# 《UserInfoManager 项目技术文档》

## 一、项目概述

UserInfoManager 项目是一个基于ASP.NET Core 的用户信息管理系统，提供用户注册、登录、个人信息管理、地址管理以及查找信息管理等核心功能。系统采用分层架构设计，实现了前后端分离的开发模式，通过 RESTful API 进行数据交互。项目支持多种部署环境，并提供了自动化的未验证用户清理机制，确保数据安全性和系统性能。

## 二、后端技术栈与实现

### 2.1 服务端框架：ASP.NET Core

* 框架特点：跨平台、高性能、模块化的开源 Web 框架，支持依赖注入、中间件和容器化部署。
* 核心组件：

控制器：处理 HTTP 请求，返回 JSON 响应。例如，AccountController处理用户认证和信息管理，LookupsController处理查找信息相关操作。

服务层：实现业务逻辑。如UsersService处理用户注册、验证和信息更新。

数据访问层：封装数据库操作，如MembersDataAccess、UserAddressDataAccess、LookupsDataAccess等。

配置管理：通过appsettings.json管理数据库连接、JWT 密钥等配置。

// 用户更新接口实现（AccountController.cs）

[HttpPost("update-user")]

public IActionResult UpdateUser([FromBody] MembersDto members)

{

var userId = User.FindFirst(ClaimTypes.PrimarySid).Value;

var user = \_userService.GetUserById(userId);

// 更新用户信息...

\_userService.Update(user);

return Ok();

}

### 2.2 数据库访问：AdoNetHelper + 存储过程

* 数据访问层：使用AdoNetHelper封装 SQL 操作，通过存储过程实现数据交互，支持参数化查询和事务处理。
* 核心模型：

Members：用户基本信息

UserAddress：用户地址信息

UserContact：用户联系方式

Lookups：查找信息

* 数据映射：通过MapToXXX方法实现数据库记录到实体对象的映射。

// 用户地址数据访问（UserAddressDataAccess.cs）

public List<UserAddress> GetData(int? id, string memberId)

{

var parameters = new List<SqlParameter> { /\* 参数设置 \*/ };

using (SqlDataReader reader = \_dbHelper.ExecuteReader("sp\_Address\_GetData", CommandType.StoredProcedure, parameters.ToArray()))

{

// 数据映射...

}

}

### 2.3 身份验证与授权：JWT

* 认证流程：

用户登录时生成 JWT 令牌（包含用户 ID 和邮箱）

客户端在请求头中携带Authorization: Bearer {token}

服务端验证令牌有效性

* 配置：在Startup.cs中配置 JWT 参数（密钥、发行者、过期时间）。

// JWT生成（AccountController.cs）

var token = new JwtSecurityToken(

issuer: issuer,

audience: audience,

claims: claims,

expires: DateTime.Now.AddMinutes(expirationMinutes),

signingCredentials: creds

);

### 2.4 定时任务：DeleteUnverifiedUsersService

* 功能：每天自动清理 7 天前未验证的用户记录。
* 实现：通过IHostedService接口和Timer实现定时触发，调用存储过程sp\_DeleteUnverifiedUsers。

//定时任务实现（DeleteUnverifiedUsersService.cs）

public Task StartAsync(CancellationToken cancellationToken)

{

\_timer = new Timer(DoWork, null, TimeSpan.Zero, TimeSpan.FromHours(1));

return Task.CompletedTask;

}

## 三、前端技术与实现

### 3.1 前端框架

* 基础技术：HTML、CSS、JavaScript
* 核心库：jQuery（DOM 操作和 AJAX 请求）
* 页面组件：

登录 / 注册页面

用户信息管理页面

地址管理页面

查找信息页面

* 交互逻辑：通过 AJAX 与后端 API 通信，处理表单提交和响应。

javascript

// 登录请求（site.js）

$.ajax({

url: '/api/account/login',

method: 'POST',

data: JSON.stringify({ email: email }),

success: function (response) {

if (response.done === 1) {

localStorage.setItem('token', response.token);

window.location.href = '/home';

}

}

});

## 四、API 接口设计

### 4.1 用户管理接口

| **接口路径** | **方法** | **功能** |
| --- | --- | --- |
| /module/account/login | POST | 用户登录 |
| /module/account/register | POST | 用户注册 |
| /module/account/update-user | POST | 更新用户信息 |
| /module/account/get-user-info/{userId} | GET | 获取用户详情 |

### 4.2 地址管理接口

| **接口路径** | **方法** | **功能** |
| --- | --- | --- |
| /module/account/get-address | GET | 获取用户地址 |
| /module/account/add-address | POST | 添加地址 |
| /module/account/update-address | POST | 更新地址 |
| /module/account/delete-address | DELETE | 删除地址 |

4.3 查找信息接口

| **接口路径** | **方法** | **功能** |
| --- | --- | --- |
| /module/lookups/get-lookups | GET | 获取查找信息 |

## 五、项目架构与部署

### 5.1 架构设计

* 分层架构：

表示层（Controllers）：负责处理 HTTP 请求和返回响应。

业务逻辑层（Services）：实现具体的业务逻辑。

数据访问层（DataAccess）：封装数据库操作。

数据模型（Models）：定义数据结构。

* 依赖注入：通过ASP.NET Core 内置的 DI 容器管理组件依赖。

### 5.2 部署配置

* 环境配置：

appsettings.json：生产环境配置

appsettings.Development.json：开发环境配置

* 连接字符串：配置 SQL Server 数据库连接（包含加密设置）。
* 启动配置：支持 HTTP/HTTPS、Docker 容器部署。

## 六、代码结构与规范

### 6.1 主要目录结构

plaintext

user-info-manager2/

├── Controllers/ # API控制器

├── Data/ # 数据访问层

├── Models/ # 数据模型

├── Service/ # 业务逻辑

├── wwwroot/ # 静态资源

├── appsettings.json # 配置文件

├── Program.cs # 应用入口

├── Startup.cs # 启动配置

└── .gitignore # 版本控制忽略文件

### 6.2 编码规范

* 命名约定：使用 PascalCase 命名类、方法和属性，camelCase 命名参数和局部变量。
* 注释：关键方法和类使用 XML 注释，解释功能和参数。
* 异常处理：使用 try-catch 块捕获数据库操作异常，记录日志。

## 七、系统亮点与优化

* 数据安全：

JWT 身份验证防止未授权访问

参数化查询防止 SQL 注入

定期清理未验证用户数据

* 性能优化：

使用存储过程提升数据库操作效率

异步操作减少线程阻塞

数据库连接池管理

* 可扩展性：

模块化设计支持功能扩展

依赖注入便于替换组件

配置文件分离便于环境切换

### 八、附录：配置示例

### 8.1 数据库连接配置（appsettings.json）

{

"ConnectionStrings": {

"DefaultConnection": "Data Source=170.64.235.76;Initial Catalog=Cobble;User ID=sa;Password=YourStrongPassword123;Integrated Security=False;Persist Security Info=True;Encrypt=True;TrustServerCertificate=True;MultipleActiveResultSets=true;"

}

}

### 8.2 JWT 配置（appsettings.json）

{

"JwtSettings": {

"SecretKey": "67ij4o6jo4i5j6io45j6i4j74p5k6i54ojoi5t9g8ergoj34ofgkrtbmreog894jbioemgropihj48rj4io5juopjgior",

"Issuer": "http://localhost:5083",

"Audience": "http://localhost:5083",

"ExpirationMinutes": 172800

}

}

### 8.3 启动配置（launchSettings.json）

{

"profiles": {

"http": {

"applicationUrl": "http://localhost:5083"

},

"https": {

"applicationUrl": "https://localhost:7011;http://localhost:5083"

}

}

}