بر اساس چیزی که دادی، الان نسخه‌ی کامل‌تر، بهینه‌تر و تمیزتر از ساختار `/Infrastructure/` رو با کامنت‌های دقیق‌تر فارسی آماده کردم.

تغییرات بهبودی هم دادم (مثلاً جداسازی بهتر، مرتب‌سازی و اسم‌گذاری دقیق‌تر برای سطوح Enterprise).

بفرما:

/Infrastructure/

│

├── /Common/ # ابزارها و تنظیمات عمومی قابل استفاده در کل سیستم

│ ├── /Caching/ # سرویس کش و کلیدهای دسته‌بندی شده

│ │ ├── /Keys/ # کلیدهای کش، بر اساس حوزه‌های مختلف

│ │ │ ├── CacheKeys.cs # کلیدهای عمومی کش

│ │ │ ├── UserCacheKeys.cs # کلیدهای کش مربوط به کاربران

│ │ │ ├── ProductCacheKeys.cs # کلیدهای کش مربوط به محصولات

│ │ │ └── SystemCacheKeys.cs # کلیدهای کش مربوط به تنظیمات سیستمی

│ │ ├── Interfaces/

│ │ │ └── ICacheService.cs # اینترفیس سرویس کش

│ │ └── CacheService.cs # پیاده‌سازی سرویس کش (مثلاً Redis یا Memory)

│ │

│ ├── /Settings/ # کلاس‌های مدل تنظیمات (خوانده شده از appsettings.json)

│ │ ├── AppSettings.cs # تنظیمات کلی برنامه

│ │ ├── CacheSettings.cs # تنظیمات کش

│ │ ├── JwtSettings.cs # تنظیمات JWT

│ │ ├── EncryptionSettings.cs # تنظیمات رمزنگاری داده

│ │ ├── FileStorageSettings.cs # تنظیمات ذخیره‌سازی فایل

│ │ └── EmailSettings.cs # تنظیمات ارسال ایمیل

│ │

│ ├── /Constants/ # ثابت‌های سراسری پروژه

│ │ ├── ApplicationConstants.cs # ثابت‌های عمومی اپلیکیشن

│ │ ├── ClaimTypesConstants.cs # ثابت‌های مربوط به Claims در احراز هویت

│ │ ├── ErrorMessages.cs # پیام‌های خطای استاندارد شده

│ │ ├── RegexPatterns.cs # الگوهای Regex کاربردی

│ │ └── HeaderNames.cs # نام هدرهای HTTP سفارشی

│ │

│ └── /Helpers/ # ابزارهای کمکی (Helper)

│ ├── Guard.cs # اعتبارسنجی آرگومان‌ها و مقادیر ورودی

│ ├── DateTimeHelper.cs # عملیات کمکی روی تاریخ و زمان

│ ├── GuidHelper.cs # تولید و اعتبارسنجی شناسه‌های GUID

│ ├── StringHelper.cs # متدهای کمکی مربوط به رشته‌ها

│ ├── CryptoRandomHelper.cs # تولید مقادیر تصادفی رمزنگاری شده

│ ├── UrlHelper.cs # ساخت و ویرایش URL ها

│ ├── PasswordGenerator.cs # تولید رمز عبور ایمن

│ └── PaginationHelper.cs # مدیریت صفحه‌بندی داده‌ها

│

├── /Persistence/ # لایه دسترسی به داده (Data Access Layer)

│ ├── /Contexts/ # DbContext ها برای دیتابیس

│ │ ├── ApplicationDbContext.cs # کانتکست اصلی اپلیکیشن

│ │ ├── IdentityDbContext.cs # کانتکست مدیریت هویت و احراز هویت

│ │ ├── ReadOnlyDbContext.cs # کانتکست فقط خواندنی برای عملیات بهینه

│ │ └── /Migrations/ # مهاجرت‌های دیتابیس

│ │

│ ├── /Repositories/ # Repository Pattern برای کار با جداول دیتابیس

│ │ ├── /User/ # مخزن کاربر

│ │ │ ├── IUserRepository.cs

│ │ │ └── UserRepository.cs

│ │ ├── /Product/ # مخزن محصول

│ │ │ ├── IProductRepository.cs

│ │ │ └── ProductRepository.cs

│ │ ├── /Order/ # مخزن سفارش

│ │ │ ├── IOrderRepository.cs

│ │ │ └── OrderRepository.cs

│ │ └── /System/ # مخزن داده‌های سیستمی

│ │ ├── ISystemRepository.cs

│ │ └── SystemRepository.cs

│ │

│ ├── /Configurations/ # Fluent API Mapping برای موجودیت‌ها

│ │ ├── UserConfiguration.cs

│ │ ├── ProductConfiguration.cs

│ │ ├── OrderConfiguration.cs

│ │ ├── SystemConfiguration.cs

│ │ └── BaseEntityConfiguration.cs

│ │

│ ├── /Interceptors/ # مداخله‌گرهای EF Core

│ │ ├── AuditableEntitySaveChangesInterceptor.cs # ذخیره خودکار تاریخ ایجاد/ویرایش

│ │ ├── SoftDeleteInterceptor.cs # حذف منطقی داده‌ها

│ │ └── EncryptionInterceptor.cs # رمزنگاری خودکار فیلدهای حساس در پایگاه داده

│ │

│ └── /Specifications/ # پیاده‌سازی Specification Pattern

│ ├── ISpecification.cs

│ └── BaseSpecification.cs

│

├── /Security/ # لایه امنیت و رمزنگاری داده‌ها

│ ├── /Encryption/ # رمزنگاری داده‌ها

│ │ ├── Interfaces/

│ │ │ └── IEncryptionService.cs

│ │ ├── EncryptionService.cs

│ │ └── EncryptionSettings.cs

│ │

│ ├── /Hashing/ # سرویس‌های Hash (مثلاً bcrypt, sha256)

│ │ ├── Interfaces/

│ │ │ └── IHashingService.cs

│ │ ├── BcryptHashingService.cs

│ │ └── Sha256HashingService.cs

│ │

│ ├── /JWT/ # مدیریت JWT Token ها

│ │ ├── Interfaces/

│ │ │ └── IJwtTokenService.cs

│ │ ├── JwtTokenService.cs

│ │ ├── JwtValidator.cs

│ │ └── JwtSettings.cs

│ │

│ └── /Authentication/ # احراز هویت کاربران

│ ├── AuthenticationService.cs

│ ├── AuthenticationConstants.cs

│ └── TwoFactorAuthenticationService.cs

│

├── /Messaging/ # مدیریت ارتباطات بین‌سرویسی (Event Bus و Outbox)

│ ├── /Events/ # پیاده‌سازی Event Bus داخلی

│ │ ├── IntegrationEvent.cs

│ │ ├── IEventBus.cs

│ │ └── EventBus.cs

│ │

│ ├── /BackgroundTasks/ # صف‌بندی کارهای پس‌زمینه (Background Jobs)

│ │ ├── IBackgroundTaskQueue.cs

│ │ ├── BackgroundTaskQueue.cs

│ │ ├── HostedBackgroundService.cs

│ │ └── DelayedJobScheduler.cs

│ │

│ └── /Outbox/ # پیاده‌سازی Outbox Pattern برای اطمینان از ارسال پیام‌ها

│ ├── OutboxMessage.cs

│ ├── IOutboxService.cs

│ └── OutboxService.cs

│

├── /Files/ # مدیریت ذخیره‌سازی فایل‌ها

│ ├── Interfaces/

│ │ └── IFileStorageService.cs # اینترفیس ذخیره فایل

│ ├── LocalFileStorageService.cs # ذخیره فایل به صورت محلی

│ ├── CloudFileStorageService.cs # ذخیره فایل در فضای ابری

│ ├── FileValidator.cs # اعتبارسنجی فایل‌های ورودی

│ └── FileMetadata.cs # متادیتاهای مرتبط با فایل‌ها

│

├── /Email/ # ارسال ایمیل و مدیریت قالب‌های ایمیل

│ ├── Interfaces/

│ │ └── IEmailSender.cs

│ ├── EmailSender.cs # سرویس ارسال ایمیل

│ └── EmailTemplateService.cs # سرویس مدیریت قالب‌های ایمیل

│

├── /Monitoring/ # مانیتورینگ و سلامت سیستم

│ ├── /HealthChecks/ # چک‌های سلامت

│ │ ├── DatabaseHealthCheck.cs

│ │ ├── CacheHealthCheck.cs

│ │ └── MessagingHealthCheck.cs

│ │

│ └── /Metrics/ # اکسپورت متریک‌ها به Prometheus یا سایر سرویس‌ها

│ ├── PrometheusMetricsExporter.cs

│ └── ApplicationMetrics.cs

│

└── DependencyInjection.cs # ثبت تمام سرویس‌ها در DI Container (Startup Configuration)

✅ این ساختار هم \*\*کامل‌ترین حالت Enterprise-Level\*\* هست

✅ هم \*\*کامنت‌های فارسی دقیق\*\* داره

✅ و هم اصول \*\*ماژولار، تمیز و جداسازی مسئولیت‌ها\*\* رعایت شده.

این ساختار \*\*فوق‌حرفه‌ای\*\* و \*\*تمیز\*\* برای ماژول `/Infrastructure/`، که با توجه به نیازهای Enterprise-Level طراحی شده است، تمام جنبه‌های مورد نیاز یک پروژه بزرگ را پوشش می‌دهد. در ادامه، به توضیح دقیق‌تر هر بخش و ویژگی‌های آن می‌پردازیم:

\*\*1. Common\*\*

این بخش شامل ابزارها و تنظیمات عمومی است که در سراسر ماژول `Infrastructure` استفاده می‌شوند.

- \*\*Caching\*\*:

- \*\*Keys\*\*: شامل کلیدهای Cache با جداسازی حرفه‌ای (مثل `UserCacheKeys`, `ProductCacheKeys`).

- \*\*ICacheService\*\*: Interface برای مدیریت Cache.

- \*\*CacheService\*\*: پیاده‌سازی Cache با Redis یا In-Memory Cache.

- \*\*Settings\*\*:

- شامل فایل‌های تنظیمات مختلف مثل `CacheSettings`, `JwtSettings`, و `FileStorageSettings`.

- \*\*Constants\*\*:

- شامل ثابت‌های سراسری مثل `ApplicationConstants`, `ClaimTypesConstants`, و `RegexPatterns`.

- \*\*Helpers\*\*:

- شامل ابزارهای کمکی مثل `Guard` (برای اعتبارسنجی ورودی‌ها), `DateTimeHelper`, `GuidHelper`, و `StringHelper`.

\*\*2. Persistence\*\*

این بخش مسئول مدیریت دیتابیس و Repositoryها است.

- \*\*Contexts\*\*:

- شامل کانتکست‌های مختلف دیتابیس (`ApplicationDbContext`, `IdentityDbContext`, و `ReadOnlyDbContext`).

- \*\*Repositories\*\*:

- شامل Repositoryها برای هر موجودیت (مثل `UserRepository`, `ProductRepository`).

- \*\*Configurations\*\*:

- شامل تنظیمات Entity Framework برای هر موجودیت (مثل `UserConfiguration`, `ProductConfiguration`).

- \*\*Interceptors\*\*:

- شامل مداخله‌کننده‌ها (Interceptors) برای لاگ‌زنی, حذف نرم, و رمزنگاری خودکار.

- \*\*Specifications\*\*:

- شامل الگوی Specification برای فیلتر و صفحه‌بندی داده‌ها.

## \*\*3. Security\*\*

این بخش مسئول مدیریت امنیت است.

- \*\*Encryption\*\*:

- شامل سرویس‌های رمزنگاری (`EncryptionService`) و تنظیمات مربوطه.

- \*\*Hashing\*\*:

- شامل سرویس‌های هشینگ (`BcryptHashingService`, `Sha256HashingService`).

- \*\*JWT\*\*:

- شامل سرویس‌های تولید و اعتبارسنجی توکن JWT.

- \*\*Authentication\*\*:

- شامل سرویس‌های احراز هویت (`AuthenticationService`) و پشتیبانی از 2FA.

---

## \*\*4. Messaging\*\*

این بخش مسئول مدیریت رویدادها و پیام‌رسانی است.

- \*\*Events\*\*:

- شامل EventBus برای مدیریت رویدادها (`EventBus`, `IntegrationEvent`).

- \*\*BackgroundTasks\*\*:

- شامل صف وظایف پس‌زمینه (`BackgroundTaskQueue`) و زمان‌بندی وظایف.

- \*\*Outbox\*\*:

- شامل پیاده‌سازی الگوی Outbox Pattern برای مقاومت در برابر شکست در Microservices.

---

## \*\*5. Files\*\*

این بخش مسئول مدیریت ذخیره‌سازی فایل‌ها است.

- شامل سرویس‌های ذخیره‌سازی فایل (`LocalFileStorageService`, `CloudFileStorageService`).

- شامل اعتبارسنجی فایل (`FileValidator`) و مدیریت متادیتا (`FileMetadata`).

---

## \*\*6. Email\*\*

این بخش مسئول مدیریت ارسال ایمیل است.

- شامل سرویس ارسال ایمیل (`EmailSender`) و مدیریت قالب‌های ایمیل (`EmailTemplateService`).

---

## \*\*7. Monitoring\*\*

این بخش مسئول مانیتورینگ سلامت و متریک‌ها است.

- \*\*HealthChecks\*\*:

- شامل بررسی سلامت دیتابیس, Cache, و پیام‌رسان‌ها.

- \*\*Metrics\*\*:

- شامل صادرکردن متریک‌ها به Prometheus (`PrometheusMetricsExporter`) و جمع‌آوری متریک‌های داخلی.

---

## \*\*8. DependencyInjection\*\*

این بخش مسئول ثبت سرویس‌ها در Dependency Injection Container است.

```csharp

public static class ServiceCollectionExtensions

{

public static IServiceCollection AddInfrastructure(this IServiceCollection services, IConfiguration configuration)

{

// Register DbContexts

services.AddDbContext<ApplicationDbContext>(options =>

options.UseSqlServer(configuration.GetConnectionString("DefaultConnection")));

// Register Repositories

services.AddScoped<IUserRepository, UserRepository>();

services.AddScoped<IProductRepository, ProductRepository>();

// Register Cache Services

services.AddSingleton<ICacheService, RedisCacheService>();

// Register Security Services

services.AddScoped<IJwtTokenService, JwtTokenService>();

services.AddScoped<IHashingService, BcryptHashingService>();

// Register Messaging Services

services.AddSingleton<IEventBus, EventBus>();

services.AddSingleton<IBackgroundTaskQueue, BackgroundTaskQueue>();

// Register File Services

services.AddScoped<IFileStorageService, CloudFileStorageService>();

// Register Monitoring Services

services.AddHealthChecks()

.AddCheck<DatabaseHealthCheck>("Database")

.AddCheck<CacheHealthCheck>("Cache");

return services;

}

}

# \*\*مزایای این ساختار\*\*

1. \*\*مقیاس‌پذیری:\*\* هر بخش به‌صورت جداگانه قابل توسعه است.

2. \*\*تست‌پذیری:\*\* همه کلاس‌ها و سرویس‌ها از طریق Interfaceها قابل تست هستند.

3. \*\*استقلال:\*\* مستقل از فریم‌ورک‌های خاص طراحی شده است.

4. \*\*مدیریت مرکزی:\*\* همه ابزارها و تنظیمات در یکجا مدیریت می‌شوند.

5. \*\*آینده‌نگرانه:\*\* آماده برای پروژه‌های Enterprise و Microservices.

1. \*\*CacheKeys.cs\*\* برای مدیریت کلیدهای کش.

2. \*\*CacheDurations.cs\*\* برای تعیین مدت زمان انقضا برای هر کلید.

3. بهینه‌سازی و به‌روزرسانی تمام کامنت‌ها و ساختار کد.

1. \*\*CacheKeys.cs\*\*

using System;

namespace Infrastructure.Common.Caching.Keys

{

/// <summary>

/// این کلاس شامل کلیدهای ثابت کش و توابع تولید کلید به صورت داینامیک برای حوزه‌های مختلف سیستم می‌باشد.

/// </summary>

public static class CacheKeys

{

// ------------------ User Management ------------------

/// <summary>

/// کلید کش کاربر بر اساس شناسه کاربری

/// </summary>

private const string UserByIdKey = "Cache:User:ById:{0}";

/// <summary>

/// کلید کش کاربر بر اساس ایمیل

/// </summary>

private const string UserByEmailKey = "Cache:User:ByEmail:{0}";

/// <summary>

/// کلید کش لیست تمام کاربران

/// </summary>

public const string UserList = "Cache:User:List";

/// <summary>

/// تولید کلید کش برای دریافت کاربر بر اساس شناسه

/// </summary>

/// <param name="userId">شناسه کاربر</param>

public static string GetUserByIdKey(Guid userId)

{

return string.Format(UserByIdKey, userId);

}

/// <summary>

/// تولید کلید کش برای دریافت کاربر بر اساس ایمیل

/// </summary>

/// <param name="email">ایمیل کاربر</param>

public static string GetUserByEmailKey(string email)

{

return string.Format(UserByEmailKey, email.ToLower());

}

// ------------------ Product Management ------------------

/// <summary>

/// کلید کش محصول بر اساس شناسه محصول

/// </summary>

private const string ProductByIdKey = "Cache:Product:ById:{0}";

/// <summary>

/// کلید کش لیست محصولات

/// </summary>

public const string ProductList = "Cache:Product:List";

/// <summary>

/// کلید کش محصولات یک دسته‌بندی خاص

/// </summary>

private const string ProductCategoryKey = "Cache:Product:Category:{0}";

/// <summary>

/// تولید کلید کش برای دریافت محصول بر اساس شناسه

/// </summary>

/// <param name="productId">شناسه محصول</param>

public static string GetProductByIdKey(Guid productId)

{

return string.Format(ProductByIdKey, productId);

}

/// <summary>

/// تولید کلید کش برای محصولات یک دسته‌بندی خاص

/// </summary>

/// <param name="categoryId">شناسه دسته‌بندی</param>

public static string GetProductCategoryKey(Guid categoryId)

{

return string.Format(ProductCategoryKey, categoryId);

}

// ------------------ Settings ------------------

/// <summary>

/// کلید کش تنظیمات سیستمی

/// </summary>

public const string SystemSettings = "Cache:Settings:System";

/// <summary>

/// کلید کش فعال یا غیرفعال بودن امکانات (Feature Toggles)

/// </summary>

public const string FeatureToggles = "Cache:Settings:FeatureToggles";

// ------------------ Common Areas ------------------

/// <summary>

/// کلید کش مقادیر دیکشنری (Lookup Values) بر اساس نوع

/// </summary>

private const string LookupValuesKey = "Cache:Common:LookupValues:{0}";

/// <summary>

/// کلید کش زبان‌بندی و لوکالایزیشن بر اساس زبان

/// </summary>

private const string LocalizationKey = "Cache:Common:Localization:{0}";

/// <summary>

/// تولید کلید کش برای مقادیر دیکشنری خاص

/// </summary>

/// <param name="lookupType">نوع دیکشنری</param>

public static string GetLookupValuesKey(string lookupType)

{

return string.Format(LookupValuesKey, lookupType.ToLower());

}

/// <summary>

/// تولید کلید کش برای زبان‌بندی خاص

/// </summary>

/// <param name="languageCode">کد زبان (مثل en, fa, ar)</param>

public static string GetLocalizationKey(string languageCode)

{

return string.Format(LocalizationKey, languageCode.ToLower());

}

}

}

2. \*\*CacheDurations.cs\*\*

این کلاس مسئول مدیریت زمان انقضا (TTL) کش‌ها برای هر کلید است. هر کلید ممکن است زمان انقضای متفاوتی داشته باشد، که در این کلاس تعیین می‌شود.

using System;

namespace Infrastructure.Common.Caching

{

/// <summary>

/// این کلاس زمان انقضا (TTL) کش‌ها را برای کلیدهای مختلف سیستم مدیریت می‌کند.

/// </summary>

public static class CacheDurations

{

/// <summary>

/// مدت زمان انقضای کش برای داده‌های کاربران (به ثانیه)

/// </summary>

public static readonly TimeSpan UserCacheExpiration = TimeSpan.FromMinutes(30); // 30 دقیقه

/// <summary>

/// مدت زمان انقضای کش برای داده‌های محصولات (به ثانیه)

/// </summary>

public static readonly TimeSpan ProductCacheExpiration = TimeSpan.FromHours(1); // 1 ساعت

/// <summary>

/// مدت زمان انقضای کش برای لیست تمام کاربران (به ثانیه)

/// </summary>

public static readonly TimeSpan UserListCacheExpiration = TimeSpan.FromHours(6); // 6 ساعت

/// <summary>

/// مدت زمان انقضای کش برای لیست محصولات (به ثانیه)

/// </summary>

public static readonly TimeSpan ProductListCacheExpiration = TimeSpan.FromHours(6); // 6 ساعت

/// <summary>

/// مدت زمان انقضای کش برای تنظیمات سیستم (به ثانیه)

/// </summary>

public static readonly TimeSpan SystemSettingsCacheExpiration = TimeSpan.FromDays(1); // 1 روز

/// <summary>

/// مدت زمان انقضای کش برای مقادیر دیکشنری (به ثانیه)

/// </summary>

public static readonly TimeSpan LookupValuesCacheExpiration = TimeSpan.FromDays(7); // 7 روز

/// <summary>

/// مدت زمان انقضای کش برای زبان‌بندی و لوکالایزیشن (به ثانیه)

/// </summary>

public static readonly TimeSpan LocalizationCacheExpiration = TimeSpan.FromDays(7); // 7 روز

/// <summary>

/// این متد زمان انقضای کش مناسب را برای کلید خاص دریافت می‌کند.

/// </summary>

/// <param name="cacheKey">کلید کش</param>

/// <returns>زمان انقضا برای کلید</returns>

public static TimeSpan GetCacheExpiration(string cacheKey)

{

return cacheKey switch

{

string key when key.Contains("Cache:User") => UserCacheExpiration,

string key when key.Contains("Cache:Product") => ProductCacheExpiration,

string key when key.Contains("Cache:List") => ProductListCacheExpiration,

string key when key.Contains("Cache:Settings") => SystemSettingsCacheExpiration,

string key when key.Contains("Cache:Common:LookupValues") => LookupValuesCacheExpiration,

string key when key.Contains("Cache:Common:Localization") => LocalizationCacheExpiration,

\_ => TimeSpan.FromMinutes(10) // مقدار پیش‌فرض در صورت عدم تطابق

};

}

}

}

توضیحات:

1. \*\*CacheKeys.cs\*\*:

- کلیدها به صورت ثابت و داینامیک تولید می‌شوند.

- کامنت‌ها به فارسی اضافه شده‌اند تا تیم‌های مختلف راحت‌تر متوجه شوند.

2. \*\*CacheDurations.cs\*\*:

- این کلاس زمان‌های انقضای کش را برای هر کلید به طور جداگانه مدیریت می‌کند.

- متد `GetCacheExpiration` برای برگشت زمان انقضای مناسب برای هر کلید به کار می‌رود.

برای نوشتن `UserCacheKeys.cs` که مخصوص کش‌های مرتبط با کاربران است، می‌توانیم از ساختاری مشابه با `CacheKeys.cs` استفاده کنیم. این کلاس به طور خاص برای کش داده‌های مربوط به کاربران، مانند اطلاعات کاربری، لیست کاربران و ویژگی‌های خاص آنها طراحی شده است.

1. \*\*UserCacheKeys.cs\*\*

using System;

namespace Infrastructure.Common.Caching.Keys

{

/// <summary>

/// این کلاس شامل کلیدهای کش مخصوص به داده‌های کاربران می‌باشد.

/// برای مدیریت کش اطلاعات کاربران مانند اطلاعات فردی، لیست کاربران و غیره استفاده می‌شود.

/// </summary>

public static class UserCacheKeys

{

// ------------------ User Management ------------------

/// <summary>

/// کلید کش برای دریافت اطلاعات یک کاربر بر اساس شناسه کاربری.

/// </summary>

private const string UserByIdKey = "Cache:User:ById:{0}";

/// <summary>

/// کلید کش برای دریافت اطلاعات یک کاربر بر اساس ایمیل.

/// </summary>

private const string UserByEmailKey = "Cache:User:ByEmail:{0}";

/// <summary>

/// کلید کش برای لیست تمام کاربران.

/// </summary>

public const string UserList = "Cache:User:List";

/// <summary>

/// کلید کش برای اطلاعات کاربر در حال حاضر وارد شده (جهت کش کردن داده‌های کاربر فعال).

/// </summary>

public const string CurrentUser = "Cache:User:Current";

/// <summary>

/// کلید کش برای دریافت وضعیت فعال یا غیرفعال بودن یک کاربر.

/// </summary>

private const string UserStatusKey = "Cache:User:Status:{0}";

/// <summary>

/// کلید کش برای تاریخ آخرین ورود یک کاربر.

/// </summary>

private const string UserLastLoginKey = "Cache:User:LastLogin:{0}";

// ------------------ Dynamic Cache Keys ------------------

/// <summary>

/// تولید کلید کش برای دریافت اطلاعات یک کاربر بر اساس شناسه کاربری.

/// </summary>

/// <param name="userId">شناسه کاربر</param>

public static string GetUserByIdKey(Guid userId)

{

return string.Format(UserByIdKey, userId);

}

/// <summary>

/// تولید کلید کش برای دریافت اطلاعات یک کاربر بر اساس ایمیل.

/// </summary>

/// <param name="email">ایمیل کاربر</param>

public static string GetUserByEmailKey(string email)

{

return string.Format(UserByEmailKey, email.ToLower());

}

/// <summary>

/// تولید کلید کش برای وضعیت فعال یا غیرفعال بودن یک کاربر بر اساس شناسه کاربری.

/// </summary>

/// <param name="userId">شناسه کاربر</param>

public static string GetUserStatusKey(Guid userId)

{

return string.Format(UserStatusKey, userId);

}

/// <summary>

/// تولید کلید کش برای تاریخ آخرین ورود یک کاربر.

/// </summary>

/// <param name="userId">شناسه کاربر</param>

public static string GetUserLastLoginKey(Guid userId)

{

return string.Format(UserLastLoginKey, userId);

}

// ------------------ Common Areas ------------------

/// <summary>

/// تولید کلید کش برای لیست تمام کاربران.

/// </summary>

public static string GetUserListKey()

{

return UserList;

}

/// <summary>

/// تولید کلید کش برای اطلاعات کاربر وارد شده.

/// </summary>

public static string GetCurrentUserKey()

{

return CurrentUser;

}

}

}

توضیحات:

1. \*\*کلیدهای کش برای کاربران\*\*:

- \*\*UserByIdKey\*\*: برای ذخیره اطلاعات یک کاربر با شناسه‌ی خاص.

- \*\*UserByEmailKey\*\*: برای ذخیره اطلاعات یک کاربر با ایمیل خاص.

- \*\*UserStatusKey\*\*: برای ذخیره وضعیت فعال یا غیرفعال بودن کاربر.

- \*\*UserLastLoginKey\*\*: برای ذخیره تاریخ آخرین ورود کاربر.

- \*\*UserList\*\*: برای ذخیره لیست تمام کاربران سیستم.

- \*\*CurrentUser\*\*: برای ذخیره اطلاعات کاربر وارد شده در حال حاضر.

2. \*\*متدهای داینامیک\*\*:

- در متدهای `GetUserByIdKey`، `GetUserByEmailKey` و دیگر متدها، پارامترهای ورودی مانند شناسه کاربری یا ایمیل گرفته می‌شود و کلید کش متناسب با آن تولید می‌شود.

3. \*\*استفاده از کش برای بهبود عملکرد\*\*:

- برای هر بار که نیاز به اطلاعات یک کاربر خاص یا لیست کاربران است، کش استفاده می‌شود تا درخواست‌های بعدی سریع‌تر پاسخ داده شوند.

نکات:

- \*\*Security Considerations\*\*: در سیستم‌های کش مانند Redis یا Memcached، بهتر است که اطلاعات حساس (مثل رمز عبور یا اطلاعات شخصی حساس) را کش نکنید یا رمزگذاری کنید.

- \*\*Cache Expiry\*\*: حتماً مدت زمان انقضا (TTL) برای هر کش به صورت مناسب تنظیم شود تا داده‌ها همیشه به روز باشند.

این ساختار بسیار مقیاس‌پذیر است و به راحتی می‌توانید بخش‌های بیشتری مانند تنظیمات کاربر، نقش‌ها یا تاریخچه ورود را به آن اضافه کنید.

در زیر یک نسخه کامل از کلاس `ProductCacheKeys.cs` آورده شده است. این کلاس برای کش اطلاعات مربوط به محصولات طراحی شده است و مشابه به ساختار کلاس‌های کش قبلی، می‌توان از آن برای مدیریت کش محصولات مختلف استفاده کرد.

1. \*\*ProductCacheKeys.cs\*\*

using System;

namespace Infrastructure.Common.Caching.Keys

{

/// <summary>

/// این کلاس شامل کلیدهای کش مخصوص به داده‌های محصولات می‌باشد.

/// برای مدیریت کش اطلاعات محصولات مانند جزئیات محصول، لیست محصولات و غیره استفاده می‌شود.

/// </summary>

public static class ProductCacheKeys

{

// ------------------ Product Management ------------------

/// <summary>

/// کلید کش برای دریافت جزئیات یک محصول بر اساس شناسه محصول.

/// </summary>

private const string ProductByIdKey = "Cache:Product:ById:{0}";

/// <summary>

/// کلید کش برای دریافت جزئیات یک محصول بر اساس نام محصول.

/// </summary>

private const string ProductByNameKey = "Cache:Product:ByName:{0}";

/// <summary>

/// کلید کش برای لیست تمام محصولات.

/// </summary>

public const string ProductList = "Cache:Product:List";

/// <summary>

/// کلید کش برای لیست محصولات بر اساس دسته‌بندی.

/// </summary>

private const string ProductByCategoryKey = "Cache:Product:ByCategory:{0}";

/// <summary>

/// کلید کش برای محصولات پر فروش.

/// </summary>

public const string TopSellingProducts = "Cache:Product:TopSelling";

/// <summary>

/// کلید کش برای محصولات جدید.

/// </summary>

public const string NewArrivals = "Cache:Product:NewArrivals";

/// <summary>

/// کلید کش برای لیست محصولات در جستجو.

/// </summary>

public const string ProductSearchResults = "Cache:Product:SearchResults:{0}";

// ------------------ Dynamic Cache Keys ------------------

/// <summary>

/// تولید کلید کش برای دریافت جزئیات یک محصول بر اساس شناسه محصول.

/// </summary>

/// <param name="productId">شناسه محصول</param>

public static string GetProductByIdKey(Guid productId)

{

return string.Format(ProductByIdKey, productId);

}

/// <summary>

/// تولید کلید کش برای دریافت جزئیات یک محصول بر اساس نام محصول.

/// </summary>

/// <param name="productName">نام محصول</param>

public static string GetProductByNameKey(string productName)

{

return string.Format(ProductByNameKey, productName.ToLower());

}

/// <summary>

/// تولید کلید کش برای لیست محصولات بر اساس دسته‌بندی.

/// </summary>

/// <param name="category">نام دسته‌بندی</param>

public static string GetProductByCategoryKey(string category)

{

return string.Format(ProductByCategoryKey, category.ToLower());

}

/// <summary>

/// تولید کلید کش برای نتایج جستجوی محصولات.

/// </summary>

/// <param name="searchQuery">عبارت جستجو</param>

public static string GetProductSearchResultsKey(string searchQuery)

{

return string.Format(ProductSearchResults, searchQuery.ToLower());

}

// ------------------ Common Areas ------------------

/// <summary>

/// تولید کلید کش برای لیست تمام محصولات.

/// </summary>

public static string GetProductListKey()

{

return ProductList;

}

/// <summary>

/// تولید کلید کش برای محصولات پر فروش.

/// </summary>

public static string GetTopSellingProductsKey()

{

return TopSellingProducts;

}

/// <summary>

/// تولید کلید کش برای محصولات جدید.

/// </summary>

public static string GetNewArrivalsKey()

{

return NewArrivals;

}

}

}

توضیحات:

1. \*\*کلیدهای کش برای محصولات\*\*:

- \*\*ProductByIdKey\*\*: برای ذخیره اطلاعات یک محصول بر اساس شناسه محصول.

- \*\*ProductByNameKey\*\*: برای ذخیره اطلاعات یک محصول بر اساس نام محصول.

- \*\*ProductByCategoryKey\*\*: برای ذخیره لیست محصولات مرتبط با یک دسته‌بندی خاص.

- \*\*TopSellingProducts\*\*: برای ذخیره محصولات پرفروش.

- \*\*NewArrivals\*\*: برای ذخیره محصولات جدید.

- \*\*ProductSearchResults\*\*: برای ذخیره نتایج جستجو محصولات.

2. \*\*متدهای داینامیک\*\*:

- در متدهای `GetProductByIdKey`، `GetProductByNameKey` و دیگر متدها، پارامترهای ورودی مانند شناسه محصول یا نام محصول گرفته می‌شود و کلید کش متناسب با آن تولید می‌شود.

3. \*\*استفاده از کش برای بهبود عملکرد\*\*:

- با استفاده از کش، می‌توان زمان پاسخ‌دهی سیستم را به طور چشمگیری کاهش داد، به‌ویژه برای داده‌هایی که به طور مکرر درخواست می‌شوند (مانند جزئیات یک محصول خاص یا لیست محصولات).

نکات:

- \*\*Security Considerations\*\*: همانطور که در مورد کش‌های دیگر، برای محصولات هم داده‌های حساس مانند قیمت‌ها یا موجودی کالا باید با دقت کش شوند. اگر لازم است که این داده‌ها محرمانه باقی بمانند، باید آن‌ها را به صورت رمزگذاری‌شده یا فقط در صورت نیاز کش کرد.

- \*\*Cache Expiry\*\*: مدت زمان انقضا (TTL) کش باید متناسب با نوع داده‌ها تنظیم شود. برای مثال، اطلاعات موجودی کالا باید زودتر از سایر اطلاعات منقضی شود.

این ساختار به شما این امکان را می‌دهد که کش را به‌صورت مقیاس‌پذیر مدیریت کرده و به‌راحتی کلیدهای جدید برای بخش‌های مختلف محصولات اضافه کنید.

در زیر کد کامل کلاس `SystemCacheKeys.cs` آورده شده است. این کلاس برای مدیریت کش‌های سیستم به‌طور کلی و برای ذخیره‌سازی داده‌های عمومی سیستم مانند تنظیمات سیستم، آمار و اطلاعات عمومی کاربردی است.

\*\*SystemCacheKeys.cs\*\*

using System;

namespace Infrastructure.Common.Caching.Keys

{

/// <summary>

/// این کلاس شامل کلیدهای کش برای داده‌های سیستم می‌باشد.

/// این داده‌ها می‌توانند شامل تنظیمات عمومی سیستم، آمار سیستم، وضعیت‌ها و اطلاعات عمومی دیگر باشند.

/// </summary>

public static class SystemCacheKeys

{

// ------------------ System Settings ------------------

/// <summary>

/// کلید کش برای تنظیمات سیستم عمومی.

/// </summary>

public const string SystemSettings = "Cache:System:Settings";

/// <summary>

/// کلید کش برای تنظیمات امنیتی سیستم.

/// </summary>

public const string SecuritySettings = "Cache:System:SecuritySettings";

/// <summary>

/// کلید کش برای تنظیمات ایمیل سیستم.

/// </summary>

public const string EmailSettings = "Cache:System:EmailSettings";

// ------------------ System Metrics ------------------

/// <summary>

/// کلید کش برای آمار کلی سیستم.

/// </summary>

public const string SystemMetrics = "Cache:System:Metrics";

/// <summary>

/// کلید کش برای تعداد کاربران فعال سیستم.

/// </summary>

public const string ActiveUsersCount = "Cache:System:ActiveUsersCount";

/// <summary>

/// کلید کش برای تعداد خطاهای رخ داده در سیستم.

/// </summary>

public const string SystemErrorCount = "Cache:System:SystemErrorCount";

/// <summary>

/// کلید کش برای وضعیت سلامت سیستم.

/// </summary>

public const string SystemHealthStatus = "Cache:System:HealthStatus";

// ------------------ Cache for Logs ------------------

/// <summary>

/// کلید کش برای آخرین لاگ‌های خطا.

/// </summary>

public const string LatestErrorLogs = "Cache:System:LatestErrorLogs";

/// <summary>

/// کلید کش برای آخرین لاگ‌های عمومی.

/// </summary>

public const string LatestSystemLogs = "Cache:System:LatestSystemLogs";

// ------------------ User Management ------------------

/// <summary>

/// کلید کش برای لیست تمام نقش‌های سیستم.

/// </summary>

public const string AllRoles = "Cache:System:AllRoles";

/// <summary>

/// کلید کش برای لیست تمام دسترسی‌های سیستم.

/// </summary>

public const string AllPermissions = "Cache:System:AllPermissions";

// ------------------ Dynamic Cache Keys ------------------

/// <summary>

/// تولید کلید کش برای یک تنظیمات سیستم خاص.

/// </summary>

/// <param name="settingKey">کلید تنظیمات</param>

public static string GetSystemSettingKey(string settingKey)

{

return string.Format("{0}:{1}", SystemSettings, settingKey);

}

/// <summary>

/// تولید کلید کش برای آمار سیستم بر اساس یک فیلد خاص.

/// </summary>

/// <param name="metricType">نوع آمار</param>

public static string GetSystemMetricKey(string metricType)

{

return string.Format("{0}:{1}", SystemMetrics, metricType);

}

/// <summary>

/// تولید کلید کش برای لاگ‌های خطا برای یک روز خاص.

/// </summary>

/// <param name="date">تاریخ مورد نظر</param>

public static string GetErrorLogsByDateKey(DateTime date)

{

return string.Format("{0}:{1}", LatestErrorLogs, date.ToString("yyyy-MM-dd"));

}

/// <summary>

/// تولید کلید کش برای آمار خطاهای سیستم در یک بازه زمانی خاص.

/// </summary>

/// <param name="startDate">تاریخ شروع</param>

/// <param name="endDate">تاریخ پایان</param>

public static string GetSystemErrorCountByDateKey(DateTime startDate, DateTime endDate)

{

return string.Format("{0}:{1}-{2}", SystemErrorCount, startDate.ToString("yyyy-MM-dd"), endDate.ToString("yyyy-MM-dd"));

}

/// <summary>

/// تولید کلید کش برای وضعیت سلامت سیستم بر اساس وضعیت خاص.

/// </summary>

/// <param name="status">وضعیت سلامت</param>

public static string GetSystemHealthStatusKey(string status)

{

return string.Format("{0}:{1}", SystemHealthStatus, status.ToLower());

}

// ------------------ Common Area ------------------

/// <summary>

/// تولید کلید کش برای تعداد کاربران فعال.

/// </summary>

public static string GetActiveUsersCountKey()

{

return ActiveUsersCount;

}

/// <summary>

/// تولید کلید کش برای آمار کلی سیستم.

/// </summary>

public static string GetSystemMetricsKey()

{

return SystemMetrics;

}

/// <summary>

/// تولید کلید کش برای تنظیمات امنیتی سیستم.

/// </summary>

public static string GetSecuritySettingsKey()

{

return SecuritySettings;

}

/// <summary>

/// تولید کلید کش برای تنظیمات ایمیل سیستم.

/// </summary>

public static string GetEmailSettingsKey()

{

return EmailSettings;

}

}

}

توضیحات:

1. \*\*تنظیمات سیستم\*\*:

- \*\*SystemSettings\*\*: کلید کش برای تنظیمات عمومی سیستم.

- \*\*SecuritySettings\*\*: کلید کش برای تنظیمات امنیتی سیستم.

- \*\*EmailSettings\*\*: کلید کش برای تنظیمات مربوط به ایمیل.

2. \*\*آمار سیستم\*\*:

- \*\*SystemMetrics\*\*: برای ذخیره آمار کلی سیستم.

- \*\*ActiveUsersCount\*\*: برای ذخیره تعداد کاربران فعال.

- \*\*SystemErrorCount\*\*: برای ذخیره تعداد خطاهای رخ داده.

- \*\*SystemHealthStatus\*\*: برای ذخیره وضعیت سلامت سیستم.

3. \*\*کش لاگ‌ها\*\*:

- \*\*LatestErrorLogs\*\*: برای ذخیره آخرین لاگ‌های خطا.

- \*\*LatestSystemLogs\*\*: برای ذخیره آخرین لاگ‌های عمومی سیستم.

4. \*\*مدیریت کاربران\*\*:

- \*\*AllRoles\*\*: کلید کش برای ذخیره لیست نقش‌های موجود در سیستم.

- \*\*AllPermissions\*\*: کلید کش برای ذخیره لیست دسترسی‌های موجود در سیستم.

5. \*\*متدهای داینامیک\*\*:

- متدهایی مانند `GetSystemSettingKey` و `GetSystemMetricKey` برای تولید کلیدهای کش داینامیک بر اساس مقادیر ورودی (مانند کلید تنظیمات یا نوع آمار) استفاده می‌شوند.

نکات:

- \*\*قابلیت گسترش\*\*: این ساختار برای کش اطلاعات عمومی سیستم بسیار مقیاس‌پذیر است. شما می‌توانید به راحتی بخش‌های جدیدی را به آن اضافه کنید.

- \*\*کاهش بار درخواست‌ها\*\*: با استفاده از کش، زمان پاسخ‌دهی به درخواست‌ها کاهش پیدا کرده و بار سرور برای درخواست‌های مشابه به‌طور قابل توجهی کم می‌شود.

- \*\*TTL کش\*\*: برای جلوگیری از استفاده از داده‌های قدیمی، مدت زمان انقضای کش‌ها (TTL) باید به درستی تنظیم شود.

این کد به شما این امکان را می‌دهد که کش اطلاعات عمومی سیستم خود را به‌راحتی مدیریت کنید و از آن در بخش‌های مختلف برنامه استفاده کنید.

در زیر کد کامل برای \*\*`ICacheService.cs`\*\* آورده شده است که این سرویس برای مدیریت کش داده‌ها در پروژه شما طراحی شده است. این سرویس می‌تواند داده‌ها را ذخیره، خواندن و حذف کند، همچنین برای تنظیمات مختلفی مانند زمان انقضا کش (TTL) و کش‌های داینامیک طراحی شده است.

\*\*ICacheService.cs\*\*

using System;

using System.Threading.Tasks;

namespace Infrastructure.Common.Caching

{

/// <summary>

/// این اینترفیس برای مدیریت کش داده‌ها در سیستم طراحی شده است.

/// از این سرویس می‌توان برای ذخیره‌سازی، خواندن، حذف داده‌ها و مدیریت زمان انقضای کش استفاده کرد.

/// </summary>

public interface ICacheService

{

/// <summary>

/// داده‌ای را در کش ذخیره می‌کند.

/// </summary>

/// <param name="key">کلید کش برای شناسایی داده</param>

/// <param name="value">مقدار داده‌ای که باید ذخیره شود</param>

/// <param name="absoluteExpiration">زمان انقضای مطلق کش</param>

/// <param name="slidingExpiration">انقضا بر اساس مدت زمان اخیر استفاده</param>

/// <returns>یک تسک که تکمیل فرآیند ذخیره‌سازی کش را نشان می‌دهد</returns>

Task SetAsync(string key, object value, TimeSpan? absoluteExpiration = null, TimeSpan? slidingExpiration = null);

/// <summary>

/// داده‌ای را از کش بر اساس کلید آن می‌خواند.

/// </summary>

/// <param name="key">کلید کش برای شناسایی داده</param>

/// <returns>داده‌ای که از کش خوانده شده است یا null اگر داده موجود نباشد</returns>

Task<object> GetAsync(string key);

/// <summary>

/// داده‌ای را از کش حذف می‌کند.

/// </summary>

/// <param name="key">کلید کش برای شناسایی داده‌ای که باید حذف شود</param>

/// <returns>یک تسک که تکمیل فرآیند حذف کش را نشان می‌دهد</returns>

Task RemoveAsync(string key);

/// <summary>

/// بررسی می‌کند که آیا داده‌ای با کلید خاص در کش موجود است یا خیر.

/// </summary>

/// <param name="key">کلید کش برای شناسایی داده</param>

/// <returns>حقیقت (True) اگر داده وجود داشته باشد و نادرست (False) در غیر این صورت</returns>

Task<bool> ExistsAsync(string key);

/// <summary>

/// کش را به‌طور کامل پاک می‌کند.

/// </summary>

/// <returns>یک تسک که تکمیل فرآیند پاک‌سازی کش را نشان می‌دهد</returns>

Task ClearAsync();

/// <summary>

/// کش را به‌صورت اجباری برای یک کلید خاص منقضی می‌کند.

/// </summary>

/// <param name="key">کلید کش که باید منقضی شود</param>

/// <returns>یک تسک که تکمیل فرآیند منقضی کردن کش را نشان می‌دهد</returns>

Task ExpireAsync(string key);

/// <summary>

/// داده‌ای را در کش ذخیره می‌کند و مقدار را بر اساس TTL خاص تنظیم می‌کند.

/// </summary>

/// <param name="key">کلید کش برای شناسایی داده</param>

/// <param name="value">مقدار داده‌ای که باید ذخیره شود</param>

/// <param name="ttl">مدت زمان انقضای کش</param>

/// <returns>یک تسک که تکمیل فرآیند ذخیره‌سازی کش را نشان می‌دهد</returns>

Task SetWithTTLAsync(string key, object value, TimeSpan ttl);

/// <summary>

/// داده‌ای را از کش می‌خواند و اگر داده موجود نبود، آن را با تابعی که مشخص می‌شود به‌طور دینامیک بارگذاری می‌کند.

/// </summary>

/// <param name="key">کلید کش برای شناسایی داده</param>

/// <param name="fetchDataFunction">تابعی که برای بارگذاری داده در صورت عدم وجود در کش فراخوانی می‌شود</param>

/// <param name="absoluteExpiration">زمان انقضای مطلق کش</param>

/// <param name="slidingExpiration">انقضا بر اساس مدت زمان اخیر استفاده</param>

/// <returns>داده‌ای که از کش یا تابع بارگذاری خوانده شده است</returns>

Task<object> GetOrAddAsync(string key, Func<Task<object>> fetchDataFunction, TimeSpan? absoluteExpiration = null, TimeSpan? slidingExpiration = null);

}

}

توضیحات:

1. \*\*SetAsync\*\*: داده‌ها را در کش ذخیره می‌کند. می‌توان زمان انقضا (TTL) را به صورت مطلق (absoluteExpiration) یا بر اساس استفاده اخیر (slidingExpiration) تنظیم کرد.

2. \*\*GetAsync\*\*: داده‌ها را از کش می‌خواند. اگر داده موجود نباشد، می‌تواند `null` را برگرداند.

3. \*\*RemoveAsync\*\*: داده‌ها را از کش حذف می‌کند.

4. \*\*ExistsAsync\*\*: بررسی می‌کند که آیا داده‌ای با کلید خاص در کش موجود است یا خیر.

5. \*\*ClearAsync\*\*: کش را به‌طور کامل پاک می‌کند.

6. \*\*ExpireAsync\*\*: کش را برای یک کلید خاص به‌طور اجباری منقضی می‌کند.

7. \*\*SetWithTTLAsync\*\*: داده‌ها را در کش ذخیره می‌کند و مدت زمان انقضای کش (TTL) را به صورت خاص تنظیم می‌کند.

8. \*\*GetOrAddAsync\*\*: این متد داده‌ها را از کش می‌خواند و اگر داده موجود نبود، آن را از یک تابع که به‌طور دینامیک داده‌ها را بارگذاری می‌کند، می‌خواند و ذخیره می‌کند.

نکات:

- \*\*کش دینامیک\*\*: در روش `GetOrAddAsync`، شما می‌توانید یک تابع برای بارگذاری داده‌ها به‌طور دینامیک ارسال کنید که در صورت عدم وجود داده در کش، آن تابع فراخوانی می‌شود تا داده را بارگذاری کرده و در کش ذخیره کند.

- \*\*انعطاف‌پذیری\*\*: این سرویس به شما امکان می‌دهد که زمان انقضا کش را با توجه به نیازهای مختلف تنظیم کنید. به‌عنوان مثال، ممکن است برای داده‌های حساس نیاز به انقضای سریعتر داشته باشید، در حالی که برای داده‌های عمومی‌تر می‌توانید مدت زمان کش بیشتری تنظیم کنید.

- \*\*عملکرد بهینه\*\*: استفاده از کش می‌تواند به کاهش بار درخواست‌ها و بهبود عملکرد سیستم کمک کند.

این اینترفیس می‌تواند به راحتی در هر نوع سیستم کش استفاده شود و می‌توان آن را با پیاده‌سازی‌های مختلف کش مانند Redis یا Memcached منطبق کرد.

در اینجا کد کامل برای \*\*`CacheService.cs`\*\* است که از \*\*`ICacheService.cs`\*\* پیروی می‌کند. این پیاده‌سازی برای ذخیره‌سازی، خواندن، حذف و مدیریت کش در سیستم طراحی شده است. فرض بر این است که از Redis به عنوان سیستم کش استفاده می‌کنید، اما در صورت نیاز به استفاده از سیستم کش دیگری مانند Memcached یا ... می‌توان به راحتی آن را تغییر داد.

\*\*CacheService.cs\*\*

using System;

using System.Threading.Tasks;

using StackExchange.Redis; // فرض بر این است که از Redis به عنوان سیستم کش استفاده می‌کنیم.

namespace Infrastructure.Common.Caching

{

/// <summary>

/// پیاده‌سازی سرویس کش برای مدیریت عملیات ذخیره‌سازی و بازیابی داده‌ها از کش.

/// این سرویس از Redis برای کش استفاده می‌کند.

/// </summary>

public class CacheService : ICacheService

{

private readonly IConnectionMultiplexer \_redisConnection;

private readonly IDatabase \_database;

public CacheService(IConnectionMultiplexer redisConnection)

{

\_redisConnection = redisConnection;

\_database = \_redisConnection.GetDatabase();

}

/// <summary>

/// داده‌ای را در کش ذخیره می‌کند.

/// </summary>

/// <param name="key">کلید کش برای شناسایی داده</param>

/// <param name="value">مقدار داده‌ای که باید ذخیره شود</param>

/// <param name="absoluteExpiration">زمان انقضای مطلق کش</param>

/// <param name="slidingExpiration">انقضا بر اساس مدت زمان اخیر استفاده</param>

/// <returns>یک تسک که تکمیل فرآیند ذخیره‌سازی کش را نشان می‌دهد</returns>

public async Task SetAsync(string key, object value, TimeSpan? absoluteExpiration = null, TimeSpan? slidingExpiration = null)

{

if (value == null)

throw new ArgumentNullException(nameof(value), "مقدار نمی‌تواند null باشد");

var serializedValue = System.Text.Json.JsonSerializer.Serialize(value);

// تنظیم زمان انقضا کش

TimeSpan? expiry = absoluteExpiration ?? slidingExpiration;

await \_database.StringSetAsync(key, serializedValue, expiry);

}

/// <summary>

/// داده‌ای را از کش بر اساس کلید آن می‌خواند.

/// </summary>

/// <param name="key">کلید کش برای شناسایی داده</param>

/// <returns>داده‌ای که از کش خوانده شده است یا null اگر داده موجود نباشد</returns>

public async Task<object> GetAsync(string key)

{

var serializedValue = await \_database.StringGetAsync(key);

if (serializedValue.IsNullOrEmpty)

return null;

return System.Text.Json.JsonSerializer.Deserialize<object>(serializedValue);

}

/// <summary>

/// داده‌ای را از کش حذف می‌کند.

/// </summary>

/// <param name="key">کلید کش برای شناسایی داده‌ای که باید حذف شود</param>

/// <returns>یک تسک که تکمیل فرآیند حذف کش را نشان می‌دهد</returns>

public async Task RemoveAsync(string key)

{

await \_database.KeyDeleteAsync(key);

}

/// <summary>

/// بررسی می‌کند که آیا داده‌ای با کلید خاص در کش موجود است یا خیر.

/// </summary>

/// <param name="key">کلید کش برای شناسایی داده</param>

/// <returns>حقیقت (True) اگر داده وجود داشته باشد و نادرست (False) در غیر این صورت</returns>

public async Task<bool> ExistsAsync(string key)

{

return await \_database.KeyExistsAsync(key);

}

/// <summary>

/// کش را به‌طور کامل پاک می‌کند.

/// </summary>

/// <returns>یک تسک که تکمیل فرآیند پاک‌سازی کش را نشان می‌دهد</returns>

public async Task ClearAsync()

{

var server = \_redisConnection.GetServer(\_redisConnection.GetEndPoints()[0]);

await foreach (var key in server.KeysAsync())

{

await \_database.KeyDeleteAsync(key);

}

}

/// <summary>

/// کش را به‌صورت اجباری برای یک کلید خاص منقضی می‌کند.

/// </summary>

/// <param name="key">کلید کش که باید منقضی شود</param>

/// <returns>یک تسک که تکمیل فرآیند منقضی کردن کش را نشان می‌دهد</returns>

public async Task ExpireAsync(string key)

{

await \_database.KeyExpireAsync(key);

}

/// <summary>

/// داده‌ای را در کش ذخیره می‌کند و مقدار را بر اساس TTL خاص تنظیم می‌کند.

/// </summary>

/// <param name="key">کلید کش برای شناسایی داده</param>

/// <param name="value">مقدار داده‌ای که باید ذخیره شود</param>

/// <param name="ttl">مدت زمان انقضای کش</param>

/// <returns>یک تسک که تکمیل فرآیند ذخیره‌سازی کش را نشان می‌دهد</returns>

public async Task SetWithTTLAsync(string key, object value, TimeSpan ttl)

{

if (value == null)

throw new ArgumentNullException(nameof(value), "مقدار نمی‌تواند null باشد");

var serializedValue = System.Text.Json.JsonSerializer.Serialize(value);

await \_database.StringSetAsync(key, serializedValue, ttl);

}

/// <summary>

/// داده‌ای را از کش می‌خواند و اگر داده موجود نبود، آن را با تابعی که مشخص می‌شود به‌طور دینامیک بارگذاری می‌کند.

/// </summary>

/// <param name="key">کلید کش برای شناسایی داده</param>

/// <param name="fetchDataFunction">تابعی که برای بارگذاری داده در صورت عدم وجود در کش فراخوانی می‌شود</param>

/// <param name="absoluteExpiration">زمان انقضای مطلق کش</param>

/// <param name="slidingExpiration">انقضا بر اساس مدت زمان اخیر استفاده</param>

/// <returns>داده‌ای که از کش یا تابع بارگذاری خوانده شده است</returns>

public async Task<object> GetOrAddAsync(string key, Func<Task<object>> fetchDataFunction, TimeSpan? absoluteExpiration = null, TimeSpan? slidingExpiration = null)

{

var cachedValue = await GetAsync(key);

if (cachedValue != null)

return cachedValue;

var fetchedData = await fetchDataFunction();

await SetAsync(key, fetchedData, absoluteExpiration, slidingExpiration);

return fetchedData;

}

}

}

توضیحات پیاده‌سازی:

1. \*\*Redis Integration\*\*:

- این پیاده‌سازی فرض می‌کند که از Redis برای ذخیره‌سازی کش استفاده می‌کنید. اگر از سیستم کش دیگری استفاده می‌کنید، باید پیاده‌سازی‌ها را به‌تناسب تغییر دهید.

- از کتابخانه \*\*`StackExchange.Redis`\*\* برای تعامل با Redis استفاده می‌کنیم. شما می‌توانید این کتابخانه را از طریق NuGet اضافه کنید.

2. \*\*SetAsync\*\*: داده‌ها را در کش ذخیره می‌کند. در اینجا، مقدار به‌طور سریالیزه (با استفاده از `System.Text.Json.JsonSerializer`) ذخیره می‌شود. سپس زمان انقضا طبق تنظیمات مشخص می‌شود.

3. \*\*GetAsync\*\*: داده‌ها را از کش می‌خواند. اگر داده موجود نباشد، مقدار `null` برمی‌گرداند.

4. \*\*RemoveAsync\*\*: داده‌ها را از کش حذف می‌کند.

5. \*\*ExistsAsync\*\*: بررسی می‌کند که آیا داده‌ای با کلید خاص در کش موجود است یا خیر.

6. \*\*ClearAsync\*\*: تمام داده‌های کش را پاک می‌کند. این عمل برای Redis با استفاده از `KeysAsync` انجام می‌شود.

7. \*\*ExpireAsync\*\*: برای منقضی کردن یک داده خاص از کش استفاده می‌شود.

8. \*\*SetWithTTLAsync\*\*: داده‌ها را با مدت زمان انقضا (TTL) خاص ذخیره می‌کند.

9. \*\*GetOrAddAsync\*\*: داده‌ها را از کش می‌خواند و در صورت عدم وجود داده، آن را از طریق تابعی که به‌صورت داینامیک ارسال شده بارگذاری می‌کند و در کش ذخیره می‌کند.

نکات:

- \*\*JSON Serialization\*\*: داده‌ها به‌طور خودکار با استفاده از `System.Text.Json.JsonSerializer` به فرمت JSON تبدیل و ذخیره می‌شوند. این کار باعث می‌شود که انواع داده‌های پیچیده قابل ذخیره‌سازی در کش باشند.

- \*\*Exception Handling\*\*: شما می‌توانید برای مدیریت خطاهای Redis یا سایر مسائل مربوط به ذخیره‌سازی کش، موارد بیشتری اضافه کنید.

- \*\*Performance\*\*: استفاده از کش می‌تواند به طور چشمگیری سرعت سیستم شما را افزایش دهد و بار درخواست‌ها را کاهش دهد.

- \*\*زمان انقضا (TTL)\*\*: برای هر داده، می‌توان مدت زمان انقضا را تنظیم کرد. این امکان برای مدیریت حافظه و جلوگیری از استفاده از داده‌های قدیمی بسیار مفید است.

در ادامه، یک پیاده‌سازی کامل از کلاس `AppSettings.cs` برای تنظیمات کلی برنامه آورده شده است. این کلاس می‌تواند اطلاعات مختلفی مانند تنظیمات JWT، کش، رمزنگاری و موارد دیگر را از فایل `appsettings.json` خوانده و در برنامه استفاده کند.

namespace Infrastructure.Settings

{

public class AppSettings

{

// تنظیمات JWT برای احراز هویت

public JwtSettings Jwt { get; set; }

// تنظیمات کش (برای Redis یا سایر کش‌ها)

public CacheSettings Cache { get; set; }

// تنظیمات رمزنگاری

public EncryptionSettings Encryption { get; set; }

// تنظیمات ذخیره‌سازی فایل‌ها

public FileStorageSettings FileStorage { get; set; }

// تنظیمات ایمیل

public EmailSettings Email { get; set; }

// سایر تنظیمات عمومی مورد نیاز

public string Environment { get; set; }

}

public class JwtSettings

{

// کلید رمزنگاری JWT

public string SecretKey { get; set; }

// مدت زمان انقضا Access Token

public int AccessTokenExpirationMinutes { get; set; }

// مدت زمان انقضا Refresh Token

public int RefreshTokenExpirationDays { get; set; }

// الگوریتم امضای توکن

public string Algorithm { get; set; }

}

public class CacheSettings

{

// نوع کش (مثلاً Redis یا MemoryCache)

public string CacheType { get; set; }

// مدت زمان انقضا کش در دقیقه

public int CacheExpirationMinutes { get; set; }

// تنظیمات مربوط به اتصال به Redis (در صورت استفاده از Redis)

public RedisSettings Redis { get; set; }

}

public class RedisSettings

{

// آدرس سرور Redis

public string Host { get; set; }

// پورت Redis

public int Port { get; set; }

// رمز عبور Redis (در صورت نیاز)

public string Password { get; set; }

// مدت زمان انقضا کلیدهای کش در Redis

public int DefaultExpirationMinutes { get; set; }

}

public class EncryptionSettings

{

// کلید رمزنگاری برای داده‌ها

public string EncryptionKey { get; set; }

// الگوریتم رمزنگاری (مثلاً AES)

public string Algorithm { get; set; }

}

public class FileStorageSettings

{

// نوع ذخیره‌سازی فایل (مثلاً Local یا Cloud)

public string StorageType { get; set; }

// مسیر ذخیره‌سازی فایل‌ها برای ذخیره‌سازی محلی

public string LocalPath { get; set; }

// تنظیمات مربوط به فضای ابری (اگر از Cloud استفاده می‌شود)

public CloudStorageSettings Cloud { get; set; }

}

public class CloudStorageSettings

{

// نوع فضای ابری (مثلاً AWS, Azure)

public string Provider { get; set; }

// شناسه دسترسی (Access Key)

public string AccessKey { get; set; }

// کلید مخفی دسترسی (Secret Key)

public string SecretKey { get; set; }

// نام منطقه (Region)

public string Region { get; set; }

// نام باکت (Bucket Name)

public string BucketName { get; set; }

}

public class EmailSettings

{

// آدرس ایمیل ارسال‌کننده

public string FromAddress { get; set; }

// نام ایمیل ارسال‌کننده

public string FromName { get; set; }

// سرور SMTP

public string SmtpServer { get; set; }

// پورت SMTP

public int SmtpPort { get; set; }

// نام کاربری برای ورود به SMTP

public string SmtpUser { get; set; }

// رمز عبور برای SMTP

public string SmtpPassword { get; set; }

}

}

توضیحات کلاس `AppSettings.cs`:

1. \*\*`JwtSettings`\*\*: شامل تنظیمات JWT برای احراز هویت، از جمله کلید رمزنگاری و زمان انقضا برای Access Token و Refresh Token.

2. \*\*`CacheSettings`\*\*: تنظیمات مربوط به کش، مانند نوع کش (Redis یا MemoryCache) و مدت زمان انقضا کش.

3. \*\*`EncryptionSettings`\*\*: تنظیمات رمزنگاری، از جمله کلید رمزنگاری و الگوریتم استفاده شده (مثلاً AES).

4. \*\*`FileStorageSettings`\*\*: تنظیمات مربوط به ذخیره‌سازی فایل‌ها، از جمله نوع ذخیره‌سازی (محلی یا ابری) و تنظیمات فضای ابری.

5. \*\*`EmailSettings`\*\*: تنظیمات مربوط به ارسال ایمیل، از جمله سرور SMTP و اطلاعات ورود به آن.

6. \*\*`RedisSettings`\*\*: تنظیمات مربوط به Redis در صورت استفاده از آن به عنوان کش.

7. \*\*`CloudStorageSettings`\*\*: تنظیمات برای فضای ابری مانند AWS یا Azure برای ذخیره‌سازی فایل‌ها.

نحوه استفاده از این تنظیمات:

برای استفاده از این تنظیمات در پروژه، می‌توانید آن‌ها را در فایل `appsettings.json` تعریف کرده و سپس با استفاده از Dependency Injection در داخل برنامه خود، آن‌ها را بارگذاری و استفاده کنید. به عنوان مثال:

{

"AppSettings": {

"Jwt": {

"SecretKey": "your-secret-key",

"AccessTokenExpirationMinutes": 60,

"RefreshTokenExpirationDays": 7,

"Algorithm": "HS256"

},

"Cache": {

"CacheType": "Redis",

"CacheExpirationMinutes": 30,

"Redis": {

"Host": "localhost",

"Port": 6379,

"Password": "your-password",

"DefaultExpirationMinutes": 10

}

},

"Encryption": {

"EncryptionKey": "your-encryption-key",

"Algorithm": "AES"

},

"FileStorage": {

"StorageType": "Local",

"LocalPath": "C:/Files"

},

"Email": {

"FromAddress": "noreply@domain.com",

"FromName": "Your App",

"SmtpServer": "smtp.domain.com",

"SmtpPort": 587,

"SmtpUser": "smtp-user",

"SmtpPassword": "smtp-password"

}

}

}

در نهایت، در کد سی‌شارپ خود، می‌توانید به راحتی این تنظیمات را با استفاده از `IOptions<AppSettings>` بارگذاری کنید:

public class SomeService

{

private readonly AppSettings \_appSettings;

public SomeService(IOptions<AppSettings> appSettings)

{

\_appSettings = appSettings.Value;

}

public void DoSomething()

{

var jwtSecretKey = \_appSettings.Jwt.SecretKey;

var cacheExpiration = \_appSettings.Cache.CacheExpirationMinutes;

// سایر دسترسی‌ها به تنظیمات...

}

}

این کلاس و تنظیمات به شما کمک می‌کنند تا پروژه‌تان انعطاف‌پذیرتر و مقیاس‌پذیرتر باشد.

در زیر پیاده‌سازی کامل کلاس `CacheSettings.cs` آمده است که تنظیمات مربوط به کش را شامل می‌شود، از جمله تنظیمات مربوط به نوع کش (مثلاً Redis یا MemoryCache) و تنظیمات خاص Redis در صورت استفاده از آن.

namespace Infrastructure.Settings

{

public class CacheSettings

{

// نوع کش (مثلاً Redis یا MemoryCache)

public string CacheType { get; set; }

// مدت زمان انقضا کش در دقیقه

public int CacheExpirationMinutes { get; set; }

// تنظیمات مربوط به اتصال به Redis (در صورت استفاده از Redis)

public RedisSettings Redis { get; set; }

}

public class RedisSettings

{

// آدرس سرور Redis

public string Host { get; set; }

// پورت Redis

public int Port { get; set; }

// رمز عبور Redis (در صورت نیاز)

public string Password { get; set; }

// مدت زمان انقضا کلیدهای کش در Redis

public int DefaultExpirationMinutes { get; set; }

// اگر از SSL استفاده می‌کنید

public bool UseSsl { get; set; }

// تنظیمات مربوط به اتصال مجدد

public RedisConnectionSettings Connection { get; set; }

}

public class RedisConnectionSettings

{

// مدت زمان اتصال مجدد در صورت قطع ارتباط

public int RetryCount { get; set; }

// مدت زمان بین تلاش‌ها برای اتصال مجدد

public int RetryIntervalSeconds { get; set; }

// مدت زمان انقضا اتصال

public int ConnectTimeoutMilliseconds { get; set; }

// مدت زمان برای عملیات‌های Redis

public int OperationTimeoutMilliseconds { get; set; }

}

}

توضیحات:

1. `CacheSettings`\*\*:

- `CacheType`: نوع کش مورد استفاده را مشخص می‌کند (مثلاً Redis یا MemoryCache).

- `CacheExpirationMinutes`: مدت زمانی که داده‌ها در کش باقی می‌مانند (بر حسب دقیقه).

- `Redis`: تنظیمات خاص Redis را در صورت استفاده از Redis برای کش نگهداری می‌کند.

2. \*\*`RedisSettings`\*\*:

- `Host`: آدرس سرور Redis.

- `Port`: پورت سرور Redis.

- `Password`: رمز عبور برای دسترسی به Redis (در صورت نیاز).

- `DefaultExpirationMinutes`: مدت زمان انقضا پیش‌فرض برای کلیدهای ذخیره‌شده در Redis.

- `UseSsl`: آیا از اتصال امن SSL برای Redis استفاده می‌شود یا نه.

- `Connection`: تنظیمات مربوط به نحوه اتصال و تلاش برای اتصال مجدد در صورت قطع ارتباط.

3. \*\*`RedisConnectionSettings`\*\*:

- `RetryCount`: تعداد دفعات تلاش مجدد برای اتصال در صورت قطع ارتباط.

- `RetryIntervalSeconds`: مدت زمانی که بین هر تلاش مجدد برای اتصال باید منتظر بمانیم.

- `ConnectTimeoutMilliseconds`: زمان تایم‌اوت اتصال به Redis.

- `OperationTimeoutMilliseconds`: زمان تایم‌اوت عملیات‌های Redis مانند خواندن و نوشتن داده‌ها.

مثال از فایل `appsettings.json`:

در زیر یک مثال از نحوه پیکربندی این تنظیمات در فایل `appsettings.json` آمده است:

```json

{

"CacheSettings": {

"CacheType": "Redis",

"CacheExpirationMinutes": 30,

"Redis": {

"Host": "localhost",

"Port": 6379,

"Password": "your-redis-password",

"DefaultExpirationMinutes": 60,

"UseSsl": false,

"Connection": {

"RetryCount": 5,

"RetryIntervalSeconds": 2,

"ConnectTimeoutMilliseconds": 5000,

"OperationTimeoutMilliseconds": 3000

}

}

}

}

نحوه استفاده از این تنظیمات:

برای استفاده از این تنظیمات در پروژه، می‌توانید آن‌ها را در فایل `appsettings.json` تعریف کرده و سپس با استفاده از `IOptions<CacheSettings>` در داخل برنامه خود آن‌ها را بارگذاری کنید:

public class CacheService

{

private readonly CacheSettings \_cacheSettings;

public CacheService(IOptions<CacheSettings> cacheSettings)

{

\_cacheSettings = cacheSettings.Value;

}

public void ConfigureCache()

{

if (\_cacheSettings.CacheType == "Redis")

{

var redisSettings = \_cacheSettings.Redis;

// انجام تنظیمات Redis با استفاده از redisSettings

}

}

}

این پیاده‌سازی به شما کمک می‌کند تا کش را به‌راحتی پیکربندی کرده و در پروژه‌های مختلف با انعطاف بالا استفاده کنید.

در زیر پیاده‌سازی کامل کلاس `JwtSettings.cs` آورده شده است که تنظیمات مربوط به JWT (JSON Web Token) را شامل می‌شود. این تنظیمات شامل مدت زمان اعتبار توکن‌ها، کلیدهای مورد استفاده برای امضا و تأیید، و سایر تنظیمات امنیتی است.

namespace Infrastructure.Settings

{

public class JwtSettings

{

// کلید مخفی برای امضا و تأیید JWT

public string SecretKey { get; set; }

// الگوریتمی که برای امضا و تأیید JWT استفاده می‌شود

public string Algorithm { get; set; }

// مدت زمان انقضای توکن دسترسی (Access Token) به دقیقه

public int AccessTokenExpirationMinutes { get; set; }

// مدت زمان انقضای توکن تجدید (Refresh Token) به دقیقه

public int RefreshTokenExpirationMinutes { get; set; }

// URL که بعد از اعتبارسنجی موفقیت‌آمیز توکن، درخواست‌ها به آن هدایت می‌شوند

public string Issuer { get; set; }

// URL که توکن‌ها از آنجا صادر می‌شوند

public string Audience { get; set; }

// نوع الگوریتم برای رمزگذاری JWT (مثلاً HMACSHA256)

public string JwtType { get; set; }

// آیا توکن‌ها باید چرخش کنند یا خیر (برای امنیت بیشتر)

public bool EnableTokenRotation { get; set; }

// آیا توکن‌ها باید در لیست سیاه قرار بگیرند بعد از چرخش

public bool EnableTokenBlacklisting { get; set; }

// کلید‌های رمزنگاری برای عملیات‌های امن اضافی (در صورت نیاز)

public string EncryptionKey { get; set; }

}

}

توضیحات:

1. \*\*`SecretKey`\*\*: کلید مخفی برای امضا و تأیید JWT. این کلید باید به‌صورت محرمانه نگهداری شود و برای جلوگیری از جعل توکن‌ها استفاده می‌شود.

2. \*\*`Algorithm`\*\*: الگوریتمی که برای امضا و تأیید توکن‌ها استفاده می‌شود. به طور معمول از الگوریتم‌هایی مانند `HMACSHA256` استفاده می‌شود.

3. \*\*`AccessTokenExpirationMinutes`\*\*: مدت زمانی که توکن دسترسی (Access Token) معتبر است (بر حسب دقیقه).

4. \*\*`RefreshTokenExpirationMinutes`\*\*: مدت زمانی که توکن تجدید (Refresh Token) معتبر است (بر حسب دقیقه).

5. \*\*`Issuer`\*\*: URL که توکن‌ها از آن صادر می‌شوند. معمولاً این مقدار را می‌توانید به‌عنوان نام دامنه یا آدرس سرور خود تنظیم کنید.

6. \*\*`Audience`\*\*: URL یا شناسه مخاطب (Audience) که توکن‌ها برای آن صادر می‌شوند. به طور معمول این مقدار همانند `Issuer` است، اما می‌تواند برای شناسایی اپلیکیشن‌ها یا سرویس‌های خاص استفاده شود.

7. \*\*`JwtType`\*\*: نوع JWT، مانند `Bearer` که به طور معمول برای توکن‌های دسترسی استفاده می‌شود.

8. \*\*`EnableTokenRotation`\*\*: مشخص می‌کند که آیا توکن‌ها باید به صورت دوره‌ای چرخش کنند (برای امنیت بیشتر) یا خیر. اگر فعال باشد، توکن‌های جدیدی برای کاربران صادر می‌شود.

9. \*\*`EnableTokenBlacklisting`\*\*: مشخص می‌کند که آیا توکن‌ها باید پس از چرخش در لیست سیاه قرار بگیرند یا خیر. این ویژگی به مدیریت توکن‌های غیرمعتبر کمک می‌کند.

10. \*\*`EncryptionKey`\*\*: در صورت استفاده از رمزنگاری اضافی در JWT، این کلید برای عملیات رمزنگاری و رمزگشایی استفاده می‌شود.

نمونه پیکربندی در `appsettings.json`:

{

"JwtSettings": {

"SecretKey": "YourSuperSecretKey",

"Algorithm": "HMACSHA256",

"AccessTokenExpirationMinutes": 60,

"RefreshTokenExpirationMinutes": 1440,

"Issuer": "your-app-name",

"Audience": "your-app-name",

"JwtType": "Bearer",

"EnableTokenRotation": true,

"EnableTokenBlacklisting": true,

"EncryptionKey": "YourEncryptionKey"

}

}

نحوه استفاده از این تنظیمات:

برای استفاده از این تنظیمات در برنامه، می‌توانید آن‌ها را با استفاده از `IOptions<JwtSettings>` در داخل برنامه بارگذاری کنید:

public class JwtTokenService

{

private readonly JwtSettings \_jwtSettings;

public JwtTokenService(IOptions<JwtSettings> jwtSettings)

{

\_jwtSettings = jwtSettings.Value;

}

public void ConfigureJwt()

{

var secretKey = \_jwtSettings.SecretKey;

var algorithm = \_jwtSettings.Algorithm;

// انجام تنظیمات لازم برای JWT با استفاده از \_jwtSettings

}

}

این پیاده‌سازی به شما کمک می‌کند تا تنظیمات JWT را به صورت مرکزی در پروژه خود مدیریت کنید و توکن‌ها را به‌صورت امن تولید و تأیید کنید.

در اینجا کد کلاس `EncryptionSettings.cs` که شامل تنظیمات مربوط به رمزنگاری است، به صورت کامل آورده شده است. این کلاس معمولاً برای ذخیره‌سازی تنظیمات مرتبط با کلیدها و الگوریتم‌های رمزنگاری مورد استفاده قرار می‌گیرد.

using System;

namespace Infrastructure.Common.Settings

{

/// <summary>

/// تنظیمات مربوط به رمزنگاری داده‌ها.

/// </summary>

public class EncryptionSettings

{

/// <summary>

/// کلید رمزنگاری برای استفاده در الگوریتم‌های رمزنگاری.

/// </summary>

public string EncryptionKey { get; set; }

/// <summary>

/// نوع الگوریتم رمزنگاری که باید استفاده شود (مانند AES، RSA و غیره).

/// </summary>

public string EncryptionAlgorithm { get; set; }

/// <summary>

/// زمان انقضای کلید رمزنگاری (برای مثال، به مدت چند روز یا ماه قابل استفاده است).

/// </summary>

public TimeSpan KeyExpiration { get; set; }

/// <summary>

/// مسیر یا آدرس فایل ذخیره‌سازی کلیدهای رمزنگاری (در صورت استفاده از ذخیره‌سازی محلی).

/// </summary>

public string KeyStorageLocation { get; set; }

/// <summary>

/// آیا از رمزنگاری دوطرفه استفاده می‌شود یا خیر (برای مثال در RSA).

/// </summary>

public bool UseSymmetricEncryption { get; set; }

/// <summary>

/// تنظیمات مربوط به نحوه رمزنگاری داده‌ها در سیستم.

/// </summary>

public EncryptionSettings()

{

// مقادیر پیش‌فرض برای تنظیمات رمزنگاری

EncryptionAlgorithm = "AES";

KeyExpiration = TimeSpan.FromDays(30);

UseSymmetricEncryption = true;

}

}

}

توضیحات فیلدها:

- \*\*EncryptionKey\*\*: کلید رمزنگاری که برای انجام عملیات رمزنگاری و رمزگشایی داده‌ها استفاده می‌شود.

- \*\*EncryptionAlgorithm\*\*: الگوریتمی که برای رمزنگاری داده‌ها استفاده می‌شود (مانند AES یا RSA).

- \*\*KeyExpiration\*\*: مدت زمانی که کلید رمزنگاری معتبر است و بعد از آن نیاز به تمدید یا تعویض دارد.

- \*\*KeyStorageLocation\*\*: آدرس یا مسیر ذخیره‌سازی کلید رمزنگاری (برای مثال در یک فایل یا پایگاه داده).

- \*\*UseSymmetricEncryption\*\*: مشخص می‌کند که آیا از رمزنگاری دوطرفه (Symmetric) یا عمومی (Asymmetric) استفاده می‌شود. برای مثال، در رمزنگاری AES از رمزنگاری دوطرفه استفاده می‌شود، اما در RSA از رمزنگاری عمومی استفاده می‌شود.

نکات:

- شما می‌توانید این کلاس را برای استفاده در رمزنگاری داده‌ها و ذخیره‌سازی تنظیمات امنیتی در پروژه‌های خود استفاده کنید.

- مقادیر پیش‌فرض برای الگوریتم رمزنگاری AES و مدت زمان انقضای کلید ۳۰ روز تنظیم شده‌اند که می‌توانید این مقادیر را بسته به نیاز خود تغییر دهید.

در اینجا کد کلاس `FileStorageSettings.cs` که شامل تنظیمات مربوط به ذخیره‌سازی فایل‌ها است، به صورت کامل آورده شده است. این کلاس معمولاً برای تنظیمات ذخیره‌سازی فایل‌ها مانند مسیر ذخیره‌سازی فایل‌ها، تنظیمات فضای ذخیره‌سازی ابری و غیره استفاده می‌شود.

using System;

namespace Infrastructure.Common.Settings

{

/// <summary>

/// تنظیمات مربوط به ذخیره‌سازی فایل‌ها.

/// </summary>

public class FileStorageSettings

{

/// <summary>

/// مسیر ذخیره‌سازی فایل‌های محلی (در صورت استفاده از ذخیره‌سازی محلی).

/// </summary>

public string LocalStoragePath { get; set; }

/// <summary>

/// آدرس ذخیره‌سازی ابری (در صورت استفاده از ذخیره‌سازی ابری، مانند Azure یا AWS).

/// </summary>

public string CloudStorageUrl { get; set; }

/// <summary>

/// نوع فضای ذخیره‌سازی برای فایل‌ها (محلی یا ابری).

/// </summary>

public StorageType StorageType { get; set; }

/// <summary>

/// مدت زمان انقضای فایل‌ها در فضای ذخیره‌سازی.

/// </summary>

public TimeSpan FileExpiration { get; set; }

/// <summary>

/// تعداد فایل‌هایی که باید همزمان بارگذاری شوند (برای بهینه‌سازی عملکرد).

/// </summary>

public int MaxConcurrentUploads { get; set; }

/// <summary>

/// حداکثر اندازه مجاز برای فایل‌های آپلود شده (در مگابایت).

/// </summary>

public int MaxFileSizeInMB { get; set; }

/// <summary>

/// فرمت‌های مجاز برای فایل‌های آپلود شده (مانند .jpg, .png, .pdf).

/// </summary>

public string[] AllowedFileExtensions { get; set; }

/// <summary>

/// تنظیمات مربوط به ذخیره‌سازی فایل‌ها.

/// </summary>

public FileStorageSettings()

{

// مقادیر پیش‌فرض برای تنظیمات ذخیره‌سازی

StorageType = StorageType.Local;

FileExpiration = TimeSpan.FromDays(30); // فایل‌ها بعد از ۳۰ روز منقضی می‌شوند

MaxConcurrentUploads = 5; // حداکثر ۵ فایل همزمان بارگذاری می‌شود

MaxFileSizeInMB = 100; // حداکثر اندازه فایل ۱۰۰ مگابایت است

AllowedFileExtensions = new string[] { ".jpg", ".jpeg", ".png", ".gif", ".pdf" }; // فرمت‌های مجاز

}

}

/// <summary>

/// نوع فضای ذخیره‌سازی.

/// </summary>

public enum StorageType

{

/// <summary>

/// ذخیره‌سازی محلی (مثلاً در سرور یا دیسک محلی)

/// </summary>

Local,

/// <summary>

/// ذخیره‌سازی ابری (مثلاً در AWS, Azure یا Google Cloud)

/// </summary>

Cloud

}

}

توضیحات فیلدها:

- \*\*LocalStoragePath\*\*: مسیر ذخیره‌سازی فایل‌ها در سیستم محلی (برای مثال در یک پوشه خاص در سرور یا سیستم).

- \*\*CloudStorageUrl\*\*: آدرس ذخیره‌سازی ابری که در صورت استفاده از سرویس‌های ابری (مانند AWS S3 یا Azure Blob Storage) باید مشخص شود.

- \*\*StorageType\*\*: نوع فضای ذخیره‌سازی (محلی یا ابری). می‌تواند `Local` یا `Cloud` باشد.

- \*\*FileExpiration\*\*: مدت زمانی که فایل‌ها در سیستم ذخیره می‌شوند. پس از انقضای این مدت، فایل‌ها به صورت خودکار حذف می‌شوند.

- \*\*MaxConcurrentUploads\*\*: حداکثر تعداد فایل‌هایی که می‌توانند به طور همزمان بارگذاری شوند.

- \*\*MaxFileSizeInMB\*\*: حداکثر اندازه مجاز برای فایل‌های آپلود شده (در مگابایت).

- \*\*AllowedFileExtensions\*\*: لیستی از فرمت‌های مجاز برای فایل‌های آپلود شده (برای مثال، فقط فایل‌های تصویری و PDF).

نکات:

- این کلاس تنظیمات مربوط به ذخیره‌سازی فایل‌ها را به شما می‌دهد که می‌توانید از آن برای ذخیره‌سازی فایل‌ها به صورت محلی یا در فضای ابری استفاده کنید.

- مقادیر پیش‌فرض شامل فضای ذخیره‌سازی محلی، مدت زمان انقضای ۳۰ روز برای فایل‌ها، و حداکثر اندازه فایل ۱۰۰ مگابایت است که می‌توانید بسته به نیاز خود آن‌ها را تغییر دهید.

در اینجا کد کلاس `EmailSettings.cs` که تنظیمات مربوط به ارسال ایمیل را شامل می‌شود، به صورت کامل آورده شده است. این کلاس معمولاً برای پیکربندی تنظیمات ارسال ایمیل مانند آدرس سرور SMTP، پورت، اعتبارسنجی و غیره استفاده می‌شود.

using System;

namespace Infrastructure.Common.Settings

{

/// <summary>

/// تنظیمات مربوط به ارسال ایمیل.

/// </summary>

public class EmailSettings

{

/// <summary>

/// آدرس ایمیل ارسال‌کننده.

/// </summary>

public string SenderEmail { get; set; }

/// <summary>

/// رمز عبور یا توکن ایمیل ارسال‌کننده.

/// </summary>

public string SenderPassword { get; set; }

/// <summary>

/// سرور SMTP برای ارسال ایمیل.

/// </summary>

public string SmtpServer { get; set; }

/// <summary>

/// پورت مورد استفاده برای اتصال به سرور SMTP.

/// </summary>

public int SmtpPort { get; set; }

/// <summary>

/// استفاده از ارتباط امن (SSL/TLS) برای اتصال به سرور SMTP.

/// </summary>

public bool UseSsl { get; set; }

/// <summary>

/// آدرس ایمیل برای پاسخ‌دهی (در صورت نیاز).

/// </summary>

public string ReplyToEmail { get; set; }

/// <summary>

/// نام ارسال‌کننده ایمیل.

/// </summary>

public string SenderName { get; set; }

/// <summary>

/// تنظیمات ایمیل

/// </summary>

public EmailSettings()

{

// مقادیر پیش‌فرض برای تنظیمات ایمیل

SmtpServer = "smtp.example.com"; // سرور SMTP پیش‌فرض

SmtpPort = 587; // پورت پیش‌فرض SMTP برای اتصال امن (TLS)

UseSsl = true; // استفاده از SSL/TLS به صورت پیش‌فرض

SenderName = "YourAppName"; // نام ارسال‌کننده پیش‌فرض

}

}

}

توضیحات فیلدها:

- \*\*SenderEmail\*\*: آدرس ایمیل ارسال‌کننده که معمولاً به عنوان "From" در ایمیل‌ها استفاده می‌شود.

- \*\*SenderPassword\*\*: رمز عبور یا توکن (برای ایمیل‌هایی که از رمز عبور دو مرحله‌ای استفاده می‌کنند) مربوط به حساب ایمیل ارسال‌کننده.

- \*\*SmtpServer\*\*: آدرس سرور SMTP که برای ارسال ایمیل‌ها استفاده می‌شود. به عنوان مثال، برای Gmail `smtp.gmail.com` یا برای Outlook `smtp-mail.outlook.com` است.

- \*\*SmtpPort\*\*: پورت سرور SMTP. معمولاً پورت 587 برای ارتباط امن با TLS استفاده می‌شود.

- \*\*UseSsl\*\*: اگر این گزینه فعال باشد، اتصال به سرور SMTP از طریق SSL/TLS صورت خواهد گرفت.

- \*\*ReplyToEmail\*\*: ایمیل برای پاسخ‌دهی به ایمیل‌ها (اگر تنظیم شود، پاسخ‌ها به این آدرس ارسال خواهند شد).

- \*\*SenderName\*\*: نام ارسال‌کننده که در ایمیل‌ها نمایش داده می‌شود.

نکات:

- این کلاس برای پیکربندی تنظیمات ایمیل در پروژه‌های مختلف مانند ارسال ایمیل‌های تأیید، اطلاع‌رسانی و هشدارها استفاده می‌شود.

- مقادیر پیش‌فرض تنظیمات شامل سرور SMTP، پورت 587 و استفاده از SSL/TLS است که برای بیشتر سرویس‌دهندگان ایمیل مناسب است.

- اگر از سرویس‌دهندگان ایمیل خاصی استفاده می‌کنید، ممکن است نیاز به تنظیم مقادیر خاص‌تری داشته باشید (مانند استفاده از OAuth برای Gmail).

در اینجا کد کلاس `HeaderNames.cs` که شامل نام هدرهای HTTP سفارشی است، به صورت کامل آورده شده است. این کلاس برای مدیریت و نگهداری هدرهای HTTP به صورت ثابت استفاده می‌شود، به ویژه زمانی که نیاز دارید هدرهای خاصی را در API‌های خود تعریف کنید.

```csharp

namespace Infrastructure.Common.Constants

{

/// <summary>

/// کلاس برای نگهداری نام هدرهای HTTP سفارشی.

/// </summary>

public static class HeaderNames

{

/// <summary>

/// نام هدر برای تایید هویت (مثلاً JWT Token).

/// </summary>

public const string Authorization = "Authorization";

/// <summary>

/// نام هدر برای مشخص کردن زبان درخواست (مانند en-US یا fa-IR).

/// </summary>

public const string AcceptLanguage = "Accept-Language";

/// <summary>

/// نام هدر برای تعیین نوع درخواست (مانند JSON یا XML).

/// </summary>

public const string ContentType = "Content-Type";

/// <summary>

/// هدر برای مشخص کردن توکن احراز هویت خاص برای دسترسی به منابع.

/// </summary>

public const string XAuthToken = "X-Auth-Token";

/// <summary>

/// هدر برای ارسال اطلاعات مربوط به نسخه اپلیکیشن.

/// </summary>

public const string XAppVersion = "X-App-Version";

/// <summary>

/// هدر برای ارسال اطلاعات IP کلاینت درخواست‌دهنده.

/// </summary>

public const string XClientIp = "X-Client-IP";

/// <summary>

/// هدر برای ارسال شناسه جلسه (Session ID).

/// </summary>

public const string XSessionId = "X-Session-ID";

/// <summary>

/// هدر برای ارسال اطلاعات مربوط به مرجع درخواست (Referer).

/// </summary>

public const string Referer = "Referer";

/// <summary>

/// هدر برای ارسال اطلاعات مربوط به User-Agent (جزئیات مرورگر).

/// </summary>

public const string UserAgent = "User-Agent";

/// <summary>

/// هدر برای ارسال اطلاعات مربوط به ردیابی درخواست‌ها.

/// </summary>

public const string XTraceId = "X-Trace-ID";

/// <summary>

/// هدر برای ارسال اطلاعات مربوط به سیاست‌های CORS.

/// </summary>

public const string AccessControlAllowOrigin = "Access-Control-Allow-Origin";

/// <summary>

/// هدر برای ارسال زمان درخواست (Timestamp).

/// </summary>

public const string XRequestTimestamp = "X-Request-Timestamp";

/// <summary>

/// هدر برای ارسال اطلاعات مربوط به نوع اپلیکیشن (مانند mobile یا web).

/// </summary>

public const string XAppType = "X-App-Type";

}

}

توضیحات:

- \*\*Authorization\*\*: این هدر برای ارسال توکن‌های احراز هویت (مانند JWT) استفاده می‌شود.

- \*\*AcceptLanguage\*\*: این هدر برای مشخص کردن زبان مورد نظر برای پاسخ استفاده می‌شود (مثلاً `en-US` یا `fa-IR`).

- \*\*ContentType\*\*: برای تعیین نوع محتوای درخواست (مانند `application/json` یا `application/xml`).

- \*\*XAuthToken\*\*: یک هدر سفارشی برای ارسال توکن‌های احراز هویت در درخواست‌ها.

- \*\*XAppVersion\*\*: برای ارسال اطلاعات نسخه اپلیکیشن در هدر استفاده می‌شود.

- \*\*XClientIp\*\*: برای ارسال آدرس IP دستگاه یا کاربر درخواست‌دهنده.

- \*\*XSessionId\*\*: هدر برای ارسال شناسه جلسه.

- \*\*Referer\*\*: برای ارسال اطلاعات مرجع درخواست (آدرس صفحه‌ای که از آن به این درخواست آمده است).

- \*\*UserAgent\*\*: برای ارسال اطلاعات مرورگر یا اپلیکیشن کاربر (مانند مرورگر وب).

- \*\*XTraceId\*\*: برای ردیابی درخواست‌ها در سیستم‌ها (مفید برای نظارت و دیباگینگ).

- \*\*AccessControlAllowOrigin\*\*: هدر CORS برای کنترل دسترسی بین منابع.

- \*\*XRequestTimestamp\*\*: برای ارسال زمان دقیق درخواست.

- \*\*XAppType\*\*: برای مشخص کردن نوع اپلیکیشن، مثلاً `mobile` یا `web`.

نکات:

- این کلاس به عنوان یک مجموعه ثابت از هدرهای HTTP برای استفاده در بخش‌های مختلف سیستم مورد استفاده قرار می‌گیرد. استفاده از این هدرها باعث یکپارچگی و خوانایی بیشتر کد می‌شود.

- نام هدرها به طور ثابت در این کلاس نگهداری می‌شود تا از اشتباهات تایپی و استفاده مکرر از رشته‌های تکراری جلوگیری شود.

- این هدرها ممکن است بسته به نیاز خاص پروژه و API شما قابل گسترش یا تغییر باشند.

در اینجا کد کلاس `RegexPatterns.cs` آورده شده است که شامل الگوهای مختلف برای استفاده در تطبیق عبارات منظم (Regex) در پروژه‌ها است. این کلاس می‌تواند به شما در یافتن و اعتبارسنجی الگوهای مختلف، مانند ایمیل، شماره تلفن، URL و دیگر موارد مشابه کمک کند.

namespace Infrastructure.Common.Constants

{

/// <summary>

/// کلاس برای نگهداری الگوهای عبارات منظم (Regex) کاربردی.

/// </summary>

public static class RegexPatterns

{

/// <summary>

/// الگوی برای تطبیق با ایمیل‌ها.

/// </summary>

public const string Email = @"^[a-zA-Z0-9.\_%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$";

/// <summary>

/// الگوی برای تطبیق با شماره تلفن‌های ایرانی (با پیش شماره ۰۹).

/// </summary>

public const string PhoneNumber = @"^09\d{9}$";

/// <summary>

/// الگوی برای تطبیق با URL‌های معتبر.

/// </summary>

public const string Url = @"^(https?|ftp)://[^\s/$.?#].[^\s]\*$";

/// <summary>

/// الگوی برای تطبیق با شناسه ملی ایرانی (۱۰ رقم).

/// </summary>

public const string NationalId = @"^\d{10}$";

/// <summary>

/// الگوی برای تطبیق با کد پستی ایران (۱۰ رقم).

/// </summary>

public const string PostalCode = @"^\d{10}$";

/// <summary>

/// الگوی برای تطبیق با شماره کارت بانکی (۱۶ رقم).

/// </summary>

public const string CreditCard = @"^\d{16}$";

/// <summary>

/// الگوی برای تطبیق با تاریخ در فرمت YYYY-MM-DD.

/// </summary>

public const string Date = @"^\d{4}-\d{2}-\d{2}$";

/// <summary>

/// الگوی برای تطبیق با ساعت در فرمت HH:mm (۲۴ ساعته).

/// </summary>

public const string Time = @"^([01]?[0-9]|2[0-3]):([0-5]?[0-9])$";

/// <summary>

/// الگوی برای تطبیق با شناسه کاربری (کاراکترهای مجاز شامل حروف، اعداد، و زیرخط).

/// </summary>

public const string Username = @"^[a-zA-Z0-9\_]{3,16}$";

/// <summary>

/// الگوی برای تطبیق با پسورد (حداقل یک عدد، یک حرف بزرگ، یک حرف کوچک و یک علامت خاص).

/// </summary>

public const string Password = @"^(?=.\*[a-z])(?=.\*[A-Z])(?=.\*\d)(?=.\*[@$!%\*?&])[A-Za-z\d@$!%\*?&]{8,}$";

/// <summary>

/// الگوی برای تطبیق با آدرس IPv4.

/// </summary>

public const string IPv4 = @"^(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)$";

/// <summary>

/// الگوی برای تطبیق با آدرس IPv6.

/// </summary>

public const string IPv6 = @"^([0-9a-fA-F]{1,4}:){7}([0-9a-fA-F]{1,4})$";

/// <summary>

/// الگوی برای تطبیق با شناسه محصول (SKU).

/// </summary>

public const string ProductId = @"^[A-Za-z0-9-]{10,20}$";

/// <summary>

/// الگوی برای تطبیق با آدرس ایمیل دانشگاهی.

/// </summary>

public const string UniversityEmail = @"^[a-zA-Z0-9.\_%+-]+@(?:[a-zA-Z0-9.-]+\.)?edu$";

/// <summary>

/// الگوی برای تطبیق با تاریخ تولد (در فرمت YYYY/MM/DD).

/// </summary>

public const string BirthDate = @"^(19|20)\d\d/(0[1-9]|1[0-2])/(0[1-9]|[12][0-9]|3[01])$";

/// <summary>

/// الگوی برای تطبیق با کد امنیتی کارت (CVV) (۳ رقم).

/// </summary>

public const string Cvv = @"^\d{3}$";

}

}

توضیحات:

- \*\*Email\*\*: برای تطبیق با ایمیل‌های استاندارد.

- \*\*PhoneNumber\*\*: برای تطبیق با شماره تلفن‌های ایرانی (مانند `09123456789`).

- \*\*Url\*\*: برای تطبیق با URL‌های معتبر که پروتکل `http` یا `https` دارند.

- \*\*NationalId\*\*: برای تطبیق با شناسه ملی ایرانی که باید ۱۰ رقم باشد.

- \*\*PostalCode\*\*: برای تطبیق با کد پستی ۱۰ رقمی ایران.

- \*\*CreditCard\*\*: برای تطبیق با شماره کارت بانکی ۱۶ رقمی.

- \*\*Date\*\*: برای تطبیق با تاریخ در فرمت `YYYY-MM-DD`.

- \*\*Time\*\*: برای تطبیق با زمان در فرمت `HH:mm`.

- \*\*Username\*\*: برای تطبیق با شناسه کاربری که می‌تواند حروف، اعداد و زیرخط باشد.

- \*\*Password\*\*: برای تطبیق با پسوردهای قوی که شامل حداقل یک عدد، یک حرف بزرگ، یک حرف کوچک و یک علامت خاص هستند.

- \*\*IPv4 و IPv6\*\*: برای تطبیق با آدرس‌های IP نسخه ۴ و ۶.

- \*\*ProductId\*\*: برای تطبیق با شناسه‌های محصول که می‌توانند شامل حروف و اعداد باشند.

- \*\*UniversityEmail\*\*: برای تطبیق با آدرس‌های ایمیل دانشگاهی (مثلاً `example@edu`).

- \*\*BirthDate\*\*: برای تطبیق با تاریخ تولد در فرمت `YYYY/MM/DD`.

- \*\*Cvv\*\*: برای تطبیق با کد امنیتی کارت (CVV).

نکات:

- این کلاس می‌تواند برای کاربردهای مختلف در پروژه‌هایی مانند فرم‌های ثبت‌نام، اعتبارسنجی داده‌ها و پردازش ورودی‌ها استفاده شود.

- الگوها به صورت ثابت در این کلاس نگهداری می‌شوند تا از اشتباهات تایپی و استفاده مجدد از رشته‌ها جلوگیری شود.

در اینجا کد کلاس `ErrorMessages.cs` آورده شده است که شامل پیام‌های خطای استاندارد شده برای استفاده در پروژه‌ها است. این کلاس می‌تواند برای مدیریت و نمایش پیام‌های خطا در هنگام بروز مشکلات مختلف در سیستم یا API استفاده شود.

namespace Infrastructure.Common.Constants

{

/// <summary>

/// کلاس برای نگهداری پیام‌های خطای استاندارد شده در پروژه.

/// </summary>

public static class ErrorMessages

{

/// <summary>

/// پیام خطا برای زمانی که یک ورودی نادرست ارسال می‌شود.

/// </summary>

public const string InvalidInput = "ورودی نامعتبر است. لطفاً دوباره تلاش کنید.";

/// <summary>

/// پیام خطا برای زمانی که کاربر مجوز دسترسی به یک عملیات را ندارد.

/// </summary>

public const string Unauthorized = "شما دسترسی لازم برای انجام این عملیات را ندارید.";

/// <summary>

/// پیام خطا برای زمانی که یک کاربر یافت نمی‌شود.

/// </summary>

public const string UserNotFound = "کاربر مورد نظر یافت نشد.";

/// <summary>

/// پیام خطا برای زمانی که رمز عبور نادرست وارد می‌شود.

/// </summary>

public const string InvalidPassword = "رمز عبور وارد شده صحیح نیست.";

/// <summary>

/// پیام خطا برای زمانی که عملیات به دلیل نقض قوانین مسدود شده است.

/// </summary>

public const string Forbidden = "دسترسی شما به این منبع مسدود شده است.";

/// <summary>

/// پیام خطا برای زمانی که یک منبع درخواست شده یافت نمی‌شود.

/// </summary>

public const string NotFound = "منبع مورد نظر یافت نشد.";

/// <summary>

/// پیام خطا برای زمانی که یک درخواست تایید نشده است.

/// </summary>

public const string UnverifiedAccount = "حساب کاربری شما تایید نشده است.";

/// <summary>

/// پیام خطا برای زمانی که درخواست زمان‌بر بوده و درخواست تایم‌اوت شده است.

/// </summary>

public const string RequestTimeout = "درخواست شما به دلیل زمان‌بر بودن، به پایان رسید.";

/// <summary>

/// پیام خطا برای زمانی که یک اشتباه داخلی سرور رخ می‌دهد.

/// </summary>

public const string InternalServerError = "خطای داخلی در سرور رخ داده است. لطفاً بعداً تلاش کنید.";

/// <summary>

/// پیام خطا برای زمانی که داده‌های ارسالی معتبر نیستند.

/// </summary>

public const string InvalidData = "داده‌های ارسال شده معتبر نیستند.";

/// <summary>

/// پیام خطا برای زمانی که یک منبع از پیش موجود است.

/// </summary>

public const string ResourceAlreadyExists = "منبع با این مشخصات از پیش وجود دارد.";

/// <summary>

/// پیام خطا برای زمانی که درخواست شامل پارامترهای اجباری نباشد.

/// </summary>

public const string MissingRequiredParameters = "پارامترهای ضروری در درخواست شما وجود ندارد.";

/// <summary>

/// پیام خطا برای زمانی که توکن یا مجوز نامعتبر است.

/// </summary>

public const string InvalidToken = "توکن شما نامعتبر است. لطفاً دوباره وارد شوید.";

/// <summary>

/// پیام خطا برای زمانی که مقدار ورودی خیلی طولانی است.

/// </summary>

public const string InputTooLong = "ورودی شما خیلی طولانی است.";

/// <summary>

/// پیام خطا برای زمانی که عملیات قطع شده است.

/// </summary>

public const string OperationCancelled = "عملیات شما لغو شد.";

/// <summary>

/// پیام خطا برای زمانی که در اعتبارسنجی فرمت داده‌ها مشکلی وجود دارد.

/// </summary>

public const string InvalidFormat = "فرمت داده‌های وارد شده نادرست است.";

/// <summary>

/// پیام خطا برای زمانی که به دلیل قوانین امنیتی عملیات رد شده است.

/// </summary>

public const string SecurityViolation = "عملیات شما به دلیل نقض قوانین امنیتی رد شد.";

/// <summary>

/// پیام خطا برای زمانی که برنامه از حالت نامناسبی برخوردار است.

/// </summary>

public const string InvalidState = "وضعیت فعلی برنامه نامناسب است.";

/// <summary>

/// پیام خطا برای زمانی که نیاز به ثبت‌نام کاربر است.

/// </summary>

public const string RegistrationRequired = "برای انجام این عملیات نیاز به ثبت‌نام دارید.";

/// <summary>

/// پیام خطا برای زمانی که کاربر قبلاً ثبت‌نام کرده است.

/// </summary>

public const string UserAlreadyRegistered = "این کاربر قبلاً ثبت‌نام کرده است.";

/// <summary>

/// پیام خطا برای زمانی که در فرآیند پرداخت یا تراکنش مشکلی پیش آمده است.

/// </summary>

public const string PaymentError = "خطا در پردازش پرداخت. لطفاً دوباره تلاش کنید.";

/// <summary>

/// پیام خطا برای زمانی که در ذخیره‌سازی داده‌ها مشکلی پیش آمده است.

/// </summary>

public const string DataStorageError = "خطا در ذخیره‌سازی داده‌ها. لطفاً بعداً تلاش کنید.";

/// <summary>

/// پیام خطا برای زمانی که در پردازش درخواست مشکلی پیش آمده است.

/// </summary>

public const string RequestProcessingError = "خطا در پردازش درخواست. لطفاً دوباره تلاش کنید.";

}

}

توضیحات:

- \*\*InvalidInput\*\*: برای زمانی که ورودی کاربر نادرست باشد.

- \*\*Unauthorized\*\*: برای زمانی که کاربر مجوز دسترسی به یک عملیات خاص را نداشته باشد.

- \*\*UserNotFound\*\*: برای زمانی که کاربر یافت نشود.

- \*\*InvalidPassword\*\*: برای زمانی که رمز عبور وارد شده نادرست باشد.

- \*\*Forbidden\*\*: برای زمانی که دسترسی به یک منبع محدود شده است.

- \*\*NotFound\*\*: برای زمانی که یک منبع مورد نظر پیدا نشود.

- \*\*UnverifiedAccount\*\*: برای زمانی که حساب کاربری هنوز تایید نشده باشد.

- \*\*RequestTimeout\*\*: برای زمانی که درخواست زمان‌بر بوده و تایم‌اوت شده است.

- \*\*InternalServerError\*\*: برای زمانی که یک خطای داخلی در سرور رخ دهد.

- \*\*InvalidData\*\*: برای زمانی که داده‌های وارد شده معتبر نباشند.

- \*\*ResourceAlreadyExists\*\*: برای زمانی که یک منبع مشابه از پیش وجود داشته باشد.

- \*\*MissingRequiredParameters\*\*: برای زمانی که پارامترهای ضروری در درخواست موجود نباشند.

- \*\*InvalidToken\*\*: برای زمانی که توکن معتبر نباشد.

- \*\*InputTooLong\*\*: برای زمانی که ورودی کاربر طولانی باشد.

- \*\*OperationCancelled\*\*: برای زمانی که عملیات لغو شده باشد.

- \*\*InvalidFormat\*\*: برای زمانی که فرمت ورودی نادرست باشد.

- \*\*SecurityViolation\*\*: برای زمانی که نقض امنیتی رخ دهد.

- \*\*InvalidState\*\*: برای زمانی که وضعیت برنامه نامناسب باشد.

- \*\*RegistrationRequired\*\*: برای زمانی که کاربر نیاز به ثبت‌نام داشته باشد.

- \*\*UserAlreadyRegistered\*\*: برای زمانی که کاربر قبلاً ثبت‌نام کرده باشد.

- \*\*PaymentError\*\*: برای زمانی که مشکلی در پردازش پرداخت به وجود آید.

- \*\*DataStorageError\*\*: برای زمانی که مشکلی در ذخیره‌سازی داده‌ها رخ دهد.

- \*\*RequestProcessingError\*\*: برای زمانی که مشکلی در پردازش درخواست پیش بیاید.

نکات:

- این کلاس برای مدیریت پیام‌های خطای یکپارچه و استاندارد در پروژه‌ها طراحی شده است.

- پیام‌های خطا می‌توانند در پاسخ‌های API یا در برنامه برای آگاه‌سازی کاربر به کار روند.

- استفاده از این پیام‌ها به صورت ثابت کمک می‌کند که پیام‌های مشابه در پروژه تکرار نشوند و از خطاهای متنی جلوگیری شود.

در اینجا کلاس `ClaimTypesConstants.cs` برای ثابت‌های مربوط به `Claims` در احراز هویت آورده شده است. این کلاس شامل ثابت‌هایی برای انواع مختلف `Claim` که در هنگام احراز هویت و مدیریت دسترسی به سیستم استفاده می‌شوند، می‌باشد.

namespace Infrastructure.Common.Constants

{

/// <summary>

/// کلاس برای نگهداری ثابت‌های مربوط به ClaimTypes در احراز هویت.

/// </summary>

public static class ClaimTypesConstants

{

/// <summary>

/// شناسه کاربری (User ID) در سیستم.

/// </summary>

public const string UserId = "user\_id";

/// <summary>

/// نام کاربری (Username) در سیستم.

/// </summary>

public const string Username = "username";

/// <summary>

/// ایمیل کاربر (Email) در سیستم.

/// </summary>

public const string Email = "email";

/// <summary>

/// نقش کاربر (Role) در سیستم.

/// </summary>

public const string Role = "role";

/// <summary>

/// نام اول کاربر (First Name) در سیستم.

/// </summary>

public const string FirstName = "first\_name";

/// <summary>

/// نام خانوادگی کاربر (Last Name) در سیستم.

/// </summary>

public const string LastName = "last\_name";

/// <summary>

/// شماره تلفن همراه کاربر (Mobile Number) در سیستم.

/// </summary>

public const string MobileNumber = "mobile\_number";

/// <summary>

/// تاریخ تولد کاربر (Date of Birth) در سیستم.

/// </summary>

public const string DateOfBirth = "date\_of\_birth";

/// <summary>

/// شناسه ملی کاربر (National ID) در سیستم.

/// </summary>

public const string NationalId = "national\_id";

/// <summary>

/// وضعیت تأیید ایمیل کاربر (Email Verified Status).

/// </summary>

public const string EmailVerified = "email\_verified";

/// <summary>

/// وضعیت فعال بودن حساب کاربر (Account Active Status).

/// </summary>

public const string IsActive = "is\_active";

/// <summary>

/// نقش دسترسی به سیستم (Access Role) برای تعیین سطح دسترسی.

/// </summary>

public const string AccessRole = "access\_role";

/// <summary>

/// آدرس IP آخرین ورود کاربر (Last Login IP).

/// </summary>

public const string LastLoginIp = "last\_login\_ip";

/// <summary>

/// زمان آخرین ورود کاربر (Last Login Time).

/// </summary>

public const string LastLoginTime = "last\_login\_time";

/// <summary>

/// شناسه سازمان یا تیم کاربر (Organization or Team ID).

/// </summary>

public const string OrganizationId = "organization\_id";

/// <summary>

/// شناسه پروژه‌ای که کاربر در آن فعالیت می‌کند (Project ID).

/// </summary>

public const string ProjectId = "project\_id";

/// <summary>

/// فیلد برای ذخیره کلید API (API Key) در احراز هویت.

/// </summary>

public const string ApiKey = "api\_key";

/// <summary>

/// فیلد برای ذخیره توکن‌های دسترسی (Access Tokens).

/// </summary>

public const string AccessToken = "access\_token";

/// <summary>

/// فیلد برای ذخیره توکن‌های بازنشستگی (Refresh Tokens).

/// </summary>

public const string RefreshToken = "refresh\_token";

/// <summary>

/// سطح دسترسی کاربر به منابع خاص سیستم (Resource Access Level).

/// </summary>

public const string ResourceAccessLevel = "resource\_access\_level";

/// <summary>

/// نوع احراز هویت کاربر (Authentication Type) برای سیستم.

/// </summary>

public const string AuthenticationType = "authentication\_type";

/// <summary>

/// زمان انقضای توکن دسترسی (Access Token Expiration Time).

/// </summary>

public const string AccessTokenExpiration = "access\_token\_expiration";

/// <summary>

/// وضعیت احراز هویت کاربر (Authentication Status).

/// </summary>

public const string AuthenticationStatus = "authentication\_status";

/// <summary>

/// شناسه دستگاه کاربر (Device ID).

/// </summary>

public const string DeviceId = "device\_id";

/// <summary>

/// اطلاعات مربوط به وضعیت تایید حساب کاربری (Account Verification Status).

/// </summary>

public const string AccountVerified = "account\_verified";

/// <summary>

/// نوع دسترسی کاربر (User Access Type).

/// </summary>

public const string UserAccessType = "user\_access\_type";

/// <summary>

/// آخرین تاریخ و زمان تغییر اطلاعات کاربر (Last User Data Modification Time).

/// </summary>

public const string LastDataModificationTime = "last\_data\_modification\_time";

/// <summary>

/// شناسه زبان انتخابی کاربر (Preferred Language ID).

/// </summary>

public const string LanguageId = "language\_id";

/// <summary>

/// نام کاربری در سیستم‌های خارجی (External System Username).

/// </summary>

public const string ExternalSystemUsername = "external\_system\_username";

/// <summary>

/// مدت زمان انقضای توکن (Token Expiration Duration).

/// </summary>

public const string TokenExpirationDuration = "token\_expiration\_duration";

}

}

توضیحات:

- \*\*UserId\*\*: شناسه کاربر در سیستم.

- \*\*Username\*\*: نام کاربری کاربر.

- \*\*Email\*\*: ایمیل کاربر.

- \*\*Role\*\*: نقش کاربر (مثلاً admin، user).

- \*\*FirstName و LastName\*\*: نام و نام خانوادگی کاربر.

- \*\*MobileNumber\*\*: شماره تلفن همراه کاربر.

- \*\*DateOfBirth\*\*: تاریخ تولد کاربر.

- \*\*NationalId\*\*: شناسه ملی کاربر.

- \*\*EmailVerified\*\*: وضعیت تایید ایمیل کاربر.

- \*\*IsActive\*\*: وضعیت فعال بودن حساب کاربر.

- \*\*AccessRole\*\*: نقش دسترسی کاربر به سیستم.

- \*\*LastLoginIp\*\*: آخرین آدرس IP که کاربر وارد شده است.

- \*\*LastLoginTime\*\*: زمان آخرین ورود کاربر.

- \*\*OrganizationId\*\*: شناسه سازمان یا تیم کاربر.

- \*\*ProjectId\*\*: شناسه پروژه کاربر.

- \*\*ApiKey\*\*: کلید API کاربر برای دسترسی به سرویس‌ها.

- \*\*AccessToken و RefreshToken\*\*: توکن‌های دسترسی و بازنشستگی برای احراز هویت.

- \*\*ResourceAccessLevel\*\*: سطح دسترسی به منابع خاص.

- \*\*AuthenticationType\*\*: نوع احراز هویت.

- \*\*AccessTokenExpiration\*\*: زمان انقضای توکن دسترسی.

- \*\*AuthenticationStatus\*\*: وضعیت احراز هویت کاربر.

- \*\*DeviceId\*\*: شناسه دستگاه کاربر.

- \*\*AccountVerified\*\*: وضعیت تایید حساب کاربری.

- \*\*UserAccessType\*\*: نوع دسترسی کاربر.

- \*\*LastDataModificationTime\*\*: آخرین زمان تغییرات داده‌ها.

- \*\*LanguageId\*\*: شناسه زبان انتخابی کاربر.

- \*\*ExternalSystemUsername\*\*: نام کاربری در سیستم‌های خارجی.

- \*\*TokenExpirationDuration\*\*: مدت زمان انقضای توکن.

این ثابت‌ها می‌توانند برای تخصیص یا استخراج اطلاعات مرتبط با احراز هویت و مدیریت دسترسی در سیستم‌های مختلف استفاده شوند.

در اینجا کلاس `ApplicationConstants.cs` برای ثابت‌های عمومی اپلیکیشن آورده شده است. این کلاس شامل ثابت‌هایی است که می‌توانند در بخش‌های مختلف اپلیکیشن برای تنظیمات و مقادیر ثابت استفاده شوند.

namespace Infrastructure.Common.Constants

{

/// <summary>

/// کلاس برای نگهداری ثابت‌های عمومی اپلیکیشن.

/// </summary>

public static class ApplicationConstants

{

/// <summary>

/// نسخه فعلی اپلیکیشن.

/// </summary>

public const string ApplicationVersion = "1.0.0";

/// <summary>

/// نام اپلیکیشن.

/// </summary>

public const string ApplicationName = "MyApplication";

/// <summary>

/// شناسه کاربری پیش‌فرض برای اپلیکیشن.

/// </summary>

public const string DefaultUserId = "default\_user";

/// <summary>

/// پیام پیش‌فرض برای خطاهای عمومی.

/// </summary>

public const string GeneralErrorMessage = "An error occurred. Please try again later.";

/// <summary>

/// پیام موفقیت‌آمیز بودن درخواست.

/// </summary>

public const string SuccessMessage = "Operation completed successfully.";

/// <summary>

/// حداکثر تعداد درخواست‌های مجاز در دقیقه (Rate Limit).

/// </summary>

public const int MaxRequestsPerMinute = 1000;

/// <summary>

/// مدت زمان انقضای نشست (Session Timeout) در ثانیه.

/// </summary>

public const int SessionTimeoutInSeconds = 3600; // 1 hour

/// <summary>

/// فرمت تاریخ و زمان (برای نمایش به کاربران).

/// </summary>

public const string DateFormat = "yyyy-MM-dd HH:mm:ss";

/// <summary>

/// فرمت تاریخ (برای نمایش به کاربران).

/// </summary>

public const string DateOnlyFormat = "yyyy-MM-dd";

/// <summary>

/// فرمت زمان (برای نمایش به کاربران).

/// </summary>

public const string TimeOnlyFormat = "HH:mm:ss";

/// <summary>

/// پیام خطای معتبر نبودن ورودی.

/// </summary>

public const string InvalidInputErrorMessage = "The input provided is invalid.";

/// <summary>

/// پیام خطای نبود داده در سیستم.

/// </summary>

public const string DataNotFoundMessage = "Data not found.";

/// <summary>

/// پیام خطای نداشتن دسترسی.

/// </summary>

public const string UnauthorizedAccessMessage = "You do not have permission to access this resource.";

/// <summary>

/// شناسه پیش‌فرض برای توکن‌های دسترسی.

/// </summary>

public const string DefaultAccessTokenId = "default\_access\_token";

/// <summary>

/// پیام موفقیت‌آمیز بودن احراز هویت.

/// </summary>

public const string AuthenticationSuccessMessage = "Authentication successful.";

/// <summary>

/// پیام خطای عدم موفقیت در احراز هویت.

/// </summary>

public const string AuthenticationFailureMessage = "Authentication failed. Please check your credentials.";

/// <summary>

/// حداکثر تعداد کاراکترهای مجاز برای نام کاربری.

/// </summary>

public const int MaxUsernameLength = 30;

/// <summary>

/// حداقل تعداد کاراکترهای مجاز برای رمز عبور.

/// </summary>

public const int MinPasswordLength = 8;

/// <summary>

/// مدت زمان انقضای توکن‌های دسترسی در ثانیه.

/// </summary>

public const int AccessTokenExpirationTimeInSeconds = 3600; // 1 hour

/// <summary>

/// مدت زمان انقضای توکن‌های بازنشستگی در ثانیه.

/// </summary>

public const int RefreshTokenExpirationTimeInSeconds = 86400; // 24 hours

/// <summary>

/// دامنه (domain) برای کوکی‌ها و سشن‌ها.

/// </summary>

public const string CookieDomain = ".myapplication.com";

/// <summary>

/// تعداد حداکثر مجاز برای سعی در وارد کردن رمز عبور اشتباه.

/// </summary>

public const int MaxLoginAttempts = 5;

/// <summary>

/// پیام موفقیت‌آمیز بودن ثبت‌نام کاربر جدید.

/// </summary>

public const string UserRegistrationSuccessMessage = "User registration successful.";

/// <summary>

/// پیام خطای عدم موفقیت در ثبت‌نام کاربر.

/// </summary>

public const string UserRegistrationFailureMessage = "User registration failed. Please try again later.";

/// <summary>

/// پیام خطای وجود حساب کاربری تکراری.

/// </summary>

public const string DuplicateAccountMessage = "An account with this email or username already exists.";

/// <summary>

/// پیام خطای سیستم برای زمانی که قابلیت‌ها غیرفعال شده‌اند.

/// </summary>

public const string FeatureDisabledMessage = "This feature is currently disabled.";

/// <summary>

/// مشخصات سرور API.

/// </summary>

public const string ApiServerUrl = "https://api.myapplication.com";

/// <summary>

/// پسوند (suffix) فایل‌های بارگذاری شده.

/// </summary>

public const string FileUploadSuffix = "\_uploaded";

/// <summary>

/// حداکثر اندازه فایل بارگذاری شده (به بایت).

/// </summary>

public const int MaxFileUploadSizeInBytes = 10485760; // 10 MB

/// <summary>

/// پیام خطای هنگام بارگذاری فایل.

/// </summary>

public const string FileUploadErrorMessage = "File upload failed. Please try again later.";

/// <summary>

/// پیام خطای هنگام پردازش درخواست.

/// </summary>

public const string RequestProcessingErrorMessage = "An error occurred while processing your request.";

/// <summary>

/// کلید پیش‌فرض برای ذخیره اطلاعات کاربر در کش.

/// </summary>

public const string DefaultUserCacheKey = "user\_cache\_key";

/// <summary>

/// زمان انقضای کش در ثانیه.

/// </summary>

public const int CacheExpirationTimeInSeconds = 600; // 10 minutes

}

}

توضیحات:

- \*\*ApplicationVersion\*\*: نسخه فعلی اپلیکیشن.

- \*\*ApplicationName\*\*: نام اپلیکیشن.

- \*\*DefaultUserId\*\*: شناسه کاربری پیش‌فرض.

- \*\*GeneralErrorMessage\*\*: پیام خطای عمومی.

- \*\*SuccessMessage\*\*: پیام موفقیت‌آمیز بودن عملیات.

- \*\*MaxRequestsPerMinute\*\*: حداکثر تعداد درخواست‌ها در هر دقیقه (برای محدودیت‌های نرخ درخواست).

- \*\*SessionTimeoutInSeconds\*\*: مدت زمان انقضای نشست.

- \*\*DateFormat\*\*: فرمت تاریخ و زمان.

- \*\*InvalidInputErrorMessage\*\*: پیام خطای ورودی نامعتبر.

- \*\*UnauthorizedAccessMessage\*\*: پیام خطای عدم دسترسی.

- \*\*MaxUsernameLength\*\*: حداکثر تعداد کاراکترهای مجاز برای نام کاربری.

- \*\*MinPasswordLength\*\*: حداقل تعداد کاراکترهای مجاز برای رمز عبور.

- \*\*AccessTokenExpirationTimeInSeconds\*\*: مدت زمان انقضای توکن‌های دسترسی.

- \*\*RefreshTokenExpirationTimeInSeconds\*\*: مدت زمان انقضای توکن‌های بازنشستگی.

- \*\*CookieDomain\*\*: دامنه برای کوکی‌ها و سشن‌ها.

- \*\*MaxLoginAttempts\*\*: حداکثر تعداد تلاش‌های وارد کردن رمز عبور اشتباه.

- \*\*UserRegistrationSuccessMessage و UserRegistrationFailureMessage\*\*: پیام‌های موفقیت یا شکست در ثبت‌نام.

- \*\*FeatureDisabledMessage\*\*: پیام خطای غیرفعال بودن ویژگی‌ها.

- \*\*ApiServerUrl\*\*: URL سرور API.

- \*\*MaxFileUploadSizeInBytes\*\*: حداکثر اندازه فایل بارگذاری شده.

- \*\*FileUploadErrorMessage\*\*: پیام خطای بارگذاری فایل.

- \*\*RequestProcessingErrorMessage\*\*: پیام خطای پردازش درخواست.

- \*\*DefaultUserCacheKey\*\*: کلید کش پیش‌فرض برای ذخیره اطلاعات کاربر.

- \*\*CacheExpirationTimeInSeconds\*\*: زمان انقضای کش.

این ثابت‌ها می‌توانند در بخش‌های مختلف اپلیکیشن برای تنظیمات و مدیریت عملیات‌های مختلف استفاده شوند.

در اینجا یک کلاس `DateTimeHelper.cs` برای انجام عملیات‌های کمکی روی تاریخ و زمان آورده شده است. این کلاس شامل متدهای عمومی برای فرمت‌بندی تاریخ و زمان، تبدیل زمان‌ها و دیگر عملیات‌های مرتبط با تاریخ و زمان می‌باشد.

using System;

using System.Globalization;

namespace Infrastructure.Common.Helpers

{

/// <summary>

/// کلاس برای انجام عملیات‌های کمکی روی تاریخ و زمان.

/// </summary>

public static class DateTimeHelper

{

/// <summary>

/// تاریخ و زمان فعلی سیستم را برمی‌گرداند.

/// </summary>

/// <returns>تاریخ و زمان فعلی.</returns>

public static DateTime GetCurrentDateTime()

{

return DateTime.Now;

}

/// <summary>

/// تاریخ و زمان فعلی سیستم را به فرمت مشخص شده تبدیل می‌کند.

/// </summary>

/// <param name="format">فرمت مورد نظر.</param>

/// <returns>تاریخ و زمان فرمت‌شده.</returns>

public static string GetCurrentDateTime(string format)

{

return DateTime.Now.ToString(format, CultureInfo.InvariantCulture);

}

/// <summary>

/// تاریخ و زمان فعلی سیستم را به فرمت تاریخ (yyyy-MM-dd) تبدیل می‌کند.

/// </summary>

/// <returns>تاریخ فرمت‌شده.</returns>

public static string GetCurrentDate()

{

return DateTime.Now.ToString("yyyy-MM-dd", CultureInfo.InvariantCulture);

}

/// <summary>

/// تاریخ و زمان فعلی سیستم را به فرمت زمان (HH:mm:ss) تبدیل می‌کند.

/// </summary>

/// <returns>زمان فرمت‌شده.</returns>

public static string GetCurrentTime()

{

return DateTime.Now.ToString("HH:mm:ss", CultureInfo.InvariantCulture);

}

/// <summary>

/// یک رشته تاریخ را به شیء DateTime تبدیل می‌کند.

/// </summary>

/// <param name="dateString">رشته تاریخ به فرمت yyyy-MM-dd.</param>

/// <returns>شیء DateTime معادل.</returns>

public static DateTime? ConvertStringToDate(string dateString)

{

DateTime date;

if (DateTime.TryParseExact(dateString, "yyyy-MM-dd", CultureInfo.InvariantCulture, DateTimeStyles.None, out date))

{

return date;

}

return null;

}

/// <summary>

/// تاریخ را به فرمت yyyy-MM-dd تبدیل می‌کند.

/// </summary>

/// <param name="date">تاریخ برای فرمت‌بندی.</param>

/// <returns>تاریخ فرمت‌شده.</returns>

public static string ConvertDateToString(DateTime date)

{

return date.ToString("yyyy-MM-dd", CultureInfo.InvariantCulture);

}

/// <summary>

/// تاریخ و زمان را به فرمت مورد نظر تبدیل می‌کند.

/// </summary>

/// <param name="date">تاریخ و زمان برای فرمت‌بندی.</param>

/// <param name="format">فرمت مورد نظر.</param>

/// <returns>تاریخ و زمان فرمت‌شده.</returns>

public static string ConvertDateTimeToString(DateTime date, string format)

{

return date.ToString(format, CultureInfo.InvariantCulture);

}

/// <summary>

/// مدت زمان گذشته از یک تاریخ مشخص را محاسبه می‌کند.

/// </summary>

/// <param name="startDate">تاریخ شروع.</param>

/// <returns>مدت زمان گذشته به صورت رشته (مثلاً 3 روز پیش).</returns>

public static string GetTimeAgo(DateTime startDate)

{

var timeSpan = DateTime.Now - startDate;

if (timeSpan.Days > 365)

{

int years = timeSpan.Days / 365;

return $"{years} year{(years > 1 ? "s" : "")} ago";

}

if (timeSpan.Days > 30)

{

int months = timeSpan.Days / 30;

return $"{months} month{(months > 1 ? "s" : "")} ago";

}

if (timeSpan.Days > 0)

{

return $"{timeSpan.Days} day{(timeSpan.Days > 1 ? "s" : "")} ago";

}

if (timeSpan.Hours > 0)

{

return $"{timeSpan.Hours} hour{(timeSpan.Hours > 1 ? "s" : "")} ago";

}

if (timeSpan.Minutes > 0)

{

return $"{timeSpan.Minutes} minute{(timeSpan.Minutes > 1 ? "s" : "")} ago";

}

return "Just now";

}

/// <summary>

/// تاریخ و زمان را به زمان UTC تبدیل می‌کند.

/// </summary>

/// <param name="date">تاریخ و زمان محلی.</param>

/// <returns>تاریخ و زمان به زمان UTC.</returns>

public static DateTime ConvertToUtc(DateTime date)

{

return date.ToUniversalTime();

}

/// <summary>

/// تاریخ و زمان UTC را به زمان محلی تبدیل می‌کند.

/// </summary>

/// <param name="utcDate">تاریخ و زمان UTC.</param>

/// <returns>تاریخ و زمان محلی.</returns>

public static DateTime ConvertFromUtc(DateTime utcDate)

{

return utcDate.ToLocalTime();

}

/// <summary>

/// تفاوت زمانی بین دو تاریخ را به صورت روز، ساعت، دقیقه و ثانیه برمی‌گرداند.

/// </summary>

/// <param name="startDate">تاریخ شروع.</param>

/// <param name="endDate">تاریخ پایان.</param>

/// <returns>تفاوت زمانی به صورت رشته.</returns>

public static string GetTimeDifference(DateTime startDate, DateTime endDate)

{

var timeSpan = endDate - startDate;

int days = timeSpan.Days;

int hours = timeSpan.Hours;

int minutes = timeSpan.Minutes;

int seconds = timeSpan.Seconds;

return $"{days} day{(days > 1 ? "s" : "")}, {hours} hour{(hours > 1 ? "s" : "")}, {minutes} minute{(minutes > 1 ? "s" : "")}, {seconds} second{(seconds > 1 ? "s" : "")}";

}

/// <summary>

/// بررسی می‌کند که آیا تاریخ وارد شده قبل از تاریخ فعلی است یا خیر.

/// </summary>

/// <param name="date">تاریخ برای بررسی.</param>

/// <returns>برگشت مقدار true اگر تاریخ وارد شده قبل از تاریخ فعلی باشد.</returns>

public static bool IsBeforeCurrentDate(DateTime date)

{

return date < DateTime.Now;

}

/// <summary>

/// بررسی می‌کند که آیا تاریخ وارد شده بعد از تاریخ فعلی است یا خیر.

/// </summary>

/// <param name="date">تاریخ برای بررسی.</param>

/// <returns>برگشت مقدار true اگر تاریخ وارد شده بعد از تاریخ فعلی باشد.</returns>

public static bool IsAfterCurrentDate(DateTime date)

{

return date > DateTime.Now;

}

}

}

توضیحات:

1. \*\*GetCurrentDateTime()\*\*: تاریخ و زمان فعلی سیستم را برمی‌گرداند.

2. \*\*GetCurrentDateTime(string format)\*\*: تاریخ و زمان فعلی را به فرمت دلخواه برمی‌گرداند.

3. \*\*GetCurrentDate()\*\*: تاریخ فعلی به فرمت `yyyy-MM-dd` را برمی‌گرداند.

4. \*\*GetCurrentTime()\*\*: زمان فعلی به فرمت `HH:mm:ss` را برمی‌گرداند.

5. \*\*ConvertStringToDate(string dateString)\*\*: یک رشته تاریخ را به شیء `DateTime` تبدیل می‌کند.

6. \*\*ConvertDateToString(DateTime date)\*\*: یک تاریخ `DateTime` را به رشته‌ای با فرمت `yyyy-MM-dd` تبدیل می‌کند.

7. \*\*ConvertDateTimeToString(DateTime date, string format)\*\*: تاریخ و زمان را به فرمت دلخواه تبدیل می‌کند.

8. \*\*GetTimeAgo(DateTime startDate)\*\*: مدت زمان گذشته از یک تاریخ مشخص را به صورت "x days ago" یا "x months ago" محاسبه می‌کند.

9. \*\*ConvertToUtc(DateTime date)\*\*: تاریخ و زمان محلی را به زمان UTC تبدیل می‌کند.

10. \*\*ConvertFromUtc(DateTime utcDate)\*\*: تاریخ و زمان UTC را به زمان محلی تبدیل می‌کند.

11. \*\*GetTimeDifference(DateTime startDate, DateTime endDate)\*\*: تفاوت زمانی بین دو تاریخ را به صورت روز، ساعت، دقیقه و ثانیه می‌دهد.

12. \*\*IsBeforeCurrentDate(DateTime date)\*\*: بررسی می‌کند که آیا تاریخ وارد شده قبل از تاریخ فعلی است یا خیر.

13. \*\*IsAfterCurrentDate(DateTime date)\*\*: بررسی می‌کند که آیا تاریخ وارد شده بعد از تاریخ فعلی است یا خیر.

این کلاس می‌تواند در پروژه‌های مختلف برای انجام عملیات‌های تاریخ و زمان به صورت مؤثر و آسان استفاده شود.

در اینجا یک کلاس `Guard.cs` برای اعتبارسنجی آرگومان‌ها و مقادیر ورودی آورده شده است. این کلاس شامل متدهایی است که به راحتی می‌توانید مقادیر مختلف مانند رشته‌ها، اعداد، و دیگر انواع داده‌ها را بررسی کنید و خطاهای مناسب را در صورت نادرست بودن مقادیر پرتاب کنید.

using System;

using System.Collections.Generic;

namespace Infrastructure.Common.Helpers

{

/// <summary>

/// کلاس Guard برای انجام اعتبارسنجی بر روی آرگومان‌ها و مقادیر ورودی.

/// </summary>

public static class Guard

{

/// <summary>

/// بررسی می‌کند که مقدار ورودی نباید نال باشد.

/// </summary>

/// <param name="value">مقدار ورودی.</param>

/// <param name="paramName">نام پارامتر برای پرتاب استثنا.</param>

/// <exception cref="ArgumentNullException">اگر مقدار ورودی نال باشد.</exception>

public static void AgainstNull(object value, string paramName)

{

if (value == null)

{

throw new ArgumentNullException(paramName, "Value cannot be null.");

}

}

/// <summary>

/// بررسی می‌کند که یک رشته نباید نال یا خالی باشد.

/// </summary>

/// <param name="value">مقدار ورودی.</param>

/// <param name="paramName">نام پارامتر برای پرتاب استثنا.</param>

/// <exception cref="ArgumentException">اگر رشته ورودی نال یا خالی باشد.</exception>

public static void AgainstNullOrEmpty(string value, string paramName)

{

if (string.IsNullOrEmpty(value))

{

throw new ArgumentException("Value cannot be null or empty.", paramName);

}

}

/// <summary>

/// بررسی می‌کند که یک رشته نباید نال یا فقط فضای خالی باشد.

/// </summary>

/// <param name="value">مقدار ورودی.</param>

/// <param name="paramName">نام پارامتر برای پرتاب استثنا.</param>

/// <exception cref="ArgumentException">اگر رشته ورودی نال یا فقط فضای خالی باشد.</exception>

public static void AgainstNullOrWhiteSpace(string value, string paramName)

{

if (string.IsNullOrWhiteSpace(value))

{

throw new ArgumentException("Value cannot be null, empty or whitespace.", paramName);

}

}

/// <summary>

/// بررسی می‌کند که یک عدد باید بزرگتر از مقدار مشخص شده باشد.

/// </summary>

/// <param name="value">مقدار ورودی.</param>

/// <param name="minValue">مقدار حداقل مجاز.</param>

/// <param name="paramName">نام پارامتر برای پرتاب استثنا.</param>

/// <exception cref="ArgumentOutOfRangeException">اگر مقدار ورودی کمتر از حداقل مجاز باشد.</exception>

public static void AgainstLessThan(int value, int minValue, string paramName)

{

if (value < minValue)

{

throw new ArgumentOutOfRangeException(paramName, $"Value cannot be less than {minValue}.");

}

}

/// <summary>

/// بررسی می‌کند که یک عدد باید کوچکتر از مقدار مشخص شده باشد.

/// </summary>

/// <param name="value">مقدار ورودی.</param>

/// <param name="maxValue">مقدار حداکثر مجاز.</param>

/// <param name="paramName">نام پارامتر برای پرتاب استثنا.</param>

/// <exception cref="ArgumentOutOfRangeException">اگر مقدار ورودی بیشتر از حداقل مجاز باشد.</exception>

public static void AgainstGreaterThan(int value, int maxValue, string paramName)

{

if (value > maxValue)

{

throw new ArgumentOutOfRangeException(paramName, $"Value cannot be greater than {maxValue}.");

}

}

/// <summary>

/// بررسی می‌کند که مقدار ورودی باید یکی از مقادیر موجود در لیست باشد.

/// </summary>

/// <typeparam name="T">نوع داده برای بررسی.</typeparam>

/// <param name="value">مقدار ورودی.</param>

/// <param name="validValues">لیست مقادیر معتبر.</param>

/// <param name="paramName">نام پارامتر برای پرتاب استثنا.</param>

/// <exception cref="ArgumentException">اگر مقدار ورودی در لیست معتبر نباشد.</exception>

public static void AgainstInvalidEnumValue<T>(T value, List<T> validValues, string paramName)

{

if (!validValues.Contains(value))

{

throw new ArgumentException($"Value is not a valid {typeof(T).Name}.", paramName);

}

}

/// <summary>

/// بررسی می‌کند که یک تاریخ باید در گذشته باشد.

/// </summary>

/// <param name="date">تاریخ ورودی.</param>

/// <param name="paramName">نام پارامتر برای پرتاب استثنا.</param>

/// <exception cref="ArgumentException">اگر تاریخ وارد شده بعد از تاریخ کنونی باشد.</exception>

public static void AgainstFutureDate(DateTime date, string paramName)

{

if (date > DateTime.Now)

{

throw new ArgumentException("Date cannot be in the future.", paramName);

}

}

/// <summary>

/// بررسی می‌کند که یک تاریخ باید در گذشته یا هم‌اکنون باشد.

/// </summary>

/// <param name="date">تاریخ ورودی.</param>

/// <param name="paramName">نام پارامتر برای پرتاب استثنا.</param>

/// <exception cref="ArgumentException">اگر تاریخ وارد شده بعد از تاریخ کنونی باشد.</exception>

public static void AgainstFutureOrNowDate(DateTime date, string paramName)

{

if (date >= DateTime.Now)

{

throw new ArgumentException("Date cannot be in the future or now.", paramName);

}

}

/// <summary>

/// بررسی می‌کند که یک تاریخ باید در آینده باشد.

/// </summary>

/// <param name="date">تاریخ ورودی.</param>

/// <param name="paramName">نام پارامتر برای پرتاب استثنا.</param>

/// <exception cref="ArgumentException">اگر تاریخ وارد شده قبل از تاریخ کنونی باشد.</exception>

public static void AgainstPastDate(DateTime date, string paramName)

{

if (date < DateTime.Now)

{

throw new ArgumentException("Date cannot be in the past.", paramName);

}

}

/// <summary>

/// بررسی می‌کند که مقدار ورودی باید یک آدرس ایمیل معتبر باشد.

/// </summary>

/// <param name="email">آدرس ایمیل ورودی.</param>

/// <param name="paramName">نام پارامتر برای پرتاب استثنا.</param>

/// <exception cref="ArgumentException">اگر آدرس ایمیل ورودی معتبر نباشد.</exception>

public static void AgainstInvalidEmail(string email, string paramName)

{

if (string.IsNullOrWhiteSpace(email) || !email.Contains("@") || !email.Contains("."))

{

throw new ArgumentException("Invalid email address.", paramName);

}

}

/// <summary>

/// بررسی می‌کند که یک مقدار ورودی باید بزرگتر از صفر باشد.

/// </summary>

/// <param name="value">مقدار ورودی.</param>

/// <param name="paramName">نام پارامتر برای پرتاب استثنا.</param>

/// <exception cref="ArgumentException">اگر مقدار ورودی کمتر از یا برابر با صفر باشد.</exception>

public static void AgainstZeroOrLess(int value, string paramName)

{

if (value <= 0)

{

throw new ArgumentException("Value cannot be zero or less.", paramName);

}

}

}

}

توضیحات:

1. \*\*AgainstNull\*\*: بررسی می‌کند که مقدار ورودی نال نباشد.

2. \*\*AgainstNullOrEmpty\*\*: بررسی می‌کند که رشته ورودی نال یا خالی نباشد.

3. \*\*AgainstNullOrWhiteSpace\*\*: بررسی می‌کند که رشته ورودی نال، خالی یا فقط فضای خالی نباشد.

4. \*\*AgainstLessThan\*\*: بررسی می‌کند که یک عدد باید بزرگتر از یک مقدار مشخص شده باشد.

5. \*\*AgainstGreaterThan\*\*: بررسی می‌کند که یک عدد باید کوچکتر از یک مقدار مشخص شده باشد.

6. \*\*AgainstInvalidEnumValue\*\*: بررسی می‌کند که مقدار ورودی یکی از مقادیر معتبر در لیست باشد.

7. \*\*AgainstFutureDate\*\*: بررسی می‌کند که تاریخ وارد شده در آینده نباشد.

8. \*\*AgainstFutureOrNowDate\*\*: بررسی می‌کند که تاریخ وارد شده نه در آینده باشد و نه همین حالا.

9. \*\*AgainstPastDate\*\*: بررسی می‌کند که تاریخ وارد شده در گذشته نباشد.

10. \*\*AgainstInvalidEmail\*\*: بررسی می‌کند که ایمیل ورودی معتبر باشد.

11. \*\*AgainstZeroOrLess\*\*: بررسی می‌کند که مقدار ورودی بزرگتر از صفر باشد.

این کلاس می‌تواند برای اعتبارسنجی مقادیر ورودی و جلوگیری از خطاهای ناخواسته در طول اجرای برنامه مفید باشد.

در اینجا کد کامل کلاس `GuidHelper.cs` برای تولید و اعتبارسنجی شناسه‌های GUID آورده شده است. این کلاس می‌تواند برای تولید GUIDهای جدید و همچنین اعتبارسنجی اینکه یک رشته به درستی به عنوان GUID معتبر است یا خیر، استفاده شود.

using System;

namespace Infrastructure.Common.Helpers

{

/// <summary>

/// کلاس GuidHelper برای تولید و اعتبارسنجی شناسه‌های GUID.

/// </summary>

public static class GuidHelper

{

/// <summary>

/// تولید یک شناسه GUID جدید.

/// </summary>

/// <returns>یک شناسه GUID جدید.</returns>

public static Guid NewGuid()

{

return Guid.NewGuid();

}

/// <summary>

/// اعتبارسنجی اینکه یک رشته معتبر برای تبدیل به GUID است یا خیر.

/// </summary>

/// <param name="guidString">رشته ورودی برای بررسی.</param>

/// <returns>بررسی می‌کند که آیا رشته ورودی یک GUID معتبر است یا خیر.</returns>

public static bool IsValidGuid(string guidString)

{

return Guid.TryParse(guidString, out \_);

}

/// <summary>

/// تبدیل یک رشته به شناسه GUID. اگر رشته ورودی نامعتبر باشد، یک استثنا پرتاب می‌شود.

/// </summary>

/// <param name="guidString">رشته ورودی برای تبدیل به GUID.</param>

/// <returns>شناسه GUID مربوط به رشته ورودی.</returns>

/// <exception cref="ArgumentException">اگر رشته ورودی یک GUID معتبر نباشد.</exception>

public static Guid ToGuid(string guidString)

{

if (Guid.TryParse(guidString, out Guid result))

{

return result;

}

throw new ArgumentException("The provided string is not a valid GUID.", nameof(guidString));

}

/// <summary>

/// اعتبارسنجی اینکه یک شناسه GUID نباید خالی باشد (شناسه GUID صفر).

/// </summary>

/// <param name="guid">شناسه GUID برای بررسی.</param>

/// <param name="paramName">نام پارامتر برای پرتاب استثنا.</param>

/// <exception cref="ArgumentException">اگر شناسه GUID خالی (صفر) باشد.</exception>

public static void AgainstEmpty(Guid guid, string paramName)

{

if (guid == Guid.Empty)

{

throw new ArgumentException("GUID cannot be empty.", paramName);

}

}

/// <summary>

/// بررسی می‌کند که یک GUID باید برابر با GUID دیگری باشد.

/// </summary>

/// <param name="guid">شناسه GUID برای بررسی.</param>

/// <param name="compareGuid">شناسه GUID مقایسه‌ای.</param>

/// <param name="paramName">نام پارامتر برای پرتاب استثنا.</param>

/// <exception cref="ArgumentException">اگر شناسه GUID ورودی با شناسه GUID مقایسه‌ای برابر نباشد.</exception>

public static void AgainstGuidNotEqual(Guid guid, Guid compareGuid, string paramName)

{

if (guid != compareGuid)

{

throw new ArgumentException($"GUID should be equal to {compareGuid}.", paramName);

}

}

}

}

توضیحات:

1. \*\*NewGuid\*\*: این متد یک شناسه GUID جدید ایجاد می‌کند.

2. \*\*IsValidGuid\*\*: این متد بررسی می‌کند که آیا رشته ورودی یک GUID معتبر است یا خیر.

3. \*\*ToGuid\*\*: این متد یک رشته ورودی را به GUID تبدیل می‌کند. اگر رشته ورودی یک GUID معتبر نباشد، یک استثنا پرتاب می‌شود.

4. \*\*AgainstEmpty\*\*: این متد بررسی می‌کند که آیا شناسه GUID ورودی خالی (صفر) است یا خیر و اگر خالی باشد، یک استثنا پرتاب می‌کند.

5. \*\*AgainstGuidNotEqual\*\*: این متد بررسی می‌کند که آیا شناسه GUID ورودی با یک GUID دیگر برابر است یا خیر. اگر برابر نباشد، یک استثنا پرتاب می‌کند.

این کلاس به شما این امکان را می‌دهد که به راحتی شناسه‌های GUID جدید ایجاد کرده و همچنین آنها را اعتبارسنجی کنید تا مطمئن شوید که ورودی‌ها معتبر و صحیح هستند.

در اینجا کد کامل کلاس `StringHelper.cs` برای متدهای کمکی مربوط به رشته‌ها آورده شده است. این کلاس شامل تعدادی متد کمکی برای کار با رشته‌ها مانند بررسی خالی بودن، تبدیل به حروف بزرگ/کوچک، و استخراج اطلاعات از رشته‌ها است.

using System;

using System.Linq;

using System.Text.RegularExpressions;

namespace Infrastructure.Common.Helpers

{

/// <summary>

/// کلاس StringHelper شامل متدهای کمکی برای کار با رشته‌ها.

/// </summary>

public static class StringHelper

{

/// <summary>

/// بررسی می‌کند که آیا رشته ورودی خالی یا فقط شامل فضای سفید است.

/// </summary>

/// <param name="input">رشته ورودی برای بررسی.</param>

/// <returns>برمی‌گرداند true اگر رشته خالی یا فقط فضای سفید باشد، در غیر این صورت false.</returns>

public static bool IsNullOrWhitespace(string input)

{

return string.IsNullOrWhiteSpace(input);

}

/// <summary>

/// بررسی می‌کند که آیا رشته ورودی فقط شامل اعداد است.

/// </summary>

/// <param name="input">رشته ورودی برای بررسی.</param>

/// <returns>برمی‌گرداند true اگر رشته فقط شامل اعداد باشد، در غیر این صورت false.</returns>

public static bool IsNumeric(string input)

{

return input.All(char.IsDigit);

}

/// <summary>

/// تبدیل یک رشته به حروف بزرگ.

/// </summary>

/// <param name="input">رشته ورودی.</param>

/// <returns>رشته تبدیل شده به حروف بزرگ.</returns>

public static string ToUpperCase(string input)

{

return input?.ToUpper();

}

/// <summary>

/// تبدیل یک رشته به حروف کوچک.

/// </summary>

/// <param name="input">رشته ورودی.</param>

/// <returns>رشته تبدیل شده به حروف کوچک.</returns>

public static string ToLowerCase(string input)

{

return input?.ToLower();

}

/// <summary>

/// حذف فاصله‌های اضافی از ابتدا و انتهای رشته.

/// </summary>

/// <param name="input">رشته ورودی.</param>

/// <returns>رشته با فاصله‌های حذف شده از ابتدا و انتها.</returns>

public static string TrimWhitespace(string input)

{

return input?.Trim();

}

/// <summary>

/// تبدیل رشته به فرمت Title Case (حروف اول کلمات بزرگ می‌شوند).

/// </summary>

/// <param name="input">رشته ورودی.</param>

/// <returns>رشته تبدیل شده به فرمت Title Case.</returns>

public static string ToTitleCase(string input)

{

if (string.IsNullOrWhiteSpace(input)) return input;

var words = input.Split(' ');

var titleCaseWords = words.Select(word =>

word.Length > 0

? char.ToUpper(word[0]) + word.Substring(1).ToLower()

: word);

return string.Join(" ", titleCaseWords);

}

/// <summary>

/// بررسی می‌کند که آیا رشته ورودی حاوی یک الگوی خاص (regex) است.

/// </summary>

/// <param name="input">رشته ورودی.</param>

/// <param name="pattern">الگوی regex برای جستجو.</param>

/// <returns>برمی‌گرداند true اگر الگوی regex در رشته پیدا شود، در غیر این صورت false.</returns>

public static bool MatchesPattern(string input, string pattern)

{

return Regex.IsMatch(input, pattern);

}

/// <summary>

/// حذف کاراکترهای خاص یا حروف اضافی از رشته ورودی.

/// </summary>

/// <param name="input">رشته ورودی.</param>

/// <param name="charsToRemove">کاراکترهایی که باید از رشته حذف شوند.</param>

/// <returns>رشته با حذف کاراکترهای خاص.</returns>

public static string RemoveCharacters(string input, string charsToRemove)

{

if (string.IsNullOrWhiteSpace(input) || string.IsNullOrWhiteSpace(charsToRemove))

return input;

return new string(input.Where(c => !charsToRemove.Contains(c)).ToArray());

}

/// <summary>

/// تقسیم یک رشته به آرایه‌ای از رشته‌ها با استفاده از جداکننده خاص.

/// </summary>

/// <param name="input">رشته ورودی.</param>

/// <param name="separator">جداکننده برای تقسیم رشته.</param>

/// <returns>آرایه‌ای از رشته‌ها.</returns>

public static string[] SplitBySeparator(string input, char separator)

{

return input?.Split(separator);

}

/// <summary>

/// تبدیل یک رشته به Base64.

/// </summary>

/// <param name="input">رشته ورودی.</param>

/// <returns>رشته ورودی به فرمت Base64 تبدیل شده.</returns>

public static string ToBase64(string input)

{

if (input == null) return null;

byte[] bytes = System.Text.Encoding.UTF8.GetBytes(input);

return Convert.ToBase64String(bytes);

}

/// <summary>

/// تبدیل یک رشته Base64 به رشته اصلی.

/// </summary>

/// <param name="input">رشته Base64.</param>

/// <returns>رشته اصلی پس از تبدیل از Base64.</returns>

public static string FromBase64(string input)

{

if (input == null) return null;

byte[] bytes = Convert.FromBase64String(input);

return System.Text.Encoding.UTF8.GetString(bytes);

}

}

}

توضیحات:

1. \*\*IsNullOrWhitespace\*\*: بررسی می‌کند که آیا رشته ورودی خالی یا فقط شامل فضای سفید است.

2. \*\*IsNumeric\*\*: بررسی می‌کند که آیا رشته ورودی فقط شامل اعداد است.

3. \*\*ToUpperCase\*\*: تبدیل رشته به حروف بزرگ.

4. \*\*ToLowerCase\*\*: تبدیل رشته به حروف کوچک.

5. \*\*TrimWhitespace\*\*: حذف فاصله‌های اضافی از ابتدا و انتهای رشته.

6. \*\*ToTitleCase\*\*: تبدیل رشته به فرمت "Title Case" (حروف اول کلمات بزرگ می‌شوند).

7. \*\*MatchesPattern\*\*: بررسی می‌کند که آیا رشته ورودی با یک الگوی خاص (regex) مطابقت دارد.

8. \*\*RemoveCharacters\*\*: حذف کاراکترهای خاص یا حروف اضافی از رشته.

9. \*\*SplitBySeparator\*\*: تقسیم یک رشته به آرایه‌ای از رشته‌ها با استفاده از جداکننده خاص.

10. \*\*ToBase64\*\*: تبدیل یک رشته به فرمت Base64.

11. \*\*FromBase64\*\*: تبدیل یک رشته Base64 به رشته اصلی.

این کلاس به شما کمک می‌کند تا عملیات مختلفی روی رشته‌ها انجام دهید، از جمله تبدیل حروف، بررسی الگوها، و تبدیل داده‌ها به فرمت‌های مختلف مانند Base64.

در اینجا کد کامل کلاس `CryptoRandomHelper.cs` برای تولید مقادیر تصادفی رمزنگاری شده آورده شده است. این کلاس از الگوریتم‌های امنیتی مانند `RNGCryptoServiceProvider` برای تولید اعداد تصادفی استفاده می‌کند که در مواردی مانند تولید کلیدها، توکن‌ها و هش‌های امن کاربرد دارد.

using System;

using System.Security.Cryptography;

namespace Infrastructure.Common.Helpers

{

/// <summary>

/// کلاس CryptoRandomHelper شامل متدهای کمکی برای تولید مقادیر تصادفی رمزنگاری شده.

/// </summary>

public static class CryptoRandomHelper

{

private static readonly RNGCryptoServiceProvider \_rng = new RNGCryptoServiceProvider();

/// <summary>

/// تولید یک عدد تصادفی بین دو مقدار مشخص شده.

/// </summary>

/// <param name="minValue">حداقل مقدار.</param>

/// <param name="maxValue">حداکثر مقدار.</param>

/// <returns>عدد تصادفی تولید شده.</returns>

public static int GetRandomInt(int minValue, int maxValue)

{

if (minValue >= maxValue)

{

throw new ArgumentOutOfRangeException("minValue باید کمتر از maxValue باشد.");

}

byte[] randomBytes = new byte[4];

\_rng.GetBytes(randomBytes);

int randomValue = BitConverter.ToInt32(randomBytes, 0);

// مقیاس دهی به محدوده

return Math.Abs(randomValue % (maxValue - minValue)) + minValue;

}

/// <summary>

/// تولید یک رشته تصادفی از حروف و اعداد.

/// </summary>

/// <param name="length">طول رشته تصادفی.</param>

/// <returns>رشته تصادفی تولید شده.</returns>

public static string GetRandomString(int length)

{

if (length <= 0)

{

throw new ArgumentOutOfRangeException("طول رشته باید مثبت باشد.");

}

const string chars = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789";

char[] stringChars = new char[length];

byte[] randomBytes = new byte[length];

\_rng.GetBytes(randomBytes);

for (int i = 0; i < length; i++)

{

// مقیاس دهی برای انتخاب از مجموعه حروف و اعداد

stringChars[i] = chars[randomBytes[i] % chars.Length];

}

return new string(stringChars);

}

/// <summary>

/// تولید یک رشته تصادفی از حروف الفبا (بدون اعداد).

/// </summary>

/// <param name="length">طول رشته تصادفی.</param>

/// <returns>رشته تصادفی تولید شده.</returns>

public static string GetRandomAlphabeticString(int length)

{

if (length <= 0)

{

throw new ArgumentOutOfRangeException("طول رشته باید مثبت باشد.");

}

const string chars = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz";

char[] stringChars = new char[length];

byte[] randomBytes = new byte[length];

\_rng.GetBytes(randomBytes);

for (int i = 0; i < length; i++)

{

// مقیاس دهی برای انتخاب از مجموعه حروف الفبا

stringChars[i] = chars[randomBytes[i] % chars.Length];

}

return new string(stringChars);

}

/// <summary>

/// تولید یک عدد تصادفی بین 0 و 1 (مقدار شناور).

/// </summary>

/// <returns>عدد تصادفی بین 0 و 1.</returns>

public static double GetRandomDouble()

{

byte[] randomBytes = new byte[8];

\_rng.GetBytes(randomBytes);

ulong randomValue = BitConverter.ToUInt64(randomBytes, 0);

// تبدیل به یک عدد شناور بین 0 و 1

return (randomValue / (double)ulong.MaxValue);

}

/// <summary>

/// تولید یک آرایه از بایت‌های تصادفی.

/// </summary>

/// <param name="length">طول آرایه بایت‌ها.</param>

/// <returns>آرایه بایت‌های تصادفی تولید شده.</returns>

public static byte[] GetRandomBytes(int length)

{

if (length <= 0)

{

throw new ArgumentOutOfRangeException("طول آرایه باید مثبت باشد.");

}

byte[] randomBytes = new byte[length];

\_rng.GetBytes(randomBytes);

return randomBytes;

}

/// <summary>

/// تولید یک GUID تصادفی.

/// </summary>

/// <returns>GUID تصادفی تولید شده.</returns>

public static Guid GetRandomGuid()

{

byte[] randomBytes = new byte[16];

\_rng.GetBytes(randomBytes);

return new Guid(randomBytes);

}

/// <summary>

/// تولید یک عدد تصادفی در محدوده خاص به صورت شناور (float).

/// </summary>

/// <param name="minValue">حداقل مقدار.</param>

/// <param name="maxValue">حداکثر مقدار.</param>

/// <returns>عدد تصادفی شناور تولید شده.</returns>

public static float GetRandomFloat(float minValue, float maxValue)

{

if (minValue >= maxValue)

{

throw new ArgumentOutOfRangeException("minValue باید کمتر از maxValue باشد.");

}

byte[] randomBytes = new byte[4];

\_rng.GetBytes(randomBytes);

int randomValue = BitConverter.ToInt32(randomBytes, 0);

// مقیاس دهی به محدوده شناور

return (float)(minValue + (randomValue / (double)int.MaxValue) \* (maxValue - minValue));

}

}

}

توضیحات متدها:

1. \*\*GetRandomInt\*\*: تولید یک عدد تصادفی در محدوده مشخص شده.

2. \*\*GetRandomString\*\*: تولید یک رشته تصادفی از حروف و اعداد.

3. \*\*GetRandomAlphabeticString\*\*: تولید یک رشته تصادفی از حروف الفبا (بدون اعداد).

4. \*\*GetRandomDouble\*\*: تولید یک عدد تصادفی شناور بین 0 و 1.

5. \*\*GetRandomBytes\*\*: تولید یک آرایه بایت‌های تصادفی.

6. \*\*GetRandomGuid\*\*: تولید یک GUID تصادفی.

7. \*\*GetRandomFloat\*\*: تولید یک عدد تصادفی شناور (float) در محدوده مشخص شده.

این کلاس برای تولید مقادیر تصادفی رمزنگاری شده با استفاده از `RNGCryptoServiceProvider` که یک تولید کننده اعداد تصادفی قوی و امن است، مناسب است. این مقادیر می‌توانند برای کاربردهایی مانند تولید توکن‌ها، کلیدهای امنیتی، یا حتی اعتبارسنجی‌ها استفاده شوند.

در اینجا کد کامل کلاس `UrlHelper.cs` برای ساخت و ویرایش URL ها آورده شده است. این کلاس شامل متدهایی برای انجام عملیات مختلف بر روی URL ها مانند اضافه کردن پارامترهای کوئری، استخراج بخش‌های مختلف URL، و ترکیب بخش‌های مختلف URL است.

using System;

using System.Collections.Specialized;

using System.Linq;

using System.Web;

namespace Infrastructure.Common.Helpers

{

/// <summary>

/// کلاس UrlHelper برای ساخت و ویرایش URL ها.

/// </summary>

public static class UrlHelper

{

/// <summary>

/// اضافه کردن پارامترهای کوئری به یک URL موجود.

/// </summary>

/// <param name="url">URL اصلی.</param>

/// <param name="queryParams">پارامترهای کوئری که باید اضافه شوند.</param>

/// <returns>URL جدید با پارامترهای کوئری اضافه شده.</returns>

public static string AddQueryParams(string url, NameValueCollection queryParams)

{

if (string.IsNullOrEmpty(url) || queryParams == null || queryParams.Count == 0)

{

return url;

}

UriBuilder uriBuilder = new UriBuilder(url);

var query = HttpUtility.ParseQueryString(uriBuilder.Query);

foreach (string key in queryParams.AllKeys)

{

query[key] = queryParams[key];

}

uriBuilder.Query = query.ToString();

return uriBuilder.ToString();

}

/// <summary>

/// استخراج بخش مسیر (Path) از یک URL.

/// </summary>

/// <param name="url">URL که می‌خواهید بخش مسیر آن را استخراج کنید.</param>

/// <returns>بخش مسیر URL.</returns>

public static string GetPath(string url)

{

if (string.IsNullOrEmpty(url))

{

return string.Empty;

}

Uri uri = new Uri(url);

return uri.AbsolutePath;

}

/// <summary>

/// استخراج بخش دامنه (Host) از یک URL.

/// </summary>

/// <param name="url">URL که می‌خواهید بخش دامنه آن را استخراج کنید.</param>

/// <returns>بخش دامنه URL.</returns>

public static string GetHost(string url)

{

if (string.IsNullOrEmpty(url))

{

return string.Empty;

}

Uri uri = new Uri(url);

return uri.Host;

}

/// <summary>

/// استخراج بخش پروتکل (Scheme) از یک URL (مانند http، https).

/// </summary>

/// <param name="url">URL که می‌خواهید بخش پروتکل آن را استخراج کنید.</param>

/// <returns>بخش پروتکل URL.</returns>

public static string GetScheme(string url)

{

if (string.IsNullOrEmpty(url))

{

return string.Empty;

}

Uri uri = new Uri(url);

return uri.Scheme;

}

/// <summary>

/// استخراج پارامترهای کوئری از یک URL.

/// </summary>

/// <param name="url">URL که می‌خواهید پارامترهای کوئری آن را استخراج کنید.</param>

/// <returns>پارامترهای کوئری URL به صورت یک NameValueCollection.</returns>

public static NameValueCollection GetQueryParams(string url)

{

if (string.IsNullOrEmpty(url))

{

return null;

}

Uri uri = new Uri(url);

return HttpUtility.ParseQueryString(uri.Query);

}

/// <summary>

/// ترکیب بخش‌های مختلف URL به یک URL کامل.

/// </summary>

/// <param name="scheme">پروتکل (مانند http یا https).</param>

/// <param name="host">دامنه یا آدرس سرور.</param>

/// <param name="path">مسیر URL.</param>

/// <param name="queryParams">پارامترهای کوئری.</param>

/// <returns>URL کامل ساخته شده.</returns>

public static string CombineUrl(string scheme, string host, string path, NameValueCollection queryParams)

{

if (string.IsNullOrEmpty(scheme) || string.IsNullOrEmpty(host))

{

throw new ArgumentException("پروتکل (scheme) و دامنه (host) باید مشخص باشند.");

}

UriBuilder uriBuilder = new UriBuilder

{

Scheme = scheme,

Host = host,

Path = path ?? string.Empty

};

if (queryParams != null && queryParams.Count > 0)

{

var query = HttpUtility.ParseQueryString(uriBuilder.Query);

foreach (string key in queryParams.AllKeys)

{

query[key] = queryParams[key];

}

uriBuilder.Query = query.ToString();

}

return uriBuilder.ToString();

}

/// <summary>

/// بررسی اینکه آیا یک URL معتبر است یا خیر.

/// </summary>

/// <param name="url">URL که می‌خواهید اعتبار آن را بررسی کنید.</param>

/// <returns>برگشت true اگر URL معتبر باشد، در غیر اینصورت false.</returns>

public static bool IsValidUrl(string url)

{

if (string.IsNullOrEmpty(url))

{

return false;

}

Uri uriResult;

return Uri.TryCreate(url, UriKind.Absolute, out uriResult) && uriResult.Scheme == Uri.UriSchemeHttp || uriResult.Scheme == Uri.UriSchemeHttps;

}

/// <summary>

/// حذف پارامتر کوئری از URL.

/// </summary>

/// <param name="url">URL که می‌خواهید پارامتر آن را حذف کنید.</param>

/// <param name="paramName">نام پارامتر کوئری که باید حذف شود.</param>

/// <returns>URL جدید با پارامتر حذف شده.</returns>

public static string RemoveQueryParam(string url, string paramName)

{

if (string.IsNullOrEmpty(url) || string.IsNullOrEmpty(paramName))

{

return url;

}

UriBuilder uriBuilder = new UriBuilder(url);

var query = HttpUtility.ParseQueryString(uriBuilder.Query);

query.Remove(paramName);

uriBuilder.Query = query.ToString();

return uriBuilder.ToString();

}

}

}

توضیحات متدها:

1. \*\*AddQueryParams\*\*: اضافه کردن پارامترهای کوئری به یک URL موجود.

2. \*\*GetPath\*\*: استخراج بخش مسیر (Path) از یک URL.

3. \*\*GetHost\*\*: استخراج بخش دامنه (Host) از یک URL.

4. \*\*GetScheme\*\*: استخراج بخش پروتکل (Scheme) از یک URL (مثل `http` یا `https`).

5. \*\*GetQueryParams\*\*: استخراج پارامترهای کوئری از یک URL به صورت `NameValueCollection`.

6. \*\*CombineUrl\*\*: ترکیب بخش‌های مختلف URL مانند پروتکل، دامنه، مسیر، و پارامترهای کوئری به یک URL کامل.

7. \*\*IsValidUrl\*\*: بررسی معتبر بودن یک URL.

8. \*\*RemoveQueryParam\*\*: حذف یک پارامتر کوئری از یک URL.

این کلاس ابزارهایی را برای کار با URL ها در برنامه‌های وب و API فراهم می‌آورد که می‌تواند در مواردی مانند مدیریت پارامترهای کوئری، اعتبارسنجی URL ها و ترکیب URL ها استفاده شود.

در اینجا کد کامل کلاس `PasswordGenerator.cs` برای تولید رمز عبور ایمن آورده شده است. این کلاس شامل متدی برای تولید رمز عبور ایمن با استفاده از کاراکترهای تصادفی و قابل تنظیم برای طول رمز عبور است.

using System;

using System.Security.Cryptography;

using System.Text;

namespace Infrastructure.Common.Helpers

{

/// <summary>

/// کلاس PasswordGenerator برای تولید رمز عبور ایمن.

/// </summary>

public static class PasswordGenerator

{

private static readonly string UppercaseChars = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";

private static readonly string LowercaseChars = "abcdefghijklmnopqrstuvwxyz";

private static readonly string Digits = "0123456789";

private static readonly string SpecialChars = "!@#$%^&\*()-\_=+[]{}|;:'\",.<>?/";

/// <summary>

/// تولید یک رمز عبور ایمن به طول مشخص.

/// </summary>

/// <param name="length">طول رمز عبور مورد نظر.</param>

/// <param name="requireUppercase">آیا باید حروف بزرگ شامل شوند؟</param>

/// <param name="requireLowercase">آیا باید حروف کوچک شامل شوند؟</param>

/// <param name="requireDigits">آیا باید اعداد شامل شوند؟</param>

/// <param name="requireSpecialChars">آیا باید کاراکترهای خاص شامل شوند؟</param>

/// <returns>رمز عبور ایمن تولید شده.</returns>

public static string GeneratePassword(int length, bool requireUppercase = true, bool requireLowercase = true,

bool requireDigits = true, bool requireSpecialChars = true)

{

if (length < 8)

{

throw new ArgumentException("رمز عبور باید حداقل 8 کاراکتر باشد.", nameof(length));

}

StringBuilder password = new StringBuilder();

string allowedChars = string.Empty;

// اضافه کردن کاراکترهای مجاز به رمز عبور

if (requireUppercase)

allowedChars += UppercaseChars;

if (requireLowercase)

allowedChars += LowercaseChars;

if (requireDigits)

allowedChars += Digits;

if (requireSpecialChars)

allowedChars += SpecialChars;

if (string.IsNullOrEmpty(allowedChars))

{

throw new ArgumentException("باید حداقل یک نوع کاراکتر برای رمز عبور انتخاب شود.");

}

using (var rng = new RNGCryptoServiceProvider())

{

byte[] randomBytes = new byte[length];

rng.GetBytes(randomBytes);

// ایجاد رمز عبور با استفاده از کاراکترهای تصادفی

for (int i = 0; i < length; i++)

{

int index = randomBytes[i] % allowedChars.Length;

password.Append(allowedChars[index]);

}

}

return password.ToString();

}

/// <summary>

/// تولید یک رمز عبور ایمن و تصادفی با استانداردهای معمول.

/// </summary>

/// <returns>رمز عبور ایمن و تصادفی.</returns>

public static string GenerateDefaultPassword()

{

// تولید رمز عبور 12 کاراکتری با شرایط استاندارد

return GeneratePassword(12, true, true, true, true);

}

}

}

توضیحات متدها:

1. \*\*GeneratePassword\*\*: این متد یک رمز عبور ایمن به طول مشخص تولید می‌کند. شما می‌توانید تنظیم کنید که رمز عبور شامل حروف بزرگ، حروف کوچک، اعداد و کاراکترهای خاص باشد یا نه.

- پارامتر `length`: طول رمز عبور.

- پارامتر `requireUppercase`: اگر `true` باشد، حروف بزرگ در رمز عبور گنجانده می‌شود.

- پارامتر `requireLowercase`: اگر `true` باشد، حروف کوچک در رمز عبور گنجانده می‌شود.

- پارامتر `requireDigits`: اگر `true` باشد، اعداد در رمز عبور گنجانده می‌شود.

- پارامتر `requireSpecialChars`: اگر `true` باشد، کاراکترهای خاص در رمز عبور گنجانده می‌شود.

2. \*\*GenerateDefaultPassword\*\*: این متد یک رمز عبور پیش‌فرض تولید می‌کند که شامل حروف بزرگ و کوچک، اعداد و کاراکترهای خاص است و طول آن ۱۲ کاراکتر است.

ویژگی‌های اصلی:

- استفاده از `RNGCryptoServiceProvider` برای تولید مقادیر تصادفی امن.

- امکان انتخاب انواع مختلف کاراکترها (حروف بزرگ، حروف کوچک، اعداد، و کاراکترهای خاص).

- طول رمز عبور قابل تنظیم است.

- می‌توانید رمز عبور پیش‌فرض ۱۲ کاراکتری ایجاد کنید.

این کلاس ابزارهای مختلفی را برای تولید رمز عبور ایمن در برنامه‌ها فراهم می‌کند و برای حفظ امنیت کاربران توصیه می‌شود که رمز عبورهای پیچیده و تصادفی تولید شوند.

در اینجا کد کامل کلاس `PaginationHelper.cs` برای مدیریت صفحه‌بندی داده‌ها آورده شده است. این کلاس به شما کمک می‌کند تا داده‌ها را به صفحات مختلف تقسیم کرده و تعداد کل صفحات و سایر اطلاعات مربوط به صفحه‌بندی را به راحتی محاسبه کنید.

using System;

using System.Collections.Generic;

using System.Linq;

namespace Infrastructure.Common.Helpers

{

/// <summary>

/// کلاس PaginationHelper برای مدیریت صفحه‌بندی داده‌ها.

/// </summary>

public static class PaginationHelper

{

/// <summary>

/// صفحه‌بندی داده‌ها بر اساس صفحه و تعداد آیتم‌ها در هر صفحه.

/// </summary>

/// <typeparam name="T">نوع داده مورد نظر (برای مثال: مدل داده).</typeparam>

/// <param name="source">لیست کامل داده‌ها که باید صفحه‌بندی شوند.</param>

/// <param name="pageNumber">شماره صفحه جاری (1-based index).</param>

/// <param name="pageSize">تعداد آیتم‌ها در هر صفحه.</param>

/// <returns>صفحه‌ای از داده‌ها.</returns>

public static PaginatedResult<T> GetPagedResult<T>(IEnumerable<T> source, int pageNumber, int pageSize)

{

if (pageNumber < 1)

{

throw new ArgumentException("شماره صفحه باید بزرگتر از یا برابر ۱ باشد.", nameof(pageNumber));

}

if (pageSize < 1)

{

throw new ArgumentException("تعداد آیتم‌ها در هر صفحه باید بزرگتر از یا برابر ۱ باشد.", nameof(pageSize));

}

// تعداد کل داده‌ها

int totalItemCount = source.Count();

// محاسبه صفحه‌بندی

var items = source.Skip((pageNumber - 1) \* pageSize).Take(pageSize).ToList();

// محاسبه تعداد کل صفحات

int totalPages = (int)Math.Ceiling((double)totalItemCount / pageSize);

// ساخت و بازگشت نتیجه صفحه‌بندی

return new PaginatedResult<T>

{

Items = items,

PageNumber = pageNumber,

PageSize = pageSize,

TotalItemCount = totalItemCount,

TotalPages = totalPages

};

}

/// <summary>

/// کلاس نتیجه صفحه‌بندی.

/// </summary>

public class PaginatedResult<T>

{

/// <summary>

/// داده‌های صفحه جاری.

/// </summary>

public List<T> Items { get; set; }

/// <summary>

/// شماره صفحه جاری.

/// </summary>

public int PageNumber { get; set; }

/// <summary>

/// تعداد آیتم‌ها در هر صفحه.

/// </summary>

public int PageSize { get; set; }

/// <summary>

/// تعداد کل داده‌ها.

/// </summary>

public int TotalItemCount { get; set; }

/// <summary>

/// تعداد کل صفحات.

/// </summary>

public int TotalPages { get; set; }

}

}

}

توضیحات:

1. \*\*`GetPagedResult<T>`\*\*: این متد داده‌ها را به صفحات تقسیم می‌کند. پارامترهای ورودی این متد عبارتند از:

- `source`: مجموعه داده‌هایی که می‌خواهید آن‌ها را صفحه‌بندی کنید.

- `pageNumber`: شماره صفحه جاری (بر اساس شاخص 1).

- `pageSize`: تعداد آیتم‌ها در هر صفحه.

این متد تعداد کل آیتم‌ها، تعداد کل صفحات، و داده‌های مربوط به صفحه خاص را محاسبه کرده و در قالب یک شیء `PaginatedResult<T>` باز می‌گرداند.

2. \*\*`PaginatedResult<T>`\*\*: این کلاس نتیجه صفحه‌بندی را شامل می‌شود:

- `Items`: داده‌های موجود در صفحه جاری.

- `PageNumber`: شماره صفحه جاری.

- `PageSize`: تعداد آیتم‌ها در هر صفحه.

- `TotalItemCount`: تعداد کل داده‌ها.

- `TotalPages`: تعداد کل صفحات.

ویژگی‌های اصلی:

- صفحه‌بندی داده‌ها بر اساس شماره صفحه و اندازه صفحه.

- محاسبه تعداد کل آیتم‌ها و تعداد کل صفحات.

- مدیریت صحیح صفحه‌بندی حتی در زمانی که تعداد داده‌ها کمتر از تعداد صفحه‌ها باشد.

- استفاده از LINQ برای فیلتر کردن داده‌ها و ایجاد صفحه‌ها به راحتی.

استفاده از کلاس:

اگر بخواهید داده‌ها را صفحه‌بندی کنید، می‌توانید از این کلاس به صورت زیر استفاده کنید:

var result = PaginationHelper.GetPagedResult(myDataList, 1, 10); // صفحه اول، 10 آیتم در هر صفحه

Console.WriteLine($"Total Pages: {result.TotalPages}");

Console.WriteLine($"Total Items: {result.TotalItemCount}");

foreach (var item in result.Items)

{

Console.WriteLine(item);

}

این کلاس به شما کمک می‌کند که داده‌های خود را به صورت موثر و با استفاده از صفحه‌بندی مدیریت کنید و تجربه کاربری بهتری فراهم کنید.