Securaa User Service - Low Level Design Document

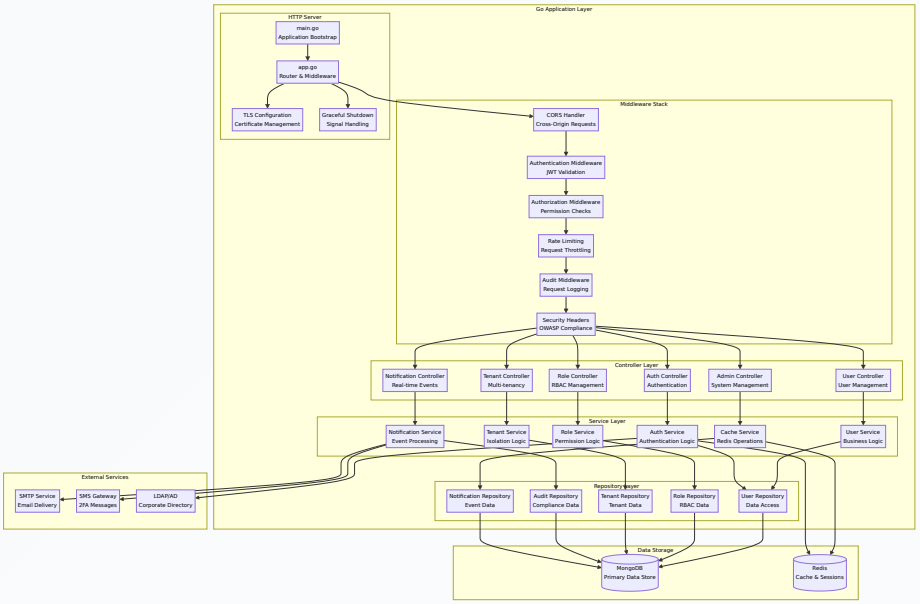
Document Information

- **Service Name**: Securaa User Service
- **Version**: 1.0
- **Date**: September 18, 2025
- **Author**: Development Team
- **Related Documents**: ZONA_USER_LLD.md

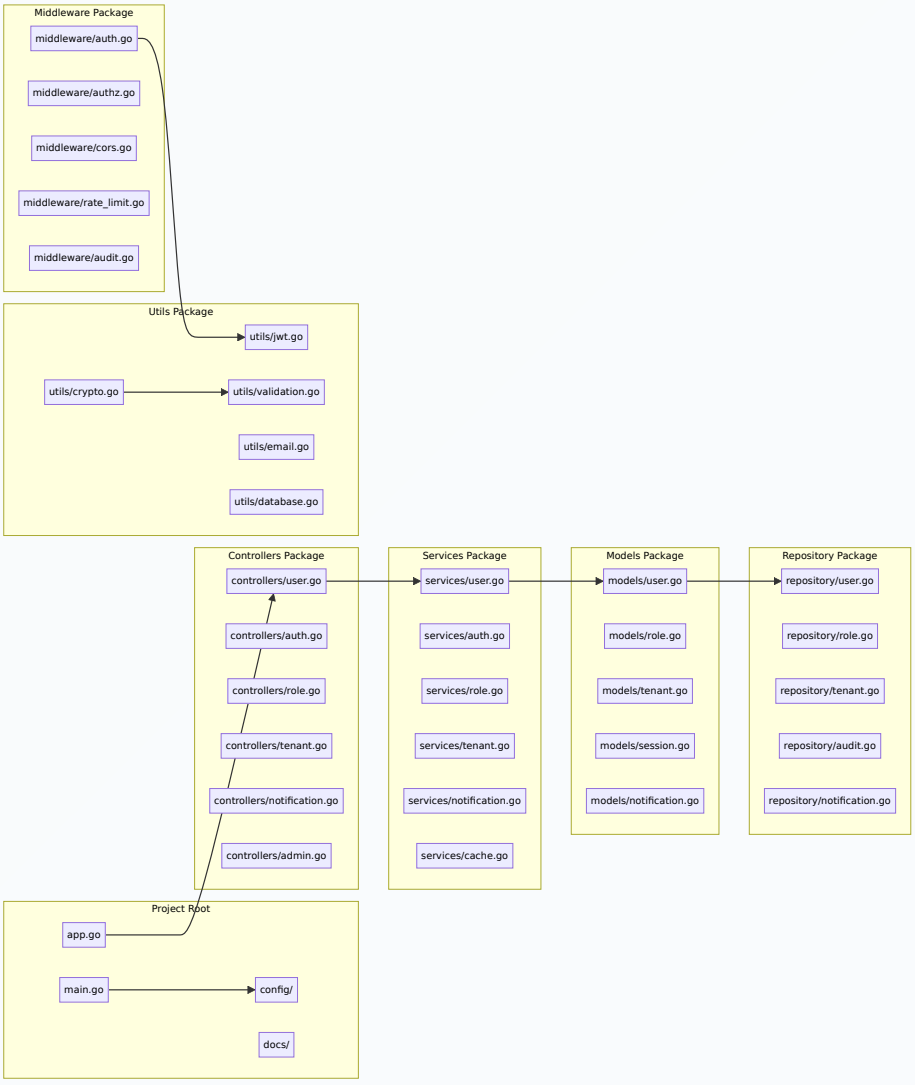
Table of Contents

🏗️ Technical Architecture Overview

Service Implementation Architecture



Code Structure & Package Organization



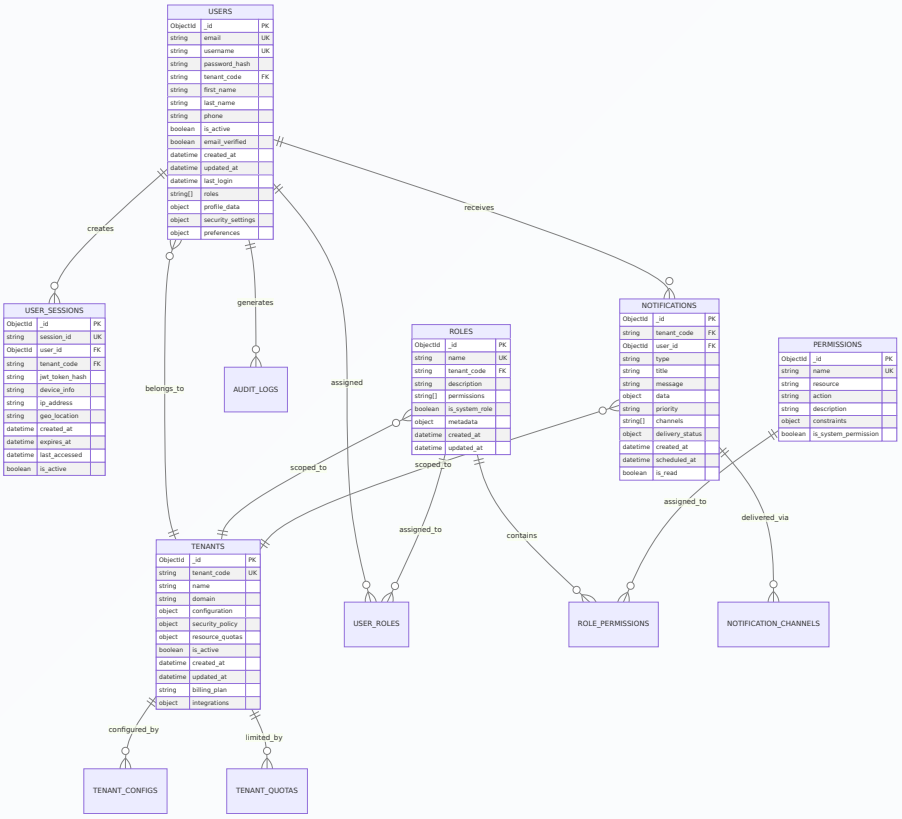
Package Responsibilities:

- **Controllers**: HTTP request handling, input validation, response formatting

- **Services**: Business logic implementation, data processing, external integrations
- **Models**: Data structures, validation rules, serialization/deserialization
- **Repository**: Database operations, query optimization, data access patterns
- **Middleware**: Cross-cutting concerns, security, logging, rate limiting
- **Utils**: Shared utilities, helper functions, common operations

🗄️ Database Design & Data Models

MongoDB Collection Schema Architecture



Collection Design Principles:

- **Multi-Tenancy**: All collections include tenant_code for data isolation
- **Indexing Strategy**: Compound indexes on tenant_code + other query fields
- **Document Embedding**: Nested objects for related data to reduce joins
- **Schema Validation**: MongoDB schema validation for data integrity

Redis Cache Schema Design

Cache Strategy:

- **TTL Management**: Different expiration times based on data sensitivity
- **Cache Invalidation**: Event-driven cache updates on data changes
- **Memory Optimization**: Compressed storage for large objects
- **High Availability**: Redis clustering for fault tolerance

API Design & Endpoint Specifications

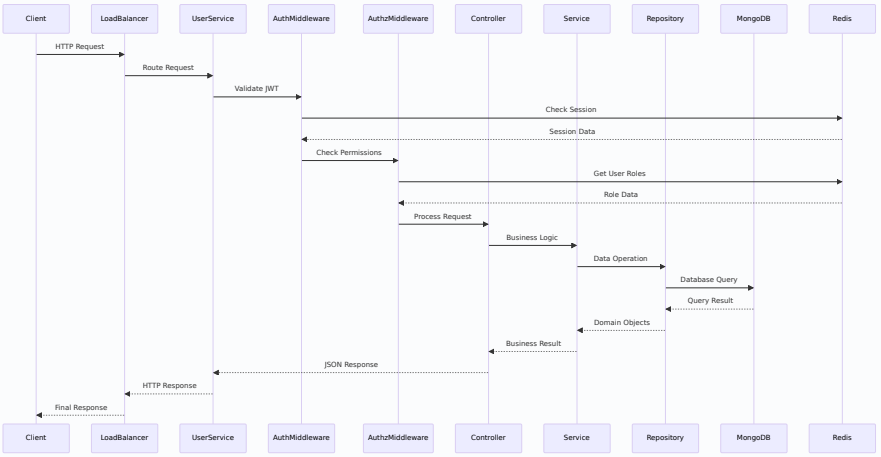
RESTful API Architecture

API Design Principles:

- **RESTful Design**: Standard HTTP methods and status codes
- **Consistent Response Format**: Unified JSON response structure

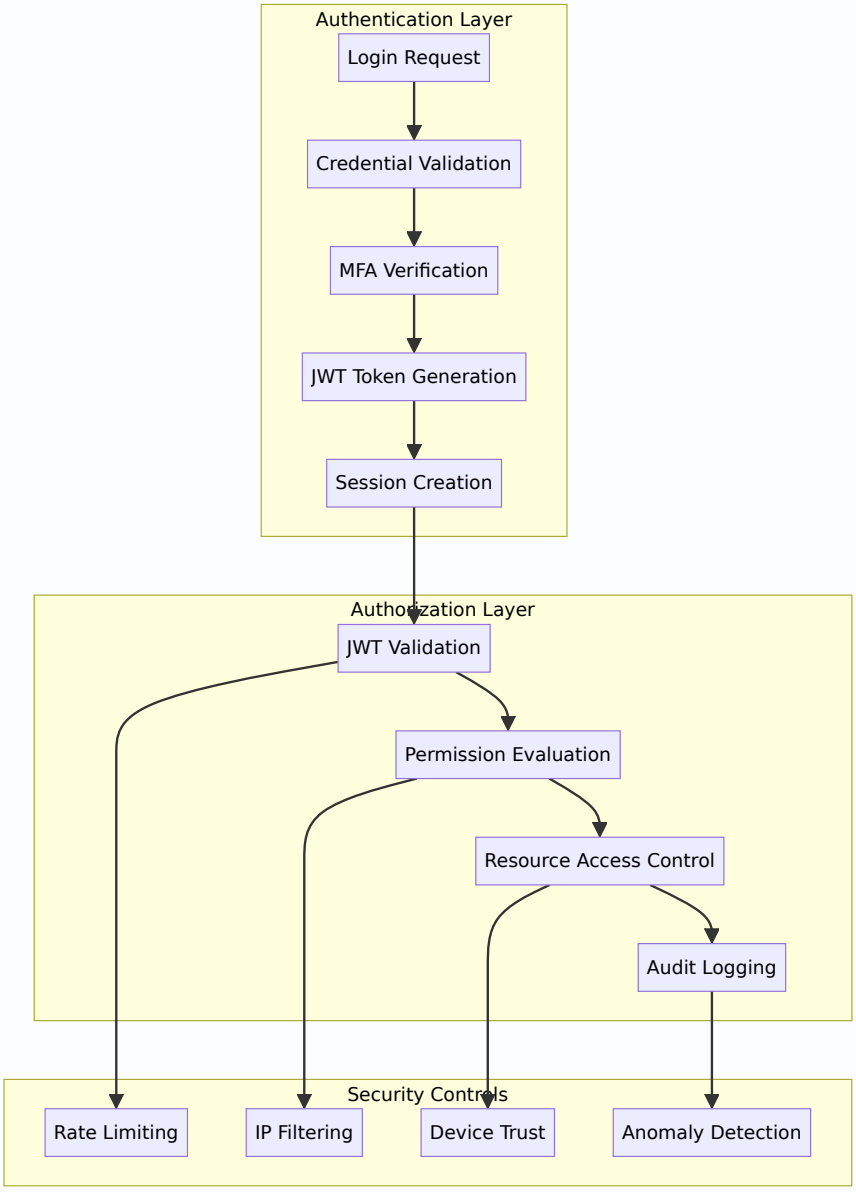
- **Pagination Support**: Cursor-based pagination for large datasets
- **Filtering & Sorting**: Query parameters for data manipulation
- **Versioning Strategy**: URL versioning for backward compatibility

API Request/Response Flow Diagram

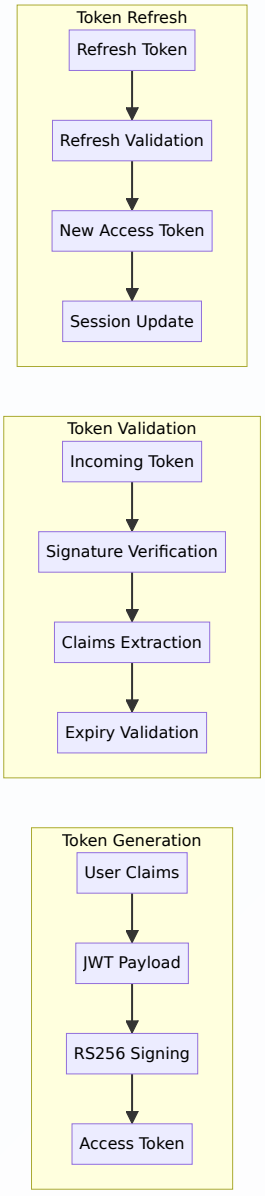


Security Implementation Details

Authentication & Authorization Flow


Authentication Layer
Login Request
Credential Validation
MFA Verification
JWT Token Generation
Session Creation
Authorization Layer
JWT Validation
Permission Evaluation
Resource Access Control
Audit Logging
Security Controls
Rate Limiting
IP Filtering
Device Trust
Anomaly Detection

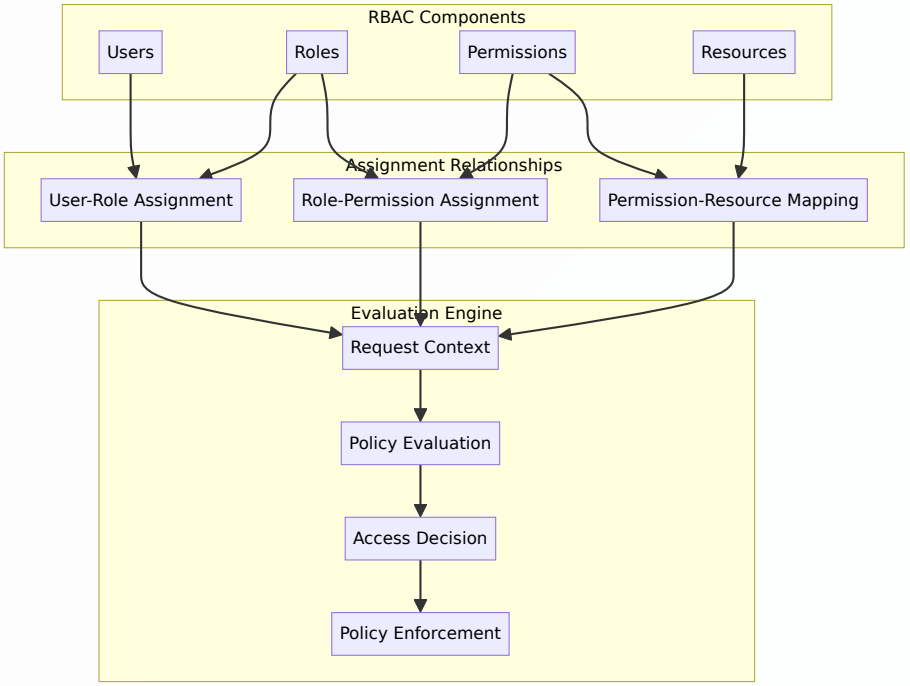
JWT Token Management Implementation



JWT Implementation Details:

- **Algorithm**: RS256 with 2048-bit RSA keys
- **Claims**: User ID, tenant, roles, permissions, device fingerprint
- **Expiration**: Configurable TTL (default 1 hour for access, 7 days for refresh)
- **Key Rotation**: Automated key rotation every 90 days

Role-Based Access Control (RBAC) Implementation



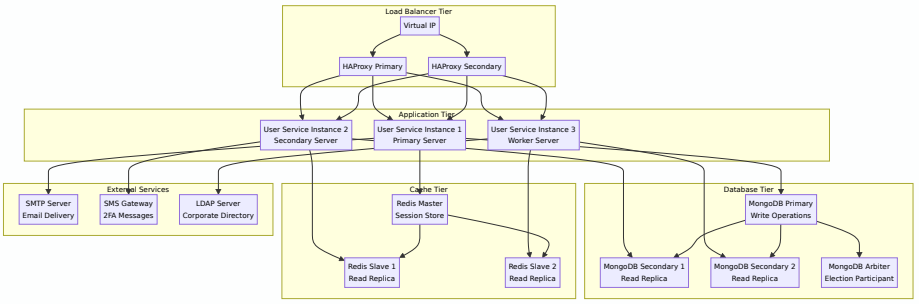
RBAC Features:

- **Hierarchical Roles**: Role inheritance and delegation
- **Fine-grained Permissions**: Resource and action-level controls

- **Dynamic Evaluation**: Runtime permission checking
- **Tenant Isolation**: Complete role separation per tenant

Deployment & Infrastructure Implementation

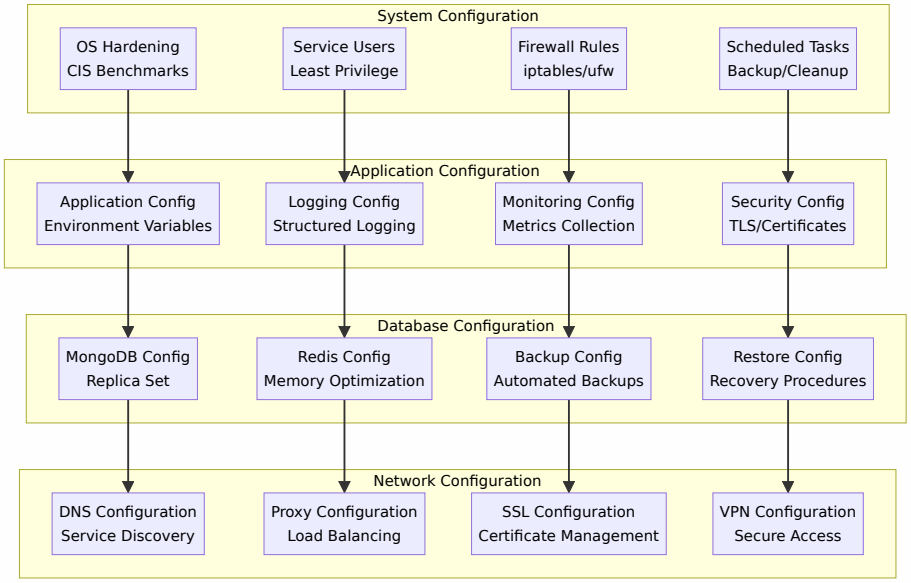
Traditional Server Deployment Architecture



Deployment Specifications:

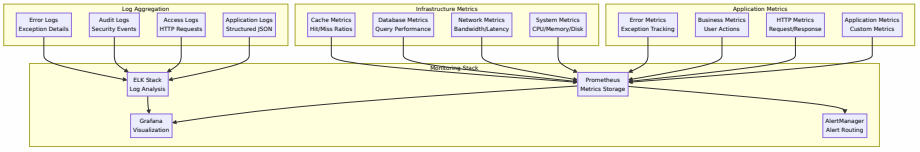
- **High Availability**: Redundant components with automatic failover
- **Load Distribution**: HAProxy with health checks and session affinity
- **Database Replication**: MongoDB replica set with automated elections
- **Cache Replication**: Redis master-slave setup with sentinel monitoring

Production Deployment Configuration



Monitoring & Observability Implementation

Comprehensive Monitoring Architecture

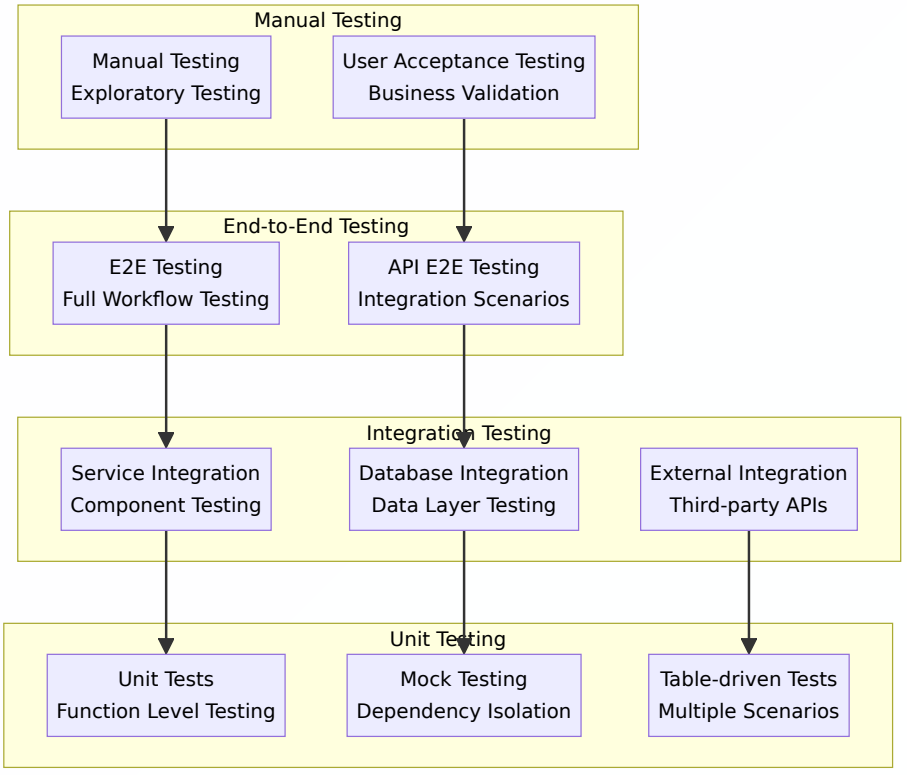


Monitoring Capabilities:

- **Real-Time Metrics**: Live dashboards with custom metrics
- **Alerting System**: Threshold-based and anomaly detection alerts
- **Log Correlation**: Centralized logging with request tracing
- **Performance Analytics**: Query optimization and bottleneck identification

Testing Strategy & Implementation

Comprehensive Testing Pyramid



Testing Implementation:

- **Test Coverage**: 80%+ code coverage with quality metrics
- **Automated Testing**: CI/CD pipeline integration with automated test execution
- **Performance Testing**: Load testing and stress testing scenarios
- **Security Testing**: Vulnerability scanning and penetration testing

Summary

The Securaa User Service Low Level Design provides comprehensive technical specifications for implementing a robust, scalable, and secure identity management platform. The detailed architecture, database schemas, API specifications, and deployment configurations ensure successful implementation and operation.

Key implementation highlights include Go-based microservice architecture, MongoDB with Redis caching, comprehensive security controls, and extensive monitoring capabilities. The modular design and clear separation of concerns facilitate maintenance, testing, and future enhancements.

This low-level design serves as a detailed blueprint for development teams to build, deploy, and maintain the Securaa User Service according to enterprise standards and best practices.