Healthcare Data Engineering Project

Technical Implementation Guide

Team Lead: Christian
Analysts: Kian Azizpour, Kiana
Project Duration: 1 Week
Date: June 20, 2025

Executive Summary

This document outlines the technical implementation strategy, task distribution, and deliverables for our healthcare data engineering project. The selected technology stack consisting of PostgreSQL, Python, and Docker addresses all project requirements while ensuring cross-platform compatibility across Linux and Windows development environments.

Strategic Technical Decisions

Database Architecture: PostgreSQL

PostgreSQL has been selected as the primary database solution based on Kian's comparative analysis. Key decision factors include superior handling of structured healthcare data, robust foreign key constraints essential for patient data integrity, mature JSON support for medications and diagnoses files, and extensive Python ecosystem integration.

Development Environment: Docker + Python

Docker containerization ensures consistent development environments across Linux (Christian) and Windows (Kiana, Kian) platforms. Python provides comprehensive data processing libraries and PostgreSQL connectivity through SQLAlchemy and psycopg2.

Security Implementation: Level 1 + Documentation

Implementation focuses on production-ready security foundations including environment variable configuration, database authentication, and user role separation. Advanced features (encryption, audit logging) are documented for compliance requirements.

Team Analysis Integration

Data Foundation (Kiana)

The systematic analysis of six data files identified critical implementation requirements including 861 missing value_text entries requiring null handling strategies, inconsistent date formats across files necessitating standardization pipelines, and zip code formatting issues requiring validation rules. These findings directly inform the data cleaning specifications and validation framework.

Technical Architecture (Kian)

The database comparison correctly identified PostgreSQL's advantages for relational healthcare data. The assessment of JSON-to-relational transformation requirements and query complexity considerations provides the foundation for schema optimization and query implementation tasks.

Security Framework (Christian)

The three-tier security approach balances educational objectives with compliance requirements. Level 1 implementation provides production-ready foundations while Level 2 documentation addresses GDPR and healthcare data protection standards.

Complete Project Structure

```
healthcare_data_project/
├── README.md                          # Project documentation (provided)
├── .env                               # Environment configuration (provided)
├── .gitignore                         # Git configuration (provided)
├── docker-compose.yml                 # PostgreSQL setup (provided)
├── Dockerfile                         # Python environment (provided)
├── requirements.txt                   # Dependencies (provided)
├── data/
│   ├── raw/                           # Original data files (team copies)
│   │   ├── patients.csv
│   │   ├── observations.csv
│   │   ├── procedures.csv
│   │   ├── diagnoses.json
│   │   ├── medications.json
│   │   └── ehr_journeys_database.sqlite
│   ├── processed/                     # Cleaned data (auto-generated)
│   └── quality_reports/               # Quality assessments (auto-generated)
├── scripts/
│   ├── main.py                        # Main execution (provided)
│   ├── data_cleaning.py               # Cleaning pipeline (provided)
│   ├── data_loader.py                 # Loading pipeline (provided)
│   ├── database_setup.py              # DB initialization (Kian creates)
│   ├── query_runner.py                # Query execution (Kian creates)
│   └── data_validation.py             # Validation rules (Kiana creates)
├── sql/
│   ├── schema.sql                     # Database schema (provided)
│   ├── queries.sql                    # Project queries (provided)
│   └── test_data.sql                  # Test data (team creates)
├── config/
│   ├── database.py                    # DB configuration (provided)
│   ├── logging_config.py              # Logging setup (provided)
│   └── settings.py                    # App settings (Christian creates)
├── tests/
│   ├── test_data_quality.py           # Quality tests (Kiana creates)
│   ├── test_queries.py                # Query tests (Kian creates)
│   └── test_security.py               # Security tests (Christian creates)
├── docs/
│   ├── security_policy.md             # GDPR compliance (Christian creates)
│   ├── system_design.md               # Architecture (team collaborates)
│   └── final_report.md                # Project report (auto-generated)
├── logs/                              # Application logs (auto-generated)
└── output/                            # Results and reports (auto-generated)
```

Implementation Timeline

Phase 1: Infrastructure Setup

- Environment setup and Docker configuration validation
- Database connectivity testing across platforms
- Security configuration implementation
- Team coordination and integration support

Phase 2: Data Processing

- Data validation framework implementation (Kiana)
- Quality assessment and testing (Kiana)
- Data cleaning pipeline refinement (Christian)
- Data loading pipeline execution (Christian)

Phase 3: Queries and Analytics

- Database schema deployment and optimization (Kian)
- Query development and performance testing (Kian)
- Cross-dataset analysis implementation (Kian)

Phase 4: Documentation and Integration

- Final testing and validation (Team)
- Documentation completion (Team)
- Presentation preparation (Team)

Detailed Technical Assignments

Christian (Team Lead) - Infrastructure & Core Development

Primary Responsibilities:

- Environment setup and distribution
- Security implementation
- Integration and coordination

Files to Create:

- `config/settings.py` - Application configuration management
- `tests/test_security.py` - Security validation tests
- `docs/security_policy.md` - GDPR compliance documentation

Infrastructure Tasks: Create complete project package with all provided files, test Docker setup on both Windows and Linux platforms, validate cross-platform compatibility, and provide setup instructions to team.

Security implementation includes developing application configuration management system, implementing environment-based configuration switching, creating comprehensive error handling for all edge cases, and setting up automated testing pipeline validation.

Integration and coordination involves monitoring team progress and resolving technical blockers, integrating individual components into main pipeline, coordinating testing and final validation, and ensuring all components work together seamlessly.

Kian (Database Specialist) - Schema & Query Implementation

Primary Responsibilities:

- Schema deployment and optimization
- Query implementation framework
- Performance optimization

Files to Create:

- `scripts/database_setup.py` - Database initialization and management
- `scripts/query_runner.py` - Query execution framework
- `tests/test_queries.py` - Query validation and performance tests

Database Schema Tasks: Execute schema.sql with error handling, create indexes for performance optimization, set up user roles and permissions, check all tables exist, validate foreign key constraints, and verify index creation.

Query implementation involves executing five required project queries, measuring performance metrics, generating result reports, timing each query execution, monitoring memory usage, and generating performance reports.

Performance Optimization: Analyze query execution plans using EXPLAIN ANALYZE, implement query caching strategies, create materialized views for complex aggregations, set up connection pooling optimization, use PostgreSQL-specific features like CTEs and window functions, and test queries with realistic data volumes.

Kiana (Data Quality Analyst) - Validation & Testing

Primary Responsibilities:

- Custom validation rules implementation
- Quality assessment implementation
- Statistical analysis implementation

Files to Create:

- `scripts/data_validation.py` - Custom validation rules and quality checks
- `tests/test_data_quality.py` - Comprehensive data quality test suite

Data Validation Framework: Implement age validation (0-150 years), gender standardization, address format validation, lab value range validation, medication dosage validation, and procedure code validation.

Generate completeness percentages, validity scores, and consistency checks. Validate glucose values with realistic ranges, calculate distribution metrics for numeric fields, identify outliers using IQR and z-score methods, implement data drift detection, and create automated quality scoring algorithms.

Healthcare-Specific Validations: Validate ICD-10 format compliance, check for common drug interactions, validate dosage ranges for age groups, flag potentially dangerous combinations, use pandas profiling for automated quality assessment, implement statistical validation using scipy.stats, create visual quality reports, and document all quality issues found with recommended remediation.

Setup Instructions

Step 1: Extract Project Package

- Extract Christian's zip file to your preferred working directory
- Navigate to the healthcare_data_project folder

Step 2: Environment Setup

```
# Windows (PowerShell):
docker-compose up -d
python -m pip install -r requirements.txt

# Linux (Bash):
docker-compose up -d
pip install -r requirements.txt

# Note: .env file is pre-configured with correct settings
```

Integration and Testing Strategy

Code Integration Protocol

1. Component Testing: Each team member tests their modules independently
2. Integration Testing: Christian coordinates component integration
3. End-to-End Testing: Full pipeline validation with all components
4. Performance Testing: Query execution and data processing benchmarks

Quality Assurance Framework Complete integration testing includes data cleaning validation, data loading verification, and query execution confirmation. Performance benchmarks target data processing completion within 15 minutes, individual query execution under 10 seconds, peak memory usage below 4GB, and database loading batch processing above 1000 records per second.

Remaining Implementation Tasks

Database Implementation (Kian)

- Five required queries implemented and optimized
- Performance benchmarks meet targets
- Cross-dataset analytics deliver meaningful insights

Data Quality (Kiana)

- Custom validation rules handle all identified issues
- Quality assessment framework provides comprehensive metrics
- Test suite validates data integrity end-to-end
- Documentation captures all quality findings and remediation

Team Integration (Christian)

- All components integrate seamlessly
- Complete pipeline executes without errors
- Performance targets achieved
- Professional documentation completed

Final Deliverables Checklist

Technical Implementation

- Complete ETL pipeline with error handling
- Normalized PostgreSQL database schema
- Five cross-dataset queries with performance metrics
- Data quality framework and validation rules
- Security implementation and compliance documentation

Documentation Package

- System architecture and technical decisions
- Data quality assessment and findings
- Performance analysis and benchmarking results
- Security policy and GDPR compliance procedures
- Complete setup and deployment instructions

Project Lead Authorization: Christian
Implementation Commences: Upon zip file distribution
Target Completion: 1 week from start date
Success Metric: Working system demonstrating all requirements with professional documentation