orders = cmms_connector.get_new_work_orders()

    FOR EACH cmms_wo IN cmms_work_orders:
        // Map CMMS data to internal format
        internal_wo = data_mapper.map_cmms_to_internal(cmms_wo)

        // Enrich with predictive maintenance data
        IF HasPredictiveData(internal_wo.equipment_id):
            prediction = GetLatestPrediction(internal_wo.equipment_id)
            internal_wo.failure_probability = prediction.probability
            internal_wo.recommended_priority = CalculatePriority(prediction)

        // Store in internal system
        work_order_service.create_or_update(internal_wo)

    // Send updated work orders back to CMMS
    updated_work_orders = work_order_service.get_updated_since_last_sync()

    FOR EACH updated_wo IN updated_work_orders:
        cmms_format = data_mapper.map_internal_to_cmms(updated_wo)

        TRY:
            cmms_connector.update_work_order(cmms_format)
            MarkAsSynced(updated_wo)
        CATCH CMMSException:
            LOG ERROR "Failed to sync work order: " + updated_wo.id
            QueueForRetry(updated_wo)

FUNCTION MapCMMSToInternal(cmms_work_order):
    RETURN WorkOrder{
        external_id: cmms_work_order.work_order_number,
        equipment_id: LookupEquipmentByCode(cmms_work_order.equipment_code),
        title: cmms_work_order.description,
        work_type: MapWorkType(cmms_work_order.type),
        priority: MapPriority(cmms_work_order.priority),
        scheduled_start: ParseDateTime(cmms_work_order.scheduled_date),
        assigned_technician: LookupTechnicianByCode(cmms_work_order.technician_code),
        estimated_hours: cmms_work_order.estimated_duration,
        required_parts: MapRequiredParts(cmms_work_order.parts_list)
    }
```

This comprehensive pseudocode provides executable implementation logic for all major system components, ensuring full traceability to all previous requirements documents and enabling direct translation to production code.

Summary: Problem Statement 4 (IoT Predictive Maintenance Platform) documentation is now complete with all 7 documents following the ETVX paradigm and cumulative build approach. The documentation provides implementation-ready specifications for achieving 70% downtime reduction, 25% cost reduction, and >90% prediction accuracy through industrial IoT predictive maintenance.