Problem Statement 15: Healthcare Patient Risk Stratification

Problem Summary

Develop an AI-powered healthcare platform that stratifies patient risk levels using multi-modal data analysis, predictive modeling, and clinical decision support to improve patient outcomes and optimize resource allocation.

Problem Statement

Healthcare systems struggle with identifying high-risk patients early and allocating resources effectively. Your task is to build an AI system that analyzes patient data from multiple sources (EHR, lab results, imaging, wearables) to stratify risk levels, predict adverse events, and provide actionable clinical insights. The system should support real-time monitoring, integrate with existing healthcare workflows, and maintain strict privacy and regulatory compliance.

Key Requirements

Core Functionality

- **Multi-modal Data Integration**: EHR, lab results, medical imaging, wearable devices, social determinants
- **Risk Stratification Models**: ML models for various conditions (sepsis, readmission, mortality, chronic disease progression)
- **Real-time Monitoring**: Continuous patient status assessment with alert systems
- **Clinical Decision Support**: Evidence-based recommendations with confidence scores
- **Workflow Integration**: Seamless integration with existing hospital systems and EMRs
- **Regulatory Compliance**: HIPAA, FDA, HL7 FHIR standards adherence

Technical Implementation Steps

1. **Data Integration Pipeline**: Multi-source data ingestion with standardization and quality validation
2. **Feature Engineering**: Clinical feature extraction, temporal pattern analysis, risk factor identification
3. **Predictive Modeling**: Ensemble models combining clinical rules with ML algorithms
4. **Real-time Processing**: Stream processing for continuous monitoring and alerting
5. **Clinical Interface**: Dashboard for clinicians with risk scores, trends, and recommendations
6. **Audit and Compliance**: Comprehensive logging, model explainability, and regulatory reporting

Data Requirements

Primary Data Sources

- **Electronic Health Records (EHR)**: Patient demographics, medical history, medications, procedures
- **Laboratory Results**: Blood work, biomarkers, diagnostic test results with temporal trends
- **Medical Imaging**: Radiology reports, DICOM metadata, AI-extracted imaging features
- **Vital Signs**: Continuous monitoring data from bedside monitors and wearable devices
- **Social Determinants**: Socioeconomic factors, geographic data, lifestyle indicators

Supporting Datasets

- **Clinical Guidelines**: Evidence-based protocols, risk assessment tools (APACHE, SOFA scores)
- **Outcome Data**: Historical patient outcomes, readmission rates, mortality data
- **Drug Interactions**: Medication databases, adverse event reporting systems
- **Population Health**: Epidemiological data, disease prevalence, demographic health trends

Technical Themes

- **Healthcare AI & Clinical Decision Support**
- **Multi-modal Data Fusion & Integration**
- **Real-time Stream Processing & Monitoring**
- **Regulatory Compliance & Privacy-Preserving AI**
- **Explainable AI for Clinical Applications**

Expected Business Outcomes

Clinical Impact

- **Early Risk Detection**: 40% improvement in identifying high-risk patients before adverse events
- **Resource Optimization**: 25% reduction in unnecessary interventions and tests
- **Patient Outcomes**: 15% reduction in preventable readmissions and complications
- **Clinical Efficiency**: 30% reduction in time spent on manual risk assessment

Operational Benefits

- **Cost Reduction**: $2M annual savings through optimized resource allocation
- **Workflow Integration**: Seamless adoption with <2 hours additional training per clinician
- **Regulatory Compliance**: 100% adherence to HIPAA and FDA requirements
- **Scalability**: Support for 10,000+ patients across multiple hospital systems

Implementation Strategy

Phase 1: Foundation (Months 1-3)

- Data integration infrastructure and FHIR compliance
- Basic risk stratification models for common conditions
- Pilot deployment in single ICU unit

Phase 2: Enhancement (Months 4-6)

- Multi-modal data fusion and advanced ML models
- Real-time monitoring and alerting system
- Expansion to additional hospital units

Phase 3: Scale (Months 7-9)

- Full hospital deployment with workflow integration
- Advanced analytics and population health insights
- Multi-site rollout and performance optimization

Phase 4: Innovation (Months 10-12)

- AI-driven treatment recommendations and care pathways

- Integration with external health systems and HIEs
- Continuous learning and model improvement capabilities

Success Metrics

- **Clinical Accuracy**: >95% sensitivity for high-risk patient identification
- **System Performance**: <100ms response time for risk score calculations
- **User Adoption**: >90% clinician satisfaction and daily active usage
- **Compliance**: Zero HIPAA violations and full FDA validation
- **ROI**: 300% return on investment within 18 months

This healthcare AI platform will transform patient care by providing clinicians with intelligent, real-time insights to improve outcomes while optimizing resource utilization and maintaining the highest standards of privacy and regulatory compliance. # Product Requirements Document (PRD) ## Healthcare Patient Risk Stratification Platform

Document Control

- **Document Version**: 1.0
- **Created**: 2025-01-XX
- **Last Updated**: 2025-01-XX
- **Document Owner**: Product Management Team
- **Stakeholders**: Clinical Leadership, IT Operations, Regulatory Affairs, Data Science Team

---

ETVX Framework Application

Entry Criteria

- âœ… **README.md completed** - Problem statement and business case established
- âœ… **Stakeholder alignment** - Clinical leadership and IT approval obtained
- âœ… **Regulatory framework** - HIPAA, FDA, HL7 FHIR requirements documented
- âœ… **Market research** - Competitive analysis and clinical needs assessment completed
- âœ… **Technical feasibility** - Infrastructure and data availability validated

Task (This Document)

Define comprehensive product requirements including business objectives, user personas, functional specifications, success metrics, and go-to-market strategy for the Healthcare Patient Risk Stratification Platform.

Verification & Validation

- **Internal Review**: Product, Engineering, Clinical, and Legal team approval
- **Stakeholder Validation**: Clinical advisory board and pilot hospital feedback
- **Regulatory Review**: Compliance team validation of healthcare requirements
- **Technical Review**: Architecture team feasibility assessment

Exit Criteria

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

---

Executive Summary

The Healthcare Patient Risk Stratification Platform represents a transformative AI-powered solution designed to revolutionize patient care through intelligent risk assessment, early intervention, and optimized resource allocation. Building upon the foundational analysis in our README, this PRD defines the comprehensive product strategy for delivering a clinically-validated, regulatory-compliant platform that integrates seamlessly with existing healthcare workflows.

Product Vision

To become the leading AI-powered clinical decision support platform that empowers healthcare providers with real-time, actionable insights for optimal patient outcomes while ensuring the highest standards of privacy, security, and regulatory compliance.

Business Objectives

1. **Clinical Excellence**: Improve patient outcomes through early risk detection and intervention
2. **Operational Efficiency**: Optimize resource allocation and reduce healthcare costs
3. **Regulatory Leadership**: Set industry standards for AI in healthcare compliance
4. **Market Expansion**: Capture 15% market share in clinical decision support systems within 3 years

---

Market Analysis and Opportunity

Market Size and Growth

- **Total Addressable Market (TAM)**: $4.2B global clinical decision support systems market
- **Serviceable Addressable Market (SAM)**: $1.8B AI-powered healthcare analytics segment
- **Serviceable Obtainable Market (SOM)**: $270M target market for risk stratification solutions
- **Growth Rate**: 22% CAGR projected through 2028

Competitive Landscape

- **Direct Competitors**: Epic Sepsis Model, Cerner HealtheLife, IBM Watson Health
- **Indirect Competitors**: Traditional clinical scoring systems (APACHE, SOFA), manual risk assessment tools
- **Competitive Advantages**: Multi-modal data fusion, real-time processing, explainable AI, comprehensive regulatory compliance

Market Drivers

- Increasing healthcare costs and pressure for efficiency
- Growing adoption of EHR systems and digital health technologies
- Regulatory push for AI transparency and clinical validation
- COVID-19 acceleration of digital health transformation

User Personas and Stakeholders

Primary Users

1. Critical Care Physicians

- **Role**: ICU attending physicians and residents
- **Goals**: Early identification of deteriorating patients, evidence-based treatment decisions
- **Pain Points**: Information overload, time constraints, alert fatigue
- **Success Metrics**: Reduced time to intervention, improved patient outcomes

2. Nursing Staff

- **Role**: ICU and floor nurses, charge nurses
- **Goals**: Continuous patient monitoring, prioritized care delivery
- **Pain Points**: High patient-to-nurse ratios, manual documentation burden
- **Success Metrics**: Improved workflow efficiency, reduced missed critical events

3. Hospital Administrators

- **Role**: CMOs, CNOs, quality improvement directors
- **Goals**: Cost reduction, quality metrics improvement, regulatory compliance
- **Pain Points**: Resource allocation challenges, readmission penalties
- **Success Metrics**: Reduced costs, improved quality scores, compliance adherence

Secondary Users

4. Clinical Pharmacists

- **Role**: Medication management and drug interaction monitoring
- **Goals**: Optimize medication therapy, prevent adverse drug events
- **Success Metrics**: Reduced medication errors, improved therapeutic outcomes

5. Quality Improvement Teams

- **Role**: Performance monitoring and process optimization
- **Goals**: Identify improvement opportunities, track quality metrics
- **Success Metrics**: Improved quality indicators, reduced variation in care

---

Product Features and Capabilities

Core Features (MVP)

1. Multi-Modal Risk Assessment Engine

- **Description**: AI-powered risk stratification using EHR, lab, and vital sign data
- **Business Value**: Early identification of high-risk patients
- **Technical Requirements**: Real-time data processing, ML model inference
- **Success Metrics**: >95% sensitivity for high-risk patient identification

2. Clinical Decision Support Dashboard

- **Description**: Intuitive interface displaying risk scores, trends, and recommendations
- **Business Value**: Improved clinical workflow and decision-making
- **Technical Requirements**: Web-based responsive design, role-based access
- **Success Metrics**: >90% user satisfaction, <2 seconds page load time

3. Real-Time Monitoring and Alerting

- **Description**: Continuous patient monitoring with intelligent alert prioritization
- **Business Value**: Reduced response time to critical events
- **Technical Requirements**: Stream processing, configurable alert thresholds
- **Success Metrics**: 50% reduction in alert fatigue, improved response times

4. Regulatory Compliance Framework

- **Description**: Comprehensive audit trails, data governance, and privacy controls
- **Business Value**: Regulatory adherence and risk mitigation
- **Technical Requirements**: HIPAA compliance, audit logging, data encryption
- **Success Metrics**: Zero compliance violations, successful regulatory audits

Advanced Features (Future Releases)

5. Predictive Analytics Suite

- **Description**: Advanced ML models for outcome prediction and intervention recommendations
- **Business Value**: Proactive care management and resource optimization
- **Timeline**: Release 2.0 (Month 6)

6. Population Health Analytics

- **Description**: Aggregate analytics for population-level insights and quality improvement
- **Business Value**: Strategic planning and performance benchmarking
- **Timeline**: Release 3.0 (Month 9)

7. Integration Ecosystem

- **Description**: APIs and connectors for third-party systems and devices
- **Business Value**: Comprehensive data integration and workflow optimization
- **Timeline**: Release 2.0 (Month 6)

Technical Requirements

Performance Requirements

- **Response Time**: <100ms for risk score calculations
- **Throughput**: Support 10,000+ concurrent patients
- **Availability**: 99.9% uptime with <4 hours planned maintenance monthly
- **Scalability**: Horizontal scaling to support multi-hospital deployments

Security and Compliance

- **Data Encryption**: AES-256 encryption at rest and in transit
- **Access Control**: Role-based access with multi-factor authentication
- **Audit Logging**: Comprehensive audit trails for all system interactions
- **Regulatory Compliance**: HIPAA, FDA 21 CFR Part 820, HL7 FHIR R4

Integration Requirements

- **EHR Systems**: Epic, Cerner, Allscripts, MEDITECH integration
- **Data Standards**: HL7 FHIR R4, DICOM, IHE profiles
- **APIs**: RESTful APIs with OAuth 2.0 authentication
- **Real-time Data**: WebSocket connections for live data streaming

---

Success Metrics and KPIs

Clinical Outcomes

- **Primary**: 15% reduction in preventable adverse events
- **Secondary**: 25% improvement in early sepsis detection
- **Tertiary**: 20% reduction in ICU length of stay

Operational Metrics

- **Cost Savings**: $2M annual savings through optimized resource allocation
- **Efficiency**: 30% reduction in manual risk assessment time
- **Quality**: 95% accuracy in risk stratification models

User Experience

- **Adoption**: >90% daily active users among target clinicians
- **Satisfaction**: >4.5/5.0 user satisfaction score
- **Training**: <2 hours required training per user

Technical Performance

- **Reliability**: 99.9% system uptime
- **Performance**: <100ms average response time
- **Scalability**: Support for 50+ hospitals without performance degradation

---

Go-to-Market Strategy

Target Market Segmentation

1. **Primary**: Large academic medical centers (500+ beds)
2. **Secondary**: Regional health systems (200-500 beds)
3. **Tertiary**: Specialty hospitals and critical access hospitals

Sales Strategy

- **Direct Sales**: Enterprise sales team for large health systems
- **Channel Partners**: Integration with EHR vendors and healthcare consultants
- **Pilot Programs**: Free pilot implementations to demonstrate value

Pricing Model

- **Subscription**: Per-bed monthly subscription ($50-100/bed/month)
- **Implementation**: One-time setup and integration fees
- **Support**: Tiered support packages with SLA guarantees

Launch Timeline

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

Risk Assessment and Mitigation

High-Risk Items

1. **Regulatory Approval Delays**
    - **Mitigation**: Early FDA engagement, regulatory consulting
    - **Contingency**: Phased approval approach, pilot exemptions
2. **Clinical Validation Challenges**
    - **Mitigation**: Robust clinical trial design, academic partnerships
    - **Contingency**: Extended validation timeline, interim results
3. **Integration Complexity**
    - **Mitigation**: Standardized APIs, experienced integration team
    - **Contingency**: Phased integration approach, fallback options

Medium-Risk Items

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

---

Resource Requirements

Team Structure

- **Product Management**: 2 FTE (Product Manager, Clinical Product Manager)
- **Engineering**: 12 FTE (Backend, Frontend, ML, DevOps)
- **Clinical Affairs**: 3 FTE (Clinical Director, Regulatory Affairs, Quality)
- **Sales & Marketing**: 4 FTE (Sales Director, Marketing Manager, Customer Success)

Budget Allocation

- **Development**: $2.5M (60% of total budget)
- **Clinical Validation**: $800K (20% of total budget)
- **Sales & Marketing**: $500K (12% of total budget)
- **Operations**: $300K (8% of total budget)

Technology Infrastructure

- **Cloud Platform**: AWS/Azure multi-region deployment
- **Development Tools**: Modern CI/CD pipeline, automated testing
- **Monitoring**: Comprehensive observability and alerting systems

---

Assumptions and Dependencies

Key Assumptions

1. **Market Demand**: Continued growth in AI adoption in healthcare
2. **Regulatory Environment**: Stable regulatory framework for AI in healthcare
3. **Technology Maturity**: Sufficient AI/ML technology maturity for clinical applications
4. **Customer Readiness**: Healthcare organizations ready for AI integration

Critical Dependencies

1. **Data Availability**: Access to high-quality, diverse clinical datasets
2. **Regulatory Approval**: Timely FDA clearance for clinical decision support
3. **Integration Partners**: Cooperation from EHR vendors for seamless integration
4. **Clinical Champions**: Strong clinical leadership support for adoption

External Factors

1. **Healthcare Policy**: Changes in healthcare regulations and reimbursement
2. **Technology Evolution**: Advances in AI/ML technologies and standards
3. **Competitive Landscape**: New entrants and competitive responses
4. **Economic Conditions**: Healthcare spending and technology investment levels

---

Out of Scope

Excluded Features

1. **Direct Patient Care**: No direct patient treatment or medication administration
2. **Diagnostic Imaging**: Advanced medical imaging analysis beyond metadata
3. **Genomic Analysis**: Genetic testing and personalized medicine applications
4. **Telemedicine**: Remote patient monitoring and virtual care delivery

Future Considerations

1. **International Markets**: Global expansion beyond US healthcare system
2. **Consumer Applications**: Direct-to-consumer health monitoring tools
3. **Research Platform**: Clinical research and drug development applications
4. **AI Model Marketplace**: Third-party AI model integration platform

---

Conclusion

This PRD establishes the foundation for developing a transformative Healthcare Patient Risk Stratification Platform that addresses critical clinical needs while ensuring regulatory compliance and commercial viability. The comprehensive requirements outlined here, building upon our README analysis, provide clear direction for the development team and stakeholders.

The success of this platform depends on our ability to deliver clinically-validated AI solutions that integrate seamlessly with existing healthcare workflows while maintaining the highest standards of patient privacy and safety. With proper execution of this product strategy, we are positioned to become the market leader in AI-powered clinical decision support systems.

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

Document Approval

Role	Name	Signature	Date
Product Manager	[Name]	[Signature]	[Date]
Clinical Director	[Name]	[Signature]	[Date]
Engineering Lead	[Name]	[Signature]	[Date]
Regulatory Affairs	[Name]	[Signature]	[Date]
Legal Counsel	[Name]	[Signature]	[Date]

This document is confidential and proprietary. Distribution is restricted to authorized personnel only. # Functional Requirements Document (FRD) ## Healthcare Patient Risk Stratification Platform

Document Control

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

ETVX Framework Application

Entry Criteria

- âœ… **PRD Approved** - Product requirements and business objectives defined
- âœ… **Stakeholder Sign-off** - Clinical and technical teams aligned on scope
- âœ… **Architecture Review** - High-level system design validated

Task (This Document)

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

Verification & Validation

- **Requirements Traceability** - All PRD features mapped to functional requirements
- **Clinical Validation** - Medical staff review of clinical workflows
- **Technical Review** - Engineering team feasibility assessment

Exit Criteria

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

System Overview

Building upon the PRD foundation, this FRD details the functional behavior of the Healthcare Patient Risk Stratification Platform across six core modules: Patient Data Management, Risk Assessment Engine, Clinical Decision Support, Real-time Monitoring, Integration Services, and Compliance Framework.

Module 1: Patient Data Management System

FR-1.1: Multi-Source Data Ingestion

Description: Ingest patient data from multiple healthcare systems **Inputs**: EHR feeds, lab systems, monitoring devices, imaging systems **Processing**: - Parse HL7 FHIR R4 messages - Validate data integrity and completeness - Apply data quality scoring algorithms **Outputs**: Standardized patient data records **Acceptance Criteria**: - Support 99.9% uptime for data ingestion - Process 10,000+ patient records per hour - Maintain data lineage for audit trails

FR-1.2: Data Standardization and Normalization

Description: Convert diverse data formats to unified clinical data model **Inputs**: Raw clinical data in various formats **Processing**: - Apply SNOMED CT and ICD-10 coding - Normalize units of measurement - Handle missing data with clinical rules **Outputs**: Standardized clinical dataset **Acceptance Criteria**: - 95% successful data mapping accuracy - Support for 50+ different data sources - Real-time processing with <5 second latency

Module 2: AI-Powered Risk Assessment Engine

FR-2.1: Multi-Modal Risk Scoring

Description: Calculate patient risk scores using ensemble ML models **Inputs**: Standardized patient data, clinical context **Processing**: - Apply gradient boosting models for sepsis risk - Use LSTM networks for temporal pattern analysis - Combine clinical rules with ML predictions **Outputs**: Risk scores with confidence intervals **Acceptance Criteria**: - >95% sensitivity for high-risk patient identification - <100ms inference time per patient - Explainable predictions with feature importance

FR-2.2: Condition-Specific Risk Models

Description: Specialized models for different clinical conditions **Inputs**: Patient data, condition-specific parameters **Processing**: - Sepsis prediction using qSOFA and ML features - Readmission risk using social determinants - Mortality prediction using severity scores **Outputs**: Condition-specific risk assessments **Acceptance Criteria**: - Support for 10+ clinical conditions - Model performance >0.85 AUC for each condition - Daily model retraining capabilities

Module 3: Clinical Decision Support Interface

FR-3.1: Risk Dashboard

Description: Real-time dashboard displaying patient risk information **Inputs**: Risk scores, patient data, clinical context **Processing**: - Render interactive visualizations - Apply role-based access controls - Generate trend analysis charts **Outputs**: Clinical dashboard interface **Acceptance Criteria**: - <2 second page load times - Mobile-responsive design - Support for 500+ concurrent users

FR-3.2: Clinical Recommendations Engine

Description: Generate evidence-based treatment recommendations **Inputs**: Risk scores, clinical guidelines, patient history **Processing**: - Apply clinical decision trees - Rank recommendations by evidence strength - Consider contraindications and allergies **Outputs**: Prioritized recommendation list **Acceptance Criteria**: - Recommendations based on current clinical guidelines - 90% clinician acceptance rate - Integration with order entry systems

Module 4: Real-Time Monitoring and Alerting

FR-4.1: Continuous Patient Monitoring

Description: Monitor patient status changes in real-time **Inputs**: Live patient data streams, risk thresholds **Processing**: - Stream processing using Apache Kafka - Apply sliding window algorithms - Detect significant status changes **Outputs**: Real-time patient status updates **Acceptance Criteria**: - <30 second detection of critical changes - Support for 1000+ simultaneous patient streams - 99.9% alert delivery reliability

FR-4.2: Intelligent Alert Management

Description: Prioritize and deliver clinical alerts **Inputs**: Risk changes, clinical context, user preferences **Processing**: - Apply alert fatigue reduction algorithms - Route alerts based on severity and role - Implement escalation procedures **Outputs**: Prioritized alert notifications **Acceptance Criteria**: - 50% reduction in false positive alerts - <10 second alert delivery time - Multi-channel notification support

Module 5: Integration Services

FR-5.1: EHR System Integration

Description: Bidirectional integration with major EHR systems **Inputs**: EHR data feeds, risk scores, recommendations **Processing**: - HL7 FHIR R4 message processing - Real-time data synchronization - Error handling and retry mechanisms **Outputs**: Integrated clinical workflows **Acceptance Criteria**: - Support for Epic, Cerner, Allscripts - 99.9% message delivery success rate - <5 second synchronization latency

FR-5.2: Device Integration Framework

Description: Connect with medical devices and IoT sensors **Inputs**: Device data streams, configuration parameters **Processing**: - Protocol translation (MQTT, HTTP, WebSocket) - Data validation and quality checks - Device status monitoring **Outputs**: Unified device data streams **Acceptance Criteria**: - Support for 20+ device types - Real-time data processing - Automatic device discovery and configuration

Module 6: Compliance and Security Framework

FR-6.1: Audit and Compliance Management

Description: Comprehensive audit trails and compliance reporting **Inputs**: All system interactions, user activities **Processing**: - Log all data access and modifications - Generate compliance reports - Monitor for policy violations **Outputs**: Audit logs and compliance reports **Acceptance Criteria**: - 100% audit trail coverage - HIPAA compliance validation - Real-time compliance monitoring

FR-6.2: Data Privacy and Security Controls

Description: Protect patient data with advanced security measures **Inputs**: User requests, data access patterns **Processing**: - Apply role-based access controls - Encrypt data at rest and in transit - Monitor for unauthorized access **Outputs**: Secure data access and protection **Acceptance Criteria**: - AES-256 encryption implementation - Multi-factor authentication support - Zero security breaches tolerance

---

Data Flow Architecture

Primary Data Flow

1. **Data Ingestion** â†’ Multi-source data collection and validation
2. **Standardization** â†’ FHIR conversion and quality scoring
3. **Risk Assessment** â†’ ML model inference and scoring
4. **Clinical Interface** â†’ Dashboard rendering and recommendations
5. **Monitoring** â†’ Real-time alerting and escalation
6. **Integration** â†’ EHR synchronization and workflow embedding

Error Handling Workflows

- **Data Quality Issues** â†’ Flagging, manual review, correction workflows
- **Model Failures** â†’ Fallback to clinical rules, alert generation
- **Integration Errors** â†’ Retry mechanisms, alternative data sources
- **System Outages** â†’ Graceful degradation, offline mode capabilities

---

Integration Requirements

External System Interfaces

- **EHR Systems** â†’ HL7 FHIR R4 APIs with OAuth 2.0 authentication
- **Laboratory Systems** â†’ Real-time result feeds with HL7 v2.x support
- **Imaging Systems** â†’ DICOM metadata extraction and analysis
- **Pharmacy Systems** â†’ Medication reconciliation and interaction checking

API Specifications

- **RESTful APIs** â†’ JSON payloads with OpenAPI 3.0 documentation
- **WebSocket Connections** â†’ Real-time data streaming capabilities
- **Webhook Support** â†’ Event-driven notifications and updates
- **GraphQL Endpoints** â†’ Flexible data querying for dashboard applications

---

Performance Requirements

Response Time Targets

- **Risk Score Calculation** â†’ <100ms per patient assessment
- **Dashboard Loading** â†’ <2 seconds for complete interface
- **Alert Generation** â†’ <30 seconds from trigger event
- **Data Synchronization** â†’ <5 seconds for EHR updates

Scalability Specifications

- **Concurrent Users** â†’ Support 500+ simultaneous clinical users
- **Patient Volume** â†’ Handle 10,000+ active patients per hospital
- **Data Throughput** â†’ Process 1M+ clinical events per hour
- **Geographic Distribution** â†’ Multi-region deployment capabilities

---

Acceptance Criteria Summary

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

- **Clinical Accuracy** â†’ >95% sensitivity for high-risk identification
- **System Performance** â†’ <100ms response times for critical functions
- **User Experience** â†’ >90% clinician satisfaction scores
- **Integration Success** â†’ 99.9% data synchronization reliability
- **Compliance Adherence** â†’ 100% regulatory requirement fulfillment

---

Conclusion

This FRD provides comprehensive functional specifications for the Healthcare Patient Risk Stratification Platform, building upon the PRD requirements with detailed system behaviors, acceptance criteria, and integration specifications. These requirements enable the development team to proceed with technical design and implementation while ensuring full traceability to business objectives.

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

This document is confidential and proprietary. Distribution is restricted to authorized personnel only. # Non-Functional Requirements Document (NFRD) ## Healthcare Patient Risk Stratification Platform

Document Control

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

ETVX Framework Application

Entry Criteria

- âœ… **PRD Approved** - Business objectives and product features defined

- âœ... **FRD Completed** - Functional specifications documented
- âœ... **Technical Architecture** - System design approach validated

Task (This Document)

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

Verification & Validation

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

Exit Criteria

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

---

System Quality Attributes

Building upon the PRD business objectives and FRD functional specifications, this NFRD defines the quality characteristics that ensure the Healthcare Patient Risk Stratification Platform meets enterprise healthcare standards for performance, reliability, security, and regulatory compliance.

Performance Requirements

NFR-1.1: Response Time Performance

Requirement: System must provide sub-second response times for critical clinical functions **Specifications**: - Risk score calculation: <100ms (95th percentile) - Dashboard page load: <2 seconds (95th percentile) - Alert generation: <30 seconds from trigger event - API response time: <500ms for standard queries **Measurement**: Application Performance Monitoring (APM) tools **Acceptance Criteria**: - 95% of requests meet specified response times - No degradation during peak usage periods - Performance maintained under 10x normal load

NFR-1.2: Throughput Capacity

Requirement: Support high-volume clinical data processing **Specifications**: - Patient data ingestion: 10,000+ records per hour - Concurrent risk assessments: 1,000+ patients simultaneously - API requests: 10,000+ requests per minute - Real-time monitoring: 5,000+ data points per second **Measurement**: Load testing and production metrics **Acceptance Criteria**: - Linear scalability up to specified limits - No data loss during peak processing - Graceful degradation beyond capacity limits

---

Scalability Requirements

NFR-2.1: Horizontal Scalability

Requirement: System must scale horizontally to support growing healthcare organizations **Specifications**: - Auto-scaling based on CPU/memory utilization (70% threshold) - Support for multi-hospital deployments (50+ facilities) - Database sharding for patient data distribution - Microservices architecture with independent scaling **Measurement**: Infrastructure monitoring and capacity planning **Acceptance Criteria**: - Automatic scaling within 5 minutes of threshold breach - No service interruption during scaling events - Cost-effective resource utilization (>80% efficiency)

NFR-2.2: Data Volume Scalability

Requirement: Handle exponential growth in clinical data volume **Specifications**: - Patient records: 1M+ active patients per deployment - Historical data: 10+ years of clinical history - Real-time streams: 100,000+ events per second - Storage growth: 10TB+ per year per hospital **Measurement**: Database performance metrics and storage utilization **Acceptance Criteria**: - Query performance maintained with data growth - Automated data archiving and lifecycle management - Cost-optimized storage tiering implementation

---

Reliability and Availability Requirements

NFR-3.1: System Availability

Requirement: Ensure continuous availability for critical patient care **Specifications**: - Uptime: 99.9% availability (8.77 hours downtime per year) - Planned maintenance: <4 hours per month - Recovery Time Objective (RTO): <15 minutes - Recovery Point Objective (RPO): <5 minutes data loss **Measurement**: Uptime monitoring and incident tracking **Acceptance Criteria**: - No single point of failure in critical path - Automated failover mechanisms implemented - Disaster recovery tested quarterly

NFR-3.2: Fault Tolerance

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

---

Security Requirements

NFR-4.1: Data Protection

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

NFR-4.2: Access Control and Authentication

Requirement: Implement robust identity and access management **Specifications**: - Multi-factor authentication (MFA) for all users - Role-based access control (RBAC) with principle of least privilege - Single sign-on (SSO) integration with hospital identity providers - Session management with automatic timeout (30 minutes idle) **Measurement**: Access logs and security monitoring **Acceptance Criteria**: - 100% MFA adoption for clinical users - Zero unauthorized access to patient data - Compliance with organizational security policies

---

Compliance and Regulatory Requirements

NFR-5.1: Healthcare Regulatory Compliance

Requirement: Full compliance with healthcare regulations and standards **Specifications**: - HIPAA Privacy and Security Rules compliance - FDA 21 CFR Part 820 quality system requirements - HL7 FHIR R4 interoperability standards - SOC 2 Type II audit compliance **Measurement**: Compliance audits and regulatory assessments **Acceptance Criteria**: - Annual compliance certification achieved - Zero regulatory violations or penalties - Successful third-party audit completion

NFR-5.2: Data Governance and Audit

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

---

Usability Requirements

NFR-6.1: User Experience

Requirement: Intuitive interface design for clinical workflows **Specifications**: - Task completion time: <30 seconds for routine operations - Learning curve: <2 hours training for proficient use - Error rate: <1% user errors in critical functions - Accessibility: WCAG 2.1 AA compliance for disabled users **Measurement**: User experience testing and feedback collection **Acceptance Criteria**: - >90% user satisfaction scores in usability testing - <5% user error rate in production usage - Successful accessibility audit completion

NFR-6.2: Mobile and Cross-Platform Support

Requirement: Consistent experience across devices and platforms **Specifications**: - Responsive design for tablets and smartphones - Cross-browser compatibility (Chrome, Firefox, Safari, Edge) - Native mobile app for iOS and Android platforms - Offline capability for critical functions **Measurement**: Cross-platform testing and user feedback **Acceptance Criteria**: - Identical functionality across all supported platforms - <2 second load times on mobile devices - Offline mode supports 4+ hours of operation

---

Maintainability Requirements

NFR-7.1: Code Quality and Architecture

Requirement: Maintainable codebase with modern development practices **Specifications**: - Code coverage: >80% automated test coverage - Technical debt: <10% of development time spent on debt reduction - Documentation: Complete API documentation and system architecture - Code review: 100% peer review for all code changes **Measurement**: Static code analysis and development metrics **Acceptance Criteria**: - Automated quality gates prevent low-quality code deployment - New developer onboarding completed within 1 week - System architecture documentation updated with each release

NFR-7.2: Deployment and Operations

Requirement: Streamlined deployment and operational management **Specifications**: - Continuous integration/continuous deployment (CI/CD) pipeline - Infrastructure as Code (IaC) for all environments - Automated monitoring and alerting for all components - Blue-green deployment strategy for zero-downtime updates **Measurement**: Deployment metrics and operational dashboards **Acceptance Criteria**: - <15 minute deployment time for routine updates - Zero-downtime deployments for all releases - Automated rollback capability within 5 minutes

---

Interoperability Requirements

NFR-8.1: Standards Compliance

Requirement: Seamless integration with healthcare ecosystem **Specifications**: - HL7 FHIR R4 API compliance for all integrations - DICOM support for medical imaging metadata - SNOMED CT and ICD-10 terminology standards - OAuth 2.0 and OpenID Connect for authentication **Measurement**: Interoperability testing and certification **Acceptance Criteria**: - Successful integration with 5+ major EHR systems - HL7 FHIR compliance certification achieved - Zero data transformation errors in production

NFR-8.2: API Design and Management

Requirement: Well-designed APIs for third-party integration **Specifications**: - RESTful API design following OpenAPI 3.0 specification - Rate limiting: 1000 requests per minute per client - API versioning strategy with backward compatibility - Comprehensive SDK support for major programming languages **Measurement**: API usage analytics and developer feedback **Acceptance Criteria**: - >95% API uptime and availability - <100ms average API response time - Successful third-party integration within 2 weeks

---

Operational Requirements

NFR-9.1: Monitoring and Observability

Requirement: Comprehensive system monitoring and alerting **Specifications**: - Application performance monitoring (APM) with distributed tracing - Infrastructure monitoring with real-time dashboards - Log aggregation and analysis with search capabilities - Proactive alerting with escalation procedures **Measurement**: Mean Time to Detection (MTTD) and Mean Time to Resolution (MTTR) **Acceptance Criteria**: - <5 minute detection time for critical issues - <15 minute resolution time for P1 incidents - 24/7 monitoring coverage with automated escalation

NFR-9.2: Backup and Disaster Recovery

Requirement: Robust data protection and business continuity **Specifications**: - Automated daily backups with 30-day retention - Cross-region replication for disaster recovery - Recovery testing performed monthly - Business continuity plan with defined procedures **Measurement**: Backup success rates and recovery testing results **Acceptance Criteria**: - 100% backup success rate with automated verification - <15 minute RTO and <5 minute RPO for disaster recovery - Quarterly disaster recovery drills completed successfully

---

Environmental Requirements

NFR-10.1: Infrastructure and Hosting

Requirement: Cloud-native deployment with enterprise-grade infrastructure **Specifications**: - Multi-region cloud deployment (AWS/Azure/GCP) - Container orchestration using Kubernetes - Auto-scaling based on demand patterns - Content delivery network (CDN) for global performance **Measurement**: Infrastructure performance metrics and cost optimization **Acceptance Criteria**: - 99.9% infrastructure availability across all regions - <50ms latency for users within geographic regions - Cost optimization achieving <20% infrastructure overhead

NFR-10.2: Capacity Planning and Resource Management

Requirement: Efficient resource utilization and capacity planning **Specifications**: - Predictive capacity planning based on usage trends - Resource utilization targets: 70-80% for optimal efficiency - Automated resource provisioning and deprovisioning - Cost monitoring and optimization recommendations **Measurement**: Resource utilization metrics and cost analysis **Acceptance Criteria**: - Proactive capacity scaling prevents performance degradation - Resource costs remain within 10% of budget projections - Automated optimization reduces manual intervention by 90%

Constraints and Limitations

Technical Constraints

- **Legacy System Integration**: Must support HL7 v2.x for older systems
- **Network Bandwidth**: Optimize for hospital networks with limited bandwidth
- **Browser Support**: Minimum support for Internet Explorer 11
- **Mobile Device Limitations**: Graceful degradation on older mobile devices

Regulatory Constraints

- **Data Residency**: Patient data must remain within specified geographic boundaries
- **Audit Requirements**: Minimum 7-year audit log retention period
- **Validation Requirements**: FDA validation for clinical decision support features
- **Privacy Regulations**: GDPR compliance for international deployments

Operational Constraints

- **Maintenance Windows**: Limited to 2-hour windows during off-peak hours
- **Change Management**: All changes require clinical stakeholder approval
- **Training Requirements**: Maximum 4 hours of training per user role
- **Support Coverage**: 24/7 support required for critical system components

---

Quality Assurance and Testing Requirements

Performance Testing

- **Load Testing**: Simulate 10x normal user load
- **Stress Testing**: Identify system breaking points
- **Endurance Testing**: 72-hour continuous operation validation
- **Spike Testing**: Handle sudden traffic increases

Security Testing

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

Reliability Testing

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

---

Acceptance Criteria Summary

The Healthcare Patient Risk Stratification Platform must meet all specified non-functional requirements to ensure enterprise-grade quality, security, and reliability. Key acceptance thresholds include:

- **Performance**: <100ms risk calculation, 99.9% uptime
- **Security**: Zero data breaches, 100% encryption coverage
- **Compliance**: Annual regulatory certification, complete audit trails
- **Usability**: >90% user satisfaction, <2 hours training time
- **Scalability**: Support 50+ hospitals, 1M+ patients per deployment

---

Conclusion

This NFRD establishes comprehensive quality attributes and operational requirements for the Healthcare Patient Risk Stratification Platform, building upon the PRD business objectives and FRD functional specifications. These non-functional requirements ensure the system meets enterprise healthcare standards for performance, security, compliance, and operational excellence.

The specified requirements provide clear guidance for architecture design, implementation decisions, and quality assurance processes while ensuring full regulatory compliance and clinical safety standards.

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

This document is confidential and proprietary. Distribution is restricted to authorized personnel only. # Architecture Diagram (AD) ## Healthcare Patient Risk Stratification Platform

Document Control

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

ETVX Framework Application

Entry Criteria

- âœ… **PRD Approved** - Business requirements and success metrics defined
- âœ… **FRD Completed** - Functional specifications documented
- âœ… **NFRD Validated** - Quality attributes and constraints established

Task (This Document)

Define comprehensive system architecture including component design, data flows, integration patterns, security framework, and deployment architecture.

Verification & Validation

- **Architecture Review** - Technical leadership and security team validation
- **Scalability Assessment** - Performance engineering team review
- **Security Audit** - Information security team approval

Exit Criteria

- âœ… **Architecture Approved** - All stakeholders signed off on design
- âœ… **Technology Stack Validated** - Implementation feasibility confirmed

- âœ… **Integration Patterns Defined** - External system interfaces specified

Executive Architecture Summary

Building upon the PRD business objectives, FRD functional specifications, and NFRD quality attributes, this Architecture Diagram defines a cloud-native, microservices-based platform that delivers enterprise-grade healthcare AI capabilities with comprehensive security, compliance, and scalability.

High-Level System Architecture

```
â”Œâ”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â
â”‚                           PRESENTATION LAYER                                    â”‚
â”œâ”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â
â”‚  Clinical Dashboard  â”‚  Mobile App  â”‚  Admin Console  â”‚  API Documentation     â”‚
â”‚  (React.js)         â”‚  (React Native) â”‚  (Vue.js)      â”‚  (Swagger UI)         â”‚
â””â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â
                                        â”‚
                                        â–¼
â”Œâ”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â
â”‚                              API GATEWAY LAYER                                  â”‚
â”œâ”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â
â”‚  Kong API Gateway â”‚ OAuth 2.0/JWT â”‚ Rate Limiting â”‚ Request Routing â”‚ SSL Term â”‚
â””â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â
                                        â”‚
                                        â–¼
â”Œâ”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â
â”‚                            MICROSERVICES LAYER                                  â”‚
â”œâ”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”¬â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”¬â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”¬â”€â”€â
â”‚ Patient Data    â”‚ Risk Assessment â”‚ Clinical Decisionâ”‚ Real-time Monitoring    â”‚
â”‚ Service         â”‚ Service         â”‚ Support Service â”‚ Service                 â”‚
â”‚ (Node.js)       â”‚ (Python/FastAPI)â”‚ (Java/Spring)   â”‚ (Go)                    â”‚
â”œâ”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”¼â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”¼â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”¼â”€â”€â
â”‚ Integration     â”‚ Notification    â”‚ Audit & Complianceâ”‚ User Management       â”‚
â”‚ Service         â”‚ Service         â”‚ Service         â”‚ Service                 â”‚
â”‚ (Python)        â”‚ (Node.js)       â”‚ (Java)          â”‚ (Node.js)               â”‚
â””â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”´â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”´â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”´â”€â”€â
                                        â”‚
                                        â–¼
â”Œâ”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â
â”‚                           DATA PROCESSING LAYER                                 â”‚
â”œâ”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â
â”‚ Apache Kafka â”‚ Apache Spark â”‚ ML Pipeline â”‚ ETL Processes â”‚ Data Validation    â”‚
â”‚ (Streaming)  â”‚ (Batch)      â”‚ (MLflow)    â”‚ (Airflow)     â”‚ (Great Expectations)â”‚
â””â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â
                                        â”‚
                                        â–¼
â”Œâ”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â
â”‚                             DATA STORAGE LAYER                                  â”‚
â”œâ”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”¬â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”¬â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”¬â”€â”€â
â”‚ PostgreSQL      â”‚ MongoDB         â”‚ InfluxDB        â”‚ Redis                   â”‚
â”‚ (Clinical Data) â”‚ (Unstructured)  â”‚ (Time Series)   â”‚ (Cache/Sessions)        â”‚
â”œâ”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”¼â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”¼â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”¼â”€â”€â
â”‚ Elasticsearch   â”‚ MinIO           â”‚ Neo4j           â”‚ Apache Cassandra        â”‚
â”‚ (Search/Logs)   â”‚ (File Storage)  â”‚ (Knowledge Graph)â”‚ (Audit Logs)           â”‚
â””â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”´â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”´â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”´â”€â”€â
```

Detailed Component Architecture

1. Presentation Layer Components

Clinical Dashboard (React.js)

- **Purpose**: Primary clinical interface for risk assessment and decision support
- **Key Features**: Real-time patient monitoring, risk visualization, clinical recommendations
- **Technology**: React 18, TypeScript, Material-UI, Chart.js
- **Integration**: WebSocket connections for real-time updates, REST APIs for data operations

Mobile Application (React Native)

- **Purpose**: Mobile access for clinicians and on-call staff
- **Key Features**: Push notifications, offline capability, secure authentication
- **Technology**: React Native, Redux Toolkit, Expo SDK
- **Integration**: Native biometric authentication, encrypted local storage

2. API Gateway Layer

Kong API Gateway

- **Purpose**: Centralized API management, security, and routing
- **Key Features**: Request/response transformation, rate limiting, analytics
- **Security**: OAuth 2.0, JWT validation, IP whitelisting
- **Monitoring**: Request logging, performance metrics, error tracking

3. Microservices Architecture

Patient Data Service (Node.js)

- **Responsibilities**: Data ingestion, standardization, patient record management
- **APIs**: FHIR R4 endpoints, patient search, data quality validation
- **Database**: PostgreSQL for structured data, MongoDB for documents
- **Integration**: HL7 message processing, EHR system connectors

Risk Assessment Service (Python/FastAPI)

- **Responsibilities**: ML model inference, risk score calculation, prediction confidence
- **APIs**: Risk assessment endpoints, model management, batch processing
- **ML Stack**: TensorFlow, scikit-learn, MLflow for model lifecycle
- **Performance**: GPU acceleration, model caching, async processing

Clinical Decision Support Service (Java/Spring)

- **Responsibilities**: Clinical recommendations, guideline enforcement, care pathways
- **APIs**: Recommendation engine, clinical rules, evidence retrieval
- **Database**: Neo4j for clinical knowledge graphs

- **Integration**: Clinical guideline databases, drug interaction APIs

Real-time Monitoring Service (Go)

- **Responsibilities**: Stream processing, alert generation, threshold monitoring
- **Technology**: Go with Goroutines, Apache Kafka consumers
- **Performance**: High-throughput event processing, low-latency alerting
- **Scalability**: Horizontal scaling with Kubernetes HPA

Data Flow Architecture

Primary Data Flow

```
EHR Systems â†’ HL7 FHIR â†’ Patient Data Service â†’ Data Validation â†’
Standardized Storage â†’ Risk Assessment Service â†’ ML Models â†’
Risk Scores â†’ Clinical Decision Support â†’ Recommendations â†’
Clinical Dashboard â†’ Clinician Actions
```

Real-time Monitoring Flow

```
Medical Devices â†’ IoT Gateway â†’ Kafka Streams â†’ Monitoring Service â†’
Threshold Analysis â†’ Alert Generation â†’ Notification Service â†’
Clinician Alerts â†’ Dashboard Updates
```

Audit and Compliance Flow

```
All System Events â†’ Audit Service â†’ Compliance Validation â†’
Audit Logs â†’ Cassandra Storage â†’ Compliance Reports â†’
Regulatory Dashboards
```

Security Architecture

Authentication and Authorization

- **Identity Provider**: Keycloak with LDAP/AD integration
- **Authentication**: OAuth 2.0 with PKCE, JWT tokens
- **Authorization**: RBAC with fine-grained permissions
- **MFA**: TOTP, SMS, biometric authentication support

Data Protection

- **Encryption at Rest**: AES-256 with AWS KMS key management
- **Encryption in Transit**: TLS 1.3 for all communications
- **Data Masking**: Dynamic masking for non-production environments
- **Key Rotation**: Automated key rotation every 90 days

Network Security

- **VPC**: Isolated network with private subnets
- **WAF**: Web Application Firewall with OWASP rules
- **DDoS Protection**: CloudFlare enterprise protection
- **Network Segmentation**: Micro-segmentation with security groups

---

Integration Architecture

EHR System Integration

```
Epic/Cerner/Allscripts â†’ HL7 FHIR R4 â†’ API Gateway â†’
Integration Service â†’ Data Transformation â†’ Patient Data Service
```

Medical Device Integration

```
Bedside Monitors â†’ MQTT/HTTP â†’ IoT Gateway â†’
Protocol Translation â†’ Kafka â†’ Real-time Monitoring Service
```

External API Integration

```
Drug Databases â†’ REST APIs â†’ Integration Service â†’
Data Enrichment â†’ Clinical Decision Support Service
```

Deployment Architecture

Cloud Infrastructure (AWS)

- **Compute**: EKS (Kubernetes) with auto-scaling node groups
- **Storage**: RDS for databases, S3 for object storage, EFS for shared files
- **Networking**: VPC with multiple AZs, ALB for load balancing
- **Monitoring**: CloudWatch, X-Ray for distributed tracing

Container Orchestration

- **Platform**: Kubernetes 1.28+ with Helm charts
- **Service Mesh**: Istio for traffic management and security
- **Ingress**: NGINX Ingress Controller with SSL termination
- **Scaling**: HPA and VPA for automatic scaling

CI/CD Pipeline

```
GitHub â†’ GitHub Actions â†’ Docker Build â†’ Security Scan â†’
Kubernetes Deploy â†’ Health Checks â†’ Production Traffic
```

Monitoring and Observability

Application Monitoring

- **APM**: New Relic for application performance monitoring
- **Metrics**: Prometheus with Grafana dashboards

- **Logging**: ELK Stack (Elasticsearch, Logstash, Kibana)
- **Tracing**: Jaeger for distributed request tracing

Infrastructure Monitoring

- **Kubernetes**: Kubernetes Dashboard, kubectl metrics
- **Cloud**: AWS CloudWatch, AWS X-Ray
- **Alerting**: PagerDuty integration with escalation policies
- **SLA Monitoring**: Uptime monitoring with synthetic transactions

Disaster Recovery and Business Continuity

Backup Strategy

- **Database Backups**: Automated daily backups with 30-day retention
- **Cross-Region Replication**: Real-time replication to secondary region
- **File Storage**: S3 cross-region replication with versioning
- **Configuration**: GitOps approach with infrastructure as code

Disaster Recovery

- **RTO**: 15 minutes for critical services
- **RPO**: 5 minutes maximum data loss
- **Failover**: Automated failover with health checks
- **Testing**: Monthly DR drills with documented procedures

Technology Stack Summary

Frontend Technologies

- **Web**: React 18, TypeScript, Material-UI, WebSocket
- **Mobile**: React Native, Expo, Redux Toolkit
- **Visualization**: D3.js, Chart.js, Plotly

Backend Technologies

- **API Gateway**: Kong, OAuth 2.0, JWT
- **Microservices**: Node.js, Python FastAPI, Java Spring Boot, Go
- **Message Queue**: Apache Kafka, Redis Pub/Sub
- **Workflow**: Apache Airflow, Temporal

Data Technologies

- **Relational**: PostgreSQL, AWS RDS
- **NoSQL**: MongoDB, Cassandra, Neo4j
- **Time Series**: InfluxDB, TimescaleDB
- **Cache**: Redis, Memcached
- **Search**: Elasticsearch, OpenSearch

ML/AI Technologies

- **Frameworks**: TensorFlow, PyTorch, scikit-learn
- **MLOps**: MLflow, Kubeflow, DVC
- **Model Serving**: TensorFlow Serving, Seldon Core
- **Feature Store**: Feast, Tecton

DevOps Technologies

- **Containers**: Docker, Kubernetes, Helm
- **CI/CD**: GitHub Actions, ArgoCD
- **Monitoring**: Prometheus, Grafana, ELK Stack
- **Security**: Vault, Falco, OPA Gatekeeper

Conclusion

This architecture provides a comprehensive, scalable, and secure foundation for the Healthcare Patient Risk Stratification Platform. The microservices-based design ensures modularity and independent scaling, while the cloud-native approach provides enterprise-grade reliability and performance.

The architecture supports all PRD business objectives, FRD functional requirements, and NFRD quality attributes while maintaining strict healthcare compliance and security standards.

Next Steps: Proceed to High Level Design (HLD) development to detail component specifications and implementation approaches.

This document is confidential and proprietary. Distribution is restricted to authorized personnel only. # High Level Design (HLD) ## Healthcare Patient Risk Stratification Platform

Document Control

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

ETVX Framework Application

Entry Criteria

- âœ… **PRD Approved** - Business objectives and product features defined
- âœ… **FRD Completed** - Functional specifications documented
- âœ… **NFRD Validated** - Quality attributes and constraints established
- âœ… **AD Approved** - System architecture and component design finalized

Task (This Document)

Define detailed component designs, API specifications, data models, processing workflows, and integration patterns based on the established architecture.

Verification & Validation

- **Design Review** - Engineering team validation of component specifications
- **API Review** - Integration team validation of interface designs
- **Data Model Review** - Database team validation of schema designs

Exit Criteria

- âœ… **Component Designs Complete** - All system components detailed
- âœ… **API Specifications Defined** - Interface contracts documented
- âœ… **Data Models Validated** - Database schemas and relationships specified

---

System Component Design

Building upon the PRD business objectives, FRD functional specifications, NFRD quality attributes, and AD system architecture, this HLD provides detailed component designs that enable implementation-ready development.

---

1. Patient Data Service Component

Component Overview

- **Technology**: Node.js 18+ with Express.js framework
- **Database**: PostgreSQL 15+ for structured data, MongoDB 6+ for documents
- **Message Queue**: Apache Kafka for event streaming
- **Caching**: Redis for session and query caching

API Design

Patient Data Ingestion API

```
POST /api/v1/patients/ingest
Content-Type: application/fhir+json

{
  "resourceType": "Bundle",
  "type": "transaction",
  "entry": [
    {
      "resource": {
        "resourceType": "Patient",
        "identifier": [{"system": "hospital-mrn", "value": "12345"}],
        "name": [{"family": "Doe", "given": ["John"]}],
        "birthDate": "1980-01-01",
        "gender": "male"
      }
    }
  ]
}
```

Patient Search API

```
GET /api/v1/patients/search?identifier=12345&active=true
Authorization: Bearer {jwt_token}

Response:
{
  "total": 1,
  "patients": [
    {
      "id": "patient-uuid",
      "mrn": "12345",
      "demographics": {...},
      "riskScore": 0.75,
      "lastUpdated": "2025-01-01T10:00:00Z"
    }
  ]
}
```

Data Processing Workflow

1. **HL7 FHIR Message Reception** â†’ Validate message structure and authentication
2. **Data Quality Assessment** â†’ Apply clinical data validation rules
3. **Standardization** â†’ Convert to internal data model with SNOMED CT coding
4. **Deduplication** â†’ Identify and merge duplicate patient records
5. **Event Publishing** â†’ Publish patient update events to Kafka
6. **Audit Logging** â†’ Record all data access and modifications

Database Schema Design

```
-- Patient Master Table
CREATE TABLE patients (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    mrn VARCHAR(50) UNIQUE NOT NULL,
    external_id VARCHAR(100),
    demographics JSONB NOT NULL,
    active BOOLEAN DEFAULT true,
    created_at TIMESTAMP DEFAULT NOW(),
    updated_at TIMESTAMP DEFAULT NOW(),
    CONSTRAINT valid_demographics CHECK (demographics ? 'name' AND demographics ? 'birthDate')
);

-- Clinical Data Table
CREATE TABLE clinical_data (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    patient_id UUID REFERENCES patients(id),
    data_type VARCHAR(50) NOT NULL, -- 'lab', 'vital', 'medication', etc.
    data_payload JSONB NOT NULL,
    effective_date TIMESTAMP NOT NULL,
    source_system VARCHAR(100),
    created_at TIMESTAMP DEFAULT NOW(),
    INDEX idx_patient_type_date (patient_id, data_type, effective_date)
);
```

---

2. Risk Assessment Service Component

Component Overview

- **Technology**: Python 3.11+ with FastAPI framework

- **ML Framework**: TensorFlow 2.13+, scikit-learn 1.3+
- **Model Management**: MLflow for experiment tracking and model registry
- **Compute**: GPU acceleration with CUDA support for deep learning models

ML Model Architecture

Sepsis Risk Model

```
class SepsisRiskModel:
    def __init__(self):
        self.feature_extractor = ClinicalFeatureExtractor()
        self.lstm_model = tf.keras.Sequential([
            tf.keras.layers.LSTM(128, return_sequences=True),
            tf.keras.layers.Dropout(0.3),
            tf.keras.layers.LSTM(64),
            tf.keras.layers.Dense(32, activation='relu'),
            tf.keras.layers.Dense(1, activation='sigmoid')
        ])

    def predict_risk(self, patient_data, time_window=24):
        features = self.feature_extractor.extract(patient_data, time_window)
        risk_score = self.lstm_model.predict(features)
        confidence = self.calculate_confidence(features)
        return {
            'risk_score': float(risk_score[0][0]),
            'confidence': float(confidence),
            'contributing_factors': self.explain_prediction(features)
        }
```

Feature Engineering Pipeline

```
class ClinicalFeatureExtractor:
    def extract(self, patient_data, time_window):
        features = {}

        # Vital signs features
        features.update(self.extract_vital_signs(patient_data, time_window))

        # Laboratory values features
        features.update(self.extract_lab_values(patient_data, time_window))

        # Medication features
        features.update(self.extract_medications(patient_data))

        # Demographic features
        features.update(self.extract_demographics(patient_data))

        return self.normalize_features(features)
```

API Design

Risk Assessment API

```
@app.post("/api/v1/risk/assess")
async def assess_patient_risk(request: RiskAssessmentRequest):
    """
    Assess patient risk for multiple conditions
    """
    patient_data = await get_patient_data(request.patient_id)

    risk_scores = {}
    for condition in request.conditions:
        model = get_model(condition)
        risk_scores[condition] = model.predict_risk(patient_data)

    return RiskAssessmentResponse(
        patient_id=request.patient_id,
        assessment_time=datetime.utcnow(),
        risk_scores=risk_scores,
        recommendations=generate_recommendations(risk_scores)
    )
```

Model Training Pipeline

```
class ModelTrainingPipeline:
    def __init__(self):
        self.mlflow_client = mlflow.tracking.MlflowClient()

    def train_model(self, model_type, training_data):
        with mlflow.start_run():
            # Data preprocessing
            X_train, X_val, y_train, y_val = self.preprocess_data(training_data)

            # Model training
            model = self.create_model(model_type)
            model.fit(X_train, y_train, validation_data=(X_val, y_val))

            # Model evaluation
            metrics = self.evaluate_model(model, X_val, y_val)

            # Log metrics and model
            mlflow.log_metrics(metrics)
            mlflow.tensorflow.log_model(model, "model")

            return model
```

---

3. Clinical Decision Support Service Component

Component Overview

- **Technology**: Java 17+ with Spring Boot 3.0+
- **Database**: Neo4j 5+ for clinical knowledge graphs
- **Rules Engine**: Drools for clinical decision rules
- **Cache**: Redis for recommendation caching

Clinical Knowledge Graph Design

```
// Clinical entities and relationships
CREATE (p:Patient {id: 'patient-123', age: 65, gender: 'M'})
CREATE (c:Condition {code: 'I50.9', name: 'Heart Failure'})
CREATE (m:Medication {code: 'RxNorm-123', name: 'Lisinopril'})
CREATE (g:Guideline {id: 'AHA-HF-2022', title: 'Heart Failure Guidelines'})
```

```
// Relationships
CREATE (p)-[:HAS_CONDITION]->(c)
CREATE (p)-[:PRESCRIBED]->(m)
CREATE (g)-[:RECOMMENDS]->(m)
CREATE (c)-[:TREATED_BY]->(m)
```

Recommendation Engine

```
@Service
public class ClinicalRecommendationService {

    @Autowired
    private Neo4jTemplate neo4jTemplate;

    @Autowired
    private DroolsRulesEngine rulesEngine;

    public List<ClinicalRecommendation> generateRecommendations(
            String patientId, Map<String, Double> riskScores) {

        // Query knowledge graph for patient context
        PatientContext context = getPatientContext(patientId);

        // Apply clinical rules
        List<ClinicalRecommendation> recommendations =
            rulesEngine.executeRules(context, riskScores);

        // Rank recommendations by evidence strength
        return rankRecommendations(recommendations);
    }

    private PatientContext getPatientContext(String patientId) {
        String cypher = """
            MATCH (p:Patient {id: $patientId})
            OPTIONAL MATCH (p)-[:HAS_CONDITION]->(c:Condition)
            OPTIONAL MATCH (p)-[:PRESCRIBED]->(m:Medication)
            RETURN p, collect(c) as conditions, collect(m) as medications
            """;

        return neo4jTemplate.findOne(cypher,
            Map.of("patientId", patientId), PatientContext.class);
    }
}
```

Clinical Rules Definition

```
// Drools rule example
rule "High Sepsis Risk Alert"
when
    $patient : PatientContext()
    $riskScore : Double(this > 0.8) from $patient.getRiskScore("sepsis")
then
    ClinicalRecommendation recommendation = new ClinicalRecommendation();
    recommendation.setType("ALERT");
    recommendation.setPriority("HIGH");
    recommendation.setMessage("High sepsis risk detected - consider immediate evaluation");
    recommendation.setActions(Arrays.asList("Order blood cultures", "Consider antibiotics"));
    insert(recommendation);
end
```

---

4. Real-time Monitoring Service Component

Component Overview

- **Technology**: Go 1.21+ with Goroutines for concurrency
- **Message Processing**: Apache Kafka consumers with consumer groups
- **Time Series Database**: InfluxDB for metrics storage
- **Alerting**: Custom alerting engine with escalation policies

Stream Processing Architecture

```
type MonitoringService struct {
    kafkaConsumer *kafka.Consumer
    influxClient  influxdb2.Client
    alertManager  *AlertManager
    ruleEngine    *RuleEngine
}

func (ms *MonitoringService) ProcessPatientStream(ctx context.Context) {
    for {
        select {
        case <-ctx.Done():
            return
        default:
            msg, err := ms.kafkaConsumer.ReadMessage(100 * time.Millisecond)
            if err != nil {
                continue
            }

            go ms.processPatientEvent(msg.Value)
        }
    }
}

func (ms *MonitoringService) processPatientEvent(data []byte) {
    var event PatientEvent
    if err := json.Unmarshal(data, &event); err != nil {
        log.Error("Failed to unmarshal event", err)
        return
    }

    // Store metrics
    ms.storeMetrics(event)

    // Evaluate alert rules
    alerts := ms.ruleEngine.EvaluateRules(event)
    for _, alert := range alerts {
        ms.alertManager.TriggerAlert(alert)
    }
}
```

Alert Management System

```
type AlertManager struct {
    notificationService *NotificationService
    escalationPolicies  map[string]EscalationPolicy
}

type Alert struct {
    ID          string    `json:"id"`
    PatientID   string    `json:"patient_id"`
    Severity    string    `json:"severity"`
    Message     string    `json:"message"`
    Timestamp   time.Time `json:"timestamp"`
    Acknowledged bool     `json:"acknowledged"`
}

func (am *AlertManager) TriggerAlert(alert Alert) {
    // Store alert
    am.storeAlert(alert)

    // Send immediate notification
    am.notificationService.SendNotification(alert)

    // Start escalation timer if not acknowledged
    if alert.Severity == "CRITICAL" {
        go am.startEscalation(alert)
    }
}
```

5. Integration Service Component

Component Overview

- **Technology**: Python 3.11+ with asyncio for concurrent processing
- **Integration Framework**: Apache Camel for enterprise integration patterns
- **Protocol Support**: HL7 v2.x, HL7 FHIR R4, REST APIs, MQTT
- **Message Transformation**: Custom transformation engine

HL7 FHIR Integration

```
class FHIRIntegrationService:
    def __init__(self):
        self.fhir_client = FHIRClient(base_url=settings.FHIR_SERVER_URL)
        self.transformer = FHIRTransformer()

    async def sync_patient_data(self, patient_id: str):
        """Synchronize patient data from EHR system"""
        try:
            # Fetch patient bundle from FHIR server
            bundle = await self.fhir_client.get_patient_bundle(patient_id)

            # Transform to internal format
            internal_data = self.transformer.transform_bundle(bundle)

            # Validate data quality
            validation_result = await self.validate_data(internal_data)

            if validation_result.is_valid:
                # Publish to patient data service
                await self.publish_patient_update(internal_data)
            else:
                await self.handle_validation_errors(validation_result.errors)

        except Exception as e:
            logger.error(f"Failed to sync patient {patient_id}: {e}")
            await self.handle_sync_error(patient_id, e)
```

Medical Device Integration

```
class DeviceIntegrationService:
    def __init__(self):
        self.mqtt_client = mqtt.Client()
        self.device_registry = DeviceRegistry()

    async def handle_device_data(self, topic: str, payload: bytes):
        """Process incoming device data"""
        device_id = self.extract_device_id(topic)
        device_config = self.device_registry.get_device(device_id)

        # Parse device-specific data format
        parsed_data = self.parse_device_data(payload, device_config.protocol)

        # Validate data ranges
        validated_data = self.validate_device_data(parsed_data, device_config.ranges)

        # Transform to standard format
        standard_data = self.transform_to_standard(validated_data, device_config.mapping)

        # Publish to monitoring service
        await self.publish_device_event(standard_data)
```

6. Data Processing Pipeline

ETL Pipeline Design

```
class ClinicalDataETL:
    def __init__(self):
        self.spark = SparkSession.builder.appName("ClinicalETL").getOrCreate()
        self.data_quality = DataQualityValidator()

    def process_clinical_data(self, source_path: str, target_path: str):
        """Process clinical data with quality checks"""

        # Extract
        raw_data = self.spark.read.json(source_path)

        # Transform
        cleaned_data = self.clean_data(raw_data)
        standardized_data = self.standardize_codes(cleaned_data)
        enriched_data = self.enrich_with_external_data(standardized_data)

        # Validate
        quality_report = self.data_quality.validate(enriched_data)

        if quality_report.passed:
            # Load
            enriched_data.write.mode("overwrite").parquet(target_path)
        else:
            self.handle_quality_failures(quality_report)
```

Feature Store Implementation

```
class ClinicalFeatureStore:
    def __init__(self):
        self.feast_client = feast.Client()

    def create_feature_views(self):
        """Define clinical feature views"""

        # Patient demographics features
        patient_demographics = FeatureView(
            name="patient_demographics",
            entities=["patient_id"],
            features=[
                Feature(name="age", dtype=ValueType.INT64),
                Feature(name="gender", dtype=ValueType.STRING),
                Feature(name="bmi", dtype=ValueType.DOUBLE)
            ],
            source=BigQuerySource(
                table="clinical_data.patient_demographics",
                timestamp_field="updated_at"
            )
        )

        # Vital signs features
        vital_signs = FeatureView(
            name="vital_signs_24h",
            entities=["patient_id"],
            features=[
                Feature(name="avg_heart_rate", dtype=ValueType.DOUBLE),
                Feature(name="max_temperature", dtype=ValueType.DOUBLE),
                Feature(name="min_blood_pressure", dtype=ValueType.DOUBLE)
            ],
            source=BigQuerySource(
                table="clinical_data.vital_signs_aggregated",
                timestamp_field="window_end"
            )
        )

        return [patient_demographics, vital_signs]
```

---

7. Security and Compliance Framework

Authentication Service

```
@RestController
@RequestMapping("/api/v1/auth")
public class AuthenticationController {

    @Autowired
    private JwtTokenProvider tokenProvider;

    @Autowired
    private UserService userService;

    @PostMapping("/login")
    public ResponseEntity<AuthResponse> authenticate(
            @RequestBody @Valid LoginRequest request) {

        // Validate credentials
        User user = userService.validateCredentials(
            request.getUsername(), request.getPassword());

        if (user == null) {
            throw new BadCredentialsException("Invalid credentials");
        }

        // Generate JWT token
        String token = tokenProvider.generateToken(user);

        // Log authentication event
        auditService.logAuthenticationEvent(user, request.getClientInfo());

        return ResponseEntity.ok(new AuthResponse(token, user.getRoles()));
    }

    @PostMapping("/refresh")
    public ResponseEntity<AuthResponse> refreshToken(
            @RequestHeader("Authorization") String refreshToken) {

        if (tokenProvider.validateToken(refreshToken)) {
            String username = tokenProvider.getUsernameFromToken(refreshToken);
            User user = userService.findByUsername(username);
            String newToken = tokenProvider.generateToken(user);

            return ResponseEntity.ok(new AuthResponse(newToken, user.getRoles()));
        }

        throw new InvalidTokenException("Invalid refresh token");
    }
}
```

Audit Logging Service

```
@Service
public class AuditService {

    @Autowired
    private AuditRepository auditRepository;

    @EventListener
    public void handleDataAccessEvent(DataAccessEvent event) {
        AuditLog auditLog = AuditLog.builder()
            .userId(event.getUserId())
            .action(event.getAction())
            .resourceType(event.getResourceType())
            .resourceId(event.getResourceId())
            .timestamp(Instant.now())
            .clientIp(event.getClientIp())
            .userAgent(event.getUserAgent())
            .success(event.isSuccess())
            .build();

        auditRepository.save(auditLog);

        // Send to compliance monitoring
        complianceMonitor.processAuditEvent(auditLog);
    }
}
```

---

8. Performance Optimization

Caching Strategy

```
@Service
public class CacheService {

    @Autowired
    private RedisTemplate<String, Object> redisTemplate;

    @Cacheable(value = "patient-risk-scores", key = "#patientId")
    public RiskAssessment getCachedRiskAssessment(String patientId) {
        return riskAssessmentService.calculateRisk(patientId);
    }

    @CacheEvict(value = "patient-risk-scores", key = "#patientId")
    public void evictPatientCache(String patientId) {
        // Cache will be evicted automatically
    }

    @Scheduled(fixedRate = 300000) // 5 minutes
    public void refreshHighRiskPatients() {
        List<String> highRiskPatients = getHighRiskPatientIds();
        for (String patientId : highRiskPatients) {
            // Warm cache for high-risk patients
            getCachedRiskAssessment(patientId);
        }
    }
}
```

Database Optimization

```
-- Optimized indexes for common queries
CREATE INDEX CONCURRENTLY idx_patients_mrn_active
ON patients(mrn) WHERE active = true;

CREATE INDEX CONCURRENTLY idx_clinical_data_patient_date
ON clinical_data(patient_id, effective_date DESC);

CREATE INDEX CONCURRENTLY idx_risk_scores_patient_condition
ON risk_scores(patient_id, condition_type, calculated_at DESC);

-- Partitioning for large tables
CREATE TABLE clinical_data_2025 PARTITION OF clinical_data
FOR VALUES FROM ('2025-01-01') TO ('2026-01-01');
```

Conclusion

This High Level Design provides comprehensive component specifications that enable implementation-ready development of the Healthcare Patient Risk Stratification Platform. Each component is designed to meet the PRD business objectives, FRD functional requirements, NFRD quality attributes, and AD architectural constraints.

The detailed API specifications, data models, and processing workflows ensure consistent implementation across all development teams while maintaining enterprise-grade security, performance, and compliance standards.

Next Steps: Proceed to Low Level Design (LLD) development to define implementation-specific details, database schemas, and deployment configurations.

This document is confidential and proprietary. Distribution is restricted to authorized personnel only. # Low Level Design (LLD) ## Healthcare Patient Risk Stratification Platform

Document Control

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

ETVX Framework Application

Entry Criteria

- âœ… **PRD Approved** - Business objectives and product features defined
- âœ… **FRD Completed** - Functional specifications documented
- âœ… **NFRD Validated** - Quality attributes and constraints established
- âœ… **AD Approved** - System architecture and component design finalized
- âœ… **HLD Completed** - Component designs and API specifications detailed

Task (This Document)

Define implementation-ready specifications including class structures, database schemas, API implementations, configuration files, and deployment scripts.

Verification & Validation

- **Code Review** - Development team validation of implementation specifications
- **Database Review** - DBA team validation of schema designs and performance
- **Security Review** - Security team validation of implementation security controls

Exit Criteria

- âœ… **Implementation Specs Complete** - All code structures and schemas defined
- âœ… **Configuration Ready** - Deployment and runtime configurations specified
- âœ… **Development Ready** - Teams can begin implementation with clear specifications

---

Implementation Overview

Building upon the PRD business objectives, FRD functional specifications, NFRD quality attributes, AD system architecture, and HLD component designs, this LLD provides implementation-ready specifications for immediate development.

1. Database Schema Implementation

PostgreSQL Schema (Clinical Data)

```
-- Database: healthcare_risk_platform
-- Version: PostgreSQL 15+

-- Enable required extensions
CREATE EXTENSION IF NOT EXISTS "uuid-ossp";
CREATE EXTENSION IF NOT EXISTS "pg_trgm";
CREATE EXTENSION IF NOT EXISTS "btree_gin";

-- Patients table with comprehensive indexing
CREATE TABLE patients (
    id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    mrn VARCHAR(50) UNIQUE NOT NULL,
    external_id VARCHAR(100),
    demographics JSONB NOT NULL,
    active BOOLEAN DEFAULT true,
    created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
    updated_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
    version INTEGER DEFAULT 1,
    CONSTRAINT valid_demographics CHECK (
        demographics ? 'name' AND
        demographics ? 'birthDate' AND
        demographics ? 'gender'
    )
);

-- Optimized indexes for patient queries
CREATE INDEX CONCURRENTLY idx_patients_mrn ON patients(mrn) WHERE active = true;
CREATE INDEX CONCURRENTLY idx_patients_demographics_gin ON patients USING gin(demographics);
CREATE INDEX CONCURRENTLY idx_patients_updated_at ON patients(updated_at DESC);

-- Clinical data with partitioning by date
CREATE TABLE clinical_data (
    id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    patient_id UUID NOT NULL REFERENCES patients(id) ON DELETE CASCADE,
    data_type VARCHAR(50) NOT NULL,
    data_payload JSONB NOT NULL,
    effective_date TIMESTAMP WITH TIME ZONE NOT NULL,
    source_system VARCHAR(100) NOT NULL,
    quality_score DECIMAL(3,2) DEFAULT 1.0,
    created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
    CONSTRAINT valid_quality_score CHECK (quality_score >= 0.0 AND quality_score <= 1.0)
) PARTITION BY RANGE (effective_date);

-- Create partitions for current and future years
CREATE TABLE clinical_data_2024 PARTITION OF clinical_data
FOR VALUES FROM ('2024-01-01') TO ('2025-01-01');

CREATE TABLE clinical_data_2025 PARTITION OF clinical_data
FOR VALUES FROM ('2025-01-01') TO ('2026-01-01');

-- Risk scores table
CREATE TABLE risk_scores (
    id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    patient_id UUID NOT NULL REFERENCES patients(id) ON DELETE CASCADE,
    condition_type VARCHAR(50) NOT NULL,
    risk_score DECIMAL(5,4) NOT NULL,
    confidence_score DECIMAL(5,4) NOT NULL,
    model_version VARCHAR(20) NOT NULL,
    contributing_factors JSONB,
    calculated_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
    expires_at TIMESTAMP WITH TIME ZONE,
    CONSTRAINT valid_risk_score CHECK (risk_score >= 0.0 AND risk_score <= 1.0),
    CONSTRAINT valid_confidence CHECK (confidence_score >= 0.0 AND confidence_score <= 1.0)
);

CREATE INDEX idx_risk_scores_patient_condition
ON risk_scores(patient_id, condition_type, calculated_at DESC);
```

2. FastAPI Implementation

Risk Assessment Service

```
# File: risk_assessment_service/main.py
from fastapi import FastAPI, HTTPException, Depends, BackgroundTasks
from fastapi.security import HTTPBearer, HTTPAuthorizationCredentials
from pydantic import BaseModel, Field
from typing import List, Dict, Optional
import asyncio
import logging
from datetime import datetime, timedelta

app = FastAPI(
    title="Healthcare Risk Assessment API",
    version="1.0.0",
    description="AI-powered patient risk stratification service"
)

security = HTTPBearer()

# Request/Response Models
class RiskAssessmentRequest(BaseModel):
    patient_id: str = Field(..., description="Unique patient identifier")
    conditions: List[str] = Field(
        default=["sepsis", "readmission", "mortality"],
        description="Conditions to assess risk for"
    )
    time_window_hours: int = Field(
        default=24,
        ge=1,
        le=168,
        description="Time window for assessment in hours"
    )
    include_explanations: bool = Field(
        default=True,
        description="Include AI explanations in response"
    )
```

```
class RiskScore(BaseModel):
    condition: str
    risk_score: float = Field(..., ge=0.0, le=1.0)
    confidence: float = Field(..., ge=0.0, le=1.0)
    risk_level: str = Field(..., regex="^(LOW|MODERATE|HIGH|CRITICAL)$")
    contributing_factors: Optional[List[Dict[str, float]]] = None
    recommendations: Optional[List[str]] = None

class RiskAssessmentResponse(BaseModel):
    patient_id: str
    assessment_timestamp: datetime
    risk_scores: List[RiskScore]
    overall_risk_level: str
    next_assessment_due: datetime
    model_versions: Dict[str, str]

# Main risk assessment endpoint
@app.post("/api/v1/risk/assess", response_model=RiskAssessmentResponse)
async def assess_patient_risk(
    request: RiskAssessmentRequest,
    background_tasks: BackgroundTasks,
    current_user = Depends(get_current_user)
):
    """Assess patient risk for specified conditions"""
    try:
        # Validate patient exists and user has access
        patient = await validate_patient_access(request.patient_id, current_user)

        # Fetch patient data
        patient_data = await get_patient_clinical_data(
            request.patient_id,
            request.time_window_hours
        )

        # Parallel risk assessment for all conditions
        risk_tasks = [
            assess_condition_risk(condition, patient_data, request.include_explanations)
            for condition in request.conditions
        ]

        risk_scores = await asyncio.gather(*risk_tasks)

        # Calculate overall risk level
        overall_risk = calculate_overall_risk(risk_scores)

        # Schedule next assessment
        next_assessment = calculate_next_assessment_time(risk_scores)

        # Log assessment for audit
        background_tasks.add_task(
            log_risk_assessment,
            request.patient_id,
            current_user.id,
            risk_scores
        )

        return RiskAssessmentResponse(
            patient_id=request.patient_id,
            assessment_timestamp=datetime.utcnow(),
            risk_scores=risk_scores,
            overall_risk_level=overall_risk,
            next_assessment_due=next_assessment,
            model_versions=get_model_versions()
        )

    except Exception as e:
        logger.error(f"Risk assessment failed for patient {request.patient_id}: {e}")
        raise HTTPException(status_code=500, detail="Risk assessment failed")

# Health check endpoint
@app.get("/health")
async def health_check():
    return {
        "status": "healthy",
        "timestamp": datetime.utcnow(),
        "version": "1.0.0",
        "models_loaded": len(await get_loaded_models())
    }
```

---

3. Docker Configuration

Docker Compose

```
# File: docker-compose.yml
version: '3.8'

services:
  # API Gateway
  kong:
    image: kong:3.4
    environment:
      KONG_DATABASE: postgres
      KONG_PG_HOST: postgres
      KONG_PG_DATABASE: kong
      KONG_PG_USER: kong
      KONG_PG_PASSWORD: ${KONG_PG_PASSWORD}
      KONG_PROXY_ACCESS_LOG: /dev/stdout
      KONG_ADMIN_ACCESS_LOG: /dev/stdout
      KONG_PROXY_ERROR_LOG: /dev/stderr
      KONG_ADMIN_ERROR_LOG: /dev/stderr
      KONG_ADMIN_LISTEN: 0.0.0.0:8001
    ports:
      - "8000:8000"
      - "8001:8001"
    depends_on:
      - postgres
    networks:
      - healthcare-network

  # Risk Assessment Service
  risk-assessment-service:
    build:
      context: ./risk-assessment-service
      dockerfile: Dockerfile
    environment:
      DATABASE_URL: postgresql://postgres:${POSTGRES_PASSWORD}@postgres:5432/healthcare
      REDIS_URL: redis://redis:6379
      MLFLOW_TRACKING_URI: http://mlflow:5000
      MODEL_REGISTRY_URI: s3://healthcare-models/
    ports:
      - "8002:8000"
    depends_on:
      - postgres
      - redis
      - mlflow
    networks:
      - healthcare-network
    deploy:
      replicas: 2
      resources:
        limits:
          memory: 4G
          cpus: '2.0'

  # Databases
  postgres:
    image: postgres:15
    environment:
      POSTGRES_DB: healthcare
      POSTGRES_USER: postgres
      POSTGRES_PASSWORD: ${POSTGRES_PASSWORD}
      POSTGRES_INITDB_ARGS: "--encoding=UTF-8 --lc-collate=C --lc-ctype=C"
    volumes:
      - postgres_data:/var/lib/postgresql/data
      - ./init-scripts:/docker-entrypoint-initdb.d
    ports:
      - "5432:5432"
    networks:
      - healthcare-network

  redis:
    image: redis:7-alpine
    command: redis-server --appendonly yes --requirepass ${REDIS_PASSWORD}
    volumes:
      - redis_data:/data
    ports:
      - "6379:6379"
    networks:
      - healthcare-network

volumes:
  postgres_data:
  redis_data:

networks:
  healthcare-network:
    driver: bridge
```

---

4. Kubernetes Deployment

Risk Assessment Deployment

```
# File: k8s/risk-assessment-deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: risk-assessment-service
  namespace: healthcare
  labels:
    app: risk-assessment-service
    version: v1.0.0
spec:
  replicas: 3
  selector:
    matchLabels:
      app: risk-assessment-service
  template:
    metadata:
      labels:
        app: risk-assessment-service
        version: v1.0.0
    spec:
      containers:
      - name: risk-assessment
        image: healthcare/risk-assessment:1.0.0
        ports:
        - containerPort: 8000
        env:
        - name: DATABASE_URL
          valueFrom:
            secretKeyRef:
              name: database-secret
              key: url
        - name: REDIS_URL
          valueFrom:
            secretKeyRef:
              name: redis-secret
              key: url
        resources:
          requests:
            memory: "2Gi"
            cpu: "1000m"
          limits:
            memory: "4Gi"
            cpu: "2000m"
        livenessProbe:
          httpGet:
            path: /health
            port: 8000
          initialDelaySeconds: 30
          periodSeconds: 10
        readinessProbe:
          httpGet:
            path: /ready
            port: 8000
          initialDelaySeconds: 5
          periodSeconds: 5
---
apiVersion: v1
kind: Service
metadata:
  name: risk-assessment-service
  namespace: healthcare
spec:
  selector:
    app: risk-assessment-service
  ports:
  - port: 80
    targetPort: 8000
    protocol: TCP
  type: ClusterIP
```

5. Security Implementation

JWT Token Validation

```
# File: auth/jwt_validator.py
import jwt
from datetime import datetime, timedelta
from typing import Optional, Dict
from cryptography.hazmat.primitives import serialization

class JWTValidator:
    def __init__(self, public_key_path: str, algorithm: str = "RS256"):
        with open(public_key_path, 'rb') as key_file:
            self.public_key = serialization.load_pem_public_key(key_file.read())
        self.algorithm = algorithm

    async def validate_token(self, token: str) -> Optional[Dict]:
        """Validate JWT token and return user claims"""
        try:
            # Decode and validate token
            payload = jwt.decode(
                token,
                self.public_key,
                algorithms=[self.algorithm],
                options={"verify_exp": True, "verify_aud": True}
            )

            # Additional validation
            if not self._validate_claims(payload):
                return None

            # Check if user is active
            if not await self._is_user_active(payload.get('sub')):
                return None

            return payload

        except jwt.ExpiredSignatureError:
            logger.warning("Token expired")
            return None
        except jwt.InvalidTokenError as e:
            logger.warning(f"Invalid token: {e}")
            return None

    def _validate_claims(self, payload: Dict) -> bool:
        """Validate required claims"""
        required_claims = ['sub', 'iat', 'exp', 'aud', 'roles']
        return all(claim in payload for claim in required_claims)
```

Data Encryption Service

```
# File: security/encryption.py
from cryptography.fernet import Fernet
from cryptography.hazmat.primitives import hashes
from cryptography.hazmat.primitives.kdf.pbkdf2 import PBKDF2HMAC
import base64
import os

class DataEncryption:
    def __init__(self, master_key: str):
        self.master_key = master_key.encode()
        self.fernet = self._create_fernet()

    def _create_fernet(self) -> Fernet:
        """Create Fernet instance with derived key"""
        salt = os.urandom(16)
        kdf = PBKDF2HMAC(
            algorithm=hashes.SHA256(),
            length=32,
            salt=salt,
            iterations=100000,
        )
        key = base64.urlsafe_b64encode(kdf.derive(self.master_key))
        return Fernet(key)

    def encrypt_pii(self, data: str) -> str:
        """Encrypt personally identifiable information"""
        encrypted_data = self.fernet.encrypt(data.encode())
        return base64.urlsafe_b64encode(encrypted_data).decode()

    def decrypt_pii(self, encrypted_data: str) -> str:
        """Decrypt personally identifiable information"""
        decoded_data = base64.urlsafe_b64decode(encrypted_data.encode())
        decrypted_data = self.fernet.decrypt(decoded_data)
        return decrypted_data.decode()
```

6. ML Model Implementation

Sepsis Risk Model

```
# File: models/sepsis_model.py
import tensorflow as tf
import numpy as np
from typing import Dict, List, Tuple

class SepsisRiskModel:
    def __init__(self, model_path: str):
        self.model = tf.keras.models.load_model(model_path)
        self.feature_columns = self._load_feature_columns()
        self.scaler = self._load_scaler()

    def predict_risk(self, patient_data: Dict) -> Dict:
        """Predict sepsis risk for a patient"""
        # Extract and normalize features
        features = self._extract_features(patient_data)
        normalized_features = self.scaler.transform([features])
        # Model prediction
        prediction = self.model.predict(normalized_features)
        risk_score = float(prediction[0][0])

        # Calculate confidence
        confidence = self._calculate_confidence(normalized_features, prediction)

        # Generate explanations
        explanations = self._generate_explanations(features, prediction)

        return {
            'risk_score': risk_score,
            'confidence': confidence,
            'contributing_factors': explanations,
            'risk_level': self._categorize_risk(risk_score)
        }

    def _extract_features(self, patient_data: Dict) -> List[float]:
        """Extract features from patient data"""
        features = []

        # Vital signs features
        vitals = patient_data.get('vitals', {})
        features.extend([
            vitals.get('heart_rate', 0),
            vitals.get('systolic_bp', 0),
            vitals.get('diastolic_bp', 0),
            vitals.get('temperature', 0),
            vitals.get('respiratory_rate', 0),
            vitals.get('oxygen_saturation', 0)
        ])

        # Lab values features
        labs = patient_data.get('labs', {})
        features.extend([
            labs.get('white_blood_cell_count', 0),
            labs.get('lactate', 0),
            labs.get('procalcitonin', 0),
            labs.get('creatinine', 0)
        ])

        # Demographics
        demographics = patient_data.get('demographics', {})
        features.extend([
            demographics.get('age', 0),
            1 if demographics.get('gender') == 'M' else 0
        ])

        return features

    def _calculate_confidence(self, features: np.ndarray, prediction: np.ndarray) -> float:
        """Calculate prediction confidence"""
        # Use model uncertainty estimation
        predictions = []
        for _ in range(100):
            pred = self.model.predict(features, training=True)
            predictions.append(pred[0][0])

        std = np.std(predictions)
        confidence = max(0.0, min(1.0, 1.0 - (std * 2)))
        return float(confidence)

    def _categorize_risk(self, risk_score: float) -> str:
        """Categorize risk score into levels"""
        if risk_score >= 0.8:
            return "CRITICAL"
        elif risk_score >= 0.6:
            return "HIGH"
        elif risk_score >= 0.3:
            return "MODERATE"
        else:
            return "LOW"
```

Conclusion

This Low Level Design provides comprehensive implementation-ready specifications for the Healthcare Patient Risk Stratification Platform. The detailed database schemas, API implementations, configuration files, and security controls enable immediate development while ensuring enterprise-grade quality and compliance.

Building upon all previous documents (PRD, FRD, NFRD, AD, HLD), this LLD completes the technical foundation for a production-ready healthcare AI system.

Next Steps: Proceed to Pseudocode development to define executable algorithms and implementation logic.

This document is confidential and proprietary. Distribution is restricted to authorized personnel only. # Pseudocode Document ## Healthcare Patient Risk Stratification Platform

Document Control

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

ETVX Framework Application

Entry Criteria

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

Task (This Document)

Define executable algorithms for core system functions.

Exit Criteria

- âœ… **Implementation Ready** - Pseudocode can be directly translated to code

---

Core Algorithms

1. Patient Data Ingestion

```
FUNCTION ingest_patient_data(fhir_bundle, source_system):
    BEGIN
        // Validate FHIR bundle structure
        IF NOT validate_fhir_structure(fhir_bundle) THEN
            RETURN error("Invalid FHIR bundle")
        END IF

        // Extract patient resource
        patient = extract_patient_resource(fhir_bundle)

        // Check for existing patient
        existing = find_patient_by_mrn(patient.mrn)
        IF existing EXISTS THEN
            patient_id = existing.id
            update_patient_demographics(patient_id, patient.demographics)
        ELSE
            patient_id = create_new_patient(patient)
        END IF

        // Process clinical data entries
        clinical_entries = extract_clinical_data(fhir_bundle)
        FOR EACH entry IN clinical_entries DO
            quality_score = assess_data_quality(entry)
            store_clinical_data(patient_id, entry, quality_score)
            publish_event("data_updated", patient_id, entry.type)
        END FOR

        RETURN success(patient_id)
    END
END FUNCTION
```

2. Risk Assessment Engine

```
FUNCTION assess_patient_risk(patient_id, conditions, time_window):
    BEGIN
        // Fetch patient clinical data
        clinical_data = get_patient_data(patient_id, time_window)

        risk_results = []
        FOR EACH condition IN conditions DO
            // Load ML model for condition
            model = load_model(condition)

            // Extract and normalize features
            features = extract_features(clinical_data, condition)
            normalized = normalize_features(features, model.scaler)

            // Model prediction
            prediction = model.predict(normalized)
            confidence = calculate_confidence(model, normalized)

            // Generate explanations
            explanations = generate_shap_explanations(model, normalized)

            risk_result = {
                condition: condition,
                risk_score: prediction.risk_score,
                confidence: confidence,
                risk_level: categorize_risk(prediction.risk_score),
                explanations: explanations
            }
            risk_results.append(risk_result)
        END FOR

        // Calculate overall risk
        overall_risk = calculate_overall_risk(risk_results)

        // Store assessment
        assessment_id = store_risk_assessment(patient_id, risk_results)

        RETURN {
            patient_id: patient_id,
            risk_scores: risk_results,
            overall_risk: overall_risk,
            timestamp: current_time()
        }
    END
END FUNCTION
```

3. Real-time Monitoring

```
FUNCTION process_real_time_data(patient_event):
    BEGIN
        // Parse incoming event
        patient_id = patient_event.patient_id
        data_type = patient_event.data_type
        values = patient_event.values

        // Store time-series data
        store_time_series_data(patient_id, data_type, values, current_time())

        // Check alert thresholds
        alert_rules = get_alert_rules(patient_id, data_type)
        FOR EACH rule IN alert_rules DO
            IF evaluate_rule(rule, values) THEN
                alert = create_alert(patient_id, rule, values)
                send_alert(alert)
                log_alert_event(alert)
            END IF
        END FOR

        // Update risk scores if significant change
        IF is_significant_change(patient_id, data_type, values) THEN
            trigger_risk_reassessment(patient_id)
        END IF
    END
END FUNCTION
```

4. Clinical Decision Support

```
FUNCTION generate_recommendations(patient_id, risk_scores):
    BEGIN
        recommendations = []

        // Load patient context
        patient_context = get_patient_context(patient_id)

        FOR EACH risk_score IN risk_scores DO
            IF risk_score.risk_level = "HIGH" OR risk_score.risk_level = "CRITICAL" THEN
                // Query clinical knowledge graph
                guidelines = query_clinical_guidelines(risk_score.condition)

                // Apply clinical rules
                applicable_rules = filter_applicable_rules(guidelines, patient_context)

                FOR EACH rule IN applicable_rules DO
                    recommendation = {
                        condition: risk_score.condition,
                        action: rule.recommended_action,
                        priority: rule.priority,
                        evidence_level: rule.evidence_level,
                        rationale: rule.rationale
                    }
                    recommendations.append(recommendation)
                END FOR
            END IF
        END FOR

        // Rank recommendations by priority and evidence
        ranked_recommendations = rank_recommendations(recommendations)

        RETURN ranked_recommendations
    END
END FUNCTION
```

5. Authentication and Authorization

```
FUNCTION authenticate_user(jwt_token):
    BEGIN
        // Validate JWT token structure
        IF NOT is_valid_jwt_format(jwt_token) THEN
            RETURN error("Invalid token format")
        END IF

        // Decode and verify signature
        payload = decode_jwt(jwt_token, public_key)
        IF payload IS NULL THEN
            RETURN error("Invalid token signature")
        END IF

        // Check expiration
        IF payload.exp < current_timestamp() THEN
            RETURN error("Token expired")
        END IF

        // Validate user status
        user = get_user_by_id(payload.sub)
        IF user IS NULL OR NOT user.active THEN
            RETURN error("User not found or inactive")
        END IF

        // Log authentication event
        log_auth_event(user.id, "login_success", current_timestamp())

        RETURN success(user)
    END
END FUNCTION
```

6. Audit Logging

```
FUNCTION log_audit_event(user_id, action, resource_type, resource_id, metadata):
    BEGIN
        audit_log = {
            id: generate_uuid(),
            user_id: user_id,
            action: action,
            resource_type: resource_type,
            resource_id: resource_id,
            timestamp: current_timestamp(),
            client_ip: get_client_ip(),
            user_agent: get_user_agent(),
            metadata: metadata,
            success: true
        }

        // Store in audit database
        store_audit_log(audit_log)

        // Send to compliance monitoring
        send_to_compliance_monitor(audit_log)

        // Check for suspicious activity
        IF detect_suspicious_activity(user_id, action) THEN
            trigger_security_alert(user_id, action)
        END IF
    END
END FUNCTION
```

Conclusion

This pseudocode provides executable algorithms for the Healthcare Patient Risk Stratification Platform, completing the comprehensive documentation suite. All algorithms are designed for immediate implementation while maintaining enterprise-grade performance, security, and compliance standards.

Implementation Ready: Development teams can now begin coding based on these detailed specifications.

This document is confidential and proprietary. Distribution is restricted to authorized personnel only.