

UserManagementApp Project Documentation

Generated from the current repository documentation. This document includes only implemented functionality.

UserManagementApp Project Documentation

1. Overview

UserManagementApp is a .NET solution for user management through a Web API and a WPF desktop client. The desktop application communicates with the backend through HTTP clients and does not access the database directly.

The current implementation contains authentication, role-based desktop navigation, user CRUD operations for admins, a regular user profile dashboard, EF Core persistence, and automated tests.

2. Architecture

2.1 Projects

Project	Responsibility
UserManagementApp.Api	ASP.NET Core Web API, controllers, Swagger/OpenAPI
UserManagementApp.Core	DTOs, models, enums, interfaces
UserManagementApp.Data	EF Core, SQLite, services, migrations
UserManagementApp.Desktop	WPF desktop client with a simple MVVM-style structure
UserManagementApp.Tests	NUnit unit, API, and integration tests

2.2 Runtime Flow

```
WPF Desktop Client
  |
  | HTTP requests
  v
ASP.NET Core Web API
  |
  | services
  v
EF Core AppDbContext
  |
  v
SQLite app.db
```

2.3 Permission Flow

Admin-only user changes use a header-based permission flow:

X-User-Id: <current-admin-user-id>

The API checks this value through `PermissionService.IsAdminAsync`.

3. API

3.1 Authentication

Endpoint:

POST /api/auth/login

The request uses username and password. Successful login returns a `LoginResponse` with user id, username, email, and role.

Invalid credentials return 401 Unauthorized.

3.2 Database Status

Endpoint:

GET /api/database/status

The endpoint checks whether the API can connect to SQLite and returns the connection status and provider name.

3.3 Users

Endpoints:

GET /api/users
GET /api/users/{id}
POST /api/users
PUT /api/users/{id}
DELETE /api/users/{id}

Create, update, and delete require an Admin X-User-Id header. Missing or invalid headers return 401 Unauthorized; non-admin users receive 403 Forbidden.

3.4 DTOs

DTO	Use
LoginRequest	Login input
LoginResponse	Login result
CreateUserRequest	Create user input
UpdateUserRequest	Update user input
UserResponse	User output

4. Core

4.1 Models

User stores identity and profile data:

- UserId
- UserName

- Email
- PasswordHash
- Role
- FirstName
- LastName
- CreatedAt
- UpdatedAt

SupportRequest exists as a model and database set with:

- Id
- UserId
- Subject
- Message
- Status
- CreatedAt

There is no support request controller or desktop support request workflow in the current codebase.

4.2 Enums

UserRole contains:

- Admin
- RegularUser

SupportRequestStatus is used by the support request model.

4.3 Interfaces

Core defines service contracts for authentication, users, permissions, and password hashing.

5. Data Layer

5.1 Database

The data layer uses EF Core with SQLite. The API connection string is configured as:

```
{
  "ConnectionStrings": {
    "DefaultConnection": "Data Source=app.db"
  }
}
```

5.2 AppDbContext

AppDbContext exposes:

- Users
- SupportRequests

User configuration includes required fields, unique username/email indexes, role conversion to string, and max lengths for profile fields.

Support request configuration includes required subject/message/status fields and a cascade relationship to User.

5.3 Services

Service	Responsibility
AuthService	Login by username and password
UserService	User CRUD operations
PermissionService	Admin role checks
PasswordHasher	Password hashing and verification through ASP.NET Core Identity

6. Desktop Application

6.1 Technology

The desktop client is a WPF application targeting net9.0-windows.

It uses a simple MVVM-style structure:

```
Commands/
Clients/
Configuration/
Services/
ViewModels/
Views/
```

6.2 LoginWindow

The login screen supports:

- username input
- password input
- show/hide password
- login action
- error message area
- light/dark theme toggle

After successful login, the app opens an admin dashboard for Admin users or a user dashboard for RegularUser.

6.3 AdminDashboardWindow

The admin dashboard shows the current logged-in user and a DataGrid with users.

Available actions:

- refresh users
- create user
- edit selected user
- delete selected user
- logout
- switch theme

6.4 UserFormWindow

The user form is used for both create and edit operations.

Fields:

- username
- email
- password in create mode
- role
- first name
- last name

6.5 UserDashboardWindow

The regular user dashboard shows profile information:

- username
- email
- role
- first name
- last name

It also supports logout and theme switching.

6.6 API Clients

The desktop client uses:

- AuthApiClient
- UserApiClient

All backend actions go through these clients.

7. Testing

7.1 Test Project

The solution contains one NUnit test project:

UserManagementApp.Tests

7.2 Test Types

Type	Location
Unit tests	UnitTests/Services
API tests	ApiTests
Integration tests	IntegrationTests/Data

7.3 Current Coverage

The test project contains 32 NUnit tests covering:

- authentication service behavior
- user service CRUD behavior
- database context behavior
- auth API login cases
- users API CRUD and permission cases

- database status endpoint

7.4 Test Support

The test project includes:

- shared API clients
- test configuration classes
- TestDataHelper
- step logging through TestStepLogger
- cleanup for temporary users in API tests

8. Local Run Commands

Restore packages:

```
dotnet restore
```

Apply migrations:

```
dotnet ef database update --project UserManagementApp.Data --startup-project UserManagementApp.Api --cont
```

Run API:

```
dotnet run --project UserManagementApp.Api
```

Run desktop client:

```
dotnet run --project UserManagementApp.Desktop
```

Run tests:

```
dotnet test
```

9. GitHub Pages Documentation

The static documentation website is located at:

```
docs/index.html
```

GitHub Pages can publish it by selecting the repository branch and the /docs folder as the Pages source.