**Documento de Especificação - Sistema FinanceSystem**

**Visão Geral do Projeto**

O FinanceSystem é um sistema financeiro com autenticação JWT e arquitetura DDD (Domain-Driven Design), composto por uma API RESTful backend e uma interface web MVC. O sistema implementa controle de acesso baseado em perfis (roles), com funcionalidades completas de gerenciamento de usuários e permissões.

**Arquitetura**

O projeto segue a arquitetura DDD com as seguintes camadas:

FinanceSystem/

├── src/

│ ├── FinanceSystem.Domain/ # Entidades, interfaces, regras de negócio

│ ├── FinanceSystem.Application/ # Serviços de aplicação, DTOs

│ ├── FinanceSystem.Infrastructure/ # Implementações concretas, EF Core

│ ├── FinanceSystem.API/ # WebAPI REST

│ └── FinanceSystem.Web/ # Frontend MVC com Razor

└── tests/

├── FinanceSystem.Domain.Tests/

├── FinanceSystem.Application.Tests/

└── FinanceSystem.Infrastructure.Tests/

**Tecnologias Utilizadas**

* **Backend**: .NET 6+
* **ORM**: Entity Framework Core
* **Banco de Dados**: SQL Server
* **Autenticação**: JWT (JSON Web Tokens) + BCrypt
* **Frontend**: ASP.NET Core MVC com Razor Views
* **CSS Framework**: Bootstrap 5
* **Comunicação**: HTTP Client para consumo da API

**Entidades Principais**

**Domain**

**User**

csharp

public class User

{

public Guid Id { get; protected set; }

public string Username { get; protected set; }

public string Email { get; protected set; }

public string PasswordHash { get; protected set; }

public bool IsActive { get; protected set; }

public DateTime CreatedAt { get; protected set; }

public DateTime? LastLogin { get; protected set; }

public ICollection<UserRole> UserRoles { get; protected set; }

*// Métodos de comportamento*

public void UpdateUsername(string username)

public void UpdateEmail(string email)

public void UpdatePassword(string passwordHash)

public void Activate()

public void Deactivate()

public void SetLastLogin()

public void AddRole(Role role)

public void RemoveRole(Role role)

}

**Role**

csharp

public class Role

{

public Guid Id { get; protected set; }

public string Name { get; protected set; }

public string Description { get; protected set; }

public DateTime CreatedAt { get; protected set; }

public ICollection<UserRole> UserRoles { get; protected set; }

*// Métodos de comportamento*

public void UpdateName(string name)

public void UpdateDescription(string description)

}

**UserRole**

csharp

public class UserRole

{

public Guid UserId { get; protected set; }

public User User { get; protected set; }

public Guid RoleId { get; protected set; }

public Role Role { get; protected set; }

public DateTime CreatedAt { get; protected set; }

}

**Interfaces do Domain**

**Repositories**

* IRepositoryBase<T>: Operações CRUD genéricas
* IUserRepository: Operações específicas para User
* IRoleRepository: Operações específicas para Role

**Services**

* IAuthService: Gerenciamento de autenticação
* IUnitOfWork: Gerenciamento de transações

**Camada de Application**

**DTOs**

* UserDto: Representação simplificada do usuário
* RoleDto: Representação simplificada do perfil
* CreateUserDto: DTO para criação de usuário
* UpdateUserDto: DTO para atualização de usuário
* LoginDto: DTO para login
* LoginResponseDto: DTO com resposta do login (contém token)

**Services**

* IUserService: Gerenciamento de usuários
* IRoleService: Gerenciamento de perfis

**Camada de Infrastructure**

**Data**

* ApplicationDbContext: Contexto EF Core
* DbInitializer: Inicialização do banco de dados

**Repositories**

* RepositoryBase<T>: Implementação genérica
* UserRepository: Implementação específica
* RoleRepository: Implementação específica

**Services**

* AuthService: Implementação de autenticação
* UnitOfWork: Implementação de transações

**Camada de API**

**Controllers**

* AuthController: Autenticação
* UsersController: CRUD de usuários
* RolesController: CRUD de perfis

**Middleware**

* ErrorHandlerMiddleware: Tratamento global de exceções

**Camada Web (MVC)**

**Controllers**

* HomeController: Dashboard
* AccountController: Login, Logout
* UsersController: CRUD usuários
* RolesController: CRUD perfis

**Services**

* ApiService: Comunicação com API
* UserService: Operações com usuários
* RoleService: Operações com perfis

**Views**

* Layout comum
* Views de Login
* Dashboard principal
* CRUD de Usuários
* CRUD de Perfis

**Autenticação e Autorização**

* **Autenticação**: JWT (JSON Web Tokens)
* **Senha**: Protegidas com BCrypt
* **Autorização**: Baseada em roles (Admin e User)
* **Web**: Cookies de autenticação a partir do JWT

**Banco de Dados**

* **Provider**: SQL Server
* **Migrations**: Gerenciadas pelo EF Core
* **Inicialização**: Automática via DbInitializer
* **Seed Data**: Admin user e roles padrão (Admin, User)

**Configurações**

**API (appsettings.json)**

json

{

"ConnectionStrings": {

"DefaultConnection": "Server=localhost;Database=FinanceSystem;User Id=sa;Password=admin;TrustServerCertificate=True;"

},

"JwtSettings": {

"Secret": "J8P3xR5tG7kL9mN2qA4zD6bV1cX3eS5w",

"Issuer": "FinanceSystem",

"Audience": "FinanceSystemClient",

"ExpiryHours": "1"

}

}

**Web (appsettings.json)**

json

{

"ApiSettings": {

"BaseUrl": "https://localhost:5001"

}

}

**Portas e URLs**

* **API**: <https://localhost:5001>
* **Web**: <https://localhost:5003>

**Pacotes NuGet Principais**

**API e Infraestrutura**

* Microsoft.EntityFrameworkCore
* Microsoft.EntityFrameworkCore.SqlServer
* Microsoft.EntityFrameworkCore.Design
* Microsoft.AspNetCore.Authentication.JwtBearer
* BCrypt.Net-Next
* AutoMapper

**Web**

* Microsoft.AspNetCore.Authentication.Cookies
* System.IdentityModel.Tokens.Jwt

**Funcionalidades Implementadas**

1. **Autenticação e Autorização**
   * Login com JWT
   * Controle de acesso baseado em roles
2. **Gerenciamento de Usuários**
   * Listagem de usuários
   * Criação de usuários
   * Edição de dados do usuário
   * Atribuição de perfis
   * Ativação/desativação
   * Exclusão
3. **Gerenciamento de Perfis**
   * Listagem de perfis
   * Criação de perfis
   * Edição de perfis
   * Exclusão de perfis

**Como Executar o Projeto**

1. **Configurar o banco de dados**
   * Confirmar string de conexão no appsettings.json
2. **Executar migrações (da pasta FinanceSystem.Infrastructure)**

bash

dotnet ef migrations add InitialCreate --startup-project ../FinanceSystem.API/FinanceSystem.API.csproj

1. **Executar a API**

bash

cd src/FinanceSystem.API

dotnet run

1. **Executar o Frontend**

bash

cd src/FinanceSystem.Web

dotnet run

1. **Acessar a aplicação**
   * Frontend: <https://localhost:5003>
   * Swagger API: <https://localhost:5001/swagger>
2. **Credenciais padrão**
   * Usuário: admin
   * Senha: admin

**Possibilidades de Expansão Futura**

1. **Adicionar módulos financeiros**
   * Contas a pagar/receber
   * Fluxo de caixa
   * Relatórios financeiros
2. **Melhorias de segurança**
   * Implementar refresh tokens
   * Implementar two-factor authentication
3. **Aprimoramentos de UI/UX**
   * Implementar tema escuro
   * Adicionar dashboard com gráficos
4. **Internacionalização**
   * Suporte para múltiplos idiomas
5. **Notificações**
   * Sistema de notificações por email
   * Notificações em tempo real