**SimRMS - Risk Management System**

**Comprehensive Software Documentation**

**Creation/Modification History Record Card**

| **Sl** | **Date** | **Version** | **Description** | **Changed by** |
| --- | --- | --- | --- | --- |
| 01 | 28/07/2025 | 1.0 | Initial system documentation created | Raihan |
|  |  |  |  |  |

**Table of Contents**

1. System Overview
2. Architecture Design
3. Technology Stack
4. Project Structure
5. API Documentation
6. Security Implementation
7. Data Access Layer
8. Development Workflow
9. Deployment Guide
10. Performance & Monitoring
11. Testing Strategy
12. Troubleshooting

**System Overview**

**Purpose**

SimRMS (Simulation Risk Management System) is an enterprise-grade web API designed to manage risk-related operations for financial institutions. The system provides comprehensive user management, authentication, authorization, and risk assessment capabilities through a modern RESTful API architecture.

**Key Features**

* **User Information Management**: Complete CRUD operations for user data
* **Multi-layered Security**: Handshake + Token-based authentication
* **API Versioning**: Support for multiple API versions (v1.0, v2.0)
* **Real-time Monitoring**: Performance tracking and health checks
* **Audit Logging**: Comprehensive activity tracking
* **Data Validation**: Enterprise-grade input validation
* **Caching Strategy**: Optimized performance with intelligent caching
* **Background Services**: Automated maintenance tasks

**Business Domain**

* Financial risk management
* User lifecycle management
* Security compliance
* Audit trail maintenance
* Performance optimization

**Architecture Design**

**Clean Architecture Implementation**

SimRMS follows Clean Architecture principles with clear separation of concerns:

┌─────────────────────────────────────────────────────────────┐

│ Presentation Layer │

│ ┌─────────────────────────────────────────────────────────┐│

│ │ SimRMS.WebAPI ││

│ │ • Controllers (V1, V2) ││

│ │ • Middleware Pipeline ││

│ │ • API Versioning ││

│ │ • Authentication & Authorization ││

│ └─────────────────────────────────────────────────────────┘│

└─────────────────────────────────────────────────────────────┘

┌─────────────────────────────────────────────────────────────┐

│ Application Layer │

│ ┌─────────────────────────────────────────────────────────┐│

│ │ SimRMS.Application ││

│ │ • CQRS Commands & Queries ││

│ │ • MediatR Handlers ││

│ │ • DTOs & Request Models ││

│ │ • Validation Pipeline ││

│ │ • Business Logic Interfaces ││

│ └─────────────────────────────────────────────────────────┘│

└─────────────────────────────────────────────────────────────┘

┌─────────────────────────────────────────────────────────────┐

│ Domain Layer │

│ ┌─────────────────────────────────────────────────────────┐│

│ │ SimRMS.Domain ││

│ │ • Entities (UsrInfo, AuditLog) ││

│ │ • Domain Events ││

│ │ • Business Rules & Validations ││

│ │ • Repository Interfaces ││

│ │ • Domain Exceptions ││

│ └─────────────────────────────────────────────────────────┘│

└─────────────────────────────────────────────────────────────┘

┌─────────────────────────────────────────────────────────────┐

│ Infrastructure Layer │

│ ┌─────────────────────────────────────────────────────────┐│

│ │ SimRMS.Infrastructure ││

│ │ • Repository Implementations ││

│ │ • External Service Clients ││

│ │ • LB.DAL Data Access ││

│ │ • Caching Services ││

│ │ • Background Services ││

│ └─────────────────────────────────────────────────────────┘│

└─────────────────────────────────────────────────────────────┘

┌─────────────────────────────────────────────────────────────┐

│ Shared Layer │

│ ┌─────────────────────────────────────────────────────────┐│

│ │ SimRMS.Shared ││

│ │ • Common Models ││

│ │ • Constants & Enums ││

│ │ • Extension Methods ││

│ │ • Utility Classes ││

│ └─────────────────────────────────────────────────────────┘│

└─────────────────────────────────────────────────────────────┘

**CQRS Pattern Implementation**

The system implements Command Query Responsibility Segregation (CQRS) using MediatR:

**Commands (Write Operations)**

* CreateUsrInfoCommand: Create new user information
* UpdateUsrInfoCommand: Update existing user information
* DeleteUsrInfoCommand: Delete user information

**Queries (Read Operations)**

* GetUsrInfoByIdQuery: Retrieve user by ID
* GetUsrInfosQuery: Get paginated user list with filtering
* GetUsrInfoStatisticsQuery: Get user statistics and analytics

**Key Design Patterns**

1. **Repository Pattern**: Clean data access abstraction
2. **Unit of Work Pattern**: Transaction management
3. **CQRS Pattern**: Separation of read and write operations
4. **Mediator Pattern**: Decoupled request handling
5. **Pipeline Pattern**: Middleware processing
6. **Strategy Pattern**: Authentication mechanisms
7. **Factory Pattern**: Service creation and injection

**Technology Stack**

**Core Framework**

{

"framework": ".NET 8.0",

"language": "C# 12",

"architecture": "Clean Architecture",

"pattern": "CQRS + MediatR"

}

**Key Libraries & Dependencies**

| **Component** | **Technology** | **Version** | **Purpose** |
| --- | --- | --- | --- |
| **Web API** | ASP.NET Core | 8.0 | RESTful API framework |
| **CQRS** | MediatR | 12.2.0 | Command/Query handling |
| **Validation** | FluentValidation | 11.8.0 | Input validation |
| **Mapping** | AutoMapper | 12.0.1 | Object-to-object mapping |
| **Data Access** | LB.DAL.Core.Common | Custom | Enterprise data layer |
| **Database** | Microsoft SQL Server | 2019+ | Primary database |
| **Caching** | Memory Cache | 8.0.1 | In-memory caching |
| **Logging** | Serilog | 3.1.1 | Structured logging |
| **Documentation** | Swagger/OpenAPI | 6.5.0 | API documentation |
| **Rate Limiting** | AspNetCoreRateLimit | 5.0.0 | API rate limiting |
| **Health Checks** | Microsoft.Extensions | 8.0.0 | System monitoring |

**Database Technology**

* **Primary Database**: Microsoft SQL Server
* **Data Access**: Custom LB.DAL.Core.Common.dll
* **Connection Management**: ADO.NET with connection pooling
* **Transaction Support**: Distributed transactions via LB.DAL

**Project Structure**

**Solution Organization**

SimRMS/

├── src/

│ ├── SimRMS.WebAPI/ # Presentation Layer

│ │ ├── Controllers/

│ │ │ └── V1/ # Version 1 Controllers

│ │ │ ├── AuthController.cs

│ │ │ ├── HandshakeController.cs

│ │ │ └── UsrInfoController.cs

│ │ ├── Middleware/ # Custom Middleware

│ │ │ ├── ExceptionHandlingMiddleware.cs

│ │ │ ├── PerformanceMiddleware.cs

│ │ │ ├── RequestLoggingMiddleware.cs

│ │ │ └── SecurityHeadersMiddleware.cs

│ │ ├── Security/ # Authentication & Authorization

│ │ │ ├── CustomAuthorizationHandler.cs

│ │ │ └── TokenAuthenticationMiddleware.cs

│ │ ├── Extensions/ # Service Extensions

│ │ ├── Filters/ # Action Filters

│ │ └── Program.cs # Application Entry Point

│ │

│ ├── SimRMS.Application/ # Application Layer

│ │ ├── Features/ # CQRS Features

│ │ │ └── UsrInfo/

│ │ │ ├── Commands/ # Write Operations

│ │ │ └── Queries/ # Read Operations

│ │ ├── Models/ # Application Models

│ │ │ ├── DTOs/ # Data Transfer Objects

│ │ │ ├── Requests/ # Request Models

│ │ │ └── Auth/ # Authentication Models

│ │ ├── Interfaces/ # Application Interfaces

│ │ ├── Mappings/ # AutoMapper Profiles

│ │ ├── Validators/ # FluentValidation Rules

│ │ └── Common/ # Common Application Logic

│ │

│ ├── SimRMS.Domain/ # Domain Layer

│ │ ├── Entities/ # Domain Entities

│ │ │ ├── UsrInfo.cs

│ │ │ ├── UsrLogin.cs

│ │ │ ├── AuditLog.cs

│ │ │ └── BaseEntity.cs

│ │ ├── Events/ # Domain Events

│ │ ├── Exceptions/ # Domain Exceptions

│ │ ├── Interfaces/ # Domain Interfaces

│ │ │ ├── Repo/ # Repository Interfaces

│ │ │ └── IUnitOfWork.cs

│ │ └── Common/ # Domain Common Types

│ │

│ ├── SimRMS.Infrastructure/ # Infrastructure Layer

│ │ ├── Repositories/ # Repository Implementations

│ │ │ ├── UsrInfoRepository.cs

│ │ │ └── UnitOfWork.cs

│ │ ├── Services/ # External Services

│ │ │ ├── CacheService.cs

│ │ │ ├── ConfigurationService.cs

│ │ │ ├── ExternalTokenService.cs

│ │ │ └── SecurityService.cs

│ │ ├── HealthChecks/ # Health Check Implementations

│ │ ├── BackgroundServices/ # Background Tasks

│ │ └── Data/ # Data Infrastructure

│ │

│ └── SimRMS.Shared/ # Shared Components

│ ├── Models/ # Shared Models

│ ├── Constants/ # Application Constants

│ └── Extensions/ # Extension Methods

│

├── lib/ # External Libraries

│ └── LB.DAL.Core.Common.dll # Custom Data Access Library

│

├── tests/ # Test Projects

├── docs/ # Documentation

└── README.md # Project README

**Layer Responsibilities**

**Presentation Layer (SimRMS.WebAPI)**

* RESTful API endpoints
* Request/response handling
* Authentication & authorization
* API versioning
* Middleware pipeline
* Error handling
* Swagger documentation

**Application Layer (SimRMS.Application)**

* Business use cases
* CQRS command/query handlers
* Data validation
* DTO mapping
* Cross-cutting concerns
* Interface definitions

**Domain Layer (SimRMS.Domain)**

* Business entities
* Domain logic
* Business rules
* Domain events
* Repository contracts
* Domain exceptions

**Infrastructure Layer (SimRMS.Infrastructure)**

* Data persistence
* External service integration
* Caching implementation
* Configuration management
* Background services
* Health monitoring

**API Documentation**

**Base URL Structure**

Production: https://api.simrms.com

Development: https://localhost:7026

Swagger UI: {baseUrl}/swagger

**API Versioning Strategy**

The system supports multiple API versions with flexible version resolution:

# Header-based versioning (Recommended)

GET /api/v1/usrinfo

X-API-Version: 1.0

# Query parameter versioning

GET /api/v1/usrinfo?version=1.0

# URL segment versioning

GET /api/v1/usrinfo

**Authentication Flow**

**1. Handshake Authentication**

POST /api/v1/handshake

Content-Type: application/json

{

"appId": "RMS\_APP\_2025",

"appSecret": "RMS\_SECRET\_2025\_SECURE\_KEY"

}

Response:

{

"success": true,

"data": {

"handshakeToken": "eyJhbGciOiJI...",

"expiresAt": "2025-07-29T04:30:00Z",

"tokenType": "Handshake",

"message": "Handshake successful"

}

}

**2. User Authentication**

POST /api/v1/auth/login

X-Handshake-Token: eyJhbGciOiJI...

Content-Type: application/json

{

"username": "EFTEST01",

"password": "password"

}

Response:

{

"success": true,

"data": {

"userToken": "eyJhbGciOiJSUzI1NiIs...",

"handshakeToken": "eyJhbGciOiJI...",

"user": {

"id": "EFTEST01",

"username": "EFTEST01",

"email": "EFTEST01@lbsbd.com",

"fullName": "EFTEST01",

"roles": ["Admin", "SuperAdmin"]

}

}

}

**Core API Endpoints**

**User Information Management**

**Get Paginated Users**

GET /api/v1/usrinfo?pageNumber=1&pageSize=10&usrStatus=A

X-Handshake-Token: {handshakeToken}

Authorization: Bearer {userToken}

Response:

{

"success": true,

"message": "Retrieved 10 records out of 150 total",

"data": [

{

"usrId": "USER001",

"usrName": "John Doe",

"usrEmail": "john.doe@company.com",

"usrStatus": "A",

"coCode": "LBSL",

"rmsType": "TRADING",

"usrCreationDate": "2025-01-15T10:30:00Z"

}

],

"pagination": {

"totalCount": 150,

"pageNumber": 1,

"pageSize": 10,

"totalPages": 15,

"hasPreviousPage": false,

"hasNextPage": true

}

}

**Get User by ID**

GET /api/v1/usrinfo/USER001

X-Handshake-Token: {handshakeToken}

Authorization: Bearer {userToken}

Response:

{

"success": true,

"message": "User information retrieved successfully",

"data": {

"usrId": "USER001",

"dlrCode": "DLR001",

"coCode": "LBSL",

"usrName": "John Doe",

"usrEmail": "john.doe@company.com",

"usrStatus": "A",

"usrCreationDate": "2025-01-15T10:30:00Z",

"usrLastUpdatedDate": "2025-07-28T04:30:00Z"

}

}

**Create New User**

POST /api/v1/usrinfo

X-Handshake-Token: {handshakeToken}

Authorization: Bearer {userToken}

Content-Type: application/json

{

"usrId": "USER002",

"usrName": "Jane Smith",

"usrEmail": "jane.smith@company.com",

"usrStatus": "A",

"coCode": "LBSL",

"rmsType": "TRADING",

"usrGender": "F",

"usrPhone": "+880123456789"

}

Response:

{

"success": true,

"message": "User information created successfully",

"data": {

"usrId": "USER002",

"usrName": "Jane Smith",

"usrEmail": "jane.smith@company.com",

"usrStatus": "A",

"usrCreationDate": "2025-07-28T04:30:00Z"

}

}

**Update User**

PUT /api/v1/usrinfo/USER002

X-Handshake-Token: {handshakeToken}

Authorization: Bearer {userToken}

Content-Type: application/json

{

"usrName": "Jane Smith Updated",

"usrPhone": "+880987654321",

"usrStatus": "A"

}

Response:

{

"success": true,

"message": "User information updated successfully",

"data": {

"usrId": "USER002",

"usrName": "Jane Smith Updated",

"usrPhone": "+880987654321",

"usrLastUpdatedDate": "2025-07-28T04:35:00Z"

}

}

**Delete User**

DELETE /api/v1/usrinfo/USER002

X-Handshake-Token: {handshakeToken}

Authorization: Bearer {userToken}

Response:

{

"success": true,

"message": "User information deleted successfully",

"data": true

}

**System Statistics (v2.0)**

GET /api/v2/usrinfo/statistics

X-API-Version: 2.0

X-Handshake-Token: {handshakeToken}

Authorization: Bearer {userToken}

Response:

{

"success": true,

"data": {

"totalUsers": 1250,

"activeUsers": 1100,

"suspendedUsers": 100,

"closedUsers": 50,

"rmsTypes": [

{ "rmsType": "TRADING", "count": 800 },

{ "rmsType": "CLEARING", "count": 450 }

],

"companyCodes": [

{ "coCode": "LBSL", "count": 1000 },

{ "coCode": "LBFL", "count": 250 }

]

}

}

**Error Response Format**

All API errors follow a consistent format:

{

"success": false,

"message": "Validation failed",

"data": null,

"errors": [

"UsrId: User ID is required",

"UsrEmail: Invalid email format"

],

"timestamp": "2025-07-28T04:30:00Z",

"traceId": "0HN7JEGHPS2K4:00000001"

}

**HTTP Status Codes**

| **Code** | **Meaning** | **Usage** |
| --- | --- | --- |
| 200 | OK | Successful operation |
| 201 | Created | Resource created successfully |
| 400 | Bad Request | Invalid request data |
| 401 | Unauthorized | Authentication required |
| 403 | Forbidden | Access denied |
| 404 | Not Found | Resource not found |
| 409 | Conflict | Resource already exists |
| 422 | Unprocessable Entity | Validation errors |
| 429 | Too Many Requests | Rate limit exceeded |
| 500 | Internal Server Error | Server error |
| 503 | Service Unavailable | External service down |

**Security Implementation**

**Multi-Layer Security Architecture**

The system implements a comprehensive security model with multiple authentication layers:

┌─────────────────────────────────────────────────────────────┐

│ Client Application │

└─────────────────┬───────────────────────────────────────────┘

│

[Step 1: Handshake]

│

┌─────────────────▼───────────────────────────────────────────┐

│ API Gateway │

│ ┌─────────────────────────────────────────────────────────┐│

│ │ Security Middleware Pipeline ││

│ │ ││

│ │ SecurityHeaders → TokenAuthentication → RateLimit ││

│ │ ↓ ↓ ↓ ││

│ │ CORS Support JWT Validation Throttling ││

│ └─────────────────────────────────────────────────────────┘│

└─────────────────┬───────────────────────────────────────────┘

│

[Step 2: User Login]

│

┌─────────────────▼───────────────────────────────────────────┐

│ External Token Service │

│ • Token Generation & Validation │

│ • User Authentication │

│ • Session Management │

└─────────────────┬───────────────────────────────────────────┘

│

[Step 3: API Access]

│

┌─────────────────▼───────────────────────────────────────────┐

│ Application Layer │

│ • Role-based Authorization │

│ • Permission Checking │

│ • Resource Access Control │

└─────────────────────────────────────────────────────────────┘

**Authentication Flow Details**

**1. Handshake Token Process**

// Handshake token generation

var handshakeToken = GenerateHandshakeToken(appId);

await \_cacheService.SetAsync($"HANDSHAKE\_TOKEN\_{handshakeToken}",

new HandshakeTokenInfo {

AppId = appId,

CreatedAt = DateTime.UtcNow,

ExpiresAt = DateTime.UtcNow.AddHours(24),

IsActive = true

}, TimeSpan.FromHours(24));

**2. User Token Validation**

// Token validation pipeline

var isValidToken = await tokenService.ValidateTokenAsync(userToken);

if (isValidToken) {

var userId = ExtractUserIdFromToken(userToken);

var userSession = await securityService.GetUserSessionAsync(userId);

if (userSession?.IsActive == true) {

// Create claims identity

var claimsIdentity = new ClaimsIdentity(new[] {

new Claim(ClaimTypes.NameIdentifier, userId),

new Claim(ClaimTypes.Name, userSession.UserName),

new Claim(ClaimTypes.Email, userSession.Email)

}, "TokenAuthentication");

// Add roles and permissions

foreach (var role in userSession.Roles)

claimsIdentity.AddClaim(new Claim(ClaimTypes.Role, role));

}

}

**Authorization Policies**

The system uses policy-based authorization with custom requirements:

// Policy definitions

services.AddAuthorization(options => {

options.AddPolicy("ViewUsers", policy =>

policy.Requirements.Add(new CustomAuthorizationRequirement {

Permission = AppConstants.Permissions.ViewUsers

}));

options.AddPolicy("ManageUsers", policy =>

policy.Requirements.Add(new CustomAuthorizationRequirement {

Permission = AppConstants.Permissions.ManageUsers

}));

});

**Security Headers**

Comprehensive security headers are automatically applied:

context.Response.Headers["X-Content-Type-Options"] = "nosniff";

context.Response.Headers["X-Frame-Options"] = "DENY";

context.Response.Headers["X-XSS-Protection"] = "1; mode=block";

context.Response.Headers["Referrer-Policy"] = "strict-origin-when-cross-origin";

context.Response.Headers["Content-Security-Policy"] = "default-src 'self'";

**Rate Limiting Configuration**

{

"IpRateLimiting": {

"EnableEndpointRateLimiting": true,

"StackBlockedRequests": false,

"RealIpHeader": "X-Real-IP",

"ClientIdHeader": "X-ClientId",

"GeneralRules": [

{

"Endpoint": "\*",

"Period": "1m",

"Limit": 100

}

]

}

}

**Security Best Practices Implemented**

1. **Input Validation**: FluentValidation with comprehensive rules
2. **SQL Injection Prevention**: Parameterized queries via LB.DAL
3. **XSS Protection**: Content-Type headers and validation
4. **CSRF Protection**: Anti-forgery tokens where applicable
5. **Rate Limiting**: IP-based throttling
6. **Session Management**: Secure token handling with expiration
7. **Audit Logging**: Comprehensive activity tracking
8. **Error Handling**: No sensitive information exposure

**Data Access Layer**

**LB.DAL Implementation**

The system uses a custom data access layer (LB.DAL.Core.Common.dll) instead of Entity Framework:

// Repository implementation example

public class UsrInfoRepository : IUsrInfoRepository

{

private readonly ILB\_DAL \_dal;

public UsrInfoRepository([FromKeyedServices("DBApplication")] ILB\_DAL dal)

{

\_dal = dal;

\_dal.LB\_GetConnectionAsync().Wait();

}

public async Task<UsrInfo?> GetByUserIdAsync(string usrId, CancellationToken cancellationToken = default)

{

string sSql = @"SELECT UsrID, DlrCode, CoCode, UsrName, UsrEmail, UsrStatus

FROM UsrInfo WHERE UsrID = @UsrID";

List<LB\_DALParam> lstParam = new List<LB\_DALParam>

{

new LB\_DALParam("UsrID", usrId)

};

using DbDataReader dr = await \_dal.LB\_GetDbDataReaderAsync(sSql, lstParam, CommandType.Text);

if (dr.Read())

{

return MapUsrInfoFromReader(dr);

}

return null;

}

}

**Database Schema**

**Primary Tables**

**UsrInfo Table**

CREATE TABLE UsrInfo (

UsrID NVARCHAR(50) PRIMARY KEY,

DlrCode NVARCHAR(10),

CoCode NVARCHAR(10),

CoBrchCode NVARCHAR(10),

UsrName NVARCHAR(100),

UsrType INT,

UsrNICNo NVARCHAR(20),

UsrEmail NVARCHAR(100),

UsrStatus NVARCHAR(1) DEFAULT 'A',

RmsType NVARCHAR(10) NOT NULL,

UsrCreationDate DATETIME2,

UsrLastUpdatedDate DATETIME2,

-- Additional fields...

);

**UsrLogin Table**

CREATE TABLE UsrLogin (

UsrID NVARCHAR(50) PRIMARY KEY,

UsrPwd NVARCHAR(255),

UsrPwdLastChgDate DATETIME2,

UsrLastLoginDate DATETIME2,

UsrTwoFactorAuth INT,

-- Security fields...

);

**AuditLog Table**

CREATE TABLE AuditLog (

Id UNIQUEIDENTIFIER PRIMARY KEY DEFAULT NEWID(),

UserId NVARCHAR(50),

UserName NVARCHAR(100),

TableName NVARCHAR(100),

Action NVARCHAR(50),

OldValues NVARCHAR(MAX),

NewValues NVARCHAR(MAX),

IpAddress NVARCHAR(50),

CreatedAt DATETIME2 DEFAULT GETUTCDATE()

);

**Transaction Management**

// Unit of Work pattern implementation

public async Task<UsrInfoDto> Handle(CreateUsrInfoCommand request, CancellationToken cancellationToken)

{

await \_unitOfWork.EnsureConnectionAsync(cancellationToken);

await \_unitOfWork.BeginTransactionAsync(cancellationToken);

try

{

var createdUser = await \_unitOfWork.UsrInfoRepository.CreateUserAsync(usrInfo, cancellationToken);

await \_unitOfWork.CommitTransactionAsync(cancellationToken);

return \_mapper.Map<UsrInfoDto>(createdUser);

}

catch

{

await \_unitOfWork.RollbackTransactionAsync(cancellationToken);

throw;

}

}

**Connection Management**

{

"ConnectionStrings": {

"DefaultConnection": "Server={server};Database={db};User Id={user};Password={pwd};TrustServerCertificate=true;MultipleActiveResultSets=true;"

},

"LB\_DAL": {

"CommandTimeout": 60,

"IsolationLevel": "ReadCommitted",

"EnableConnectionPooling": true,

"MaxPoolSize": 100,

"MinPoolSize": 5,

"ConnectionTimeout": 30

}

}

**CQRS Mapping Flow: Domain Entity to API Model**

**Complete Data Flow in SimRMS Clean Architecture**

**Overview of Mapping Layers**

In SimRMS, the CQRS pattern with Clean Architecture creates multiple mapping layers to maintain separation of concerns:

┌─────────────────────────────────────────────────────────────┐

│ API Layer (Controllers) │

│ ┌─────────────────────────────────────────────────────────┐│

│ │ API Response Models ││

│ │ • ApiResponse<UsrInfoDto> ││

│ │ • ApiResponse<IEnumerable<UsrInfoDto>> ││

│ │ • Pagination info, success flags ││

│ └─────────────────────────────────────────────────────────┘│

└─────────────────┬───────────────────────────────────────────┘

│ Controller calls MediatR

▼

┌─────────────────────────────────────────────────────────────┐

│ Application Layer (CQRS) │

│ ┌─────────────────────────────────────────────────────────┐│

│ │ DTOs ││

│ │ • UsrInfoDto (for queries/responses) ││

│ │ • CreateUsrInfoRequest (for commands) ││

│ │ • UpdateUsrInfoRequest (for commands) ││

│ └─────────────────────────────────────────────────────────┘│

└─────────────────┬───────────────────────────────────────────┘

│ AutoMapper converts

▼

┌─────────────────────────────────────────────────────────────┐

│ Domain Layer │

│ ┌─────────────────────────────────────────────────────────┐│

│ │ Domain Entities ││

│ │ • UsrInfo (pure business entity) ││

│ │ • Business rules and validations ││

│ │ • Domain events ││

│ └─────────────────────────────────────────────────────────┘│

└─────────────────┬───────────────────────────────────────────┘

│ Repository maps

▼

┌─────────────────────────────────────────────────────────────┐

│ Infrastructure Layer │

│ ┌─────────────────────────────────────────────────────────┐│

│ │ Database Tables ││

│ │ • UsrInfo table (SQL Server) ││

│ │ • Raw data storage ││

│ └─────────────────────────────────────────────────────────┘│

└─────────────────────────────────────────────────────────────┘

**1. Domain Entity (Pure Business Model)**

The domain entity represents the core business concept:

// src/SimRMS.Domain/Entities/UsrInfo.cs

public class UsrInfo

{

[Key] // Primary key

public string UsrId { get; set; } = null!;

public string? DlrCode { get; set; }

public string? CoCode { get; set; }

public string? CoBrchCode { get; set; }

public string? UsrName { get; set; }

public int? UsrType { get; set; }

public string? UsrNicno { get; set; }

public string? UsrPassNo { get; set; }

public string? UsrGender { get; set; }

public DateTime? UsrDob { get; set; }

public string? UsrRace { get; set; }

public string? UsrEmail { get; set; }

public string? UsrAddr { get; set; }

public string? UsrPhone { get; set; }

public string? UsrMobile { get; set; }

public string? UsrFax { get; set; }

public DateTime? UsrLastUpdatedDate { get; set; }

public DateTime? UsrCreationDate { get; set; }

public string? UsrStatus { get; set; }

public string? UsrQualify { get; set; }

public DateTime? UsrRegisterDate { get; set; }

public DateTime? UsrTdrdate { get; set; }

public DateTime? UsrResignDate { get; set; }

public string? ClntCode { get; set; }

public string? UsrLicenseNo { get; set; }

public DateTime? UsrExpiryDate { get; set; }

public string RmsType { get; set; } = null!;

public int? UsrSuperiorId { get; set; }

public int? UsrNotifierId { get; set; }

// Additional business properties...

public int? UsrAssct { get; set; }

public string? UsrAssctPwd { get; set; }

public string? UsrAccessFa { get; set; }

public string? UsrBtxmode { get; set; }

public string? WithoutClntList { get; set; }

public string? Bfesname { get; set; }

public string? ThirdPartyUsrId { get; set; }

public int? GtcexpiryPeriod { get; set; }

public string? UsrGtdmode { get; set; }

public int? MarketDepth { get; set; }

public DateTime? MktDepthStartDate { get; set; }

public DateTime? MktDepthEndDate { get; set; }

public int? CrOrderDealer { get; set; }

public string? Category { get; set; }

public string? UsrChannel { get; set; }

public string? Pid { get; set; }

public string? PidRms { get; set; }

public string? Pidflag { get; set; }

public string? PidflagRms { get; set; }

public string? ChannelUpdFlag { get; set; }

public DateTime? MimosMigrateDt { get; set; }

public DateTime? MimosMigrateDtRms { get; set; }

public string? OriUsrEmail { get; set; }

}

**Purpose**: Pure business entity with all domain-specific properties, no API concerns.

**2. Application DTOs (API Contract Models)**

**Response DTO (Query Results)**

// src/SimRMS.Application/Models/DTOs/UsrInfoDto.cs

public class UsrInfoDto

{

public string UsrId { get; set; } = null!;

public string? DlrCode { get; set; }

public string? CoCode { get; set; }

public string? CoBrchCode { get; set; }

public string? UsrName { get; set; }

public int? UsrType { get; set; }

public string? UsrNicno { get; set; }

public string? UsrPassNo { get; set; }

public string? UsrGender { get; set; }

public DateTime? UsrDob { get; set; }

public string? UsrRace { get; set; }

public string? UsrEmail { get; set; }

public string? UsrAddr { get; set; }

public string? UsrPhone { get; set; }

public string? UsrMobile { get; set; }

public string? UsrFax { get; set; }

public DateTime? UsrLastUpdatedDate { get; set; }

public DateTime? UsrCreationDate { get; set; }

public string? UsrStatus { get; set; }

public string? UsrQualify { get; set; }

public DateTime? UsrRegisterDate { get; set; }

public DateTime? UsrTdrdate { get; set; }

public DateTime? UsrResignDate { get; set; }

public string? ClntCode { get; set; }

public string? UsrLicenseNo { get; set; }

public DateTime? UsrExpiryDate { get; set; }

public string RmsType { get; set; } = null!;

public int? UsrSuperiorId { get; set; }

public int? UsrNotifierId { get; set; }

// All other properties that should be exposed to API...

// (Same as domain entity in this case, but could be different)

}

**Request Models (Command Input)**

// src/SimRMS.Application/Models/Requests/CreateUsrInfoRequest.cs

public class CreateUsrInfoRequest

{

public string UsrId { get; set; } = null!;

public string? DlrCode { get; set; }

public string? CoCode { get; set; }

public string? CoBrchCode { get; set; }

public string? UsrName { get; set; }

public int? UsrType { get; set; }

public string? UsrNicno { get; set; }

public string? UsrPassNo { get; set; }

public string? UsrGender { get; set; }

public DateTime? UsrDob { get; set; }

public string? UsrRace { get; set; }

public string? UsrEmail { get; set; }

public string? UsrAddr { get; set; }

public string? UsrPhone { get; set; }

public string? UsrMobile { get; set; }

public string? UsrFax { get; set; }

public string? UsrStatus { get; set; }

public string? UsrQualify { get; set; }

public DateTime? UsrRegisterDate { get; set; }

public DateTime? UsrTdrdate { get; set; }

public DateTime? UsrResignDate { get; set; }

public string? ClntCode { get; set; }

public string? UsrLicenseNo { get; set; }

public DateTime? UsrExpiryDate { get; set; }

public string RmsType { get; set; } = null!;

public int? UsrSuperiorId { get; set; }

public int? UsrNotifierId { get; set; }

public string? Category { get; set; }

public string? UsrChannel { get; set; }

public string? Pid { get; set; }

public string? PidRms { get; set; }

}

// src/SimRMS.Application/Models/Requests/UpdateUsrInfoRequest.cs

public class UpdateUsrInfoRequest

{

// Only fields that can be updated (no UsrId, no creation dates)

public string? DlrCode { get; set; }

public string? CoCode { get; set; }

public string? CoBrchCode { get; set; }

public string? UsrName { get; set; }

public int? UsrType { get; set; }

public string? UsrNicno { get; set; }

public string? UsrPassNo { get; set; }

public string? UsrGender { get; set; }

public DateTime? UsrDob { get; set; }

public string? UsrRace { get; set; }

public string? UsrEmail { get; set; }

public string? UsrAddr { get; set; }

public string? UsrPhone { get; set; }

public string? UsrMobile { get; set; }

public string? UsrFax { get; set; }

public string? UsrStatus { get; set; }

public string? UsrQualify { get; set; }

public DateTime? UsrRegisterDate { get; set; }

public DateTime? UsrTdrdate { get; set; }

public DateTime? UsrResignDate { get; set; }

public string? ClntCode { get; set; }

public string? UsrLicenseNo { get; set; }

public DateTime? UsrExpiryDate { get; set; }

public string? RmsType { get; set; }

public int? UsrSuperiorId { get; set; }

public int? UsrNotifierId { get; set; }

public string? Category { get; set; }

public string? UsrChannel { get; set; }

public string? Pid { get; set; }

public string? PidRms { get; set; }

}

**Purpose**: API contract models that define exactly what data flows in/out of the API.

**3. AutoMapper Configuration**

The mapping between layers is handled by AutoMapper:

// src/SimRMS.Application/Mappings/MappingProfile.cs

public class MappingProfile : Profile

{

public MappingProfile()

{

// Bidirectional mapping between Domain Entity and DTO

CreateMap<UsrInfo, UsrInfoDto>().ReverseMap();

// Request to Domain Entity mapping (for CREATE)

CreateMap<CreateUsrInfoRequest, UsrInfo>()

.ForMember(dest => dest.UsrCreationDate, opt => opt.MapFrom(src => DateTime.UtcNow))

.ForMember(dest => dest.UsrLastUpdatedDate, opt => opt.MapFrom(src => DateTime.UtcNow));

// Request to Domain Entity mapping (for UPDATE)

CreateMap<UpdateUsrInfoRequest, UsrInfo>()

.ForMember(dest => dest.UsrId, opt => opt.Ignore()) // Don't update primary key

.ForMember(dest => dest.UsrCreationDate, opt => opt.Ignore()) // Don't update creation date

.ForMember(dest => dest.UsrLastUpdatedDate, opt => opt.MapFrom(src => DateTime.UtcNow))

.ForAllMembers(opt => opt.Condition((src, dest, srcMember) => srcMember != null)); // Only map non-null values

}

}

**Purpose**: Centralized mapping configuration with business rules (timestamps, conditional mapping, etc.).

**4. CQRS Command/Query Flow**

**Query Flow (Read Operations)**

// src/SimRMS.Application/Features/UsrInfo/Queries/GetUsrInfoByIdQuery.cs

// 1. Query Definition

public class GetUsrInfoByIdQuery : IRequest<UsrInfoDto>

{

public string UsrId { get; set; } = null!;

}

// 2. Query Handler

public class GetUsrInfoByIdQueryHandler : IRequestHandler<GetUsrInfoByIdQuery, UsrInfoDto>

{

private readonly IUnitOfWork \_unitOfWork;

private readonly IMapper \_mapper;

private readonly ILogger<GetUsrInfoByIdQueryHandler> \_logger;

public GetUsrInfoByIdQueryHandler(

IUnitOfWork unitOfWork,

IMapper mapper,

ILogger<GetUsrInfoByIdQueryHandler> logger)

{

\_unitOfWork = unitOfWork;

\_mapper = mapper;

\_logger = logger;

}

public async Task<UsrInfoDto> Handle(GetUsrInfoByIdQuery request, CancellationToken cancellationToken)

{

\_logger.LogInformation("Getting UsrInfo by UsrId: {UsrId}", request.UsrId);

try

{

await \_unitOfWork.EnsureConnectionAsync(cancellationToken);

// 3. Repository returns Domain Entity

var usrInfo = await \_unitOfWork.UsrInfoRepository.GetByUserIdAsync(request.UsrId.Trim(), cancellationToken);

if (usrInfo == null)

{

throw new NotFoundException(nameof(UsrInfo), request.UsrId);

}

// 4. AutoMapper converts Domain Entity -> DTO

var result = \_mapper.Map<UsrInfoDto>(usrInfo);

\_logger.LogInformation("Successfully retrieved UsrInfo for UsrId: {UsrId}", request.UsrId);

return result;

}

catch (Exception ex) when (!(ex is NotFoundException || ex is ArgumentException))

{

\_logger.LogError(ex, "Error getting UsrInfo by UsrId: {UsrId}", request.UsrId);

throw;

}

}

}

**Command Flow (Write Operations)**

// src/SimRMS.Application/Features/UsrInfo/Commands/CreateUsrInfoCommand.cs

// 1. Command Definition

public class CreateUsrInfoCommand : IRequest<UsrInfoDto>

{

public CreateUsrInfoRequest Request { get; set; } = new();

}

// 2. Command Handler

public class CreateUsrInfoCommandHandler : IRequestHandler<CreateUsrInfoCommand, UsrInfoDto>

{

private readonly IUnitOfWork \_unitOfWork;

private readonly IMapper \_mapper;

private readonly ICurrentUserService \_currentUserService;

private readonly ILogger<CreateUsrInfoCommandHandler> \_logger;

public CreateUsrInfoCommandHandler(

IUnitOfWork unitOfWork,

IMapper mapper,

ICurrentUserService currentUserService,

ILogger<CreateUsrInfoCommandHandler> logger)

{

\_unitOfWork = unitOfWork;

\_mapper = mapper;

\_currentUserService = currentUserService;

\_logger = logger;

}

public async Task<UsrInfoDto> Handle(CreateUsrInfoCommand request, CancellationToken cancellationToken)

{

\_logger.LogInformation("Creating new UsrInfo: {UsrId}", request.Request.UsrId);

try

{

await \_unitOfWork.EnsureConnectionAsync(cancellationToken);

// Business validation

var existingUser = await \_unitOfWork.UsrInfoRepository.GetByUserIdAsync(request.Request.UsrId, cancellationToken);

if (existingUser != null)

{

throw new DomainException($"User with ID '{request.Request.UsrId}' already exists");

}

// 3. AutoMapper converts Request -> Domain Entity

var usrInfo = \_mapper.Map<SimRMS.Domain.Entities.UsrInfo>(request.Request);

await \_unitOfWork.BeginTransactionAsync(cancellationToken);

try

{

// 4. Repository saves Domain Entity

var createdUser = await \_unitOfWork.UsrInfoRepository.CreateUserAsync(usrInfo, cancellationToken);

await \_unitOfWork.CommitTransactionAsync(cancellationToken);

// 5. AutoMapper converts Domain Entity -> DTO for response

var result = \_mapper.Map<UsrInfoDto>(createdUser);

\_logger.LogInformation("Successfully created UsrInfo: {UsrId}", request.Request.UsrId);

return result;

}

catch

{

await \_unitOfWork.RollbackTransactionAsync(cancellationToken);

throw;

}

}

catch (Exception ex)

{

\_logger.LogError(ex, "Error creating UsrInfo: {UsrId}", request.Request.UsrId);

throw;

}

}

}

**5. Controller Integration (API Layer)**

// src/SimRMS.WebAPI/Controllers/V1/UsrInfoController.cs

[Route("api/v{version:apiVersion}/[controller]")]

[ApiController]

[ApiVersion("1.0")]

[Authorize]

public class UsrInfoController : BaseController

{

private readonly IMediator \_mediator;

private readonly ILogger<UsrInfoController> \_logger;

public UsrInfoController(

IMediator mediator,

IConfigurationService configurationService,

ILogger<UsrInfoController> logger)

: base(configurationService)

{

\_mediator = mediator;

\_logger = logger;

}

// GET: api/v1/usrinfo/{usrId}

[HttpGet("{usrId}")]

[MapToApiVersion("1.0")]

[Authorize(Policy = "ViewUsers")]

public async Task<ActionResult<ApiResponse<UsrInfoDto>>> GetUsrInfo(string usrId)

{

if (string.IsNullOrWhiteSpace(usrId))

{

return BadRequest<UsrInfoDto>("User ID is required");

}

\_logger.LogInformation("Getting UsrInfo by ID: {UsrId}", usrId);

// 6. MediatR sends query to handler

var result = await \_mediator.Send(new GetUsrInfoByIdQuery { UsrId = usrId });

// 7. BaseController wraps DTO in ApiResponse

return Ok(result, "User information retrieved successfully");

}

// POST: api/v1/usrinfo

[HttpPost]

[MapToApiVersion("1.0")]

[Authorize(Policy = "ManageUsers")]

public async Task<ActionResult<ApiResponse<UsrInfoDto>>> CreateUsrInfo([FromBody] CreateUsrInfoRequest request)

{

if (request == null)

{

return BadRequest<UsrInfoDto>("Request body is required");

}

\_logger.LogInformation("Creating new UsrInfo: {UsrId}", request.UsrId);

// 6. MediatR sends command to handler

var result = await \_mediator.Send(new CreateUsrInfoCommand { Request = request });

// 7. BaseController wraps DTO in ApiResponse

return Ok(result, "User information created successfully");

}

// PUT: api/v1/usrinfo/{usrId}

[HttpPut("{usrId}")]

[MapToApiVersion("1.0")]

[Authorize(Policy = "ManageUsers")]

public async Task<ActionResult<ApiResponse<UsrInfoDto>>> UpdateUsrInfo(string usrId, [FromBody] UpdateUsrInfoRequest request)

{

if (string.IsNullOrWhiteSpace(usrId) || request == null)

{

return BadRequest<UsrInfoDto>("User ID and request body are required");

}

\_logger.LogInformation("Updating UsrInfo: {UsrId}", usrId);

// 6. MediatR sends command to handler

var result = await \_mediator.Send(new UpdateUsrInfoCommand { UsrId = usrId, Request = request });

// 7. BaseController wraps DTO in ApiResponse

return Ok(result, "User information updated successfully");

}

}

**6. Complete Flow Examples**

**Example 1: Query Flow (GET User)**

1. HTTP Request:

GET /api/v1/usrinfo/USER001

2. Controller:

UsrInfoController.GetUsrInfo("USER001")

3. MediatR Command:

GetUsrInfoByIdQuery { UsrId = "USER001" }

4. Handler:

GetUsrInfoByIdQueryHandler.Handle()

5. Repository Call:

\_unitOfWork.UsrInfoRepository.GetByUserIdAsync("USER001")

6. Database Query:

SELECT \* FROM UsrInfo WHERE UsrID = @UsrID

7. Domain Entity Created:

UsrInfo { UsrId = "USER001", UsrName = "John Doe", ... }

8. AutoMapper Conversion:

\_mapper.Map<UsrInfoDto>(usrInfo)

9. DTO Result:

UsrInfoDto { UsrId = "USER001", UsrName = "John Doe", ... }

10. API Response Wrapping:

ApiResponse<UsrInfoDto> {

Success = true,

Data = UsrInfoDto { ... },

Message = "User information retrieved successfully"

}

11. JSON Response:

{

"success": true,

"message": "User information retrieved successfully",

"data": {

"usrId": "USER001",

"usrName": "John Doe",

"usrEmail": "john@company.com",

...

}

}

**Example 2: Command Flow (CREATE User)**

1. HTTP Request:

POST /api/v1/usrinfo

{

"usrId": "USER002",

"usrName": "Jane Smith",

"usrEmail": "jane@company.com",

"rmsType": "TRADING"

}

2. Model Binding:

CreateUsrInfoRequest { UsrId = "USER002", UsrName = "Jane Smith", ... }

3. Controller:

UsrInfoController.CreateUsrInfo(request)

4. MediatR Command:

CreateUsrInfoCommand { Request = CreateUsrInfoRequest { ... } }

5. Handler:

CreateUsrInfoCommandHandler.Handle()

6. AutoMapper: Request -> Domain Entity

\_mapper.Map<UsrInfo>(request.Request)

UsrInfo {

UsrId = "USER002",

UsrName = "Jane Smith",

UsrCreationDate = DateTime.UtcNow, // Auto-set by mapping

UsrLastUpdatedDate = DateTime.UtcNow

}

7. Repository Call:

\_unitOfWork.UsrInfoRepository.CreateUserAsync(usrInfo)

8. Database Insert:

INSERT INTO UsrInfo (UsrID, UsrName, UsrEmail, ...) VALUES (...)

9. AutoMapper: Domain Entity -> DTO

\_mapper.Map<UsrInfoDto>(createdUser)

10. DTO Result:

UsrInfoDto { UsrId = "USER002", UsrName = "Jane Smith", ... }

11. API Response:

{

"success": true,

"message": "User information created successfully",

"data": {

"usrId": "USER002",

"usrName": "Jane Smith",

"usrCreationDate": "2025-07-28T10:30:00Z",

...

}

}

**7. Key Benefits of This Mapping Architecture**

**1. Separation of Concerns**

* **Domain Entity**: Pure business logic, no API concerns
* **DTOs**: API contract, versioning, client-specific data
* **Request Models**: Input validation, command-specific fields

**2. API Evolution**

// You can evolve APIs without changing domain

public class UsrInfoDtoV2 // New version

{

public string UsrId { get; set; }

public string UsrName { get; set; }

public string DisplayName { get; set; } // New computed field

public UserStatus Status { get; set; } // Enum instead of string

// Domain entity stays the same!

}

**3. Validation Separation**

// API validation (CreateUsrInfoRequestValidator)

RuleFor(x => x.UsrId)

.NotEmpty().WithMessage("User ID is required")

.MaximumLength(50).WithMessage("User ID must not exceed 50 characters")

.Matches("^[A-Z0-9]+$").WithMessage("User ID must contain only uppercase letters and numbers");

// Domain validation would be different (business rules)

**4. Security & Data Filtering**

// You can exclude sensitive fields from DTOs

public class UsrInfoPublicDto // For external APIs

{

public string UsrId { get; set; }

public string UsrName { get; set; }

public string UsrEmail { get; set; }

// No password, sensitive data, internal fields

}

**5. Testing Benefits**

// Test mapping in isolation

[TestMethod]

public void Should\_Map\_CreateRequest\_To\_Domain\_Entity()

{

var request = new CreateUsrInfoRequest

{

UsrId = "TEST001",

UsrName = "Test User"

};

var entity = \_mapper.Map<UsrInfo>(request);

Assert.AreEqual("TEST001", entity.UsrId);

Assert.AreEqual("Test User", entity.UsrName);

Assert.IsTrue(entity.UsrCreationDate > DateTime.UtcNow.AddMinutes(-1));

}

**Summary**

The CQRS mapping in SimRMS follows this pattern:

1. **API Request** → **Request Model** (Model binding)
2. **Request Model** → **Domain Entity** (AutoMapper with business rules)
3. **Domain Entity** → **Database** (Repository with LB.DAL)
4. **Database** → **Domain Entity** (Repository mapping)
5. **Domain Entity** → **DTO** (AutoMapper)
6. **DTO** → **API Response** (Controller wrapping)

This architecture provides clean separation, testability, and API evolution capabilities while maintaining strong typing throughout the entire flow.

**Development Workflow**

**Local Development Setup**

**Prerequisites**

* .NET 8.0 SDK or later
* Microsoft SQL Server 2019 or later
* Visual Studio 2022 or VS Code
* Git for version control

**Environment Setup**

1. **Clone Repository**

git clone https://github.com/company/simrms.git

cd simrms

1. **Restore Dependencies**

dotnet restore

1. **Configure Database**

// appsettings.Development.json

{

"ConnectionStrings": {

"DefaultConnection": "Server=localhost;Database=SimRMS\_Dev;Integrated Security=true;TrustServerCertificate=true;"

},

"TokenService": {

"GenerateUrl": "http://localhost:9092/token",

"ValidateUrl": "http://localhost:9092/validate"

}

}

1. **Run Application**

cd src/SimRMS.WebAPI

dotnet run

1. **Access Swagger UI**

https://localhost:7026/swagger

**Development Standards**

**Code Style Guidelines**

* **Naming Conventions**: PascalCase for public members, camelCase for private
* **File Organization**: One class per file, logical folder structure
* **Comments**: XML documentation for public APIs
* **Async/Await**: All I/O operations must be asynchronous
* **Error Handling**: Use specific exception types, never catch generic Exception

**Git Workflow**

# Feature development

git checkout -b feature/user-management-enhancement

git add .

git commit -m "feat: add user profile picture support"

git push origin feature/user-management-enhancement

# Create pull request for code review

**Code Review Checklist**

* [ ] Clean Architecture compliance
* [ ] SOLID principles followed
* [ ] Proper error handling
* [ ] Unit tests included
* [ ] Performance considerations
* [ ] Security implications reviewed
* [ ] API documentation updated

**Debugging & Troubleshooting**

**Logging Configuration**

// Structured logging with Serilog

Log.Information("Processing {RequestType} for user {UserId}",

typeof(CreateUsrInfoCommand).Name, request.UsrId);

Log.Error(ex, "Database operation failed for user {UserId}", userId);

**Health Check Endpoints**

GET /health

Response: {"status": "Healthy", "totalDuration": "00:00:00.0234567"}

GET /health/database

Response: {"status": "Healthy", "description": "LB.DAL database connection is healthy"}

**Performance Monitoring**

// Performance middleware automatically tracks:

// - Request duration

// - Slow request warnings (>2 seconds)

// - Response times in headers

X-Response-Time: 145ms

**Testing Strategy**

**Unit Testing Structure**

[TestClass]

public class CreateUsrInfoCommandHandlerTests

{

private readonly Mock<IUnitOfWork> \_mockUnitOfWork;

private readonly Mock<IMapper> \_mockMapper;

private readonly CreateUsrInfoCommandHandler \_handler;

[TestMethod]

public async Task Handle\_ValidRequest\_ReturnsUsrInfoDto()

{

// Arrange

var command = new CreateUsrInfoCommand { /\* test data \*/ };

// Act

var result = await \_handler.Handle(command, CancellationToken.None);

// Assert

Assert.IsNotNull(result);

Assert.AreEqual("USER001", result.UsrId);

}

}

**Integration Testing**

[TestClass]

public class UsrInfoControllerIntegrationTests

{

private readonly WebApplicationFactory<Program> \_factory;

[TestMethod]

public async Task GetUsrInfo\_ValidId\_ReturnsUser()

{

var client = \_factory.CreateClient();

var response = await client.GetAsync("/api/v1/usrinfo/USER001");

response.EnsureSuccessStatusCode();

var content = await response.Content.ReadAsStringAsync();

var result = JsonSerializer.Deserialize<ApiResponse<UsrInfoDto>>(content);

Assert.IsTrue(result.Success);

Assert.AreEqual("USER001", result.Data.UsrId);

}

}

**Deployment Guide**

**Environment Configuration**

**Production Environment Variables**

# Database

CONNECTIONSTRING\_DEFAULTCONNECTION="Server=prod-db;Database=SimRMS;User Id=rms\_app;Password=\*\*\*"

# Token Service

TOKENSERVICE\_GENERATEURL="https://tokenservice.prod.company.com/token"

TOKENSERVICE\_VALIDATEURL="https://tokenservice.prod.company.com/validate"

TOKENSERVICE\_LOGINID="prod\_service\_account"

TOKENSERVICE\_PASSWORD="\*\*\*"

# Security

APPCREDENTIALS\_APPID="RMS\_PROD\_2025"

APPCREDENTIALS\_APPSECRET="\*\*\*"

# Logging

SERILOG\_MINIMUMLEVEL\_DEFAULT="Warning"

**Docker Deployment**

**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 ["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/"]

COPY ["lib/LB.DAL.Core.Common.dll", "lib/"]

RUN dotnet restore "src/SimRMS.WebAPI/SimRMS.WebAPI.csproj"

COPY . .

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

FROM base AS final

WORKDIR /app

COPY --from=publish /app/publish .

ENTRYPOINT ["dotnet", "SimRMS.WebAPI.dll"]

**docker-compose.yml**

version: '3.8'

services:

simrms-api:

build: .

ports:

- "8080:80"

- "8443:443"

environment:

- ASPNETCORE\_ENVIRONMENT=Production

- CONNECTIONSTRING\_DEFAULTCONNECTION=${DB\_CONNECTION\_STRING}

- TOKENSERVICE\_GENERATEURL=${TOKEN\_SERVICE\_URL}

volumes:

- ./logs:/app/logs

depends\_on:

- sqlserver

sqlserver:

image: mcr.microsoft.com/mssql/server:2019-latest

environment:

- SA\_PASSWORD=${SA\_PASSWORD}

- ACCEPT\_EULA=Y

ports:

- "1433:1433"

volumes:

- sqlserver\_data:/var/opt/mssql

volumes:

sqlserver\_data:

**Kubernetes Deployment**

**deployment.yaml**

apiVersion: apps/v1

kind: Deployment

metadata:

name: simrms-api

spec:

replicas: 3

selector:

matchLabels:

app: simrms-api

template:

metadata:

labels:

app: simrms-api

spec:

containers:

- name: simrms-api

image: simrms:latest

ports:

- containerPort: 80

env:

- name: ASPNETCORE\_ENVIRONMENT

value: "Production"

- name: CONNECTIONSTRING\_DEFAULTCONNECTION

valueFrom:

secretKeyRef:

name: simrms-secrets

key: database-connection

resources:

requests:

memory: "256Mi"

cpu: "250m"

limits:

memory: "512Mi"

cpu: "500m"

livenessProbe:

httpGet:

path: /health

port: 80

initialDelaySeconds: 30

periodSeconds: 10

readinessProbe:

httpGet:

path: /health

port: 80

initialDelaySeconds: 5

periodSeconds: 5

**Monitoring & Observability**

**Application Insights Integration**

{

"ApplicationInsights": {

"ConnectionString": "InstrumentationKey=\*\*\*",

"EnableSqlCommandTextInstrumentation": true,

"EnablePerformanceCountersCollection": true

}

}

**Custom Metrics**

// Custom telemetry tracking

telemetryClient.TrackEvent("UserCreated", new Dictionary<string, string>

{

{"UserId", userId},

{"Company", coCode},

{"Channel", channel}

});

telemetryClient.TrackMetric("ActiveUsers", activeUserCount);

**Performance & Monitoring**

**Performance Optimization Strategies**

**1. Caching Implementation**

// Multi-level caching strategy

public async Task<T> GetOrSetAsync<T>(string key, Func<Task<T>> factory, TimeSpan? expiration = null)

{

var cachedValue = await GetAsync<T>(key);

if (cachedValue != null) return cachedValue;

// Cache stampede prevention

var semaphore = \_locks.GetOrAdd(key, \_ => new SemaphoreSlim(1, 1));

await semaphore.WaitAsync();

try

{

cachedValue = await GetAsync<T>(key);

if (cachedValue != null) return cachedValue;

var value = await factory();

await SetAsync(key, value, expiration);

return value;

}

finally

{

semaphore.Release();

}

}

**2. Database Optimization**

* **Connection Pooling**: Configured for optimal throughput
* **Parameterized Queries**: All queries use parameters for performance and security
* **Pagination**: Large datasets use efficient OFFSET/FETCH pagination
* **Indexing Strategy**: Optimized indexes on frequently queried columns

**3. Async/Await Best Practices**

// Proper async implementation

public async Task<PagedResult<UsrInfo>> GetUsersPagedAsync(int pageNumber, int pageSize,

CancellationToken cancellationToken = default)

{

// Non-blocking I/O operations

var totalCountTask = GetTotalCountAsync(cancellationToken);

var dataTask = GetPagedDataAsync(pageNumber, pageSize, cancellationToken);

await Task.WhenAll(totalCountTask, dataTask);

return new PagedResult<UsrInfo>

{

Data = await dataTask,

TotalCount = await totalCountTask,

PageNumber = pageNumber,

PageSize = pageSize

};

}

**Monitoring Dashboard**

**Key Performance Indicators (KPIs)**

| **Metric** | **Target** | **Alert Threshold** |
| --- | --- | --- |
| **Response Time** | < 200ms | > 1000ms |
| **Throughput** | > 1000 RPS | < 500 RPS |
| **Error Rate** | < 0.1% | > 1% |
| **Database Connection Pool** | < 80% | > 90% |
| **Memory Usage** | < 70% | > 85% |
| **CPU Usage** | < 60% | > 80% |

**Health Check Implementation**

public class LBDALHealthCheck : IHealthCheck

{

public async Task<HealthCheckResult> CheckHealthAsync(HealthCheckContext context,

CancellationToken cancellationToken = default)

{

try

{

await \_dal.LB\_GetConnectionAsync();

var result = await \_dal.LB\_ExecuteScalarAsync("SELECT 1", null, CommandType.Text);

return result?.ToString() == "1"

? HealthCheckResult.Healthy("Database connection is healthy")

: HealthCheckResult.Degraded("Database returned unexpected result");

}

catch (Exception ex)

{

return HealthCheckResult.Unhealthy("Database connection failed", ex);

}

}

}

**Background Services**

**Cache Cleanup Service**

public class CacheCleanupService : BackgroundService

{

protected override async Task ExecuteAsync(CancellationToken stoppingToken)

{

while (!stoppingToken.IsCancellationRequested)

{

try

{

await CleanupExpiredCaches();

await Task.Delay(TimeSpan.FromMinutes(30), stoppingToken);

}

catch (Exception ex)

{

\_logger.LogError(ex, "Error during cache cleanup");

}

}

}

private async Task CleanupExpiredCaches()

{

await \_cacheService.RemovePatternAsync("TOKEN\_VALIDATION\_");

await \_cacheService.RemovePatternAsync("HANDSHAKE\_TOKEN\_");

await \_cacheService.RemovePatternAsync($"{AppConstants.UserSessionCacheKey}");

}

}

**Testing Strategy**

**Testing Pyramid**

┌─────────────────────┐

│ E2E Tests │ ← Few, High Value

│ (Integration) │

└─────────────────────┘

┌─────────────────────────────┐

│ Integration Tests │ ← Some, Key Flows

│ (API + Database) │

└─────────────────────────────┘

┌───────────────────────────────────┐

│ Unit Tests │ ← Many, Fast

│ (Business Logic + Handlers) │

└───────────────────────────────────┘

**Unit Testing Examples**

**Command Handler Testing**

[TestClass]

public class CreateUsrInfoCommandHandlerTests

{

private readonly Mock<IUnitOfWork> \_mockUnitOfWork;

private readonly Mock<IUsrInfoRepository> \_mockRepository;

private readonly Mock<IMapper> \_mockMapper;

private readonly Mock<ICurrentUserService> \_mockCurrentUser;

private readonly CreateUsrInfoCommandHandler \_handler;

[TestInitialize]

public void Setup()

{

\_mockUnitOfWork = new Mock<IUnitOfWork>();

\_mockRepository = new Mock<IUsrInfoRepository>();

\_mockMapper = new Mock<IMapper>();

\_mockCurrentUser = new Mock<ICurrentUserService>();

\_mockUnitOfWork.Setup(x => x.UsrInfoRepository).Returns(\_mockRepository.Object);

\_handler = new CreateUsrInfoCommandHandler(

\_mockUnitOfWork.Object,

\_mockMapper.Object,

\_mockCurrentUser.Object,

Mock.Of<ILogger<CreateUsrInfoCommandHandler>>());

}

[TestMethod]

public async Task Handle\_ValidRequest\_CreatesUserSuccessfully()

{

// Arrange

var request = new CreateUsrInfoRequest

{

UsrId = "TEST001",

UsrName = "Test User",

UsrEmail = "test@company.com",

RmsType = "TRADING"

};

var command = new CreateUsrInfoCommand { Request = request };

var usrInfo = new UsrInfo { UsrId = "TEST001", UsrName = "Test User" };

var usrInfoDto = new UsrInfoDto { UsrId = "TEST001", UsrName = "Test User" };

\_mockRepository.Setup(x => x.GetByUserIdAsync("TEST001", It.IsAny<CancellationToken>()))

.ReturnsAsync((UsrInfo)null);

\_mockRepository.Setup(x => x.ExistsByEmailAsync("test@company.com", null, It.IsAny<CancellationToken>()))

.ReturnsAsync(false);

\_mockMapper.Setup(x => x.Map<UsrInfo>(request)).Returns(usrInfo);

\_mockRepository.Setup(x => x.CreateUserAsync(usrInfo, It.IsAny<CancellationToken>()))

.ReturnsAsync(usrInfo);

\_mockMapper.Setup(x => x.Map<UsrInfoDto>(usrInfo)).Returns(usrInfoDto);

// Act

var result = await \_handler.Handle(command, CancellationToken.None);

// Assert

Assert.IsNotNull(result);

Assert.AreEqual("TEST001", result.UsrId);

Assert.AreEqual("Test User", result.UsrName);

\_mockUnitOfWork.Verify(x => x.BeginTransactionAsync(It.IsAny<CancellationToken>()), Times.Once);

\_mockUnitOfWork.Verify(x => x.CommitTransactionAsync(It.IsAny<CancellationToken>()), Times.Once);

\_mockRepository.Verify(x => x.CreateUserAsync(usrInfo, It.IsAny<CancellationToken>()), Times.Once);

}

[TestMethod]

public async Task Handle\_DuplicateUser\_ThrowsDomainException()

{

// Arrange

var request = new CreateUsrInfoRequest { UsrId = "EXISTING001" };

var command = new CreateUsrInfoCommand { Request = request };

var existingUser = new UsrInfo { UsrId = "EXISTING001" };

\_mockRepository.Setup(x => x.GetByUserIdAsync("EXISTING001", It.IsAny<CancellationToken>()))

.ReturnsAsync(existingUser);

// Act & Assert

var exception = await Assert.ThrowsExceptionAsync<DomainException>(

() => \_handler.Handle(command, CancellationToken.None));

Assert.AreEqual("User with ID 'EXISTING001' already exists", exception.Message);

\_mockUnitOfWork.Verify(x => x.BeginTransactionAsync(It.IsAny<CancellationToken>()), Times.Never);

}

}

**Validation Testing**

[TestClass]

public class CreateUsrInfoRequestValidatorTests

{

private readonly CreateUsrInfoRequestValidator \_validator;

[TestInitialize]

public void Setup()

{

\_validator = new CreateUsrInfoRequestValidator();

}

[TestMethod]

public void Validate\_ValidRequest\_ReturnsValid()

{

// Arrange

var request = new CreateUsrInfoRequest

{

UsrId = "VALID001",

UsrName = "Valid User",

UsrEmail = "valid@company.com",

RmsType = "TRADING"

};

// Act

var result = \_validator.Validate(request);

// Assert

Assert.IsTrue(result.IsValid);

Assert.AreEqual(0, result.Errors.Count);

}

[TestMethod]

public void Validate\_EmptyUsrId\_ReturnsInvalid()

{

// Arrange

var request = new CreateUsrInfoRequest

{

UsrId = "",

RmsType = "TRADING"

};

// Act

var result = \_validator.Validate(request);

// Assert

Assert.IsFalse(result.IsValid);

Assert.IsTrue(result.Errors.Any(e => e.PropertyName == "UsrId"));

Assert.IsTrue(result.Errors.Any(e => e.ErrorMessage.Contains("User ID is required")));

}

}

**Integration Testing**

**API Integration Tests**

[TestClass]

public class UsrInfoControllerIntegrationTests

{

private readonly WebApplicationFactory<Program> \_factory;

private readonly HttpClient \_client;

[TestInitialize]

public void Setup()

{

\_factory = new WebApplicationFactory<Program>()

.WithWebHostBuilder(builder =>

{

builder.ConfigureServices(services =>

{

// Replace services for testing

services.AddScoped<IExternalTokenService, MockTokenService>();

});

});

\_client = \_factory.CreateClient();

}

[TestMethod]

public async Task GetUsrInfo\_ValidId\_ReturnsUser()

{

// Arrange

await AuthenticateAsync();

// Act

var response = await \_client.GetAsync("/api/v1/usrinfo/EFTEST01");

// Assert

response.EnsureSuccessStatusCode();

var content = await response.Content.ReadAsStringAsync();

var result = JsonSerializer.Deserialize<ApiResponse<UsrInfoDto>>(content, new JsonSerializerOptions

{

PropertyNamingPolicy = JsonNamingPolicy.CamelCase

});

Assert.IsNotNull(result);

Assert.IsTrue(result.Success);

Assert.AreEqual("EFTEST01", result.Data.UsrId);

}

[TestMethod]

public async Task CreateUsrInfo\_ValidData\_ReturnsCreated()

{

// Arrange

await AuthenticateAsync();

var request = new CreateUsrInfoRequest

{

UsrId = "INTEGRATION\_TEST\_001",

UsrName = "Integration Test User",

UsrEmail = "integration@test.com",

RmsType = "TESTING"

};

var json = JsonSerializer.Serialize(request, new JsonSerializerOptions

{

PropertyNamingPolicy = JsonNamingPolicy.CamelCase

});

var content = new StringContent(json, Encoding.UTF8, "application/json");

// Act

var response = await \_client.PostAsync("/api/v1/usrinfo", content);

// Assert

response.EnsureSuccessStatusCode();

var responseContent = await response.Content.ReadAsStringAsync();

var result = JsonSerializer.Deserialize<ApiResponse<UsrInfoDto>>(responseContent, new JsonSerializerOptions

{

PropertyNamingPolicy = JsonNamingPolicy.CamelCase

});

Assert.IsNotNull(result);

Assert.IsTrue(result.Success);

Assert.AreEqual("INTEGRATION\_TEST\_001", result.Data.UsrId);

}

private async Task AuthenticateAsync()

{

// Perform handshake

var handshakeRequest = new HandshakeRequest

{

AppId = "RMS\_APP\_2025",

AppSecret = "RMS\_SECRET\_2025\_SECURE\_KEY"

};

var handshakeJson = JsonSerializer.Serialize(handshakeRequest);

var handshakeContent = new StringContent(handshakeJson, Encoding.UTF8, "application/json");

var handshakeResponse = await \_client.PostAsync("/api/v1/handshake", handshakeContent);

var handshakeResult = JsonSerializer.Deserialize<ApiResponse<HandshakeResponse>>(

await handshakeResponse.Content.ReadAsStringAsync(),

new JsonSerializerOptions { PropertyNamingPolicy = JsonNamingPolicy.CamelCase });

var handshakeToken = handshakeResult.Data.HandshakeToken;

// Perform login

var loginRequest = new LoginRequest

{

Username = "EFTEST01",

Password = "password"

};

var loginJson = JsonSerializer.Serialize(loginRequest);

var loginContent = new StringContent(loginJson, Encoding.UTF8, "application/json");

\_client.DefaultRequestHeaders.Add("X-Handshake-Token", handshakeToken);

var loginResponse = await \_client.PostAsync("/api/v1/auth/login", loginContent);

var loginResult = JsonSerializer.Deserialize<ApiResponse<LoginResponse>>(

await loginResponse.Content.ReadAsStringAsync(),

new JsonSerializerOptions { PropertyNamingPolicy = JsonNamingPolicy.CamelCase });

// Set authentication headers for subsequent requests

\_client.DefaultRequestHeaders.Authorization =

new System.Net.Http.Headers.AuthenticationHeaderValue("Bearer", loginResult.Data.UserToken);

}

}

**Troubleshooting**

**Common Issues & Solutions**

**1. Authentication Issues**

**Problem**: "Invalid or expired handshake token"

Status: 401 Unauthorized

Message: "Invalid or expired handshake token"

**Solution**:

// Check handshake token cache

var cacheKey = $"HANDSHAKE\_TOKEN\_{handshakeToken}";

var tokenInfo = await \_cacheService.GetAsync<HandshakeTokenInfo>(cacheKey);

// Regenerate if expired

if (tokenInfo == null || tokenInfo.ExpiresAt < DateTime.UtcNow)

{

// Request new handshake token

POST /api/v1/handshake

}

**2. Database Connection Issues**

**Problem**: "LB.DAL database connection failed"

HealthCheckResult: Unhealthy

Exception: Unable to connect to SQL Server

**Solution**:

// Check connection string format

{

"ConnectionStrings": {

"DefaultConnection": "Server=server;Database=db;User Id=user;Password=pwd;TrustServerCertificate=true;MultipleActiveResultSets=true;Connection Timeout=30;"

}

}

**3. Performance Issues**

**Problem**: Slow API responses (>2 seconds)

X-Response-Time: 3245ms

Warning: SLOW REQUEST detected

**Solution**:

// Check database query performance

EXEC sp\_who2 -- Check for blocking processes

// Review query execution plans

SET STATISTICS IO ON

SELECT \* FROM UsrInfo WHERE UsrStatus = 'A'

// Implement caching for frequently accessed data

await \_cacheService.SetAsync($"USER\_{userId}", user, TimeSpan.FromMinutes(15));

**4. Memory Issues**

**Problem**: High memory usage and OutOfMemoryException

System.OutOfMemoryException: Insufficient memory to continue execution

**Solution**:

// Implement proper disposal patterns

public async Task<PagedResult<UsrInfo>> GetUsersPagedAsync(...)

{

using DbDataReader dr = await \_dal.LB\_GetDbDataReaderAsync(sql, parameters, CommandType.Text);

// Process data

} // DbDataReader automatically disposed

// Use pagination for large datasets

var pageSize = Math.Min(requestedPageSize, 100); // Cap at 100 records

**5. Token Validation Failures**

**Problem**: "Token validation failed" with valid token

External token service returned: Invalid token

**Solution**:

// Debug token validation

var userId = ExtractUserIdFromToken(token);

\_logger.LogDebug("Validating token for user: {UserId}", userId);

// Check token service connectivity

var healthCheck = await \_tokenService.HandshakeAsync();

if (!healthCheck)

{

\_logger.LogError("Token service is not available");

// Implement fallback or circuit breaker

}

**Logging & Diagnostics**

**Application Logs**

# View application logs

tail -f logs/rms-log-20250728.txt

# Filter for errors

grep "ERROR" logs/rms-log-\*.txt

# Check performance issues

grep "SLOW REQUEST" logs/rms-log-\*.txt

**Health Check Monitoring**

# Check overall system health

GET /health

Response: {

"status": "Healthy",

"totalDuration": "00:00:00.0234567",

"entries": {

"database": { "status": "Healthy" },

"external-api": { "status": "Healthy" }

}

}

# Check specific component

GET /health/database

**Database Diagnostics**

-- Check active connections

SELECT

DB\_NAME(dbid) as DatabaseName,

COUNT(dbid) as ConnectionCount

FROM sys.sysprocesses

WHERE dbid > 0

GROUP BY dbid, DB\_NAME(dbid);

-- Check long-running queries

SELECT

r.session\_id,

r.start\_time,

r.status,

r.command,

s.text

FROM sys.dm\_exec\_requests r

CROSS APPLY sys.dm\_exec\_sql\_text(sql\_handle) s

WHERE r.total\_elapsed\_time > 30000; -- > 30 seconds

**Error Code Reference**

| **Error Code** | **HTTP Status** | **Description** | **Resolution** |
| --- | --- | --- | --- |
| **AUTH001** | 401 | Missing handshake token | Include X-Handshake-Token header |
| **AUTH002** | 401 | Invalid handshake token | Request new handshake |
| **AUTH003** | 401 | Expired handshake token | Request new handshake |
| **AUTH004** | 401 | Missing user token | Include Authorization: Bearer header |
| **AUTH005** | 401 | Invalid user token | Re-authenticate user |
| **VAL001** | 400 | Validation failed | Check request data format |
| **DB001** | 500 | Database connection failed | Check connection string |
| **DB002** | 500 | Database query timeout | Optimize query or increase timeout |
| **EXT001** | 503 | External service unavailable | Check external service status |
| **RATE001** | 429 | Rate limit exceeded | Reduce request frequency |

**Conclusion**

SimRMS represents a comprehensive, enterprise-grade Risk Management System built with modern .NET technologies and Clean Architecture principles. The system provides:

**Key Benefits**

* **Scalability**: Clean Architecture enables easy scaling and maintenance
* **Security**: Multi-layered authentication and authorization
* **Performance**: Optimized data access and intelligent caching
* **Reliability**: Comprehensive error handling and monitoring
* **Maintainability**: SOLID principles and separation of concerns
* **Testability**: Extensive unit and integration testing capabilities

**Future Roadmap**

* **Enhanced Analytics**: Advanced reporting and dashboard capabilities
* **Real-time Notifications**: WebSocket-based real-time updates
* **Mobile Support**: Mobile API optimizations
* **Multi-tenancy**: Support for multiple organizations
* **Machine Learning**: Risk prediction and anomaly detection
* **Microservices Evolution**: Gradual decomposition into microservices

**Support & Contact**

For technical support, feature requests, or contributions:

* **Documentation**: Internal Wiki/Confluence
* **Issue Tracking**: Jira/Azure DevOps
* **Code Repository**: Internal Git Repository
* **Team Contact**: Development Team Lead

*This documentation is maintained by the SimRMS Development Team and is updated with each major release. Last updated: July 28, 2025*