# SimRMS - Developer Guidelines & Implementation Guide

## 🚀 Quick Start for New Developers

### Project Architecture Overview

SimRMS follows **\*\*Clean Architecture\*\*** with strict layer separation. Each layer has specific responsibilities:

```

src/

├── SimRMS.Domain/         # Core business entities, exceptions, interfaces

├── SimRMS.Application/    # Use cases, DTOs, request models, validators, service interfaces

├── SimRMS.Infrastructure/ # Data access, services implementation, external integrations

├── SimRMS.Shared/        # Cross-cutting concerns, common models, constants, extensions

└── SimRMS.WebAPI/        # Controllers, middleware, security, configuration

```

**\*\*Key Rule\*\***: Dependencies flow inward only. No circular references between layers.

---

## 📋 Naming Conventions & Standards

### 1. **\*\*Entities\*\*** (Domain Layer)

- **\*\*Pattern\*\***: Use exact database table names

- **\*\*Examples\*\***: `MstCoBrch`, `MstCo`, `UsrInfo`

- **\*\*Inheritance\*\***: All inherit from `BaseEntity`

### 2. **\*\*DTOs\*\*** (Application Layer)

- **\*\*Pattern\*\***: `{BusinessContext}Dto`

- **\*\*Examples\*\***: `MstCoBrchDto`, `BrokerBranchDto`, `UserStatisticsDto`

- **\*\*Inheritance\*\***: Inherit from `BaseEntityDto` when applicable

### 3. **\*\*Request Models\*\*** (Application Layer)

- **\*\*Pattern\*\***: `{Action}{BusinessContext}Request`

- **\*\*Examples\*\***:

  - `CreateMstCoBrchRequest`

  - `UpdateMstCoBrchRequest`

  - `DeleteMstCoBrchRequest`

  - `AuthorizeMstCoBrchRequest`

### 4. **\*\*Services\*\*** (Infrastructure Layer)

- **\*\*Interface Pattern\*\***: `I{BusinessContext}Service`

- **\*\*Implementation Pattern\*\***: `{BusinessContext}Service`

- **\*\*Examples\*\***:

  - Interface: `IBrokerBranchService`

  - Implementation: `BrokerBranchService`

### 5. **\*\*Controllers\*\*** (WebAPI Layer)

- **\*\*Pattern\*\***: `{BusinessContext}Controller`

- **\*\*Examples\*\***: `BrokerBranchController`, `WorkFlowController`, `CompanyExposureController`

- **\*\*Base\*\***: All inherit from `BaseController`

### 6. **\*\*Validators\*\*** (Application Layer)

- **\*\*Pattern\*\***: Single file per business area with multiple validators

- **\*\*File Name\*\***: `{BusinessContext}RequestValidators.cs`

- **\*\*Class Names\*\***: `{Action}{BusinessContext}RequestValidator`

- **\*\*Examples\*\***:

  ```csharp

*// File: MstCoBrchRequestValidators.cs*

  public class CreateMstCoBrchRequestValidator : AbstractValidator<CreateMstCoBrchRequest>

  public class UpdateMstCoBrchRequestValidator : AbstractValidator<UpdateMstCoBrchRequest>

  public class DeleteMstCoBrchRequestValidator : AbstractValidator<DeleteMstCoBrchRequest>

  ```

### 7. **\*\*Properties & Methods\*\***

- **\*\*Properties\*\***: PascalCase (`CoCode`, `CoBrchDesc`)

- **\*\*Methods\*\***: PascalCase with descriptive verbs (`GetMstCoBrchListAsync`, `CreateMstCoBrchAsync`)

- **\*\*Private Methods\*\***: PascalCase (`ValidateCreateRequestAsync`)

- **\*\*Parameters\*\***: camelCase (`pageNumber`, `pageSize`, `cancellationToken`)

### 8. **\*\*Constants\*\*** (Shared Layer)

- **\*\*Files\*\***: `AppConstants.cs`, `ActionTypeEnum.cs`

- **\*\*Usage\*\***: `ActionTypeEnum.INSERT`, `AppConstants.Headers.Authorization`

---

## 🛠️ Implementation Patterns & Guidelines

### Database Access Rules

- ✅ **\*\*ONLY use stored procedures\*\*** - Never write inline SQL

- ✅ **\*\*Use generic repository\*\*** for all data operations

- ✅ **\*\*Handle OUTPUT parameters\*\*** for result status tracking

- ❌ Never use Entity Framework or raw SQL queries

**\*\*Stored Procedure Conventions:\*\***

```csharp

*// CRUD: LB\_SP\_Crud{EntityName} with Action parameter (1=Create, 2=Update, 3=Delete)*

"LB\_SP\_CrudMstCoBrch"

*// List: LB\_SP\_Get{EntityName}List with pagination*

"LB\_SP\_GetBrokerBranchList"

*// Single: LB\_SP\_Get{EntityName}\_By{KeyField}*

"LB\_SP\_GetMstCoBrch\_ByBranchCode"

*// Workflow: {Base}WF suffix*

"LB\_SP\_GetBrokerBranchListWF"

```

### Service Layer Pattern

```csharp

public class BrokerBranchService : IBrokerBranchService

{

*// Standard CRUD methods*

    Task<PagedResult<TDto>> Get{Entity}ListAsync(params...)

    Task<TDto?> Get{Entity}ByIdAsync(params...)

    Task<TDto> Create{Entity}Async(params...)

    Task<TDto> Update{Entity}Async(params...)

    Task<bool> Delete{Entity}Async(params...)

    Task<bool> {Entity}ExistsAsync(params...)

*// Workflow methods (if applicable)*

    Task<PagedResult<TDto>> GetUnauthorizedListAsync(params...)

    Task<bool> Authorize{Entity}Async(params...)

}

```

### Controller Pattern

```csharp

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

[ApiController]

[ApiVersion("1.0")]

[Authorize]

public class BrokerBranchController : BaseController

{

*// Standard REST endpoints*

    [HttpGet] *// GET api/v1/brokerbranch*

    [HttpGet("{id}")] *// GET api/v1/brokerbranch/{id}*

    [HttpPost] *// POST api/v1/brokerbranch*

    [HttpPut("{id}")] *// PUT api/v1/brokerbranch/{id}*

    [HttpDelete("{id}")] *// DELETE api/v1/brokerbranch/{id}*

    [HttpHead("{id}")] *// HEAD api/v1/brokerbranch/{id} (exists check)*

*// Workflow endpoints*

    [HttpGet("wf/unauthorized")] *// Workflow-specific routes*

    [HttpPost("wf/authorize/{id}")]

}

```

### Exception Handling Rules

- ✅ **\*\*Services throw domain exceptions\*\*** - Never catch and return false success

- ✅ **\*\*Controllers let exceptions bubble up\*\*** - Middleware handles all errors

- ✅ **\*\*Use domain-specific exceptions\*\***: `ValidationException`, `DomainException`, `NotFoundException`

- ❌ **\*\*NEVER catch exceptions in controllers\*\*** and return `BadRequest()`

```csharp

*// ✅ CORRECT - Service throws exceptions*

public async Task<EntityDto> GetEntityAsync(int id)

{

    try

    {

        return await \_repository.GetAsync(id);

    }

    catch (Exception ex)

    {

        \_logger.LogError(ex, "Error getting entity");

        throw new DomainException($"Failed to retrieve entity: {ex.Message}");

    }

}

*// ✅ CORRECT - Controller lets exceptions bubble up*

[HttpGet("{id}")]

public async Task<ActionResult<EntityDto>> GetEntity(int id)

{

    var entity = await \_service.GetEntityAsync(id);

    return Ok(entity);

}

```

### Validation Pattern

- **\*\*Single validator file per business area\*\*** with extension methods for reusability

- **\*\*Validation occurs at service layer\*\*** before business logic

- **\*\*FluentValidation\*\*** for all request validation

```csharp

*// Extension methods for reusable rules*

public static class EntityValidationRules

{

    public static IRuleBuilderOptions<T, string> ValidEntityCode<T>(this IRuleBuilder<T, string> ruleBuilder)

    {

        return ruleBuilder.NotEmpty().MaximumLength(10).Matches("^[A-Z0-9]+$");

    }

}

```

---

## 🔐 Security & Middleware

### Authentication Flow

1. **\*\*TokenAuthenticationMiddleware\*\*** - Validates handshake + user tokens

2. **\*\*ExceptionHandlingMiddleware\*\*** - Global error handling with consistent responses

3. **\*\*Authorization\*\*** - Policy-based with permissions

### Middleware Registration Order

```csharp

*// Program.cs - Correct order is critical*

app.UseMiddleware<TokenAuthenticationMiddleware>();

app.UseAuthentication();

app.UseAuthorization();

app.UseMiddleware<ExceptionHandlingMiddleware>();

```

---

## 📝 Implementing Trader Section - Step-by-Step Guide

Based on the BrokerBranch implementation pattern, here's how to implement the Trader section:

### Step 1: Create Domain Entity

```csharp

*// src/SimRMS.Domain/Entities/MstTrader.cs*

public class MstTrader : BaseEntity

{

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

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

    public string? TraderEmail { get; set; }

    public string? TraderPhone { get; set; }

*// Add other trader-specific properties*

}

```

### Step 2: Create DTOs

```csharp

*// src/SimRMS.Application/Models/DTOs/TraderDtos.cs*

public class TraderDto : BaseEntityDto

{

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

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

    public string? TraderEmail { get; set; }

    public string? TraderPhone { get; set; }

}

```

### Step 3: Create Request Models

```csharp

*// src/SimRMS.Application/Models/Requests/TraderRequests.cs*

public class CreateTraderRequest

{

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

    public string? TraderEmail { get; set; }

    public string? TraderPhone { get; set; }

    public string? Remarks { get; set; }

}

public class UpdateTraderRequest

{

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

    public string? TraderName { get; set; }

    public string? TraderEmail { get; set; }

    public string? TraderPhone { get; set; }

    public string? Remarks { get; set; }

}

*// ... DeleteTraderRequest, AuthorizeTraderRequest*

```

### Step 4: Create Validators (Single File)

```csharp

*// src/SimRMS.Application/Validators/TraderRequestValidators.cs*

public static class TraderValidationRules

{

    public static IRuleBuilderOptions<T, string> ValidTraderCode<T>(this IRuleBuilder<T, string> ruleBuilder)

    {

        return ruleBuilder.NotEmpty().MaximumLength(20).Matches("^[A-Z0-9]+$");

    }

}

public class CreateTraderRequestValidator : AbstractValidator<CreateTraderRequest> { }

public class UpdateTraderRequestValidator : AbstractValidator<UpdateTraderRequest> { }

public class DeleteTraderRequestValidator : AbstractValidator<DeleteTraderRequest> { }

```

### Step 5: Create Service Interface & Implementation

```csharp

*// src/SimRMS.Application/Interfaces/Services/ITraderService.cs*

public interface ITraderService

{

    Task<PagedResult<TraderDto>> GetTraderListAsync(int pageNumber, int pageSize, string? searchTerm, CancellationToken cancellationToken);

    Task<TraderDto?> GetTraderByIdAsync(string traderCode, CancellationToken cancellationToken);

    Task<TraderDto> CreateTraderAsync(CreateTraderRequest request, CancellationToken cancellationToken);

    Task<TraderDto> UpdateTraderAsync(string traderCode, UpdateTraderRequest request, CancellationToken cancellationToken);

    Task<bool> DeleteTraderAsync(string traderCode, DeleteTraderRequest request, CancellationToken cancellationToken);

    Task<bool> TraderExistsAsync(string traderCode, CancellationToken cancellationToken);

}

*// src/SimRMS.Infrastructure/Services/TraderService.cs*

public class TraderService : ITraderService

{

*// Implement following BrokerBranchService pattern*

*// Use stored procedures: LB\_SP\_CrudTrader, LB\_SP\_GetTraderList, etc.*

}

```

### Step 6: Create Controller

```csharp

*// src/SimRMS.WebAPI/Controllers/V1/TraderController.cs*

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

[ApiController]

[ApiVersion("1.0")]

[Authorize]

public class TraderController : BaseController

{

*// Follow BrokerBranchController pattern exactly*

*// Implement GET, POST, PUT, DELETE, HEAD endpoints*

*// Add workflow endpoints if needed*

}

```

### Step 7: Register Services

```csharp

*// src/SimRMS.Infrastructure/DependencyInjection.cs*

services.AddScoped<ITraderService, TraderService>();

*// Register validators*

services.AddScoped<IValidator<CreateTraderRequest>, CreateTraderRequestValidator>();

services.AddScoped<IValidator<UpdateTraderRequest>, UpdateTraderRequestValidator>();

services.AddScoped<IValidator<DeleteTraderRequest>, DeleteTraderRequestValidator>();

```

### Step 8: Create Database Stored Procedures

Create the following stored procedures in your database:

- `LB\_SP\_CrudTrader` (Action: 1=Insert, 2=Update, 3=Delete)

- `LB\_SP\_GetTraderList` (with pagination support)

- `LB\_SP\_GetTrader\_ByCode` (single record retrieval)

- `LB\_SP\_GetTraderListWF` (workflow unauthorized list)

- `LB\_SP\_AuthTrader` (workflow authorization)

---

## ✅ Development Checklist

When implementing any new business section:

- [ ] **\*\*Entity\*\*** created in Domain/Entities (use database table name)

- [ ] **\*\*DTOs\*\*** created in Application/Models/DTOs

- [ ] **\*\*Request models\*\*** created in Application/Models/Requests

- [ ] **\*\*Single validator file\*\*** created with extension methods

- [ ] **\*\*Service interface\*\*** created in Application/Interfaces/Services

- [ ] **\*\*Service implementation\*\*** created in Infrastructure/Services

- [ ] **\*\*Controller\*\*** created in WebAPI/Controllers/V1

- [ ] **\*\*Services registered\*\*** in DI containers

- [ ] **\*\*Stored procedures\*\*** created in database

- [ ] **\*\*Exception handling\*\*** implemented (throw domain exceptions)

- [ ] **\*\*Logging\*\*** added throughout service methods

- [ ] **\*\*Authorization\*\*** attributes applied to controller actions

- [ ] **\*\*API documentation\*\*** added with XML comments

**\*\*Remember\*\***: Follow the BrokerBranch implementation as your reference template. Keep it simple, consistent, and maintain the established patterns.