# High-Level Design (High Level Design) - Securaa Custom Services

## Document Information

- **Service Name**: Securaa Custom Services
- **Version**: 1.0
- **Date**: September 2025
- **Author**: Development Team
- **Related Documents**: Low Level Design

## Table of Contents

# 1. Executive Summary

## 1.1 System Purpose and Overview

Securaa Custom Services represents a sophisticated Go-based microservice architecture designed specifically for security automation platforms. This system serves as the backbone for integration and task management capabilities, enabling organizations to create, manage, and execute custom security applications and workflows. The service is architected to handle complex multi-tenant environments while maintaining high performance, security, and scalability standards.

The primary mission of Securaa Custom Services is to provide a unified platform where security teams can:

- Design and deploy custom security applications tailored to their specific needs
- Create and manage complex task workflows with interdependencies
- Import and export complete application configurations for portability and backup
- Process security events in real-time with automated response capabilities
- Maintain complete data isolation between different organizational tenants
- Scale seamlessly from small deployments to enterprise-level installations

## 1.2 Architecture Philosophy and Design Principles

The system is built on several key architectural principles that guide its design and implementation:

**Microservices Architecture**: Securaa Custom Services follows a microservices architectural pattern, operating as a standalone, independently deployable service. This approach provides several benefits including independent scaling, technology flexibility, and fault isolation. The service communicates with other components through well-defined REST APIs, ensuring loose coupling and high cohesion.

**Layered Architecture Pattern**: The internal structure follows a strict layered architecture with clear separation of concerns:

- **Presentation Layer** (Controllers): Handles HTTP requests, input validation, and response formatting
- **Business Logic Layer** (Services): Contains core business rules, workflows, and processing logic
- **Data Access Layer** (Models): Manages data structures, database interactions, and data validation
- **Infrastructure Layer**: Provides cross-cutting concerns like logging, security, and configuration management

**Multi-Tenant by Design**: The architecture is inherently designed for multi-tenancy, supporting both shared database models for smaller deployments and completely isolated database instances for enterprise MSSP (Managed Security Service Provider) environments. This flexibility allows the system to adapt to different organizational requirements and compliance needs.

**Security-First Approach**: Security is not an afterthought but a core design principle. Every component includes built-in security measures including field-level encryption, tenant isolation, comprehensive authentication and authorization mechanisms, and audit logging capabilities.

## 1.3 Technology Foundation

**Programming Language and Runtime**: Built with Go 1.17, leveraging Go's excellent concurrency model, memory efficiency, and strong standard library. Go's garbage collection and built-in HTTP server capabilities make it ideal for building high-performance web services.

**Web Framework**: Utilizes Gorilla Mux for HTTP routing, providing powerful URL routing capabilities, middleware support, and flexible request handling. Gorilla Mux offers better performance and flexibility compared to Go's standard HTTP router while maintaining simplicity.

**Database Technology**: MongoDB serves as the primary data store, chosen for its:

- Document-oriented structure that naturally fits complex security data models
- Built-in sharding capabilities for horizontal scaling
- Flexible schema design supporting evolving security requirements
- Excellent performance for read-heavy workloads common in security applications
- Native support for multi-tenant architectures

**Caching Strategy**: Redis provides high-performance caching capabilities for:

- Application configuration caching to reduce database load
- Session management and token storage
- Frequently accessed integration listings
- Real-time data that requires sub-second access times

## 1.4 Core Capabilities and Features

**Custom Application Management**: The system provides comprehensive capabilities for managing custom security applications. Users can create, configure, and deploy applications with complex parameter sets, file uploads (including logos), and integration-specific configurations. The system supports both generic applications that can be reused across tenants and highly customized applications tailored for specific use cases.

**Generic Task Creation and Execution**: One of the most powerful features is the ability to create and manage generic tasks that can be composed into complex workflows. These tasks support:

- Complex input parameter definitions with validation rules
- Dependency management between tasks
- Real-time execution monitoring and status tracking
- Output field mapping for downstream task consumption

- Error handling and retry mechanisms

**Data Export/Import Functionality**: The system includes sophisticated export/import capabilities that enable:

- Complete application configuration portability between environments
- Backup and disaster recovery scenarios
- Template-based application deployment across multiple tenants
- Version control and change management for application configurations

**Integration Management**: Comprehensive integration management features allow for:

- Connection management with external systems
- Credential storage with encryption and secure access
- API endpoint configuration and testing
- Health monitoring and status reporting
- Rate limiting and throttling capabilities

**Event Processing and Case Ingestion**: Real-time event processing capabilities include:

- High-volume security event ingestion from multiple sources
- Event correlation and analysis workflows
- Automated case creation and management
- Custom event processing rules and filters
- Integration with SIEM and other security platforms

**Multi-Tenant Support**: Enterprise-grade multi-tenancy includes:

- Complete data isolation between tenants
- Tenant-specific database instances (MSSP mode)
- Shared infrastructure with logical separation (single-tenant mode)
- Per-tenant configuration and customization
- Independent scaling and resource allocation

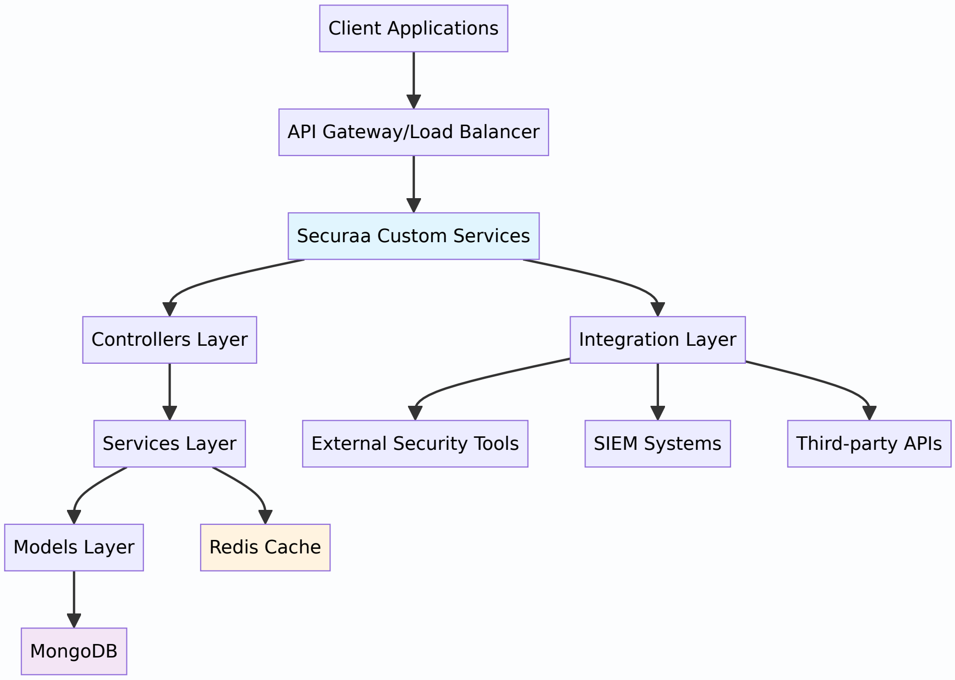
**License Management**: Sophisticated licensing capabilities support:

- Trial license limitations and usage tracking
- Enterprise license validation and feature enabling
- Usage monitoring and compliance reporting
- Automated license renewal and notification systems

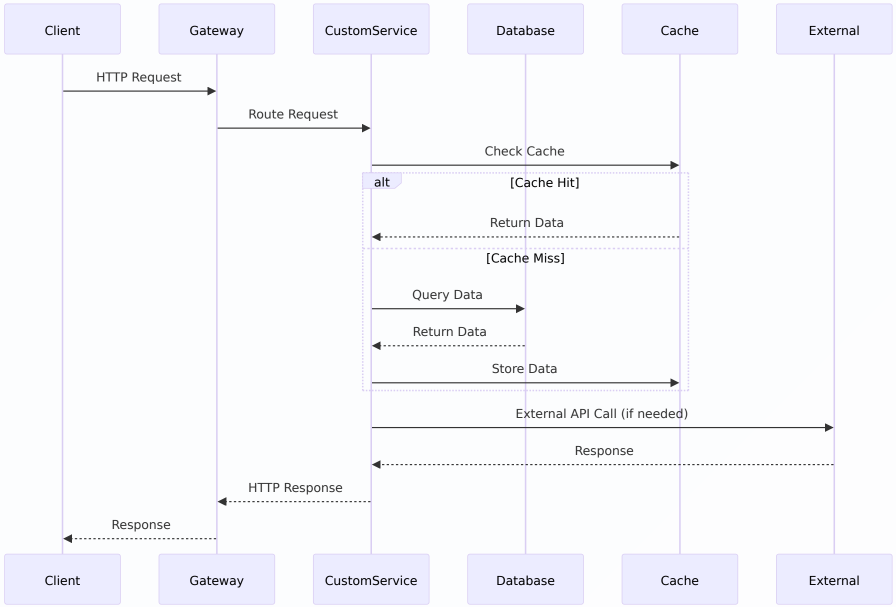
# 2. System Architecture

## 2.1 Overall System Architecture

The Securaa Custom Services architecture is designed as a modern, cloud-native microservice that operates within a larger security automation ecosystem. The system is structured in multiple layers, each with specific responsibilities and clear interfaces between them. This architectural approach ensures maintainability, scalability, and testability while providing the flexibility needed for complex security workflows.



## 2.2 Service Communication Architecture



# 3. Component Design

The system is composed of several key components, each responsible for specific aspects of the custom services functionality.

## 3.1 Controller Components

- **Custom App Controller**: Manages custom application lifecycle operations
- **Generic App Controller**: Handles generic task and application operations
- **Export Controller**: Manages data export/import functionality
- **Integration Controller**: Handles external system integrations
- **Events Controller**: Processes security events and case management

## 3.2 Service Components

- **Export Service**: Business logic for data export operations
- **Import Service**: Handles data import and validation
- **Integration Service**: Manages external system connections
- **Events Service**: Processes events and case creation

# 4. Data Architecture

The data architecture is designed to support multi-tenant operations with high performance and security.

## 4.1 Database Design Principles

- **Document-Oriented Storage**: Leverages MongoDB's flexible schema
- **Tenant Isolation**: Ensures complete data separation between tenants
- **Performance Optimization**: Strategic indexing and query optimization
- **Data Encryption**: Field-level encryption for sensitive data

# 5. API Design

The API follows RESTful principles with comprehensive error handling and security measures.

## 5.1 API Categories

- **Application Management APIs**: Create, update, delete custom applications
- **Task Management APIs**: Generic task creation and execution
- **Export/Import APIs**: Data portability operations
- **Integration APIs**: External system connectivity
- **Event Processing APIs**: Security event handling

# 6. Security Architecture

Security is implemented at multiple layers with comprehensive protection mechanisms.

## 6.1 Authentication and Authorization

- **JWT Token Authentication**: Secure token-based authentication
- **SAML Integration**: Enterprise single sign-on support
- **Role-Based Access Control**: Granular permission management
- **Multi-Factor Authentication**: Additional security layer

## 6.2 Data Protection

- **Encryption at Rest**: AES encryption for sensitive data
- **Encryption in Transit**: TLS/SSL for all communications
- **Field-Level Encryption**: Granular data protection
- **Secure Key Management**: Enterprise key management integration

# 7. Scalability & Performance

The architecture is designed for horizontal scaling and high performance.

## 7.1 Scaling Strategies

- **Horizontal Scaling**: Multiple service instances
- **Database Sharding**: MongoDB sharding for data distribution
- **Caching Strategy**: Redis for high-performance data access
- **Load Balancing**: Intelligent request distribution

# 8. Deployment Architecture

Cloud-native deployment with container orchestration support.

## 8.1 Containerization

- **Docker Containers**: Consistent deployment packaging
- **Docker Swarm Support**: Container orchestration
- **Health Checks**: Automated health monitoring
- **Rolling Updates**: Zero-downtime deployments

# 9. Integration Points

Comprehensive integration capabilities with external systems.

## 9.1 External Integrations

- **SIEM Platforms**: Security information and event management
- **Security Tools**: Vulnerability scanners, threat intelligence
- **Identity Providers**: LDAP, Active Directory, SAML

- **Notification Systems**: Email, Slack, webhooks

# 10. Monitoring & Observability

Comprehensive monitoring and observability features for operational excellence.

## 10.1 Monitoring Components

- **Application Metrics**: Performance and usage metrics
- **Health Checks**: Service and dependency monitoring
- **Audit Logging**: Comprehensive audit trail
- **Alerting**: Real-time issue notification