**SimRMS - Risk Management System**

**Comprehensive Developer Documentation**

**Document Information**

**Document Title: SimRMS - Risk Management System - Comprehensive Developer Documentation  
Document Code: SIMRMS-DEV-DOC-001  
Version: 1.0  
Classification: Internal Use Only  
Department: Information Technology  
Prepared By: Md Raihan Sharif   
Approved By:   
Date of Approval:**

**Creation/Modification History Record Card**

| **Version** | **Date** | **Author** | **Description of Changes** | **Review Status** | **Approver** |
| --- | --- | --- | --- | --- | --- |
| **0.1** | **2025-08-14** | **Md. Raihan Sharif** | **Initial draft creation with basic structure** | **Draft** | **-** |
| **0.2** | **2025-08-14** | **Md. Raihan Sharif** | **Added architecture documentation and API reference** | **Draft** | **-** |
| **0.3** | **2025-08-14** | **Md. Raihan Sharif** | **Completed development guidelines and troubleshooting** | **Draft** | **-** |
| **1.0** | **2025-08-14** | **SimRMS Development Team** | **Final version with complete documentation** | **Ready for Review** | **-** |

**Document Control Information**

| **Field** | **Value** |
| --- | --- |
| **Next Review Date** | **2025-11-14** |
| **Review Frequency** | **Quarterly** |
| **Document Owner** | **Lead Developer** |
| **Distribution** | **Development Team, Technical Leads, DevOps Team** |
| **Storage Location** | **Project Repository /docs folder** |
| **Related Documents** | **API Specification, Database Schema** |

**Document Approval**

| **Role** | **Name** | **Signature** | **Date** |
| --- | --- | --- | --- |
| **Document Author** | **SimRMS Development Team** |  | **2025-08-14** |
| **Technical Lead** |  |  |  |
| **Project Manager** |  |  |  |
| **Quality Assurance** |  |  |  |

**Table of Contents**

1. [***Executive Summary***](file:///C:\Users\raihan.sharif\Downloads\trypandoc.docx#executive-summary)
2. [***Overview***](file:///C:\Users\raihan.sharif\Downloads\trypandoc.docx#overview)
3. [***Architecture***](file:///C:\Users\raihan.sharif\Downloads\trypandoc.docx#architecture)
4. [***Project Structure***](file:///C:\Users\raihan.sharif\Downloads\trypandoc.docx#project-structure)
5. [***Technology Stack***](file:///C:\Users\raihan.sharif\Downloads\trypandoc.docx#technology-stack)
6. [***Setup and Installation***](file:///C:\Users\raihan.sharif\Downloads\trypandoc.docx#setup-and-installation)
7. [***Configuration***](file:///C:\Users\raihan.sharif\Downloads\trypandoc.docx#configuration)
8. [***Development Guidelines***](file:///C:\Users\raihan.sharif\Downloads\trypandoc.docx#development-guidelines)
9. [***API Documentation***](file:///C:\Users\raihan.sharif\Downloads\trypandoc.docx#api-documentation)
10. [***Database Integration***](file:///C:\Users\raihan.sharif\Downloads\trypandoc.docx#database-integration)
11. [***Authentication & Security***](file:///C:\Users\raihan.sharif\Downloads\trypandoc.docx#authentication--security)
12. [***Error Handling***](file:///C:\Users\raihan.sharif\Downloads\trypandoc.docx#error-handling)
13. [***Performance Considerations***](file:///C:\Users\raihan.sharif\Downloads\trypandoc.docx#performance-considerations)
14. [***Testing***](file:///C:\Users\raihan.sharif\Downloads\trypandoc.docx#testing)
15. [***Deployment***](file:///C:\Users\raihan.sharif\Downloads\trypandoc.docx#deployment)
16. [***Troubleshooting***](file:///C:\Users\raihan.sharif\Downloads\trypandoc.docx#troubleshooting)
17. [***Appendices***](file:///C:\Users\raihan.sharif\Downloads\trypandoc.docx#appendices)

**Executive Summary**

**This document serves as the comprehensive developer documentation for the SimRMS (Simulation Risk Management System) application. SimRMS is an enterprise-grade risk management platform built using .NET 8 and Clean Architecture principles, designed to provide robust risk assessment, monitoring, and management capabilities for financial institutions.**

**Document Purpose**

**This documentation provides developers, technical leads, and DevOps teams with: - Complete understanding of the system architecture and design patterns - Detailed implementation guidelines and coding standards - Comprehensive API reference and usage examples - Setup, configuration, and deployment procedures - Troubleshooting guides and best practices**

**Target Audience**

* **Software Developers: Implementation guidelines and coding patterns**
* **Technical Leads: Architecture overview and design decisions**
* **DevOps Engineers: Deployment and monitoring configurations**
* **Quality Assurance: Testing strategies and validation procedures**
* **Project Managers: Technical overview and capability understanding**

**Key Benefits of This Documentation**

1. **Standardization: Ensures consistent development practices across the team**
2. **Onboarding: Accelerates new developer integration into the project**
3. **Maintenance: Facilitates ongoing system maintenance and updates**
4. **Quality Assurance: Provides clear testing and validation guidelines**
5. **Operational Excellence: Enables effective deployment and monitoring**

**Overview**

**SimRMS (Simulation Risk Management System) is a comprehensive, enterprise-grade risk management platform built with .NET 8 and designed using Clean Architecture principles. The system provides robust risk assessment, monitoring, and management capabilities for financial institutions.**

**Key Features**

* **Clean Architecture: Follows Clean Architecture principles for maintainability and testability**
* **Custom Authentication: Token-based authentication with handshake mechanism**
* **Performance Optimized: Built with performance considerations from ground up**
* **Scalable Design: Horizontally scalable architecture with proper separation of concerns**
* **Comprehensive Logging: Structured logging with Serilog**
* **API Versioning: Full API versioning support with deprecation handling**
* **File Management: Multi-server file upload and management system**
* **Health Monitoring: Built-in health checks for all dependencies**

**Business Domain**

**SimRMS handles: - Company master data management - Broker branch operations - User information and access control - Company exposure tracking - File management and storage - Risk assessment and monitoring**

**Architecture**

**Clean Architecture Overview**

**SimRMS follows the Clean Architecture pattern with the following layers:**

***┌─────────────────────────────────────────────────────────────┐*  
*│ SimRMS.WebAPI │*  
*│ (Controllers, Middleware, Security) │*  
*└─────────────────────────────────────────────────────────────┘*  
 *│*  
*┌─────────────────────────────────────────────────────────────┐*  
*│ SimRMS.Infrastructure │*  
*│ (Services, Repositories, External) │*  
*└─────────────────────────────────────────────────────────────┘*  
 *│*  
*┌─────────────────────────────────────────────────────────────┐*  
*│ SimRMS.Application │*  
*│ (Use Cases, DTOs, Interfaces, Validators) │*  
*└─────────────────────────────────────────────────────────────┘*  
 *│*  
*┌─────────────────────────────────────────────────────────────┐*  
*│ SimRMS.Domain │*  
*│ (Entities, Business Rules, Exceptions) │*  
*└─────────────────────────────────────────────────────────────┘*  
  
*┌─────────────────────────────────────────────────────────────┐*  
*│ SimRMS.Shared │*  
*│ (Common Models, Constants, Extensions) │*  
*└─────────────────────────────────────────────────────────────┘***

**Core Principles**

1. **Dependency Inversion: Dependencies flow inward toward the domain**
2. **Single Responsibility: Each layer has a clear, single responsibility**
3. **Separation of Concerns: Business logic is separate from infrastructure**
4. **Testability: Easy to unit test with proper abstraction**
5. **Maintainability: Changes in one layer don’t affect others**

**Project Structure**

***SimRMS/*  
*├── src/*  
*│ ├── SimRMS.Domain/ # Core business entities and logic*  
*│ │ ├── Entities/ # Domain entities*  
*│ │ │ ├── BaseEntity.cs # Base entity with audit fields*  
*│ │ │ ├── MstCo.cs # Company master entity*  
*│ │ │ ├── MstCoBrch.cs # Company branch entity*  
*│ │ │ ├── UsrInfo.cs # User information entity*  
*│ │ │ └── ...*  
*│ │ ├── Exceptions/ # Domain-specific exceptions*  
*│ │ │ ├── DomainException.cs # Base domain exception*  
*│ │ │ ├── NotFoundException.cs # Entity not found exception*  
*│ │ │ ├── ValidationException.cs # Business validation exception*  
*│ │ │ └── ...*  
*│ │ ├── Interfaces/ # Core interfaces*  
*│ │ │ └── Common/*  
*│ │ │ ├── IGenericRepository.cs # Generic repository interface*  
*│ │ │ └── IUnitOfWork.cs # Unit of work pattern*  
*│ │ └── Common/ # Common domain types*  
*│ │*  
*│ ├── SimRMS.Application/ # Application business logic*  
*│ │ ├── Interfaces/ # Application interfaces*  
*│ │ │ ├── Services/ # Service interfaces*  
*│ │ │ │ ├── IBrokerBranchService.cs*  
*│ │ │ │ ├── IMstCoService.cs*  
*│ │ │ │ ├── IUsrInfoService.cs*  
*│ │ │ │ └── ...*  
*│ │ │ ├── ICacheService.cs # Caching interface*  
*│ │ │ ├── IFileService.cs # File management interface*  
*│ │ │ └── ...*  
*│ │ ├── Models/ # Application models*  
*│ │ │ ├── DTOs/ # Data Transfer Objects*  
*│ │ │ │ ├── MstCoBrchDtos.cs # Branch DTOs*  
*│ │ │ │ ├── MstCoDtos.cs # Company DTOs*  
*│ │ │ │ ├── UsrInfoDto.cs # User DTOs*  
*│ │ │ │ └── ...*  
*│ │ │ ├── Requests/ # Request models*  
*│ │ │ │ ├── MstCoBrchRequests.cs # Branch request models*  
*│ │ │ │ ├── MstCoRequests.cs # Company request models*  
*│ │ │ │ └── ...*  
*│ │ │ └── Auth/ # Authentication models*  
*│ │ ├── Validators/ # FluentValidation validators*  
*│ │ │ ├── MstCoBrchRequestValidators.cs # Consolidated validators*  
*│ │ │ ├── UsrInfoRequestValidator.cs*  
*│ │ │ └── ...*  
*│ │ └── DependencyInjection.cs # DI configuration*  
*│ │*  
*│ ├── SimRMS.Infrastructure/ # External concerns implementation*  
*│ │ ├── Services/ # Service implementations*  
*│ │ │ ├── BrokerBranchService.cs # Branch service implementation*  
*│ │ │ ├── MstCoService.cs # Company service implementation*  
*│ │ │ ├── FileService.cs # File management service*  
*│ │ │ ├── CacheService.cs # Caching implementation*  
*│ │ │ └── ...*  
*│ │ ├── Common/ # Infrastructure common*  
*│ │ │ ├── GenericRepository.cs # Generic repository implementation*  
*│ │ │ └── UnitOfWork.cs # Unit of work implementation*  
*│ │ ├── HealthChecks/ # Health check implementations*  
*│ │ ├── BackgroundServices/ # Background services*  
*│ │ └── DependencyInjection.cs # Infrastructure DI*  
*│ │*  
*│ ├── SimRMS.Shared/ # Shared components*  
*│ │ ├── Constants/ # Application constants*  
*│ │ │ ├── AppConstants.cs # General constants*  
*│ │ │ └── ActionTypeEnum.cs # Action type enumeration*  
*│ │ ├── Models/ # Shared models*  
*│ │ │ ├── ApiResponse.cs # Standard API response*  
*│ │ │ ├── PagedResult.cs # Pagination model*  
*│ │ │ ├── BulkOperationResult.cs # Bulk operation result*  
*│ │ │ └── ...*  
*│ │ └── Extensions/ # Utility extensions*  
*│ │ └── ExpressionExtensions.cs # Expression tree extensions*  
*│ │*  
*│ └── SimRMS.WebAPI/ # Web API layer*  
*│ ├── Controllers/ # API controllers*  
*│ │ ├── BaseController.cs # Base controller with common functionality*  
*│ │ └── V1/ # API version 1 controllers*  
*│ │ ├── BrokerBranchController.cs*  
*│ │ ├── MstCoController.cs*  
*│ │ ├── AuthController.cs*  
*│ │ ├── FileController.cs*  
*│ │ └── ...*  
*│ ├── Middleware/ # Custom middleware*  
*│ │ ├── ExceptionHandlingMiddleware.cs*  
*│ │ ├── PerformanceMiddleware.cs*  
*│ │ ├── RequestLoggingMiddleware.cs*  
*│ │ └── SecurityHeadersMiddleware.cs*  
*│ ├── Security/ # Security implementations*  
*│ │ ├── TokenAuthenticationMiddleware.cs*  
*│ │ ├── TokenAuthenticationSchemeHandler.cs*  
*│ │ └── CustomAuthorizationHandler.cs*  
*│ ├── Services/ # Web-specific services*  
*│ │ └── CurrentUserService.cs*  
*│ ├── Extensions/ # Service collection extensions*  
*│ ├── Program.cs # Application entry point*  
*│ └── appsettings.json # Configuration file*  
*│*  
*├── lib/ # External libraries*  
*│ ├── LB.DAL.Core.Common.dll # Database access library*  
*│ └── LB.DAL.Core.Common.pdb*  
*│*  
*├── docs/ # Documentation*  
*└── README.md # Project overview***

**Technology Stack**

**Core Technologies**

* **.NET 8: Latest LTS version of .NET**
* **ASP.NET Core Web API: For building REST APIs**
* **C# 12: Latest C# language features**
* **SQL Server: Primary database**

**Key Libraries and Frameworks**

| **Category** | **Library** | **Purpose** |
| --- | --- | --- |
| **Validation** | **FluentValidation** | **Request validation and business rules** |
| **Logging** | **Serilog** | **Structured logging** |
| **Documentation** | **Swagger/OpenAPI** | **API documentation** |
| **Caching** | **IMemoryCache** | **In-memory caching** |
| **Authentication** | **Custom Token Service** | **Token-based authentication** |
| **Database** | **LB.DAL.Core.Common** | **Custom data access layer** |
| **API Versioning** | **ASP.NET Core API Versioning** | **API version management** |
| **Health Checks** | **ASP.NET Core Health Checks** | **Application health monitoring** |

**Development Tools**

* **Visual Studio 2022: Primary IDE**
* **SQL Server Management Studio: Database management**
* **Postman: API testing**
* **Git: Version control**

**Setup and Installation**

**Prerequisites**

* **.NET 8 SDK (8.0 or later)**
* **SQL Server (2019 or later)**
* **Visual Studio 2022 (recommended) or VS Code**
* **Git for version control**

**Installation Steps**

1. **Clone the Repository**

* ***git clone [repository-url]*  
  *cd SimRMS***

1. **Restore Dependencies**

* ***dotnet restore***

1. **Configure Database Connection**
   * **Update *appsettings.json* with your SQL Server connection string**
   * **Ensure the database *DB\_EFBTX\_LBSL* exists and is accessible**
2. **Build the Solution**

* ***dotnet build***

1. **Run the Application**

* ***dotnet run --project src/SimRMS.WebAPI***

1. **Verify Installation**
   * **Navigate to *https://localhost:7000/swagger* to access API documentation**
   * **Check health endpoint: *https://localhost:7000/health***

**Development Environment Setup**

1. **Configure appsettings.Development.json**

* ***{*  
   *"ConnectionStrings": {*  
   *"DefaultConnection": "Server=localhost;Database=DB\_EFBTX\_LBSL;Integrated Security=true;TrustServerCertificate=true;"*  
   *},*  
   *"Serilog": {*  
   *"MinimumLevel": {*  
   *"Default": "Debug"*  
   *}*  
   *}*  
  *}***

1. **Set Environment Variables (Optional)**

* ***set ASPNETCORE\_ENVIRONMENT=Development*  
  *set ASPNETCORE\_URLS=https://localhost:7000;http://localhost:5000***

**Configuration**

**appsettings.json Structure**

**The application uses a comprehensive configuration system:**

***Database Configuration***

***{*  
 *"ConnectionStrings": {*  
 *"DefaultConnection": "Server=server;Database=db;User Id=user;Password=pass;TrustServerCertificate=true;"*  
 *},*  
 *"LB\_DAL": {*  
 *"CommandTimeout": 60,*  
 *"IsolationLevel": "ReadCommitted",*  
 *"EnableConnectionPooling": true,*  
 *"MaxPoolSize": 100,*  
 *"MinPoolSize": 5,*  
 *"ConnectionTimeout": 30*  
 *}*  
*}***

***Security Configuration***

***{*  
 *"Security": {*  
 *"AllowedRoutes": ["/health", "/api/V1/handshake"],*  
 *"AllowedRoutePrefixes": ["/api/V1/auth/login", "/swagger"],*  
 *"EnableStrictPathMatching": true,*  
 *"CaseSensitiveMatching": false*  
 *}*  
*}***

***API Versioning***

***{*  
 *"ApiVersioning": {*  
 *"Enabled": true,*  
 *"DefaultVersion": "1.0",*  
 *"StrictVersionValidation": false,*  
 *"SupportedVersions": [*  
 *{*  
 *"Version": "1.0",*  
 *"Status": "Current",*  
 *"Description": "Initial release with core functionality"*  
 *}*  
 *]*  
 *}*  
*}***

***File Upload Configuration***

***{*  
 *"FileUpload": {*  
 *"Default": {*  
 *"UploadPath": "D:\\Uploads\\SimRMS",*  
 *"MaxFileSize": 10485760,*  
 *"AllowedExtensions": [".jpg", ".pdf", ".doc", ".xlsx"]*  
 *},*  
 *"Servers": {*  
 *"dbserver": {*  
 *"UploadPath": "\\\\server\\path",*  
 *"RequiresImpersonation": true,*  
 *"Username": "user",*  
 *"Password": "password"*  
 *}*  
 *}*  
 *}*  
*}***

**Environment-Specific Configuration**

* **Development: *appsettings.Development.json***
* **Staging: *appsettings.Staging.json***
* **Production: *appsettings.Production.json***

**Development Guidelines**

**Core Development Rules**

1. **Database Access: ONLY use stored procedures - never inline SQL**
2. **Repository Pattern: Use the generic *IGenericRepository* for all data operations**
3. **Service Pattern: One service interface/implementation per business area**
4. **Validation: Consolidated validators with extension methods for reusability**
5. **Clean Architecture: Maintain strict layer separation**
6. **Shared Components: Use *SimRMS.Shared* for cross-cutting concerns**

**Coding Standards**

***Naming Conventions***

| **Component** | **Convention** | **Example** |
| --- | --- | --- |
| **Entities** | **Database table names** | ***MstCoBrch*** |
| **DTOs** | **Descriptive names + Dto** | ***BrokerBranchDto*** |
| **Services** | **Business names + Service** | ***BrokerBranchService*** |
| **Controllers** | **Business names + Controller** | ***BrokerBranchController*** |
| **Interfaces** | **I + Name** | ***IBrokerBranchService*** |

***Entity Pattern***

***public class MstCoBrch : BaseEntity*  
*{*  
 *[Key]*  
 *[MaxLength(5)]*  
 *public string CoCode { get; set; } = string.Empty;*  
  
 *[Key]*  
 *[MaxLength(6)]*  
 *public string CoBrchCode { get; set; } = string.Empty;*  
  
 *[MaxLength(100)]*  
 *public string? CoBrchDesc { get; set; }*  
  
 *public bool? CoBrchSts { get; set; }*  
*}***

***DTO Pattern***

***public class BrokerBranchDto*  
*{*  
 *public string CoCode { get; set; } = string.Empty;*  
 *public string CoBrchCode { get; set; } = string.Empty;*  
 *public string? CoBrchDesc { get; set; }*  
 *public bool? CoBrchSts { get; set; }*  
 *public DateTime? CreatedAt { get; set; }*  
*}***

***Service Interface Pattern***

***public interface IBrokerBranchService*  
*{*  
 *Task<PagedResult<BrokerBranchDto>> GetMstCoBrchListAsync(*  
 *int pageNumber, int pageSize, string? searchTerm, string? coCode,*   
 *CancellationToken cancellationToken);*  
  
 *Task<BrokerBranchDto?> GetMstCoBrchByIdAsync(*  
 *string coCode, string coBrchCode, CancellationToken cancellationToken);*  
  
 *Task<BrokerBranchDto> CreateMstCoBrchAsync(*  
 *CreateMstCoBrchRequest request, CancellationToken cancellationToken);*  
  
 *Task<BrokerBranchDto> UpdateMstCoBrchAsync(*  
 *string coCode, string coBrchCode, UpdateMstCoBrchRequest request,*   
 *CancellationToken cancellationToken);*  
  
 *Task<bool> DeleteMstCoBrchAsync(*  
 *string coCode, string coBrchCode, DeleteMstCoBrchRequest request,*   
 *CancellationToken cancellationToken);*  
*}***

***Controller Pattern***

***[Route("api/v{version:apiVersion}/[controller]")]*  
*[ApiController]*  
*[ApiVersion("1.0")]*  
*[Authorize]*  
*public class BrokerBranchController : BaseController*  
*{*  
 *private readonly IBrokerBranchService \_brokerBranchService;*  
 *private readonly ILogger<BrokerBranchController> \_logger;*  
  
 *public BrokerBranchController(*  
 *IBrokerBranchService brokerBranchService,*  
 *IConfigurationService configurationService,*  
 *ILogger<BrokerBranchController> logger)*  
 *: base(configurationService)*  
 *{*  
 *\_brokerBranchService = brokerBranchService;*  
 *\_logger = logger;*  
 *}*  
  
 *[HttpGet]*  
 *[ProducesResponseType(typeof(ApiResponse<IEnumerable<BrokerBranchDto>>), 200)]*  
 *public async Task<ActionResult<ApiResponse<IEnumerable<BrokerBranchDto>>>> GetList(*  
 *[FromQuery, Range(1, int.MaxValue)] int pageNumber = 1,*  
 *[FromQuery, Range(1, 100)] int pageSize = 10,*  
 *CancellationToken cancellationToken = default)*  
 *{*  
 *// Implementation*  
 *}*  
*}***

***Validation Pattern***

***// Single file with multiple validators and extension methods*  
*public static class MstCoBrchValidationRules*  
*{*  
 *public static IRuleBuilderOptions<T, string> ValidCoCode<T>(this IRuleBuilder<T, string> ruleBuilder)*  
 *{*  
 *return ruleBuilder*  
 *.NotEmpty().WithMessage("Company code is required")*  
 *.MaximumLength(5).WithMessage("Company code cannot exceed 5 characters")*  
 *.Matches("^[A-Z0-9]+$").WithMessage("Company code must contain only uppercase letters and numbers");*  
 *}*  
  
 *public static IRuleBuilderOptions<T, string> ValidCoBrchCode<T>(this IRuleBuilder<T, string> ruleBuilder)*  
 *{*  
 *return ruleBuilder*  
 *.NotEmpty().WithMessage("Branch code is required")*  
 *.MaximumLength(6).WithMessage("Branch code cannot exceed 6 characters");*  
 *}*  
*}*  
  
*public class CreateMstCoBrchRequestValidator : AbstractValidator<CreateMstCoBrchRequest>*  
*{*  
 *public CreateMstCoBrchRequestValidator()*  
 *{*  
 *RuleFor(x => x.CoCode).ValidCoCode();*  
 *RuleFor(x => x.CoBrchCode).ValidCoBrchCode();*  
 *RuleFor(x => x.CoBrchDesc)*  
 *.MaximumLength(100).WithMessage("Description cannot exceed 100 characters");*  
 *}*  
*}*  
  
*public class UpdateMstCoBrchRequestValidator : AbstractValidator<UpdateMstCoBrchRequest>*  
*{*  
 *public UpdateMstCoBrchRequestValidator()*  
 *{*  
 *RuleFor(x => x.CoCode).ValidCoCode();*  
 *RuleFor(x => x.CoBrchCode).ValidCoBrchCode();*  
 *RuleFor(x => x.CoBrchDesc)*  
 *.MaximumLength(100).WithMessage("Description cannot exceed 100 characters");*  
 *}*  
*}***

**Adding New Business Section Workflow**

1. **Create Entity in *Domain/Entities/* (use database table name)**
2. **Create DTOs in *Application/Models/DTOs/* (descriptive names)**
3. **Create Request Models in *Application/Models/Requests/***
4. **Create Consolidated Validator in *Application/Validators/* with extension methods**
5. **Create Service Interface in *Application/Interfaces/Services/***
6. **Implement Service in *Infrastructure/Services/***
7. **Create Controller in *WebAPI/Controllers/V1/* (meaningful name)**
8. **Register Services in DI containers**
9. **Add Constants to *Shared/Constants/* if needed**

**API Documentation**

**Base API Information**

* **Base URL: *https://localhost:7000/api/v1***
* **Authentication: Bearer Token**
* **Content-Type: *application/json***
* **API Version: Specified in URL path (*v1*, *v2*, etc.)**

**Standard Response Format**

**All API responses follow a consistent format:**

***{*  
 *"success": true,*  
 *"data": { /\* response data \*/ },*  
 *"message": "Operation completed successfully",*  
 *"errors": null,*  
 *"timestamp": "2025-08-14T10:30:00Z",*  
 *"pagination": {*  
 *"pageNumber": 1,*  
 *"pageSize": 10,*  
 *"totalRecords": 100,*  
 *"totalPages": 10*  
 *}*  
*}***

**Error Response Format**

***{*  
 *"success": false,*  
 *"data": null,*  
 *"message": "Operation failed",*  
 *"errors": [*  
 *{*  
 *"field": "CoCode",*  
 *"message": "Company code is required"*  
 *}*  
 *],*  
 *"timestamp": "2025-08-14T10:30:00Z"*  
*}***

**Core API Endpoints**

***Authentication Endpoints***

| **Method** | **Endpoint** | **Description** |
| --- | --- | --- |
| **POST** | ***/api/v1/handshake*** | **Initiate authentication handshake** |
| **POST** | ***/api/v1/handshake/validate*** | **Validate handshake token** |
| **POST** | ***/api/v1/auth/login*** | **User login** |
| **POST** | ***/api/v1/auth/logout*** | **User logout** |
| **POST** | ***/api/v1/auth/refresh*** | **Refresh access token** |

***Company Management***

| **Method** | **Endpoint** | **Description** |
| --- | --- | --- |
| **GET** | ***/api/v1/companies*** | **Get paginated company list** |
| **GET** | ***/api/v1/companies/{id}*** | **Get company by ID** |
| **POST** | ***/api/v1/companies*** | **Create new company** |
| **PUT** | ***/api/v1/companies/{id}*** | **Update company** |
| **DELETE** | ***/api/v1/companies/{id}*** | **Delete company (soft delete)** |
| **HEAD** | ***/api/v1/companies/{id}*** | **Check if company exists** |

***Broker Branch Management***

| **Method** | **Endpoint** | **Description** |
| --- | --- | --- |
| **GET** | ***/api/v1/brokerBranch*** | **Get paginated branch list** |
| **GET** | ***/api/v1/brokerBranch/{coCode}/{coBrchCode}*** | **Get branch by composite key** |
| **POST** | ***/api/v1/brokerBranch*** | **Create new branch** |
| **PUT** | ***/api/v1/brokerBranch/{coCode}/{coBrchCode}*** | **Update branch** |
| **DELETE** | ***/api/v1/brokerBranch/{coCode}/{coBrchCode}*** | **Delete branch** |
| **HEAD** | ***/api/v1/brokerBranch/{coCode}/{coBrchCode}*** | **Check if branch exists** |

***User Management***

| **Method** | **Endpoint** | **Description** |
| --- | --- | --- |
| **GET** | ***/api/v1/users*** | **Get paginated user list** |
| **GET** | ***/api/v1/users/{id}*** | **Get user by ID** |
| **POST** | ***/api/v1/users*** | **Create new user** |
| **PUT** | ***/api/v1/users/{id}*** | **Update user** |
| **DELETE** | ***/api/v1/users/{id}*** | **Delete user** |
| **GET** | ***/api/v1/users/statistics*** | **Get user statistics** |

***File Management***

| **Method** | **Endpoint** | **Description** |
| --- | --- | --- |
| **POST** | ***/api/v1/files/upload*** | **Upload file** |
| **GET** | ***/api/v1/files/{id}*** | **Download file** |
| **DELETE** | ***/api/v1/files/{id}*** | **Delete file** |
| **GET** | ***/api/v1/files/{id}/info*** | **Get file information** |

***Health and Monitoring***

| **Method** | **Endpoint** | **Description** |
| --- | --- | --- |
| **GET** | ***/health*** | **Overall health status** |
| **GET** | ***/health/ready*** | **Readiness probe** |
| **GET** | ***/health/live*** | **Liveness probe** |

**Request/Response Examples**

***Create Broker Branch***

**Request:**

***POST /api/v1/brokerBranch*  
*Content-Type: application/json*  
*Authorization: Bearer {token}*  
  
*{*  
 *"coCode": "LBSL",*  
 *"coBrchCode": "HEAD01",*  
 *"coBrchDesc": "Head Office Branch",*  
 *"coBrchSts": true*  
*}***

**Response:**

***HTTP/1.1 201 Created*  
*Content-Type: application/json*  
  
*{*  
 *"success": true,*  
 *"data": {*  
 *"coCode": "LBSL",*  
 *"coBrchCode": "HEAD01",*   
 *"coBrchDesc": "Head Office Branch",*  
 *"coBrchSts": true,*  
 *"createdAt": "2025-08-14T10:30:00Z",*  
 *"createdBy": "admin"*  
 *},*  
 *"message": "Market Stock Company Branch created successfully",*  
 *"timestamp": "2025-08-14T10:30:00Z"*  
*}***

***Get Paginated Branch List***

**Request:**

***GET /api/v1/brokerBranch?pageNumber=1&pageSize=10&searchTerm=HEAD&coCode=LBSL*  
*Authorization: Bearer {token}***

**Response:**

***HTTP/1.1 200 OK*  
*Content-Type: application/json*  
  
*{*  
 *"success": true,*  
 *"data": [*  
 *{*  
 *"coCode": "LBSL",*  
 *"coBrchCode": "HEAD01",*  
 *"coBrchDesc": "Head Office Branch",*  
 *"coBrchSts": true,*  
 *"createdAt": "2025-08-14T10:30:00Z"*  
 *}*  
 *],*  
 *"message": "Market Stock Company Branches retrieved successfully",*  
 *"pagination": {*  
 *"pageNumber": 1,*  
 *"pageSize": 10,*  
 *"totalRecords": 1,*  
 *"totalPages": 1*  
 *},*  
 *"timestamp": "2025-08-14T10:30:00Z"*  
*}***

**Database Integration**

**Stored Procedure Conventions**

**SimRMS exclusively uses stored procedures for all database operations. No inline SQL is permitted.**

***Naming Conventions***

| **Operation Type** | **Pattern** | **Example** |
| --- | --- | --- |
| **CRUD Operations** | ***LB\_SP\_Crud{EntityName}*** | ***LB\_SP\_CrudMstCoBrch*** |
| **List Operations** | ***LB\_SP\_Get{EntityName}List*** | ***LB\_SP\_GetMstCoBrchList*** |
| **Single Record** | ***LB\_SP\_Get{EntityName}\_By{Key}*** | ***LB\_SP\_GetMstCoBrch\_ById*** |

***CRUD Stored Procedure Pattern***

***CREATE PROCEDURE [dbo].[LB\_SP\_CrudMstCoBrch]*  
 *@Action INT, -- 1=Create, 2=Update, 3=Delete*  
 *@CoCode NVARCHAR(5),*  
 *@CoBrchCode NVARCHAR(6),*  
 *@CoBrchDesc NVARCHAR(100) = NULL,*  
 *@CoBrchSts BIT = NULL,*  
 *@CreatedBy NVARCHAR(50) = NULL,*  
 *@UpdatedBy NVARCHAR(50) = NULL,*  
  
 *-- OUTPUT Parameters*  
 *@RowsAffected INT OUTPUT,*  
 *@StatusCode INT OUTPUT,*  
 *@StatusMsg NVARCHAR(500) OUTPUT*  
*AS*  
*BEGIN*  
 *SET NOCOUNT ON;*  
  
 *BEGIN TRY*  
 *IF @Action = 1 -- Create*  
 *BEGIN*  
 *INSERT INTO MstCoBrch (CoCode, CoBrchCode, CoBrchDesc, CoBrchSts, CreatedBy, CreatedAt)*  
 *VALUES (@CoCode, @CoBrchCode, @CoBrchDesc, @CoBrchSts, @CreatedBy, GETDATE())*  
  
 *SET @RowsAffected = @@ROWCOUNT*  
 *SET @StatusCode = 200*  
 *SET @StatusMsg = 'Branch created successfully'*  
 *END*  
 *ELSE IF @Action = 2 -- Update*  
 *BEGIN*  
 *UPDATE MstCoBrch*   
 *SET CoBrchDesc = @CoBrchDesc,*  
 *CoBrchSts = @CoBrchSts,*  
 *UpdatedBy = @UpdatedBy,*  
 *UpdatedAt = GETDATE()*  
 *WHERE CoCode = @CoCode AND CoBrchCode = @CoBrchCode*  
  
 *SET @RowsAffected = @@ROWCOUNT*  
 *SET @StatusCode = CASE WHEN @@ROWCOUNT > 0 THEN 200 ELSE 404 END*  
 *SET @StatusMsg = CASE WHEN @@ROWCOUNT > 0 THEN 'Branch updated successfully' ELSE 'Branch not found' END*  
 *END*  
 *ELSE IF @Action = 3 -- Delete*  
 *BEGIN*  
 *UPDATE MstCoBrch*   
 *SET IsDeleted = 1,*  
 *DeletedBy = @UpdatedBy,*  
 *DeletedAt = GETDATE()*  
 *WHERE CoCode = @CoCode AND CoBrchCode = @CoBrchCode*  
  
 *SET @RowsAffected = @@ROWCOUNT*  
 *SET @StatusCode = CASE WHEN @@ROWCOUNT > 0 THEN 200 ELSE 404 END*  
 *SET @StatusMsg = CASE WHEN @@ROWCOUNT > 0 THEN 'Branch deleted successfully' ELSE 'Branch not found' END*  
 *END*  
 *END TRY*  
 *BEGIN CATCH*  
 *SET @RowsAffected = 0*  
 *SET @StatusCode = 500*  
 *SET @StatusMsg = ERROR\_MESSAGE()*  
 *END CATCH*  
*END***

***List Stored Procedure Pattern***

***CREATE PROCEDURE [dbo].[LB\_SP\_GetMstCoBrchList]*  
 *@PageNumber INT = 1,*  
 *@PageSize INT = 10,*  
 *@SearchTerm NVARCHAR(100) = NULL,*  
 *@CoCode NVARCHAR(5) = NULL,*  
  
 *-- OUTPUT Parameters*  
 *@TotalCount INT OUTPUT,*  
 *@StatusCode INT OUTPUT,*  
 *@StatusMsg NVARCHAR(500) OUTPUT*  
*AS*  
*BEGIN*  
 *SET NOCOUNT ON;*  
  
 *DECLARE @Offset INT = (@PageNumber - 1) \* @PageSize*  
  
 *BEGIN TRY*  
 *-- Get total count*  
 *SELECT @TotalCount = COUNT(1)*  
 *FROM MstCoBrch*  
 *WHERE IsDeleted = 0*  
 *AND (@CoCode IS NULL OR CoCode = @CoCode)*  
 *AND (@SearchTerm IS NULL OR*   
 *CoBrchCode LIKE '%' + @SearchTerm + '%' OR*   
 *CoBrchDesc LIKE '%' + @SearchTerm + '%')*  
  
 *-- Get paginated results*  
 *SELECT CoCode, CoBrchCode, CoBrchDesc, CoBrchSts,*   
 *CreatedBy, CreatedAt, UpdatedBy, UpdatedAt*  
 *FROM MstCoBrch*  
 *WHERE IsDeleted = 0*  
 *AND (@CoCode IS NULL OR CoCode = @CoCode)*  
 *AND (@SearchTerm IS NULL OR*   
 *CoBrchCode LIKE '%' + @SearchTerm + '%' OR*   
 *CoBrchDesc LIKE '%' + @SearchTerm + '%')*  
 *ORDER BY CoCode, CoBrchCode*  
 *OFFSET @Offset ROWS*  
 *FETCH NEXT @PageSize ROWS ONLY*  
  
 *SET @StatusCode = 200*  
 *SET @StatusMsg = 'Data retrieved successfully'*  
 *END TRY*  
 *BEGIN CATCH*  
 *SET @StatusCode = 500*  
 *SET @StatusMsg = ERROR\_MESSAGE()*  
 *END CATCH*  
*END***

**Generic Repository Usage**

**The *IGenericRepository* provides a unified interface for all database operations:**

***public interface IGenericRepository*  
*{*  
 *Task<T> ExecuteScalarAsync<T>(string sql, object? parameters = null, bool isStoredProcedure = false);*  
 *Task<IEnumerable<T>> QueryAsync<T>(string sql, object? parameters = null, bool isStoredProcedure = false);*  
 *Task<T?> QuerySingleOrDefaultAsync<T>(string sql, object? parameters = null, bool isStoredProcedure = false);*  
 *Task<int> ExecuteAsync(string sql, object? parameters = null, bool isStoredProcedure = false);*  
 *Task<(IEnumerable<T> Data, IDictionary<string, object> OutputValues)> ExecuteWithOutputAsync<T>(*  
 *string sql, object? parameters = null, bool isStoredProcedure = false);*  
*}***

***Service Implementation Example***

***public class BrokerBranchService : IBrokerBranchService*  
*{*  
 *private readonly IGenericRepository \_repository;*  
  
 *public async Task<BrokerBranchDto> CreateMstCoBrchAsync(*  
 *CreateMstCoBrchRequest request, CancellationToken cancellationToken)*  
 *{*  
 *var parameters = new*  
 *{*  
 *Action = 1, // Create*  
 *CoCode = request.CoCode,*  
 *CoBrchCode = request.CoBrchCode,*  
 *CoBrchDesc = request.CoBrchDesc,*  
 *CoBrchSts = request.CoBrchSts,*  
 *CreatedBy = \_currentUserService.GetUserId()*  
 *};*  
  
 *var (data, outputValues) = await \_repository.ExecuteWithOutputAsync<BrokerBranchDto>(*  
 *"LB\_SP\_CrudMstCoBrch", parameters, isStoredProcedure: true);*  
  
 *var statusCode = outputValues.GetValueOrDefault("StatusCode", 500);*  
 *var statusMsg = outputValues.GetValueOrDefault("StatusMsg", "Unknown error")?.ToString();*  
  
 *if ((int)statusCode != 200)*  
 *{*  
 *throw new DomainException(statusMsg ?? "Failed to create branch");*  
 *}*  
  
 *// Return the created branch*  
 *return await GetMstCoBrchByIdAsync(request.CoCode, request.CoBrchCode, cancellationToken);*  
 *}*  
  
 *public async Task<PagedResult<BrokerBranchDto>> GetMstCoBrchListAsync(*  
 *int pageNumber, int pageSize, string? searchTerm, string? coCode,*   
 *CancellationToken cancellationToken)*  
 *{*  
 *var parameters = new*  
 *{*  
 *PageNumber = pageNumber,*  
 *PageSize = pageSize,*  
 *SearchTerm = searchTerm,*  
 *CoCode = coCode*  
 *};*  
  
 *var (data, outputValues) = await \_repository.ExecuteWithOutputAsync<BrokerBranchDto>(*  
 *"LB\_SP\_GetMstCoBrchList", parameters, isStoredProcedure: true);*  
  
 *var totalCount = (int)(outputValues.GetValueOrDefault("TotalCount", 0) ?? 0);*  
 *var statusCode = (int)(outputValues.GetValueOrDefault("StatusCode", 500) ?? 500);*  
  
 *if (statusCode != 200)*  
 *{*  
 *throw new DomainException("Failed to retrieve branch list");*  
 *}*  
  
 *return new PagedResult<BrokerBranchDto>*  
 *{*  
 *Data = data.ToList(),*  
 *PageNumber = pageNumber,*  
 *PageSize = pageSize,*  
 *TotalRecords = totalCount,*  
 *TotalPages = (int)Math.Ceiling((double)totalCount / pageSize)*  
 *};*  
 *}*  
*}***

**Authentication & Security**

**Custom Token Authentication**

**SimRMS implements a sophisticated token-based authentication system with the following components:**

***Authentication Flow***

1. **Handshake Initiation: Client requests a handshake token**
2. **Token Generation: External token service generates user token**
3. **Token Validation: Each request validates the bearer token**
4. **Claims Resolution: Token is resolved to user claims and permissions**

***Authentication Components***

**TokenAuthenticationMiddleware**

***public class TokenAuthenticationMiddleware*  
*{*  
 *public async Task InvokeAsync(HttpContext context, RequestDelegate next)*  
 *{*  
 *// Check if route requires authentication*  
 *if (IsAuthenticationRequired(context.Request.Path))*  
 *{*  
 *var token = ExtractTokenFromHeader(context.Request);*  
 *if (string.IsNullOrEmpty(token))*  
 *{*  
 *await HandleUnauthorized(context);*  
 *return;*  
 *}*  
  
 *var validationResult = await \_externalTokenService.ValidateTokenAsync(token);*  
 *if (!validationResult.IsValid)*  
 *{*  
 *await HandleUnauthorized(context);*  
 *return;*  
 *}*  
  
 *// Set user context*  
 *context.Items["User"] = validationResult.UserInfo;*  
 *}*  
  
 *await next(context);*  
 *}*  
*}***

**TokenAuthenticationSchemeHandler**

***public class TokenAuthenticationSchemeHandler : AuthenticationHandler<TokenAuthenticationSchemeOptions>*  
*{*  
 *protected override async Task<AuthenticateResult> HandleAuthenticateAsync()*  
 *{*  
 *// Extract token from request*  
 *var token = Request.Headers["Authorization"]*  
 *.FirstOrDefault()?.Replace("Bearer ", "");*  
  
 *if (string.IsNullOrEmpty(token))*  
 *return AuthenticateResult.NoResult();*  
  
 *// Validate token with external service*  
 *var validationResult = await \_externalTokenService.ValidateTokenAsync(token);*  
  
 *if (!validationResult.IsValid)*  
 *return AuthenticateResult.Fail("Invalid token");*  
  
 *// Create claims principal*  
 *var claims = CreateClaimsFromTokenData(validationResult.UserInfo);*  
 *var identity = new ClaimsIdentity(claims, Scheme.Name);*  
 *var principal = new ClaimsPrincipal(identity);*  
  
 *return AuthenticateResult.Success(new AuthenticationTicket(principal, Scheme.Name));*  
 *}*  
*}***

***Security Configuration***

***{*  
 *"Security": {*  
 *"AllowedRoutes": [*  
 *"/health",*  
 *"/api/V1/handshake",*  
 *"/api/V1/handshake/validate"*  
 *],*  
 *"AllowedRoutePrefixes": [*  
 *"/api/V1/auth/login",*  
 *"/swagger",*  
 *"/health"*  
 *],*  
 *"EnableStrictPathMatching": true,*  
 *"CaseSensitiveMatching": false*  
 *}*  
*}***

***Token Service Integration***

***public interface IExternalTokenService*  
*{*  
 *Task<HandshakeResult> InitiateHandshakeAsync(HandshakeRequest request);*  
 *Task<TokenValidationResult> ValidateTokenAsync(string token);*  
 *Task<GenerateTokenResult> GenerateTokenAsync(GenerateTokenRequest request);*  
 *Task<bool> RevokeTokenAsync(string token);*  
*}*  
  
*public class ExternalTokenService : IExternalTokenService*  
*{*  
 *private readonly HttpClient \_httpClient;*  
 *private readonly IConfiguration \_configuration;*  
  
 *public async Task<TokenValidationResult> ValidateTokenAsync(string token)*  
 *{*  
 *var request = new HttpRequestMessage(HttpMethod.POST, \_configuration["TokenService:ValidateUrl"])*  
 *{*  
 *Content = JsonContent.Create(new { token })*  
 *};*  
  
 *var response = await \_httpClient.SendAsync(request);*  
  
 *if (!response.IsSuccessStatusCode)*  
 *{*  
 *return new TokenValidationResult { IsValid = false };*  
 *}*  
  
 *var result = await response.Content.ReadFromJsonAsync<TokenValidationResponse>();*  
 *return new TokenValidationResult*  
 *{*  
 *IsValid = result.IsValid,*  
 *UserInfo = result.UserInfo,*  
 *ExpiresAt = result.ExpiresAt*  
 *};*  
 *}*  
*}***

**Authorization**

***Policy-Based Authorization***

***services.AddAuthorization(options =>*  
*{*  
 *options.AddPolicy("AdminOnly", policy =>*   
 *policy.RequireClaim("Role", "Admin"));*  
  
 *options.AddPolicy("BranchManager", policy =>*   
 *policy.RequireClaim("Permission", "BranchManagement"));*  
  
 *options.AddPolicy("ReadOnly", policy =>*   
 *policy.RequireClaim("Permission", "ReadAccess"));*  
*});***

***Controller Authorization***

***[Authorize(Policy = "BranchManager")]*  
*[HttpPost]*  
*public async Task<ActionResult> CreateBranch([FromBody] CreateBranchRequest request)*  
*{*  
 *// Only users with BranchManagement permission can access*  
*}*  
  
*[Authorize(Policy = "ReadOnly")]*  
*[HttpGet]*  
*public async Task<ActionResult> GetBranches()*  
*{*  
 *// Users with ReadAccess permission can access*  
*}***

**Security Headers Middleware**

***public class SecurityHeadersMiddleware*  
*{*  
 *public async Task InvokeAsync(HttpContext context, RequestDelegate next)*  
 *{*  
 *// Add security headers*  
 *context.Response.Headers.Add("X-Content-Type-Options", "nosniff");*  
 *context.Response.Headers.Add("X-Frame-Options", "DENY");*  
 *context.Response.Headers.Add("X-XSS-Protection", "1; mode=block");*  
 *context.Response.Headers.Add("Referrer-Policy", "strict-origin-when-cross-origin");*  
 *context.Response.Headers.Add("Content-Security-Policy",*   
 *"default-src 'self'; script-src 'self' 'unsafe-inline'");*  
  
 *await next(context);*  
 *}*  
*}***

**Error Handling**

**Exception Hierarchy**

**SimRMS implements a comprehensive exception handling system:**

***// Base domain exception*  
*public abstract class DomainException : Exception*  
*{*  
 *protected DomainException(string message) : base(message) { }*  
 *protected DomainException(string message, Exception innerException) : base(message, innerException) { }*  
*}*  
  
*// Specific exceptions*  
*public class NotFoundException : DomainException*  
*{*  
 *public NotFoundException(string message) : base(message) { }*  
 *public NotFoundException(string entityName, object key)*   
 *: base($"{entityName} with key '{key}' was not found") { }*  
*}*  
  
*public class ValidationException : DomainException*  
*{*  
 *public IEnumerable<ValidationErrorDetail> ValidationErrors { get; }*  
  
 *public ValidationException(IEnumerable<ValidationErrorDetail> validationErrors)*  
 *: base("One or more validation errors occurred")*  
 *{*  
 *ValidationErrors = validationErrors;*  
 *}*  
*}*  
  
*public class FileOperationException : DomainException*  
*{*  
 *public string? FileName { get; }*  
 *public string? FilePath { get; }*  
  
 *public FileOperationException(string message, string? fileName = null, string? filePath = null)*  
 *: base(message)*  
 *{*  
 *FileName = fileName;*  
 *FilePath = filePath;*  
 *}*  
*}***

**Global Exception Handling Middleware**

***public class ExceptionHandlingMiddleware*  
*{*  
 *private readonly RequestDelegate \_next;*  
 *private readonly ILogger<ExceptionHandlingMiddleware> \_logger;*  
  
 *public async Task InvokeAsync(HttpContext context)*  
 *{*  
 *try*  
 *{*  
 *await \_next(context);*  
 *}*  
 *catch (Exception ex)*  
 *{*  
 *\_logger.LogError(ex, "An unhandled exception occurred");*  
 *await HandleExceptionAsync(context, ex);*  
 *}*  
 *}*  
  
 *private static async Task HandleExceptionAsync(HttpContext context, Exception exception)*  
 *{*  
 *var response = context.Response;*  
 *response.ContentType = "application/json";*  
  
 *var apiResponse = exception switch*  
 *{*  
 *ValidationException validationEx => new ApiResponse<object>*  
 *{*  
 *Success = false,*  
 *Message = validationEx.Message,*  
 *Errors = validationEx.ValidationErrors.Select(e => new*   
 *{*   
 *Field = e.PropertyName,*   
 *Message = e.ErrorMessage*   
 *}).ToList()*  
 *},*  
 *NotFoundException notFoundEx => new ApiResponse<object>*  
 *{*  
 *Success = false,*  
 *Message = notFoundEx.Message*  
 *},*  
 *DomainException domainEx => new ApiResponse<object>*  
 *{*  
 *Success = false,*  
 *Message = domainEx.Message*  
 *},*  
 *\_ => new ApiResponse<object>*  
 *{*  
 *Success = false,*  
 *Message = "An internal server error occurred"*  
 *}*  
 *};*  
  
 *response.StatusCode = exception switch*  
 *{*  
 *ValidationException => StatusCodes.Status400BadRequest,*  
 *NotFoundException => StatusCodes.Status404NotFound,*  
 *DomainException => StatusCodes.Status400BadRequest,*  
 *\_ => StatusCodes.Status500InternalServerError*  
 *};*  
  
 *var jsonResponse = JsonSerializer.Serialize(apiResponse, new JsonSerializerOptions*  
 *{*  
 *PropertyNamingPolicy = JsonNamingPolicy.CamelCase*  
 *});*  
  
 *await response.WriteAsync(jsonResponse);*  
 *}*  
*}***

**Controller Error Handling**

***public class BrokerBranchController : BaseController*  
*{*  
 *[HttpPost]*  
 *public async Task<ActionResult<ApiResponse<BrokerBranchDto>>> CreateBranch(*  
 *[FromBody] CreateBranchRequest request)*  
 *{*  
 *try*  
 *{*  
 *var result = await \_brokerBranchService.CreateMstCoBrchAsync(request, cancellationToken);*  
 *return Ok(result, "Branch created successfully");*  
 *}*  
 *catch (ArgumentNullException ex)*  
 *{*  
 *\_logger.LogWarning(ex, "Invalid create request");*  
 *return BadRequest<BrokerBranchDto>(ex.Message);*  
 *}*  
 *catch (ValidationException ex)*  
 *{*  
 *\_logger.LogWarning(ex, "Validation failed");*  
 *var errors = ex.ValidationErrors?.Select(e => e.ErrorMessage).ToList();*  
 *return BadRequest<BrokerBranchDto>("Validation failed", errors);*  
 *}*  
 *catch (DomainException ex)*  
 *{*  
 *\_logger.LogWarning(ex, "Business rule violation");*  
 *return BadRequest<BrokerBranchDto>(ex.Message);*  
 *}*  
 *catch (Exception ex)*  
 *{*  
 *\_logger.LogError(ex, "Error creating branch");*  
 *return BadRequest<BrokerBranchDto>("Failed to create branch");*  
 *}*  
 *}*  
*}***

**Performance Considerations**

**Database Performance**

***Connection Pooling Configuration***

***{*  
 *"LB\_DAL": {*  
 *"EnableConnectionPooling": true,*  
 *"MaxPoolSize": 100,*  
 *"MinPoolSize": 5,*  
 *"ConnectionTimeout": 30,*  
 *"CommandTimeout": 60*  
 *}*  
*}***

***Pagination Strategy***

***// Always implement server-side pagination*  
*public async Task<PagedResult<T>> GetPagedResultAsync<T>(*  
 *string storedProcedure,*   
 *object parameters,*   
 *int pageNumber,*   
 *int pageSize)*  
*{*  
 *var (data, outputValues) = await \_repository.ExecuteWithOutputAsync<T>(*  
 *storedProcedure,*   
 *new {*   
 *PageNumber = pageNumber,*   
 *PageSize = pageSize,*  
 *...parameters*   
 *},*   
 *isStoredProcedure: true);*  
  
 *var totalCount = (int)(outputValues.GetValueOrDefault("TotalCount", 0) ?? 0);*  
  
 *return new PagedResult<T>*  
 *{*  
 *Data = data.ToList(),*  
 *PageNumber = pageNumber,*  
 *PageSize = pageSize,*  
 *TotalRecords = totalCount,*  
 *TotalPages = (int)Math.Ceiling((double)totalCount / pageSize)*  
 *};*  
*}***

***Caching Strategy***

***public class CacheService : ICacheService*  
*{*  
 *private readonly IMemoryCache \_memoryCache;*  
 *private readonly ILogger<CacheService> \_logger;*  
  
 *public async Task<T> GetOrSetAsync<T>(string key, Func<Task<T>> getItem, TimeSpan? expiry = null)*  
 *{*  
 *if (\_memoryCache.TryGetValue(key, out T cachedValue))*  
 *{*  
 *return cachedValue;*  
 *}*  
  
 *var item = await getItem();*  
  
 *var options = new MemoryCacheEntryOptions*  
 *{*  
 *SlidingExpiration = expiry ?? TimeSpan.FromMinutes(30),*  
 *Priority = CacheItemPriority.Normal*  
 *};*  
  
 *\_memoryCache.Set(key, item, options);*  
 *return item;*  
 *}*  
  
 *public void Remove(string key)*  
 *{*  
 *\_memoryCache.Remove(key);*  
 *}*  
  
 *public void RemoveByPattern(string pattern)*  
 *{*  
 *// Implementation for pattern-based cache invalidation*  
 *}*  
*}*  
  
*// Usage in service*  
*public async Task<BrokerBranchDto?> GetMstCoBrchByIdAsync(string coCode, string coBrchCode, CancellationToken cancellationToken)*  
*{*  
 *var cacheKey = $"branch:{coCode}:{coBrchCode}";*  
  
 *return await \_cacheService.GetOrSetAsync(cacheKey, async () =>*  
 *{*  
 *var parameters = new { CoCode = coCode, CoBrchCode = coBrchCode };*  
 *return await \_repository.QuerySingleOrDefaultAsync<BrokerBranchDto>(*  
 *"LB\_SP\_GetMstCoBrch\_ById", parameters, isStoredProcedure: true);*  
 *}, TimeSpan.FromMinutes(15));*  
*}***

**Background Services**

***public class CacheCleanupService : BackgroundService*  
*{*  
 *private readonly ICacheService \_cacheService;*  
 *private readonly ILogger<CacheCleanupService> \_logger;*  
  
 *protected override async Task ExecuteAsync(CancellationToken stoppingToken)*  
 *{*  
 *while (!stoppingToken.IsCancellationRequested)*  
 *{*  
 *try*  
 *{*  
 *// Cleanup expired cache entries*  
 *\_cacheService.CleanupExpiredEntries();*  
  
 *// Wait for 1 hour before next cleanup*  
 *await Task.Delay(TimeSpan.FromHours(1), stoppingToken);*  
 *}*  
 *catch (Exception ex)*  
 *{*  
 *\_logger.LogError(ex, "Error during cache cleanup");*  
 *await Task.Delay(TimeSpan.FromMinutes(5), stoppingToken);*  
 *}*  
 *}*  
 *}*  
*}***

**Performance Monitoring Middleware**

***public class PerformanceMiddleware*  
*{*  
 *private readonly RequestDelegate \_next;*  
 *private readonly ILogger<PerformanceMiddleware> \_logger;*  
  
 *public async Task InvokeAsync(HttpContext context)*  
 *{*  
 *var stopwatch = Stopwatch.StartNew();*  
  
 *await \_next(context);*  
  
 *stopwatch.Stop();*  
  
 *var elapsedMilliseconds = stopwatch.ElapsedMilliseconds;*  
  
 *if (elapsedMilliseconds > 1000) // Log slow requests*  
 *{*  
 *\_logger.LogWarning(*  
 *"Slow request: {Method} {Path} took {ElapsedMilliseconds}ms",*  
 *context.Request.Method,*  
 *context.Request.Path,*  
 *elapsedMilliseconds);*  
 *}*  
  
 *// Add performance header*  
 *context.Response.Headers.Add("X-Response-Time", $"{elapsedMilliseconds}ms");*  
 *}*  
*}***

**Testing**

**Testing Strategy**

**SimRMS follows a comprehensive testing approach:**

1. **Unit Tests: Test business logic in services and domain entities**
2. **Integration Tests: Test complete workflows including database operations**
3. **API Tests: Test controller endpoints and API contracts**

**Unit Testing Examples**

***Service Unit Tests***

***public class BrokerBranchServiceTests*  
*{*  
 *private readonly Mock<IGenericRepository> \_mockRepository;*  
 *private readonly Mock<ICacheService> \_mockCacheService;*  
 *private readonly Mock<ICurrentUserService> \_mockCurrentUserService;*  
 *private readonly BrokerBranchService \_service;*  
  
 *public BrokerBranchServiceTests()*  
 *{*  
 *\_mockRepository = new Mock<IGenericRepository>();*  
 *\_mockCacheService = new Mock<ICacheService>();*  
 *\_mockCurrentUserService = new Mock<ICurrentUserService>();*  
  
 *\_service = new BrokerBranchService(*  
 *\_mockRepository.Object,*  
 *\_mockCacheService.Object,*  
 *\_mockCurrentUserService.Object);*  
 *}*  
  
 *[Fact]*  
 *public async Task CreateMstCoBrchAsync\_ValidRequest\_ReturnsCreatedBranch()*  
 *{*  
 *// Arrange*  
 *var request = new CreateMstCoBrchRequest*  
 *{*  
 *CoCode = "LBSL",*  
 *CoBrchCode = "HEAD01",*  
 *CoBrchDesc = "Head Office",*  
 *CoBrchSts = true*  
 *};*  
  
 *var outputValues = new Dictionary<string, object>*  
 *{*  
 *["StatusCode"] = 200,*  
 *["StatusMsg"] = "Success",*  
 *["RowsAffected"] = 1*  
 *};*  
  
 *\_mockRepository*  
 *.Setup(r => r.ExecuteWithOutputAsync<BrokerBranchDto>(*  
 *It.IsAny<string>(), It.IsAny<object>(), true))*  
 *.ReturnsAsync((new List<BrokerBranchDto>(), outputValues));*  
  
 *\_mockCurrentUserService*  
 *.Setup(u => u.GetUserId())*  
 *.Returns("test-user");*  
  
 *// Act*  
 *var result = await \_service.CreateMstCoBrchAsync(request, CancellationToken.None);*  
  
 *// Assert*  
 *Assert.NotNull(result);*  
 *\_mockRepository.Verify(r => r.ExecuteWithOutputAsync<BrokerBranchDto>(*  
 *"LB\_SP\_CrudMstCoBrch", It.IsAny<object>(), true), Times.Once);*  
 *}*  
  
 *[Fact]*  
 *public async Task CreateMstCoBrchAsync\_DatabaseError\_ThrowsDomainException()*  
 *{*  
 *// Arrange*  
 *var request = new CreateMstCoBrchRequest*  
 *{*  
 *CoCode = "LBSL",*  
 *CoBrchCode = "HEAD01"*  
 *};*  
  
 *var outputValues = new Dictionary<string, object>*  
 *{*  
 *["StatusCode"] = 500,*  
 *["StatusMsg"] = "Database error"*  
 *};*  
  
 *\_mockRepository*  
 *.Setup(r => r.ExecuteWithOutputAsync<BrokerBranchDto>(*  
 *It.IsAny<string>(), It.IsAny<object>(), true))*  
 *.ReturnsAsync((new List<BrokerBranchDto>(), outputValues));*  
  
 *// Act & Assert*  
 *var exception = await Assert.ThrowsAsync<DomainException>(*  
 *() => \_service.CreateMstCoBrchAsync(request, CancellationToken.None));*  
  
 *Assert.Equal("Database error", exception.Message);*  
 *}*  
*}***

***Validation Unit Tests***

***public class CreateMstCoBrchRequestValidatorTests*  
*{*  
 *private readonly CreateMstCoBrchRequestValidator \_validator;*  
  
 *public CreateMstCoBrchRequestValidatorTests()*  
 *{*  
 *\_validator = new CreateMstCoBrchRequestValidator();*  
 *}*  
  
 *[Theory]*  
 *[InlineData("", false)] // Empty code*  
 *[InlineData("TOOLONG", false)] // Too long*  
 *[InlineData("LBSL", true)] // Valid*  
 *[InlineData("TEST1", true)] // Valid with number*  
 *public void CoCode\_Validation\_WorksCorrectly(string coCode, bool shouldBeValid)*  
 *{*  
 *// Arrange*  
 *var request = new CreateMstCoBrchRequest*  
 *{*  
 *CoCode = coCode,*  
 *CoBrchCode = "HEAD01",*  
 *CoBrchDesc = "Test Branch"*  
 *};*  
  
 *// Act*  
 *var result = \_validator.Validate(request);*  
  
 *// Assert*  
 *if (shouldBeValid)*  
 *{*  
 *Assert.True(result.IsValid);*  
 *}*  
 *else*  
 *{*  
 *Assert.False(result.IsValid);*  
 *Assert.Contains(result.Errors, e => e.PropertyName == nameof(request.CoCode));*  
 *}*  
 *}*  
*}***

**Integration Testing**

***Database Integration Tests***

***public class BrokerBranchIntegrationTests : IClassFixture<DatabaseFixture>*  
*{*  
 *private readonly DatabaseFixture \_fixture;*  
 *private readonly IBrokerBranchService \_service;*  
  
 *public BrokerBranchIntegrationTests(DatabaseFixture fixture)*  
 *{*  
 *\_fixture = fixture;*  
 *\_service = \_fixture.GetService<IBrokerBranchService>();*  
 *}*  
  
 *[Fact]*  
 *public async Task CreateAndRetrieveBranch\_EndToEnd\_WorksCorrectly()*  
 *{*  
 *// Arrange*  
 *var createRequest = new CreateMstCoBrchRequest*  
 *{*  
 *CoCode = "TEST",*  
 *CoBrchCode = "BR001",*  
 *CoBrchDesc = "Test Branch",*  
 *CoBrchSts = true*  
 *};*  
  
 *// Act - Create*  
 *var createdBranch = await \_service.CreateMstCoBrchAsync(*  
 *createRequest, CancellationToken.None);*  
  
 *// Act - Retrieve*  
 *var retrievedBranch = await \_service.GetMstCoBrchByIdAsync(*  
 *"TEST", "BR001", CancellationToken.None);*  
  
 *// Assert*  
 *Assert.NotNull(createdBranch);*  
 *Assert.NotNull(retrievedBranch);*  
 *Assert.Equal(createRequest.CoCode, retrievedBranch.CoCode);*  
 *Assert.Equal(createRequest.CoBrchCode, retrievedBranch.CoBrchCode);*  
 *Assert.Equal(createRequest.CoBrchDesc, retrievedBranch.CoBrchDesc);*  
  
 *// Cleanup*  
 *await \_service.DeleteMstCoBrchAsync("TEST", "BR001",*   
 *new DeleteMstCoBrchRequest(), CancellationToken.None);*  
 *}*  
*}*  
  
*public class DatabaseFixture : IDisposable*  
*{*  
 *private readonly IServiceProvider \_serviceProvider;*  
  
 *public DatabaseFixture()*  
 *{*  
 *var services = new ServiceCollection();*  
  
 *// Configure test database connection*  
 *services.AddSingleton<IConfiguration>(new ConfigurationBuilder()*  
 *.AddJsonFile("appsettings.Test.json")*  
 *.Build());*  
  
 *// Register dependencies*  
 *services.AddApplication();*  
 *services.AddInfrastructure();*  
  
 *\_serviceProvider = services.BuildServiceProvider();*  
  
 *// Initialize test database*  
 *InitializeDatabase();*  
 *}*  
  
 *public T GetService<T>() => \_serviceProvider.GetRequiredService<T>();*  
  
 *private void InitializeDatabase()*  
 *{*  
 *// Set up test data if needed*  
 *}*  
  
 *public void Dispose()*  
 *{*  
 *// Cleanup test database*  
 *\_serviceProvider?.Dispose();*  
 *}*  
*}***

**API Testing**

***Controller Integration Tests***

***public class BrokerBranchControllerTests : IClassFixture<WebApplicationFactory<Program>>*  
*{*  
 *private readonly WebApplicationFactory<Program> \_factory;*  
 *private readonly HttpClient \_client;*  
  
 *public BrokerBranchControllerTests(WebApplicationFactory<Program> factory)*  
 *{*  
 *\_factory = factory;*  
 *\_client = \_factory.CreateClient();*  
  
 *// Add authentication header for tests*  
 *\_client.DefaultRequestHeaders.Authorization =*   
 *new AuthenticationHeaderValue("Bearer", "test-token");*  
 *}*  
  
 *[Fact]*  
 *public async Task GetBranches\_ReturnsSuccessWithData()*  
 *{*  
 *// Act*  
 *var response = await \_client.GetAsync("/api/v1/brokerBranch?pageNumber=1&pageSize=10");*  
  
 *// Assert*  
 *response.EnsureSuccessStatusCode();*  
 *var content = await response.Content.ReadAsStringAsync();*  
 *var apiResponse = JsonSerializer.Deserialize<ApiResponse<IEnumerable<BrokerBranchDto>>>(*  
 *content, new JsonSerializerOptions { PropertyNameCaseInsensitive = true });*  
  
 *Assert.True(apiResponse.Success);*  
 *Assert.NotNull(apiResponse.Data);*  
 *}*  
  
 *[Fact]*  
 *public async Task CreateBranch\_ValidRequest\_ReturnsCreated()*  
 *{*  
 *// Arrange*  
 *var request = new CreateMstCoBrchRequest*  
 *{*  
 *CoCode = "TEST",*  
 *CoBrchCode = "BR001",*  
 *CoBrchDesc = "Test Branch",*  
 *CoBrchSts = true*  
 *};*  
  
 *var json = JsonSerializer.Serialize(request);*  
 *var content = new StringContent(json, Encoding.UTF8, "application/json");*  
  
 *// Act*  
 *var response = await \_client.PostAsync("/api/v1/brokerBranch", content);*  
  
 *// Assert*  
 *Assert.Equal(HttpStatusCode.Created, response.StatusCode);*  
  
 *var responseContent = await response.Content.ReadAsStringAsync();*  
 *var apiResponse = JsonSerializer.Deserialize<ApiResponse<BrokerBranchDto>>(*  
 *responseContent, new JsonSerializerOptions { PropertyNameCaseInsensitive = true });*  
  
 *Assert.True(apiResponse.Success);*  
 *Assert.Equal(request.CoCode, apiResponse.Data.CoCode);*  
 *}*  
*}***

**Test Configuration**

***appsettings.Test.json***

***{*  
 *"ConnectionStrings": {*  
 *"DefaultConnection": "Server=(localdb)\\mssqllocaldb;Database=SimRMS\_Test;Integrated Security=true;TrustServerCertificate=true;"*  
 *},*  
 *"Serilog": {*  
 *"MinimumLevel": {*  
 *"Default": "Warning"*  
 *}*  
 *},*  
 *"TokenService": {*  
 *"ValidateUrl": "http://localhost:9999/validate"*  
 *}*  
*}***

**Deployment**

**Build Configuration**

***Release Configuration***

***<Project Sdk="Microsoft.NET.Sdk.Web">*  
 *<PropertyGroup>*  
 *<TargetFramework>net8.0</TargetFramework>*  
 *<Nullable>enable</Nullable>*  
 *<ImplicitUsings>enable</ImplicitUsings>*  
 *<GenerateDocumentationFile>true</GenerateDocumentationFile>*  
 *<TreatWarningsAsErrors>true</TreatWarningsAsErrors>*  
 *<WarningsNotAsErrors>CS1591</WarningsNotAsErrors>*  
 *</PropertyGroup>*  
*</Project>***

***Build Scripts***

***# Build script for production*  
*#!/bin/bash*  
  
*# Clean previous builds*  
*dotnet clean --configuration Release*  
  
*# Restore dependencies*  
*dotnet restore*  
  
*# Build solution*  
*dotnet build --configuration Release --no-restore*  
  
*# Run tests*  
*dotnet test --configuration Release --no-build --logger trx*  
  
*# Publish application*  
*dotnet publish src/SimRMS.WebAPI/SimRMS.WebAPI.csproj \*  
 *--configuration Release \*  
 *--output ./publish \*  
 *--no-build \*  
 *--self-contained false \*  
 *--runtime win-x64***

**Environment Configuration**

***Production appsettings.Production.json***

***{*  
 *"ConnectionStrings": {*  
 *"DefaultConnection": "Server=prod-server;Database=SimRMS\_Prod;User Id=prod\_user;Password=prod\_password;TrustServerCertificate=true;Encrypt=true;"*  
 *},*  
 *"Serilog": {*  
 *"MinimumLevel": {*  
 *"Default": "Information",*  
 *"Override": {*  
 *"Microsoft": "Warning",*  
 *"System": "Warning"*  
 *}*  
 *},*  
 *"WriteTo": [*  
 *{*  
 *"Name": "File",*  
 *"Args": {*  
 *"path": "/var/log/simrms/rms-log-.txt",*  
 *"rollingInterval": "Day",*  
 *"retainedFileCountLimit": 30*  
 *}*  
 *}*  
 *]*  
 *},*  
 *"TokenService": {*  
 *"ValidateUrl": "https://prod-token-service.company.com/validate",*  
 *"ApiKey": "prod-api-key"*  
 *},*  
 *"FileUpload": {*  
 *"Default": {*  
 *"UploadPath": "/var/uploads/simrms",*  
 *"BaseUrl": "https://files.company.com/simrms"*  
 *}*  
 *}*  
*}***

***Docker Configuration***

**Dockerfile**

***FROM mcr.microsoft.com/dotnet/aspnet:8.0 AS base*  
*WORKDIR /app*  
*EXPOSE 80*  
*EXPOSE 443*  
  
*FROM mcr.microsoft.com/dotnet/sdk:8.0 AS build*  
*WORKDIR /src*  
  
*# Copy project files*  
*COPY ["src/SimRMS.WebAPI/SimRMS.WebAPI.csproj", "src/SimRMS.WebAPI/"]*  
*COPY ["src/SimRMS.Application/SimRMS.Application.csproj", "src/SimRMS.Application/"]*  
*COPY ["src/SimRMS.Domain/SimRMS.Domain.csproj", "src/SimRMS.Domain/"]*  
*COPY ["src/SimRMS.Infrastructure/SimRMS.Infrastructure.csproj", "src/SimRMS.Infrastructure/"]*  
*COPY ["src/SimRMS.Shared/SimRMS.Shared.csproj", "src/SimRMS.Shared/"]*  
  
*# Restore dependencies*  
*RUN dotnet restore "src/SimRMS.WebAPI/SimRMS.WebAPI.csproj"*  
  
*# Copy source code*  
*COPY . .*  
  
*# Build application*  
*WORKDIR "/src/src/SimRMS.WebAPI"*  
*RUN dotnet build "SimRMS.WebAPI.csproj" -c Release -o /app/build*  
  
*FROM build AS publish*  
*RUN dotnet publish "SimRMS.WebAPI.csproj" -c Release -o /app/publish /p:UseAppHost=false*  
  
*FROM base AS final*  
*WORKDIR /app*  
*COPY --from=publish /app/publish .*  
  
*# Copy external libraries*  
*COPY lib/ ./lib/*  
  
*# Create logs directory*  
*RUN mkdir -p /app/logs*  
  
*ENTRYPOINT ["dotnet", "SimRMS.WebAPI.dll"]***

**docker-compose.yml**

***version: '3.8'*  
  
*services:*  
 *simrms-api:*  
 *build:*  
 *context: .*  
 *dockerfile: Dockerfile*  
 *ports:*  
 *- "8080:80"*  
 *- "8443:443"*  
 *environment:*  
 *- ASPNETCORE\_ENVIRONMENT=Production*  
 *- ASPNETCORE\_URLS=https://+:443;http://+:80*  
 *- ASPNETCORE\_Kestrel\_\_Certificates\_\_Default\_\_Password=password*  
 *- ASPNETCORE\_Kestrel\_\_Certificates\_\_Default\_\_Path=/https/cert.pfx*  
 *volumes:*  
 *- ./certs:/https/:ro*  
 *- ./logs:/app/logs*  
 *- ./uploads:/var/uploads/simrms*  
 *networks:*  
 *- simrms-network*  
 *depends\_on:*  
 *- sqlserver*  
 *- redis*  
  
 *sqlserver:*  
 *image: mcr.microsoft.com/mssql/server:2022-latest*  
 *environment:*  
 *- ACCEPT\_EULA=Y*  
 *- SA\_PASSWORD=YourStrong!Passw0rd*  
 *ports:*  
 *- "1433:1433"*  
 *volumes:*  
 *- sqlserver-data:/var/opt/mssql*  
 *networks:*  
 *- simrms-network*  
  
 *redis:*  
 *image: redis:7-alpine*  
 *ports:*  
 *- "6379:6379"*  
 *volumes:*  
 *- redis-data:/data*  
 *networks:*  
 *- simrms-network*  
  
*volumes:*  
 *sqlserver-data:*  
 *redis-data:*  
  
*networks:*  
 *simrms-network:*  
 *driver: bridge***

**Health Checks Configuration**

***Health Check Implementation***

***public class LBDALHealthCheck : IHealthCheck*  
*{*  
 *private readonly IGenericRepository \_repository;*  
  
 *public async Task<HealthCheckResult> CheckHealthAsync(*  
 *HealthCheckContext context,*   
 *CancellationToken cancellationToken = default)*  
 *{*  
 *try*  
 *{*  
 *var result = await \_repository.ExecuteScalarAsync<int>(*  
 *"SELECT 1", isStoredProcedure: false);*  
  
 *return result == 1*   
 *? HealthCheckResult.Healthy("Database connection is healthy")*  
 *: HealthCheckResult.Unhealthy("Database connection failed");*  
 *}*  
 *catch (Exception ex)*  
 *{*  
 *return HealthCheckResult.Unhealthy("Database connection failed", ex);*  
 *}*  
 *}*  
*}*  
  
*public class ExternalApiHealthCheck : IHealthCheck*  
*{*  
 *private readonly IExternalTokenService \_tokenService;*  
  
 *public async Task<HealthCheckResult> CheckHealthAsync(*  
 *HealthCheckContext context,*   
 *CancellationToken cancellationToken = default)*  
 *{*  
 *try*  
 *{*  
 *var result = await \_tokenService.ValidateTokenAsync("health-check-token");*  
  
 *return result != null*   
 *? HealthCheckResult.Healthy("External API is responsive")*  
 *: HealthCheckResult.Degraded("External API returned unexpected response");*  
 *}*  
 *catch (Exception ex)*  
 *{*  
 *return HealthCheckResult.Unhealthy("External API is not responding", ex);*  
 *}*  
 *}*  
*}***

***Health Check Registration***

***services.AddHealthChecks()*  
 *.AddCheck<LBDALHealthCheck>("database")*  
 *.AddCheck<ExternalApiHealthCheck>("external-api")*  
 *.AddUrlGroup(new Uri("https://external-service.com/health"), "external-service")*  
 *.AddDiskStorageHealthCheck(setup => setup.AddDrive("C:\\", 1024)) // At least 1GB free*  
 *.AddMemoryHealthCheck(1024 \* 1024 \* 1024); // At least 1GB available memory*  
  
*app.MapHealthChecks("/health", new HealthCheckOptions*  
*{*  
 *ResponseWriter = UIResponseWriter.WriteHealthCheckUIResponse,*  
 *ResultStatusCodes =*  
 *{*  
 *[HealthStatus.Healthy] = StatusCodes.Status200OK,*  
 *[HealthStatus.Degraded] = StatusCodes.Status200OK,*  
 *[HealthStatus.Unhealthy] = StatusCodes.Status503ServiceUnavailable*  
 *}*  
*});*  
  
*app.MapHealthChecks("/health/ready", new HealthCheckOptions*  
*{*  
 *Predicate = check => check.Tags.Contains("ready"),*  
 *ResponseWriter = UIResponseWriter.WriteHealthCheckUIResponse*  
*});*  
  
*app.MapHealthChecks("/health/live", new HealthCheckOptions*  
*{*  
 *Predicate = \_ => false,*  
 *ResponseWriter = UIResponseWriter.WriteHealthCheckUIResponse*  
*});***

**Monitoring and Observability**

***Structured Logging with Serilog***

***Log.Logger = new LoggerConfiguration()*  
 *.MinimumLevel.Information()*  
 *.MinimumLevel.Override("Microsoft", LogEventLevel.Warning)*  
 *.MinimumLevel.Override("System", LogEventLevel.Warning)*  
 *.Enrich.FromLogContext()*  
 *.Enrich.WithProperty("Application", "SimRMS")*  
 *.Enrich.WithProperty("Environment", builder.Environment.EnvironmentName)*  
 *.WriteTo.Console(outputTemplate:*   
 *"[{Timestamp:HH:mm:ss} {Level:u3}] {Message:lj} {Properties:j}{NewLine}{Exception}")*  
 *.WriteTo.File(*  
 *path: "logs/simrms-log-.txt",*  
 *rollingInterval: RollingInterval.Day,*  
 *retainedFileCountLimit: 30,*  
 *outputTemplate:*   
 *"{Timestamp:yyyy-MM-dd HH:mm:ss.fff zzz} [{Level:u3}] {Message:lj} {Properties:j}{NewLine}{Exception}")*  
 *.WriteTo.Seq("http://seq-server:5341", apiKey: "seq-api-key")*  
 *.CreateLogger();***

***Custom Metrics***

***public class MetricsService*  
*{*  
 *private readonly IMetrics \_metrics;*  
  
 *public void RecordApiRequest(string endpoint, string method, int statusCode, long duration)*  
 *{*  
 *\_metrics.Measure.Counter.Increment(*  
 *"api\_requests\_total",*  
 *new MetricTags("endpoint", endpoint, "method", method, "status", statusCode.ToString()));*  
  
 *\_metrics.Measure.Histogram.Update(*  
 *"api\_request\_duration\_ms",*  
 *duration,*  
 *new MetricTags("endpoint", endpoint));*  
 *}*  
  
 *public void RecordDatabaseOperation(string operation, long duration, bool success)*  
 *{*  
 *\_metrics.Measure.Counter.Increment(*  
 *"database\_operations\_total",*  
 *new MetricTags("operation", operation, "success", success.ToString()));*  
  
 *\_metrics.Measure.Histogram.Update(*  
 *"database\_operation\_duration\_ms",*  
 *duration,*  
 *new MetricTags("operation", operation));*  
 *}*  
*}***

**Troubleshooting**

**Common Issues and Solutions**

***1. Database Connection Issues***

**Problem: Connection timeout or access denied errors**

**Solution:**

***// Check connection string in appsettings.json*  
*{*  
 *"ConnectionStrings": {*  
 *"DefaultConnection": "Server=server;Database=db;User Id=user;Password=pass;TrustServerCertificate=true;MultipleActiveResultSets=true;ConnectTimeout=30;CommandTimeout=60;"*  
 *}*  
*}*  
  
*// Test connection in health check*  
*public async Task<HealthCheckResult> CheckDatabaseAsync()*  
*{*  
 *try*  
 *{*  
 *using var connection = new SqlConnection(\_connectionString);*  
 *await connection.OpenAsync();*  
  
 *using var command = new SqlCommand("SELECT 1", connection);*  
 *var result = await command.ExecuteScalarAsync();*  
  
 *return HealthCheckResult.Healthy("Database connected successfully");*  
 *}*  
 *catch (Exception ex)*  
 *{*  
 *return HealthCheckResult.Unhealthy($"Database connection failed: {ex.Message}");*  
 *}*  
*}***

**Common Fixes: - Verify SQL Server is running and accessible - Check firewall settings - Validate credentials and permissions - Ensure database exists and user has access**

***2. Authentication/Authorization Issues***

**Problem: 401 Unauthorized or 403 Forbidden responses**

**Solution:**

***// Debug authentication in middleware*  
*public async Task InvokeAsync(HttpContext context, RequestDelegate next)*  
*{*  
 *var token = ExtractToken(context.Request);*  
  
 *if (string.IsNullOrEmpty(token))*  
 *{*  
 *\_logger.LogWarning("No token found in request for {Path}", context.Request.Path);*  
 *await HandleUnauthorized(context, "No authentication token provided");*  
 *return;*  
 *}*  
  
 *try*  
 *{*  
 *var validationResult = await \_externalTokenService.ValidateTokenAsync(token);*  
  
 *if (!validationResult.IsValid)*  
 *{*  
 *\_logger.LogWarning("Invalid token for {Path}: {Reason}",*   
 *context.Request.Path, validationResult.Reason);*  
 *await HandleUnauthorized(context, "Invalid authentication token");*  
 *return;*  
 *}*  
  
 *context.Items["User"] = validationResult.UserInfo;*  
 *await next(context);*  
 *}*  
 *catch (Exception ex)*  
 *{*  
 *\_logger.LogError(ex, "Error validating token for {Path}", context.Request.Path);*  
 *await HandleUnauthorized(context, "Authentication service unavailable");*  
 *}*  
*}***

**Common Fixes: - Verify token service is running and accessible - Check token expiration - Validate API keys and credentials - Review allowed routes configuration**

***3. File Upload Issues***

**Problem: File upload failures or access denied errors**

**Solution:**

***public class FileUploadTroubleshootingService*  
*{*  
 *public async Task<FileUploadDiagnostics> DiagnoseFileUploadIssueAsync(string serverName)*  
 *{*  
 *var config = \_configuration.GetFileUploadConfig(serverName);*  
 *var diagnostics = new FileUploadDiagnostics();*  
  
 *// Check upload directory exists*  
 *try*  
 *{*  
 *if (Directory.Exists(config.UploadPath))*  
 *{*  
 *diagnostics.DirectoryExists = true;*  
  
 *// Test write permissions*  
 *var testFile = Path.Combine(config.UploadPath, $"test\_{Guid.NewGuid()}.tmp");*  
 *await File.WriteAllTextAsync(testFile, "test");*  
 *File.Delete(testFile);*  
  
 *diagnostics.HasWritePermission = true;*  
 *}*  
 *}*  
 *catch (UnauthorizedAccessException)*  
 *{*  
 *diagnostics.HasWritePermission = false;*  
 *diagnostics.Issues.Add("No write permission to upload directory");*  
 *}*  
 *catch (DirectoryNotFoundException)*  
 *{*  
 *diagnostics.DirectoryExists = false;*  
 *diagnostics.Issues.Add("Upload directory does not exist");*  
 *}*  
  
 *// Check available space*  
 *var driveInfo = new DriveInfo(Path.GetPathRoot(config.UploadPath));*  
 *diagnostics.AvailableSpaceGB = driveInfo.AvailableFreeSpace / (1024 \* 1024 \* 1024);*  
  
 *if (diagnostics.AvailableSpaceGB < 1)*  
 *{*  
 *diagnostics.Issues.Add("Low disk space available");*  
 *}*  
  
 *return diagnostics;*  
 *}*  
*}***

**Common Fixes: - Ensure upload directories exist - Check file system permissions - Verify sufficient disk space - Validate network path accessibility for UNC paths**

***4. Performance Issues***

**Problem: Slow API responses or high resource usage**

**Solution:**

***// Performance monitoring*  
*public class PerformanceDiagnosticsService*  
*{*  
 *public async Task<PerformanceDiagnostics> DiagnosePerformanceAsync()*  
 *{*  
 *var diagnostics = new PerformanceDiagnostics();*  
  
 *// Check database performance*  
 *var stopwatch = Stopwatch.StartNew();*  
 *await \_repository.ExecuteScalarAsync<int>("SELECT 1");*  
 *stopwatch.Stop();*  
  
 *diagnostics.DatabaseResponseTimeMs = stopwatch.ElapsedMilliseconds;*  
  
 *if (diagnostics.DatabaseResponseTimeMs > 1000)*  
 *{*  
 *diagnostics.Issues.Add("Database response time is slow");*  
 *}*  
  
 *// Check memory usage*  
 *var process = Process.GetCurrentProcess();*  
 *diagnostics.MemoryUsageMB = process.WorkingSet64 / (1024 \* 1024);*  
  
 *if (diagnostics.MemoryUsageMB > 512)*  
 *{*  
 *diagnostics.Issues.Add("High memory usage detected");*  
 *}*  
  
 *// Check cache efficiency*  
 *var cacheStats = \_cacheService.GetStatistics();*  
 *diagnostics.CacheHitRatio = cacheStats.HitRatio;*  
  
 *if (diagnostics.CacheHitRatio < 0.7)*  
 *{*  
 *diagnostics.Issues.Add("Low cache hit ratio");*  
 *}*  
  
 *return diagnostics;*  
 *}*  
*}***

**Common Fixes: - Enable query optimization in stored procedures - Implement proper indexing strategy - Review and optimize cache usage - Consider connection pooling adjustments**

***5. Logging and Monitoring Issues***

**Problem: Missing logs or monitoring data**

**Solution:**

***// Verify logging configuration*  
*public class LoggingDiagnosticsService*  
*{*  
 *public LoggingDiagnostics DiagnoseLogging()*  
 *{*  
 *var diagnostics = new LoggingDiagnostics();*  
  
 *// Check log file permissions*  
 *var logPath = \_configuration["Serilog:WriteTo:1:Args:path"];*  
 *if (!string.IsNullOrEmpty(logPath))*  
 *{*  
 *try*  
 *{*  
 *var logDirectory = Path.GetDirectoryName(logPath);*  
 *diagnostics.LogDirectoryExists = Directory.Exists(logDirectory);*  
  
 *if (diagnostics.LogDirectoryExists)*  
 *{*  
 *// Test write permissions*  
 *var testFile = Path.Combine(logDirectory, $"test\_{Guid.NewGuid()}.tmp");*  
 *File.WriteAllText(testFile, "test");*  
 *File.Delete(testFile);*  
  
 *diagnostics.CanWriteToLogDirectory = true;*  
 *}*  
 *}*  
 *catch (Exception ex)*  
 *{*  
 *diagnostics.Issues.Add($"Cannot write to log directory: {ex.Message}");*  
 *}*  
 *}*  
  
 *// Check if Serilog is configured*  
 *diagnostics.SerilogConfigured = Log.Logger != null && Log.Logger.GetType() != typeof(SilentLogger);*  
  
 *return diagnostics;*  
 *}*  
*}***

**Diagnostic Endpoints**

***[ApiController]*  
*[Route("api/v1/diagnostics")]*  
*[Authorize(Roles = "Admin")]*  
*public class DiagnosticsController : ControllerBase*  
*{*  
 *[HttpGet("database")]*  
 *public async Task<ActionResult<DatabaseDiagnostics>> GetDatabaseDiagnostics()*  
 *{*  
 *// Return database connection and performance diagnostics*  
 *}*  
  
 *[HttpGet("performance")]*  
 *public async Task<ActionResult<PerformanceDiagnostics>> GetPerformanceDiagnostics()*  
 *{*  
 *// Return performance metrics and resource usage*  
 *}*  
  
 *[HttpGet("configuration")]*  
 *public ActionResult<ConfigurationDiagnostics> GetConfigurationDiagnostics()*  
 *{*  
 *// Return sanitized configuration validation*  
 *}*  
  
 *[HttpGet("health-detailed")]*  
 *public async Task<ActionResult<DetailedHealthStatus>> GetDetailedHealth()*  
 *{*  
 *// Return comprehensive health status*  
 *}*  
*}***

**Log Analysis**

***Important Log Patterns to Monitor***

***// Performance issues*  
*"Slow request: {Method} {Path} took {ElapsedMilliseconds}ms"*  
  
*// Authentication failures*  
*"Invalid token for {Path}: {Reason}"*  
  
*// Database issues*  
*"Database operation failed: {Operation} - {Error}"*  
  
*// File upload issues*  
*"File upload failed for {FileName}: {Error}"*  
  
*// Business logic errors*  
*"Business rule violation: {Rule} - {Details}"***

***Log Queries for Troubleshooting***

***-- Find slow requests*  
*SELECT \* FROM Logs*   
*WHERE Message LIKE '%Slow request%'*   
*AND Timestamp > DATEADD(hour, -1, GETDATE())*  
  
*-- Authentication failures*  
*SELECT \* FROM Logs*   
*WHERE Message LIKE '%Invalid token%'*   
*AND Level = 'Warning'*  
*AND Timestamp > DATEADD(hour, -1, GETDATE())*  
  
*-- Database errors*  
*SELECT \* FROM Logs*   
*WHERE Message LIKE '%Database%'*   
*AND Level = 'Error'*  
*AND Timestamp > DATEADD(hour, -24, GETDATE())***

**Conclusion**

**This comprehensive documentation provides a complete guide for developing, deploying, and maintaining the SimRMS application. The system follows industry best practices with Clean Architecture, comprehensive security, robust error handling, and extensive monitoring capabilities.**

**Summary of Key Benefits**

**The SimRMS system offers the following advantages:**

1. **Robust Architecture: Clean Architecture ensures maintainability and scalability**
2. **Security First: Comprehensive authentication and authorization mechanisms**
3. **Performance Optimized: Caching strategies and database optimization**
4. **Developer Friendly: Clear patterns and comprehensive documentation**
5. **Production Ready: Health monitoring and diagnostic capabilities**
6. **Quality Assured: Comprehensive testing strategies and validation**

**Development Best Practices Summary**

| **Category** | **Best Practice** | **Implementation** |
| --- | --- | --- |
| **Architecture** | **Clean Architecture** | **Strict layer separation with dependency inversion** |
| **Database** | **Stored Procedures Only** | **No inline SQL, use *IGenericRepository*** |
| **Validation** | **Consolidated Validators** | **Single file per business area with extensions** |
| **Security** | **Token Authentication** | **Custom middleware with external validation** |
| **Performance** | **Caching & Optimization** | **Memory cache with proper invalidation** |
| **Testing** | **Comprehensive Coverage** | **Unit, integration, and API tests** |
| **Monitoring** | **Structured Logging** | **Serilog with health checks** |

**Implementation Checklist for New Developers**

* **Environment Setup: Complete development environment configuration**
* **Code Review: Understand existing code patterns and structure**
* **Database Access: Review stored procedure conventions and usage**
* **Security Model: Understand authentication and authorization flow**
* **Testing Strategy: Review testing approaches and write comprehensive tests**
* **Documentation: Keep documentation updated with any changes**
* **Performance: Monitor and optimize code for performance**
* **Deployment: Understand deployment processes and monitoring**

**Continuous Improvement**

**This documentation should be treated as a living document that evolves with the system:**

1. **Regular Updates: Update documentation with new features and changes**
2. **Community Feedback: Incorporate developer feedback and suggestions**
3. **Best Practice Evolution: Update patterns as technology and practices evolve**
4. **Training Material: Use as foundation for developer onboarding programs**
5. **Knowledge Transfer: Facilitate knowledge sharing across team members**

**Support and Resources**

**For additional support or questions:**

* **Primary Contact: Lead Developer**
* **Documentation Issues: Create issues in project repository**
* **Technical Questions: Development Team Lead**
* **Process Questions: Project Manager**
* **Emergency Support: On-call Development Team**

**Appendices**

**Appendix A: Glossary of Terms**

| **Term** | **Definition** |
| --- | --- |
| **Clean Architecture** | **Software design philosophy that separates concerns into distinct layers** |
| **DTO** | **Data Transfer Object - Simple objects used to transfer data between layers** |
| **Repository Pattern** | **Design pattern that encapsulates data access logic** |
| **Stored Procedure** | **Precompiled SQL code stored in the database** |
| **JWT** | **JSON Web Token - Standard for securely transmitting information** |
| **API Versioning** | **Practice of managing changes to APIs over time** |
| **Health Check** | **Automated test to verify system component status** |
| **Middleware** | **Software that sits between different applications or services** |

**Appendix B: Common Error Codes**

| **Error Code** | **Description** | **Resolution** |
| --- | --- | --- |
| **AUTH001** | **Invalid authentication token** | **Refresh token or re-authenticate** |
| **AUTH002** | **Token expired** | **Obtain new token** |
| **VALID001** | **Validation failure** | **Review request data format** |
| **DB001** | **Database connection error** | **Check database connectivity** |
| **DB002** | **Stored procedure error** | **Review procedure parameters** |
| **FILE001** | **File upload error** | **Check file size and permissions** |
| **PERF001** | **Performance threshold exceeded** | **Review and optimize code** |

**Appendix C: Configuration Templates**

***Development Configuration Template***

***{*  
 *"ConnectionStrings": {*  
 *"DefaultConnection": "Server=localhost;Database=SimRMS\_Dev;Integrated Security=true;"*  
 *},*  
 *"Serilog": {*  
 *"MinimumLevel": { "Default": "Debug" }*  
 *},*  
 *"TokenService": {*  
 *"ValidateUrl": "http://localhost:9092/validate"*  
 *}*  
*}***

***Production Configuration Template***

***{*  
 *"ConnectionStrings": {*  
 *"DefaultConnection": "Server=prod-server;Database=SimRMS\_Prod;User Id=prod\_user;Password=\*\*\*;"*  
 *},*  
 *"Serilog": {*  
 *"MinimumLevel": { "Default": "Information" }*  
 *},*  
 *"TokenService": {*  
 *"ValidateUrl": "https://token-service.company.com/validate"*  
 *}*  
*}***

**Appendix D: Database Schema Overview**

***Core Tables***

* **MstCo: Company master data**
* **MstCoBrch: Company branch information**
* **UsrInfo: User information and access control**
* **UsrLogin: User login and session management**

***Relationship Overview***

***MstCo (1) -----> (N) MstCoBrch*  
*UsrInfo (1) -----> (N) UsrLogin***

**Appendix E: API Quick Reference**

| **Endpoint Category** | **Base Path** | **Authentication Required** |
| --- | --- | --- |
| **Authentication** | ***/api/v1/auth*** | **Partial** |
| **Companies** | ***/api/v1/companies*** | **Yes** |
| **Broker Branches** | ***/api/v1/brokerBranch*** | **Yes** |
| **Users** | ***/api/v1/users*** | **Yes** |
| **Files** | ***/api/v1/files*** | **Yes** |
| **Health** | ***/health*** | **No** |

**Appendix F: Performance Benchmarks**

| **Operation** | **Target Response Time** | **Acceptable Range** |
| --- | --- | --- |
| **Authentication** | **< 200ms** | **< 500ms** |
| **Data Retrieval** | **< 300ms** | **< 1000ms** |
| **Data Creation** | **< 500ms** | **< 2000ms** |
| **File Upload** | **< 2000ms** | **< 5000ms** |
| **Health Check** | **< 100ms** | **< 250ms** |

**Document Revision Log**

**Version 1.0 - August 14, 2025**

* **Initial Release: Complete documentation with all sections**
* **Features Added: Architecture, API reference, development guidelines**
* **Review Status: Ready for technical review**
* **Next Review: November 14, 2025**

**Future Planned Updates**

* **Version 1.1: Addition of advanced deployment scenarios**
* **Version 1.2: Integration with CI/CD pipeline documentation**
* **Version 2.0: Major updates for .NET 9 migration (if applicable)**

**Document Classification and Disclaimer**

**Classification: Internal Use Only  
Confidentiality: This document contains proprietary information and is intended solely for use by authorized personnel of the organization. Distribution outside the organization is prohibited without explicit written approval.**

**Disclaimer: The information contained in this document is provided “as is” without warranty of any kind. The development team reserves the right to make changes to this document without notice.**

**Copyright: © 2025 SimRMS Development Team. All rights reserved.**

**Document Control Number: SIMRMS-DEV-DOC-001  
Version: 1.0  
Issue Date: August 14, 2025  
Review Date: November 14, 2025  
Classification: Internal Use Only**