Auth Signer System - Development Architecture & Implementation Guide

Project Overview

Mission: Build a centralized, automated, and auditable system for managing authorized signers across the bank, starting with CME business line MVP by Q4 2025.

Current State Problem: 500+ monthly CME requests processed manually via email, leading to lost requests, weeks-long delays, and no tracking visibility.

Target State Solution: Self-service digital platform with automated workflows, real-time status tracking, and integrated compliance validation.

---

1. TECHNICAL STACK & INFRASTRUCTURE

Backend Technology Stack

```
Language: Java 17+
Framework: Spring Boot 3.x
Build Tool: Maven
Database: SQL Server 2019+
Message Queue: Apache Kafka (for bulk processing)
API Standard: RESTful APIs with OpenAPI 3.0 documentation
```

Frontend Technology Stack

```
Framework: React 18+ with TypeScript
UI Library: Material-UI (Corporate Connect standards)
State Management: React Context API + useReducer
Build Tool: Vite or Create React App
Integration: Embedded within Corporate Connect portal
```

Infrastructure Requirements

```
Cloud Platform: Bank's approved cloud environment
Authentication: Corporate Connect SSO integration
Database: SQL Server cluster with read replicas
Load Balancing: Application Gateway with SSL termination
```

```
Monitoring: Application Insights + Custom dashboards
Security: KMS for encryption, OAuth 2.0 for APIs
```

---

2. SYSTEM ARCHITECTURE COMPONENTS

Core Microservice Architecture

```
auth-signer-service/
├── src/main/java/com/usbank/authsigner/
│   ├── controller/          # REST API endpoints
│   ├── service/            # Business logic layer
│   ├── repository/          # Data access layer
│   ├── model/              # Domain entities
│   ├── dto/                # Data transfer objects
│   ├── config/             # Configuration classes
│   ├── security/           # Auth & authorization
│   ├── integration/        # External service clients
│   ├── validation/         # Business rule validators
│   └── exception/          # Error handling
├── src/main/resources/
│   ├── application.yml     # Configuration
│   ├── db/migration/       # Flyway SQL scripts
│   └── static/             # Static resources
└── src/test/               # Unit & integration tests
```

Frontend Application Structure

```
auth-signer-ui/
├── src/
│   ├── components/
│   │   ├── dashboard/      # ASL dashboard components
│   │   ├── forms/          # Add/edit signer forms
│   │   ├── guest/          # Guest user components
│   │   ├── bulk/           # Bulk upload interface
│   │   └── common/         # Shared UI components
│   ├── services/
│   │   ├── authSignerApi.ts    # API client
│   │   ├── guestTokenService.ts # Guest auth
│   │   └── bulkUploadService.ts # File processing
│   ├── hooks/              # Custom React hooks
│   ├── utils/              # Helper functions
│   └── types/          # TypeScript definitions
└── public/             # Static assets
```

---

3. DATABASE DESIGN & IMPLEMENTATION

Core Database Schema (Priority Order)

```
sql
```

```sql
-- 1. FOUNDATION TABLES (Build First)
CREATE TABLE accounts (
    account_id VARCHAR(50) PRIMARY KEY,
    business_line VARCHAR(20) NOT NULL,
    account_name VARCHAR(255) NOT NULL,
    client_id VARCHAR(50) NOT NULL,
    status VARCHAR(20) DEFAULT 'ACTIVE',
    created_date DATETIME2 DEFAULT GETDATE(),
    updated_date DATETIME2 DEFAULT GETDATE()
);

-- 2. CORE ENTITY TABLES
CREATE TABLE authorized_signers (
    signer_id UNIQUEIDENTIFIER PRIMARY KEY DEFAULT NEWID(),
    account_id VARCHAR(50) NOT NULL,
    first_name VARCHAR(100) NOT NULL,
    last_name VARCHAR(100) NOT NULL,
    email VARCHAR(255),
    phone VARCHAR(20),
    title VARCHAR(100),
    privilege_level VARCHAR(50) NOT NULL,
    authorization_limit DECIMAL(15,2),
    effective_date DATE NOT NULL,
    expiration_date DATE,
    status VARCHAR(20) DEFAULT 'ACTIVE',
    created_date DATETIME2 DEFAULT GETDATE(),
    updated_date DATETIME2 DEFAULT GETDATE(),
    FOREIGN KEY (account_id) REFERENCES accounts(account_id)
);

-- 3. WORKFLOW TABLES
CREATE TABLE asl_requests (
    request_id UNIQUEIDENTIFIER PRIMARY KEY DEFAULT NEWID(),
    account_id VARCHAR(50) NOT NULL,
    request_type VARCHAR(20) NOT NULL,
    initiated_by VARCHAR(100) NOT NULL,
    initiated_date DATETIME2 DEFAULT GETDATE(),
    status VARCHAR(20) DEFAULT 'DRAFT',
    approval_required BOOLEAN DEFAULT FALSE,
    completed_date DATETIME2,
    business_justification TEXT,
    FOREIGN KEY (account_id) REFERENCES accounts(account_id)
);
```

```
-- 4. AUDIT & COMPLIANCE TABLES
CREATE TABLE asl_audit_log (
    log_id UNIQUEIDENTIFIER PRIMARY KEY DEFAULT NEWID(),
    account_id VARCHAR(50) NOT NULL,
    signer_id UNIQUEIDENTIFIER,
    action VARCHAR(50) NOT NULL,
    changed_by VARCHAR(100) NOT NULL,
    change_timestamp DATETIME2 DEFAULT GETDATE(),
    before_state NVARCHAR(MAX),
    after_state NVARCHAR(MAX),
    source_system VARCHAR(50),
    ip_address VARCHAR(45),
    session_id VARCHAR(100)
);
```

Database Migration Strategy

```
Tool: Flyway for version-controlled migrations
Naming: V1__Create_foundation_tables.sql
Environment: Dev → Test → Staging → Production
Rollback: Each migration includes rollback scripts
Testing: Automated tests for each migration
```

---

4. API DESIGN & IMPLEMENTATION

Core API Endpoints (MVP Phase 1)

yaml

```
```

```
# Account & Signer Management
GET    /api/v1/accounts/{accountId}/signers    # Get current ASL
POST   /api/v1/accounts/{accountId}/signers    # Add new signer
PUT    /api/v1/accounts/{accountId}/signers/{signerId}  # Update signer
DELETE /api/v1/accounts/{accountId}/signers/{signerId}  # Remove signer

# Request Management
POST   /api/v1/requests                        # Create new request
GET    /api/v1/requests/{requestId}            # Get request status
PUT    /api/v1/requests/{requestId}/status     # Update request status
GET    /api/v1/requests/pending                # Get pending approvals

# Client Self-Service
POST   /api/v1/accounts/{accountId}/attest     # Annual attestation
GET    /api/v1/accounts/{accountId}/export     # Export ASL as PDF

# Guest User Management
POST   /api/v1/guest/provision                 # Create guest token
GET    /api/v1/guest/verify/{token}            # Validate guest token
```

API Security Implementation

```java
@RestController
@RequestMapping("/api/v1/accounts")
@PreAuthorize("hasRole('AUTHORIZED_SIGNER') or hasRole('INTERNAL_BANKER')")
public class AuthSignerController {

  @GetMapping("/{accountId}/signers")
  @PreAuthorize("@securityService.canAccessAccount(#accountId)")
  public ResponseEntity<List<SignerDto>> getSigners(@PathVariable String accountId) {
    // Implementation
  }
}
```

---

5. INTEGRATION ARCHITECTURE

External System Integrations (Build Order)

```
1. WebKYC Integration (Critical Path)
```

```
   - API: POST /webkyc/signers/validate
   - Purpose: Compliance validation
   - Retry: 3 attempts with exponential backoff
   - Fallback: Manual review queue

2. FileNet Integration (Core Functionality)
   - API: POST /filenet/documents
   - Purpose: Document storage
   - Async: Queue failed uploads for retry
   - Format: PDF generation with business-line verbiage

3. OCR Service Integration (Enhancement)
   - API: POST /ocr/process
   - Purpose: Document digitization
   - Confidence: Threshold-based validation
   - Fallback: Manual data entry

4. Corporate Connect SSO (Authentication)
   - Protocol: OAuth 2.0 / SAML
   - Token: JWT with role-based claims
   - Session: Timeout and refresh handling
   - Security: Rate limiting and monitoring
```

Integration Client Implementation

```java
@Service
public class WebKYCIntegrationService {

    @Retryable(value = {Exception.class}, maxAttempts = 3, backoff = @Backoff(delay = 1000))
    public WebKYCResponse validateSigner(SignerValidationRequest request) {
        // HTTP client implementation with circuit breaker
    }

    @Recover
    public WebKYCResponse recover(Exception ex, SignerValidationRequest request) {
        // Fallback logic - queue for manual review
    }
}
```

---

6. DEVELOPMENT PHASES & SPRINT PLANNING

Phase 1: Foundation (Sprints 1-3)

Sprint 1: Project Setup & Core Infrastructure

- [ ] Repository setup with CI/CD pipeline
- [ ] Database schema creation and migrations
- [ ] Spring Boot project structure
- [ ] Basic CRUD operations for accounts and signers
- [ ] Unit test framework setup

Sprint 2: Authentication & Authorization

- [ ] Corporate Connect SSO integration
- [ ] Role-based access control
- [ ] JWT token validation
- [ ] Basic security configurations
- [ ] API authentication middleware

Sprint 3: Core API Development

- [ ] Signer management endpoints
- [ ] Request workflow APIs
- [ ] Basic validation logic
- [ ] Error handling framework
- [ ] API documentation (OpenAPI)

Phase 2: Client Experience (Sprints 4-6)

Sprint 4: React Application Setup

- [ ] React project within Corporate Connect
- [ ] Component library integration
- [ ] API client service layer
- [ ] Basic dashboard layout
- [ ] Authentication integration

Sprint 5: Self-Service Functionality

- [ ] Add/remove signer forms
- [ ] ASL dashboard with real-time data
- [ ] Status tracking interface
- [ ] Form validation and error handling

- [ ] Responsive design implementation

Sprint 6: Advanced Features

- [ ] Annual attestation workflow
- [ ] Export/print functionality with business-line verbiage
- [ ] Bulk upload interface
- [ ] Guest user workflow (basic)
- [ ] Notification system

Phase 3: Integrations (Sprints 7-9)

Sprint 7: WebKYC Integration

- [ ] WebKYC API client implementation
- [ ] Compliance validation workflow
- [ ] Retry and error handling
- [ ] Manual review fallback
- [ ] Integration testing

Sprint 8: Document Management

- [ ] FileNet integration
- [ ] PDF generation with templates
- [ ] Document upload handling
- [ ] OCR service integration (basic)
- [ ] File validation and processing

Sprint 9: Advanced Workflows

- [ ] Approval workflow engine
- [ ] Task management system
- [ ] Bulk processing with Kafka
- [ ] Guest user token management
- [ ] Email notification service

---

7. TESTING STRATEGY

Testing Pyramid Implementation

```
Unit Tests (70%):
- Service layer business logic
- Validation rules and workflows
- Data access layer operations
- Utility functions and helpers
- Target: 90%+ code coverage

Integration Tests (20%):
- API endpoint testing
- Database integration
- External service mocking
- End-to-end workflow testing
- Authentication and authorization

System Tests (10%):
- Full user journey testing
- Performance and load testing
- Security penetration testing
- Browser compatibility testing
- Accessibility compliance testing
```

Test Data Management

```yaml
Test Environments:
  Development:
    - Local H2 database for unit tests
    - Docker containers for integration tests
    - Mock external services

  QA:
    - Dedicated SQL Server instance
    - Stubbed external integrations
    - Synthetic test data

  Staging:
    - Production-like environment
    - Real integration endpoints (test mode)
    - Anonymized production data subset
```

---

8. DEPLOYMENT & DEVOPS STRATEGY

CI/CD Pipeline Design

```yaml
Source Control: Git with feature branch workflow
Build: Maven for backend, npm/yarn for frontend
Quality Gates:
  - Unit test execution (minimum 80% coverage)
  - SonarQube code quality analysis
  - Security vulnerability scanning
  - Integration test execution

Deployment Stages:
  1. Development: Automatic on merge to develop
  2. QA: Manual trigger with smoke tests
  3. Staging: Automatic with full test suite
  4. Production: Manual approval with rollback capability
```

Infrastructure as Code

```
Container: Docker images for consistent deployment
Orchestration: Kubernetes or Docker Swarm
Configuration: External configuration management
Secrets: Azure Key Vault or equivalent
Monitoring: Application Insights + custom dashboards
Logging: Centralized logging with ELK stack
```

---

9. TEAM STRUCTURE & RESPONSIBILITIES

Development Team Roles

```
Tech Lead (1):
- Architecture decisions and code reviews
- Integration design and external API coordination
- Performance optimization and scalability planning

Backend Developers (2-3):
- Spring Boot service implementation
- Database design and optimization
- Integration service development
- API design and security implementation

Frontend Developers (2):
```

- React application development
- Corporate Connect integration
- User experience implementation
- Responsive design and accessibility

DevOps Engineer (1):
- CI/CD pipeline setup and maintenance
- Infrastructure provisioning and monitoring
- Security compliance and vulnerability management
- Deployment automation and rollback procedures

QA Engineers (2):
- Test strategy development and execution
- Automated testing framework setup
- Performance and security testing
- User acceptance testing coordination

Team Collaboration Tools

Project Management: Jira with Agile workflows
Communication: Microsoft Teams or Slack
Documentation: Confluence wiki
Code Review: GitHub/GitLab pull requests
Knowledge Sharing: Weekly architecture reviews

---

10. RISK MITIGATION & CONTINGENCY PLANNING

Technical Risks & Mitigation

Risk: WebKYC Integration Delays
Mitigation: Build mock service for parallel development
Fallback: Manual approval workflow as temporary solution

Risk: Corporate Connect Integration Complexity
Mitigation: Early POC with authentication team
Fallback: Standalone authentication with future migration

Risk: Performance Issues with Bulk Processing
Mitigation: Kafka implementation with proper sizing
Fallback: Synchronous processing with progress tracking

```
Risk: Data Migration Quality Issues
Mitigation: Multiple validation layers and business review
Fallback: Gradual migration with manual verification
```

Business Continuity Planning

```
System Downtime: Manual process documentation maintained
Data Corruption: Point-in-time recovery with audit trail validation
Security Breach: Incident response plan with stakeholder communication
Performance Degradation: Auto-scaling and load balancing implementation
```

---

11. SUCCESS METRICS & MONITORING

Key Performance Indicators

```
Technical Metrics:
- API response time < 500ms for 95% of requests
- System uptime > 99.5%
- Error rate < 0.1%
- Database query performance < 100ms average

Business Metrics:
- Processing time reduction: weeks → hours
- Request error rate reduction: 50%+ improvement
- Client satisfaction score improvement
- Banker productivity increase (requests per hour)

Adoption Metrics:
- Self-service usage rate > 70%
- Guest user completion rate > 80%
- Mobile/responsive usage tracking
- Feature utilization analysis
```

Monitoring Implementation

```
java
```

```
@Component
public class BusinessMetricsCollector {

  @EventListener
  public void trackRequestProcessing(RequestCompletedEvent event) {
    // Track processing time, success rate, user satisfaction
  }

  @Scheduled(fixedRate = 300000) // 5 minutes
  public void collectSystemHealth() {
    // API response times, database performance, integration health
  }
}
```

---

12. IMMEDIATE NEXT STEPS

Week 1-2: Project Initiation

- [ ] Development environment setup
- [ ] Repository creation and access permissions
- [ ] Database environment provisioning
- [ ] CI/CD pipeline basic setup
- [ ] Team onboarding and architecture review

Week 3-4: Foundation Development

- [ ] Database schema implementation
- [ ] Basic Spring Boot service structure
- [ ] Authentication integration POC
- [ ] React application skeleton
- [ ] Initial API endpoint development

Critical Dependencies to Resolve

- [ ] WebKYC API documentation and test environment access
- [ ] Corporate Connect integration specifications
- [ ] Guest user provisioning service requirements
- [ ] OCR service partnership details (Cognizant)
- [ ] Business line approval workflow definitions

This architecture provides a solid foundation for building the Auth Signer system while maintaining flexibility for future enhancements and business line expansion.