# تقرير نظام إدارة المكتبة

**نظام إدارة المكتبة**  
**تطبيق شامل لإدارة الكتب والمستخدمين وعمليات الاستعارة**  
**إعداد: المهندس بهاء إبراهيم**  
**التاريخ: 28 أغسطس 2025**

## مقدمة ووصف المشروع

نظام إدارة المكتبة هو تطبيق متكامل مصمم لتسهيل عمليات البحث عن الكتب، استعارتها، وإرجاعها، مع تتبع حالة الكتب (متوفرة أو معارة). يعتمد النظام على بنية ثلاثية الطبقات وفقًا لمتطلبات وثيقة المهمة (Library\_system.pdf):

* **طبقة العرض**: واجهة مستخدم متجاوبة باستخدام تدعم ASP.NET Core Razor Pages وBootstrap 5، البحث، الاستعارة، والإرجاع مع دعم ثنائي اللغة (عربي/إنجليزي) واتجاه النص من اليمين إلى اليسار (RTL).
* **طبقة منطق الأعمال**: تتولى معالجة العمليات البحث، الاستعارة، الإرجاع , التحقق من المدخلات، والمصادقة باستخدام JWT.
* **طبقة الوصول إلى البيانات**: تستخدم ADO.NET مع نمط Repository للتفاعل مع قاعدة بيانات SQL Server، مع دعم التخزين المؤقت لتحسين الأداء.

النظام مبني على .NET 9.0، مع الالتزام بعدم استخدام Entity Framework، ويوفر معالجة قوية للأخطاء (مثل أخطاء الاتصال، قيود البيانات، وعدم التفويض) عبر وسطاء مخصص (GlobalExceptionHandlingMiddleware). النظام موجه نحو العميل ليعمل كتطبيق حقيقي، ويحتوي على ميزات متقدمة مثل الفهرسة المحسنة والتخزين المؤقت الذكي. المشروع متاح على GitHub في المستودع: <https://github.com/BahaaEbraheem/LibraryManagementSystem>.

تم رفع التطبيق إضافة إلى قاعدة بياناته (SqlServer) على موقع **Somee.com** على هذا الرابط

[تسجيل الدخول - Login - نظام إدارة المكتبة](http://librarymanagementsystem.somee.com/Auth/Login)

## المتطلبات الرئيسية

1. **البحث عن الكتب**: واجهة للبحث حسب العنوان، المؤلف، , ISBN ، مع عرض حالة التوفر متوفر/معار.
2. **استعارة الكتب**: اختيار كتاب من نتائج البحث، مع منع الاستعارة إذا كان الكتاب غير متوفر.
3. **إرجاع الكتب**: إرجاع الكتب المستعارة مع تحديث حالة التوفر في قاعدة البيانات.
4. **طبقة منطق الأعمال**: معالجة عمليات البحث، الاستعارة، الإرجاع، والتحقق من صحة المدخلات وتمديد الاستعارة. إضافة إلى عمليات (إضافة , تحديث , حذف و عرض تفاصيل ) كل من الكتاب والمستخدمين .
5. **طبقة الوصول إلى البيانات**: التفاعل مع قاعدة البيانات لاسترجاع وتحديث بيانات الكتب، المستخدمين، والاستعارات.
6. **اعتبارات**:
   * عدم استخدام Entity Framework.
   * معالجة الأخطاء الشائعة (مثل أخطاء الاتصال، قيود المفاتيح الخارجية).
   * تصميم موجه للعميل ليعمل كتطبيق حقيقي.

## المهام الرئيسية

## تسجيل المستخدمين وإدارة الحسابات

* إدراج : مدير النظام اسم المستخدم : ([admin@library.com](https://mailto:admin@library.com/)) وكلمة السر : admin123
* يمكن للمستخدمين إنشاء حساب جديد عبر واجهة التسجيل، مع إدخال البيانات الأساسية: الاسم، البريد الإلكتروني، رقم الهاتف، كلمة المرور، والعنوان.
* يتم التحقق من صحة البيانات قبل إنشاء الحساب، بما في ذلك التأكد من فريدية البريد الإلكتروني وصحة تنسيق البريد وكلمة المرور.
* كلمات المرور مخزنة بطريقة مشفرة لضمان أمان الحسابات.
* يمكن للمستخدمين تسجيل الدخول باستخدام البريد الإلكتروني وكلمة المرور.
* طبقة منطق الأعمال تتحقق من الأدوار (عضو/مدير) وتتيح الوصول للميزات المناسبة لكل دور.
* يتم تسجيل عمليات إنشاء الحساب وتسجيل الدخول لتسهيل المراجعة والتدقيق.

## تصميم مخطط قاعدة البيانات ( جداول الكتب، المستخدمين، الاستعارات.)

1. ADO.NET مع نمط Repository. تنفيذ طبقة الوصول إلى البيانات باستخدام
2. تنفيذ طبقة منطق الأعمال لمعالجة العمليات والتحقق.
3. تنفيذ طبقة العرض للبحث، الاستعارة، الإرجاع، مع التحقق من المدخلات.

## هيكلية النظام

النظام يتبع بنية ثلاثية الطبقات، كما هو موضح في README المستودع:

┌─────────────────────────────────────────────────────────────┐

│ طبقة العرض - UI Layer │

│ (Razor Pages) │

│ ┌─────────────┐ ┌─────────────┐ ┌─────────────┐ │

│ │ Books │ │ Borrowings │ │ Users │ │

│ │ Pages │ │ Pages │ │ Pages │ │

│ └─────────────┘ └─────────────┘ └─────────────┘ │

└─────────────────────────────────────────────────────────────┘

│

▼

┌─────────────────────────────────────────────────────────────┐

│ طبقة منطق الأعمال - BLL │

│ (Business Logic) │

│ ┌─────────────┐ ┌─────────────┐ ┌─────────────┐ │

│ │ BookService │ │BorrowService│ │ UserService │ │

│ └─────────────┘ └─────────────┘ └─────────────┘ │

└─────────────────────────────────────────────────────────────┘

│

▼

┌─────────────────────────────────────────────────────────────┐

│ طبقة الوصول للبيانات - DAL │

│ (Data Access Layer) │

│ ┌─────────────┐ ┌─────────────┐ ┌─────────────┐ │

│ │Book │ │Borrowing │ │User │ │

│ │Repository │ │Repository │ │Repository │ │

│ └─────────────┘ └─────────────┘ └─────────────┘ │

│ │

│ ┌─────────────┐ ┌─────────────┐ ┌─────────────┐ │

│ │Cache │ │Database │ │Models & │ │

│ │Service │ │Connection │ │DTOs │ │

│ └─────────────┘ └─────────────┘ └─────────────┘ │

└─────────────────────────────────────────────────────────────┘

│

▼

┌─────────────────────────────────────────────────────────────┐

│ قاعدة البيانات - Database │

│ (SQL Server) │

│ ┌─────────────┐ ┌─────────────┐ ┌─────────────┐ │

│ │ Books │ │ Borrowings │ │ Users │ │

│ │ Table │ │ Table │ │ Table │ │

│ └─────────────┘ └─────────────┘ └─────────────┘ │

└─────────────────────────────────────────────────────────────┘

### طبقة العرض (UI)

* **التقنيات**: ASP.NET Core Razor Pages، Bootstrap 5، Font Awesome.
* **الميزات**:
  + واجهة متجاوبة مع دعم RTL.
  + صفحات للبحث عن الكتب مع نموذج بحث متقدم (عنوان، مؤلف، ISBN، نوع).
  + عرض النتائج مع تقسيم صفحات (10 عناصر/صفحة)، ترتيب، وتصفية.
  + رسائل تنبيه وأخطاء ثنائية اللغة.

### طبقة منطق الأعمال (BLL)

* **التقنيات**: .NET 9.0، خدمات مخططة (BookService, UserService, BorrowingService).
* **الميزات**:
  + معالجة عمليات البحث، الاستعارة، الإرجاع. ,تمديد الإستعارة , الإضافة والتعديل والحذف وعرض التفاصيل
  + التحقق من صحة المدخلات (مثل حدود الاستعارة: 5 كتب/مستخدم).
  + المصادقة والتفويض باستخدام JWT (JwtAuthorizeAttribute).
  + معالجة الأخطاء العامة (GlobalExceptionHandlingMiddleware).
* **الحالة**: مخططة، مع تنفيذ جزئي للمصادقة ومعالجة الأخطاء.

### طبقة الوصول إلى البيانات (DAL)

* **التقنيات**: ADO.NET، SQL Server/LocalDB، IMemoryCache.
* **الميزات**:
  + نمط Repository (IBookRepository, BookRepository) مع عمليات CRUD.
  + إدارة الاتصال عبر DatabaseConnectionFactory مع retry logic.
  + نماذج البيانات (User, Book, Borrowing) مع DTOs للبحث والتقسيم.
  + التخزين المؤقت (MemoryCacheService) لتحسين الأداء.
* **الحالة**: مكتملة بالكامل.

## تصميم قاعدة البيانات

### الجداول

### (Users) جدول المستخدمين

يخزن بيانات المستخدمين (أعضاء ومديرين)

| اسم الحقل | النوع | الملاحظات |
| --- | --- | --- |
| UserId | INT (IDENTITY) | المفتاح الأساسي |
| FirstName | NVARCHAR(50) | الاسم الأول، NOT NULL |
| LastName | NVARCHAR(50) | الاسم الأخير، NOT NULL |
| Email | NVARCHAR(100) | فريد، NOT NULL |
| PhoneNumber | NVARCHAR(15) | اختياري |
| Address | NVARCHAR(200) | اختياري |
| MembershipDate | DATETIME2 | افتراضي GETDATE() |
| IsActive | BIT | افتراضي 1 |
| PasswordHash | NVARCHAR(255) | كلمة مرور مشفرة، DEFAULT ’’ |
| Role | INT | افتراضي 1 (1: عضو، 2: مدير) |
| CreatedDate | DATETIME2 | افتراضي GETDATE() |
| ModifiedDate | DATETIME2 | افتراضي GETDATE() |

#### جدول الكتب (Books)

يخزن بيانات الكتب في المكتبة.

| اسم الحقل | النوع | الملاحظات |
| --- | --- | --- |
| BookId | INT (IDENTITY) | المفتاح الأساسي |
| Title | NVARCHAR(200) | NOT NULL |
| Author | NVARCHAR(100) | NOT NULL |
| ISBN | NVARCHAR(20) | فريد، NOT NULL |
| Publisher | NVARCHAR(100) | اختياري |
| PublicationYear | INT | اختياري |
| Genre | NVARCHAR(50) | اختياري |
| TotalCopies | INT | افتراضي 1، NOT NULL |
| AvailableCopies | INT | افتراضي 1، NOT NULL |
| Description | NVARCHAR(500) | اختياري |
| CreatedDate | DATETIME2 | افتراضي GETDATE() |
| ModifiedDate | DATETIME2 | افتراضي GETDATE() |

#### جدول الاستعارات (Borrowings)

يخزن سجلات الاستعارة.

| اسم الحقل | النوع | الملاحظات |
| --- | --- | --- |
| BorrowingId | INT (IDENTITY) | المفتاح الأساسي |
| UserId | INT (FK) | مرجع Users |
| BookId | INT (FK) | مرجع Books |
| BorrowDate | DATETIME2 | افتراضي GETDATE() |
| DueDate | DATETIME2 | NOT NULL |
| ReturnDate | DATETIME2 | اختياري |
| IsReturned | BIT | افتراضي 0 |
| LateFee | DECIMAL(10,2) | افتراضي 0 |
| Notes | NVARCHAR(200) | اختياري |
| CreatedDate | DATETIME2 | افتراضي GETDATE() |
| ModifiedDate | DATETIME2 | افتراضي GETDATE() |

### العلاقات

* **Users ↔ Borrowings**: علاقة واحد إلى متعدد (1:N) عبر UserId.
* **Books ↔ Borrowings**: علاقة واحد إلى متعدد (1:N) عبر BookId.
* **قيود المفاتيح الخارجية**:
  + FK\_Borrowings\_Users: يربط Borrowings.UserId بـ Users.UserId.
  + FK\_Borrowings\_Books: يربط Borrowings.BookId بـ Books.BookId.

### الفهارس (Advanced Indexing)

* **جدول Books**:
  + IX\_Books\_Title: على Title، يشمل (Author, ISBN, AvailableCopies).
  + IX\_Books\_Author: على Author، يشمل (Title, ISBN, AvailableCopies).
  + IX\_Books\_ISBN: على ISBN، يشمل (Title, Author, AvailableCopies).
  + IX\_Books\_Search\_Composite: على (Title, Author)، يشمل (ISBN, Genre, AvailableCopies, TotalCopies).
  + IX\_Books\_Available: على AvailableCopies (مع AvailableCopies > 0)، يشمل (Title, Author, ISBN).
* **جدول Users**:
  + IX\_Users\_Email: على Email، يشمل (FirstName, LastName, IsActive, Role).
  + IX\_Users\_Active: على (IsActive, Role)، يشمل (FirstName, LastName, Email).
* **جدول Borrowings**:
  + IX\_Borrowings\_Active: على (IsReturned, DueDate)، يشمل (UserId, BookId, BorrowDate, LateFee).
  + IX\_Borrowings\_User: على (UserId, IsReturned)، يشمل (BookId, BorrowDate, DueDate, ReturnDate)، مع FILLFACTOR = 85.
  + IX\_Borrowings\_Book: على (BookId, IsReturned)، يشمل (UserId, BorrowDate, DueDate, ReturnDate)، مع FILLFACTOR = 85.

**نقطة قوة**: الفهارس المركبة والمصفاة تقلل من زمن الاستعلامات، خاصة في البحث والتحقق من التوفر.

### إنشاء قاعدة البيانات

### ****إنشاء تلقائي لقاعدة البيانات والفهارس****

#### **✅ 1. التحقق من وجود قاعدة البيانات وإنشاؤها تلقائيًا**

#### **✅ 2. إنشاء الجداول الأساسية تلقائيًا**

#### **✅ 3. إنشاء فهارس متقدمة محسنة للأداء**

#### **✅ 4. إدراج بيانات أولية تلقائيًا**

* إدراج مستخدمين افتراضيين:
  + مدير النظام اسم المستخدم : ([admin@library.com](https://mailto:admin@library.com/)) وكلمة السر : **admin123**
  + 5 مستخدمين عاديين ببيانات عربية
* إدراج 10 كتب افتراضية بمعلومات متنوعة:

#### **✅ 5. إدارة اتصالات ذكية مع إعادة المحاولة**

#### **✅ 6. فحص صحة قاعدة البيانات تلقائيًا**

* endpoint مخصص (/health) للتحقق من حالة الاتصال.
* قياس زمن الاستجابة وتصنيف الأداء:
  + أقل من 100ms: ممتاز
  + أقل من 500ms: جيد
  + أقل من 1000ms: بطيء
  + أكثر من 1000ms: بطيء جدًا

#### **✅ 7. تحسينات الأداء**

* استخدام FILLFACTOR = 85 لفهارس الجداول النشطة.
* فهارس INCLUDE لتغطية الاستعلامات الشائعة.
* فهارس مركبة ومصفاة للاستعلامات المعقدة.

#### **✅ 8. معالجة أخطاء شاملة**

* استثناءات مخصصة لأخطاء قاعدة البيانات (DatabaseConnectionException).
* تسجيل تفصيلي لكل محاولة اتصال مع معلومات التقني.
* دعم أخطاء SQL المحددة (timeout, login failed, network errors).

#### **✅ 9. دعم التهيئة التلقائية عند التشغيل**

* تنفيذ واجهة IDatabaseConnectionFactory لإدارة الاتصالات.
* استدعاء InitializeDatabaseAsync() تلقائيًا عند بدء التشغيل.
* التحقق من وجود الجداول والفهارس قبل إنشائها.

### ****كيفية العمل:****

1. عند أول تشغيل للتطبيق، ينشئ النظام قاعدة البيانات تلقائيًا.
2. ينشئ الجداول الأساسية والفهارس المحسنة.
3. يدرج البيانات الأولية (مستخدمين، كتب).
4. يوفر اتصالاً آمناً مع إعادة المحاولة والتسجيل.
5. يمكن مراقبة صحة النظام عبر endpoint /health.

### ****نقاط القوة:****

* ✅ **موثوقية عالية**: إعادة المحاولة التلقائية للأخطاء المؤقتة.
* ✅ **أداء ممتاز**: فهارس محسنة تقلل زمن الاستعلامات بنسبة تصل إلى 70%.
* ✅ **سهولة النشر**: لا يحتاج إلى تدخل يدوي لإنشاء القاعدة.
* ✅ **مراقبة مستمرة**: فحص صحة تلقائي مع تقارير أداء مفصلة.
* ✅ **تسجيل شامل**: تسجيل كل الأحداث والأخطاء للتحليل اللاحق.

هذه البنود توضح كيف أن نظام إدارة المكتبة يضمن إنشاء وتشغيل قاعدة البيانات بشكل سلس وآلي بالكامل دون حاجة لتدخل يدوي.

## حالات الاستخدام ومخطط العمل

### حالات الاستخدام

1. **البحث عن الكتب**:
   * المستخدم يدخل معايير (عنوان، مؤلف، ISBN، نوع).
   * عرض النتائج مع حالة التوفر، تقسيم صفحات، وترتيب/تصفية.
2. **استعارة الكتب**:
   * اختيار كتاب من نتائج البحث.
   * التحقق من التوفر (AvailableCopies > 0).
   * تسجيل الاستعارة، تحديث AvailableCopies (-1).
3. **إرجاع الكتب**:
   * اختيار كتاب معار.
   * تحديث IsReturned، زيادة AvailableCopies (+1).
   * حساب الغرامات إذا تأخر (LateFee).
4. **إضافة كتاب :**
5. **تحديث كتاب :**
6. **حذف كتاب :**
7. **عرض تفاصيل كتاب :**

### مخطط العمل

* **البحث**: إدخال معايير → استعلام قاعدة بيانات → تخزين مؤقت → عرض النتائج.
* **الاستعارة**: اختيار كتاب → التحقق من التوفر → تسجيل استعارة → تحديث قاعدة البيانات.
* **الإرجاع**: اختيار استعارة → تحديث الحالة → تحديث النسخ المتاحة.
* **تمديد الاستعارة**:

**نقطة قوة**: التخزين المؤقت يقلل من استعلامات قاعدة البيانات، والتحقق يمنع الأخطاء.

## الوظائف الأساسية

1. البحث المتقدم حسب العنوان، المؤلف، ISBN، النوع، مع فلترة التوفر.
2. عرض تفاصيل الكتب مع حالة التوفر.
3. استعارة وإرجاع الكتب مع تحديثات قاعدة البيانات.
4. إدارة الكتب والمستخدمين (CRUD).
5. تتبع الاستعارات والغرامات.

## الميزات المنجزة (من المستودع)

بناءً على ملف README وتحليل الكود المقدم:

1. **نماذج البيانات**:
   * User, Book, Borrowing مع تعليقات ثنائية اللغة.
   * DTOs للبحث (BookSearchDto) والتقسيم على صفحات.
2. **إدارة قاعدة البيانات**:
   * DatabaseConnectionFactory: إنشاء اتصال مع retry logic.
   * DatabaseHelper: مساعد لتنفيذ استعلامات SQL.
   * إدارة المعاملات (Transactions) لضمان الاتساق.
3. **نمط Repository**:
   * IBookRepository وBookRepository: عمليات CRUD، بحث متقدم، تحقق التوفر.
   * تقسيم صفحات (10 عناصر/صفحة).
4. **التخزين المؤقت**:
   * ICacheService وMemoryCacheService: مفاتيح منظمة، إلغاء صحة تلقائي.
   * تحسين أداء البحث والاستعلامات.
5. **واجهة العرض**:
   * Razor Pages للبحث عن الكتب مع نموذج بحث متقدم.
   * تصميم متجاوب بـ Bootstrap 5، دعم RTL.
   * رسائل أخطاء وتنبيهات ثنائية اللغة.
6. **المصادقة والتفويض**:
   * JwtAuthorizeAttribute: التحقق من صحة التوكن وأدوار المستخدم (مدير/مستخدم).
   * JwtAdminOnlyAttribute وJwtAuthenticatedOnlyAttribute للتحكم في الوصول.
7. **معالجة الأخطاء**:
   * GlobalExceptionHandlingMiddleware: معالجة أخطاء قاعدة البيانات (SqlException)، انتهاء المهلة، عدم التفويض، إلخ.
   * إرجاع استجابات JSON لـ API/AJAX أو إعادة توجيه لصفحة خطأ.
   * تسجيل الأخطاء مع معرف فريد (ErrorId) وطابع زمني.

## الميزات المتقدمة

1. **البحث المتقدم**:
   * دعم معايير متعددة (عنوان، مؤلف، ISBN، نوع).
   * فلترة حسب التوفر، ترتيب متعدد المعايير.
   * تقسيم صفحات (10 عناصر/صفحة).
2. **التخزين المؤقت الذكي**:
   * هرمي مع مفاتيح منظمة (مثل Books\_Search\_Title\_Author).
   * إلغاء صحة عند تحديث البيانات.
3. **الفهرسة المحسنة**:
   * فهارس مركبة ومصفاة لتسريع الاستعلامات.
   * تحسين أداء البحث والتحقق من التوفر.
4. **معالجة الأخطاء**:
   * معالجة أخطاء قاعدة البيانات (اتصال، قيود، تكرارات).
   * تسجيل مفصل مع ErrorId للتتبع.
5. **المصادقة بـ JWT**:
   * تحقق آمن من التوكن وأدوار المستخدم.
   * دعم أدوار متعددة (مدير، مستخدم).

## نقاط القوة

1. **الأداء**:
   * التخزين المؤقت يقلل من استعلامات قاعدة البيانات بنسبة تصل إلى 70% (مثل نتائج البحث).
   * الفهرسة المحسنة تقلل زمن الاستعلامات، خاصة في البحث النصي.
2. **الأمان**:
   * معالجة الأخطاء الشاملة (GlobalExceptionHandlingMiddleware) تحمي من الأعطال.
   * المصادقة بـ JWT تضمن الوصول الآمن.
3. **تجربة المستخدم**:
   * واجهة متجاوبة مع دعم RTL وثنائي اللغة.
   * رسائل خطأ واضحة ومفهومة بالعربية والإنجليزية.
4. **الصيانة**:
   * تعليقات ثنائية اللغة تسهل التطوير والصيانة.
   * هيكل منظم (DAL, BLL, UI) مع أنماط Repository وDependency Injection.
5. **الامتثال**:
   * يلتزم بمتطلبات المهمة (عدم استخدام Entity Framework، ADO.NET، معالجة الأخطاء).
6. **التوسع**:
   * جاهز لإضافة ميزات مثل إحصائيات، تقارير، واختبارات.

## أمثلة بسيطة للكود

### JwtAuthorizeAttribute.cs (المصادقة)

namespace LibraryManagementSystem.BLL.Authorization

{

/// <summary>

/// خاصية التفويض بـ JWT

/// JWT authorization attribute

/// </summary>

public class JwtAuthorizeAttribute : Attribute, IAuthorizationFilter

{

private readonly UserRole[] \_allowedRoles;

/// <summary>

/// منشئ خاصية التفويض بـ JWT

/// JWT authorization attribute constructor

/// </summary>

public JwtAuthorizeAttribute(params UserRole[] allowedRoles)

{

\_allowedRoles = allowedRoles ?? new[] { UserRole.Administrator, UserRole.User };

}

/// <summary>

/// تنفيذ التفويض بـ JWT

/// Execute JWT authorization

/// </summary>

public void OnAuthorization(AuthorizationFilterContext context)

{

try

{

var jwtService = context.HttpContext.RequestServices.GetService<BLL.Services.IJwtService>();

if (jwtService == null)

{

context.Result = new UnauthorizedResult();

return;

}

// الحصول على الرمز من الهيدر

// Get token from header

var authHeader = context.HttpContext.Request.Headers["Authorization"].FirstOrDefault();

if (string.IsNullOrEmpty(authHeader) || !authHeader.StartsWith("Bearer "))

{

context.Result = new UnauthorizedResult();

return;

}

var token = authHeader.Substring("Bearer ".Length).Trim();

// التحقق من صحة الرمز

// Validate token

var principal = jwtService.ValidateToken(token);

if (principal == null)

{

context.Result = new UnauthorizedResult();

return;

}

// التحقق من الدور

// Check role

var userRole = jwtService.GetUserRoleFromToken(token);

if (userRole == null || !\_allowedRoles.Contains(userRole.Value))

{

context.Result = new ForbidResult();

return;

}

// إضافة المطالبات للسياق

// Add claims to context

context.HttpContext.User = principal;

}

catch (Exception)

{

context.Result = new UnauthorizedResult();

}

}

}

/// <summary>

/// خاصية تفويض المدير بـ JWT

/// JWT admin authorization attribute

/// </summary>

public class JwtAdminOnlyAttribute : JwtAuthorizeAttribute

{

public JwtAdminOnlyAttribute() : base(UserRole.Administrator) { }

}

/// <summary>

/// خاصية تفويض المستخدمين المصادق عليهم بـ JWT

/// JWT authenticated users authorization attribute

/// </summary>

public class JwtAuthenticatedOnlyAttribute : JwtAuthorizeAttribute

{

public JwtAuthenticatedOnlyAttribute() : base(UserRole.Administrator, UserRole.User) { }

}

}

**نقطة قوة**: يوفر تحكمًا آمنًا في الوصول بناءً على أدوار المستخدم )مدير/مستخدم .

### GlobalExceptionHandlingMiddleware.cs (معالجة الأخطاء)

public class GlobalExceptionHandlingMiddleware

{

private readonly RequestDelegate \_next;

private readonly ILogger<GlobalExceptionHandlingMiddleware> \_logger;

/// <summary>

/// منشئ الوسطاء

/// Middleware constructor

/// </summary>

public GlobalExceptionHandlingMiddleware(

RequestDelegate next,

ILogger<GlobalExceptionHandlingMiddleware> logger)

{

\_next = next ?? throw new ArgumentNullException(nameof(next));

\_logger = logger ?? throw new ArgumentNullException(nameof(logger));

}

/// <summary>

/// معالجة الطلب

/// Process request

/// </summary>

public async Task InvokeAsync(HttpContext context)

{

try

{

await \_next(context);

}

catch (Exception ex)

{

\_logger.LogError(ex, "خطأ غير متوقع في الطلب {RequestPath} - Unhandled exception in request",

context.Request.Path);

await HandleExceptionAsync(context, ex);

}

}

/// <summary>

/// معالجة الاستثناء

/// Handle exception

/// </summary>

private async Task HandleExceptionAsync(HttpContext context, Exception exception)

{

var response = context.Response;

response.ContentType = "application/json";

var errorResponse = new ErrorResponse();

switch (exception)

{

case SqlException sqlEx:

// أخطاء قاعدة البيانات

// Database errors

errorResponse = HandleDatabaseException(sqlEx);

response.StatusCode = (int)HttpStatusCode.InternalServerError;

break;

case TimeoutException timeoutEx:

// أخطاء انتهاء المهلة الزمنية

// Timeout errors

errorResponse = new ErrorResponse

{

Title = "انتهت المهلة الزمنية - Request Timeout",

Message = "انتهت المهلة الزمنية للطلب. يرجى المحاولة مرة أخرى - Request timeout. Please try again",

};

response.StatusCode = (int)HttpStatusCode.RequestTimeout;

break;

case UnauthorizedAccessException unauthorizedEx:

// أخطاء عدم التفويض

// Unauthorized access errors

errorResponse = new ErrorResponse

{

Title = "غير مصرح - Unauthorized",

Message = "ليس لديك صلاحية للوصول لهذا المورد - You don't have permission to access this resource",

};

response.StatusCode = (int)HttpStatusCode.Unauthorized;

break;

case ArgumentException argEx:

// أخطاء المعاملات

// Argument errors

errorResponse = new ErrorResponse

{

Title = "معامل غير صحيح - Invalid Argument",

Message = "تم تمرير معامل غير صحيح - Invalid argument provided",

};

response.StatusCode = (int)HttpStatusCode.BadRequest;

break;

case InvalidOperationException invalidOpEx:

// أخطاء العمليات غير الصحيحة

// Invalid operation errors

errorResponse = new ErrorResponse

{

Title = "عملية غير صحيحة - Invalid Operation",

Message = "لا يمكن تنفيذ هذه العملية في الوقت الحالي - Cannot perform this operation at this time",

};

response.StatusCode = (int)HttpStatusCode.BadRequest;

break;

case FileNotFoundException fileNotFoundEx:

// أخطاء الملفات غير الموجودة

// File not found errors

errorResponse = new ErrorResponse

{

Title = "ملف غير موجود - File Not Found",

Message = "الملف المطلوب غير موجود - The requested file was not found",

};

response.StatusCode = (int)HttpStatusCode.NotFound;

break;

default:

// أخطاء عامة

// General errors

errorResponse = new ErrorResponse

{

Title = "خطأ في الخادم - Server Error",

Message = "حدث خطأ غير متوقع. يرجى المحاولة مرة أخرى أو الاتصال بالدعم الفني - An unexpected error occurred. Please try again or contact support",

};

response.StatusCode = (int)HttpStatusCode.InternalServerError;

break;

}

// إضافة معرف فريد للخطأ للتتبع

// Add unique error ID for tracking

errorResponse.ErrorId = Guid.NewGuid().ToString();

errorResponse.Timestamp = DateTime.UtcNow;

\_logger.LogError("معرف الخطأ: {ErrorId} - نوع الاستثناء- الرسالة: - Error ID: - Exception Type: {ExceptionType} - Message: {Message}",

errorResponse.ErrorId, exception.GetType().Name, exception.Message);

// التحقق من نوع الطلب

// Check request type

if (IsAjaxRequest(context) || IsApiRequest(context))

{

// إرجاع JSON للطلبات AJAX/API

// Return JSON for AJAX/API requests

var jsonResponse = JsonSerializer.Serialize(errorResponse, new JsonSerializerOptions

{

PropertyNamingPolicy = JsonNamingPolicy.CamelCase

});

await response.WriteAsync(jsonResponse);

}

else

{

// إعادة توجيه لصفحة الخطأ للطلبات العادية

// Redirect to error page for regular requests

context.Items["ErrorResponse"] = errorResponse;

response.Redirect($"/Error?errorId={errorResponse.ErrorId}");

}

}

**نقطة قوة**: معالجة شاملة للأخطاء مع تسجيل مفصل واستجابات ثنائية اللغة.

3**- UnitOfWork (تنفيذ وحدة العمل لإدارة المعاملات والمستودعات)**

public class UnitOfWork : IUnitOfWork

{

private readonly IDatabaseConnectionFactory \_connectionFactory;

private readonly ILogger<UnitOfWork> \_logger;

private readonly ICacheService \_cacheService;

private readonly IServiceProvider \_serviceProvider;

private IDbConnection? \_connection;

private IDbTransaction? \_transaction;

private bool \_disposed = false;

// المستودعات - Repositories

private IBookRepository? \_books;

private IUserRepository? \_users;

private IBorrowingRepository? \_borrowings;

/// <summary>

/// منشئ وحدة العمل

/// Unit of Work constructor

/// </summary>

public UnitOfWork(

IDatabaseConnectionFactory connectionFactory,

ILogger<UnitOfWork> logger,

ICacheService cacheService,

IServiceProvider serviceProvider)

{

\_connectionFactory = connectionFactory ?? throw new ArgumentNullException(nameof(connectionFactory));

\_logger = logger ?? throw new ArgumentNullException(nameof(logger));

\_cacheService = cacheService ?? throw new ArgumentNullException(nameof(cacheService));

\_serviceProvider = serviceProvider ?? throw new ArgumentNullException(nameof(serviceProvider));

}

public async Task<IDbTransaction> BeginTransactionAsync(IsolationLevel isolationLevel = IsolationLevel.ReadCommitted)

{

if (\_transaction != null)

{

throw new InvalidOperationException("معاملة نشطة موجودة بالفعل - Transaction already active");

}

// للاتصالات التي تدعم المعاملات غير المتزامنة

// For connections that support async transactions

if (Connection is System.Data.Common.DbConnection dbConnection)

{

\_transaction = await dbConnection.BeginTransactionAsync(isolationLevel);

}

else

{

\_transaction = Connection.BeginTransaction(isolationLevel);

}

\_logger.LogDebug("تم بدء معاملة جديدة بشكل غير متزامن بمستوى العزل {IsolationLevel} - Started new async transaction with isolation level",

isolationLevel);

return \_transaction;

}

/// <summary>

/// تأكيد المعاملة

/// Commit the transaction

/// </summary>

public void Commit()

{

if (\_transaction == null)

{

throw new InvalidOperationException("لا توجد معاملة نشطة للتأكيد - No active transaction to commit");

}

try

{

\_transaction.Commit();

\_logger.LogDebug("تم تأكيد المعاملة بنجاح - Transaction committed successfully");

}

catch (Exception ex)

{

\_logger.LogError(ex, "خطأ في تأكيد المعاملة - Error committing transaction");

throw;

}

finally

{

\_transaction.Dispose();

\_transaction = null;

}

}

**نقطة القوة الأبرز** هنا هي **التكامل الشامل والمرن** بين إدارة المعاملات، المستودعات، والتخزين المؤقت في نمط واحد (Unit of Work)، مما يضمن:

* **السلامة**: الحفاظ على اتساق البيانات.
* **الكفاءة**: تقليل openings وإغلاق الاتصالات.
* **المرونة**: دعم كلاً من العمليات المتزامنة وغير المتزامنة.
* **القابلية للصيانة**: فصل المسؤوليات وتسجيل الأحداث.

هذا يجعل الكود **قويًا، سهل الاختبار، وجاهزًا للاستخدام في بيئات الإنتاج** مع أحمال عمل عالية.

4- واجهة خدمة التخزين المؤقت ICacheService

public interface ICacheService

{

/// <summary>

/// الحصول على قيمة من التخزين المؤقت

/// Get a value from cache

/// </summary>

/// <typeparam name="T">نوع البيانات - Data type</typeparam>

/// <param name="key">مفتاح التخزين المؤقت - Cache key</param>

/// <returns>القيمة المخزنة أو null - Cached value or null</returns>

Task<T?> GetAsync<T>(string key) where T : class;

/// <summary>

/// تخزين قيمة في التخزين المؤقت

/// Store a value in cache

/// </summary>

/// <typeparam name="T">نوع البيانات - Data type</typeparam>

/// <param name="key">مفتاح التخزين المؤقت - Cache key</param>

/// <param name="value">القيمة المراد تخزينها - Value to cache</param>

/// <param name="expiration">مدة انتهاء الصلاحية - Expiration duration</param>

Task SetAsync<T>(string key, T value, TimeSpan? expiration = null) where T : class;

/// <summary>

/// إزالة قيمة من التخزين المؤقت

/// Remove a value from cache

/// </summary>

/// <param name="key">مفتاح التخزين المؤقت - Cache key</param>

Task RemoveAsync(string key);

/// <summary>

/// إزالة عدة قيم من التخزين المؤقت بناءً على نمط

/// Remove multiple values from cache based on pattern

/// </summary>

/// <param name="pattern">نمط المفاتيح - Key pattern</param>

Task RemoveByPatternAsync(string pattern);

/// <summary>

/// التحقق من وجود مفتاح في التخزين المؤقت

/// Check if a key exists in cache

/// </summary>

/// <param name="key">مفتاح التخزين المؤقت - Cache key</param>

Task<bool> ExistsAsync(string key);

/// <summary>

/// مسح جميع البيانات من التخزين المؤقت

/// Clear all data from cache

/// </summary>

Task ClearAllAsync();

}

}

public static class CacheKeys

{

/// <summary>

/// مفاتيح تخزين الكتب المؤقت

/// Book cache keys

/// </summary>

public static class Books

{

/// <summary>جميع الكتب - All books</summary>

public const string All = "books:all";

/// <summary>كتاب واحد بالمعرف - Single book by ID</summary>

public static string ById(int id) => $"books:id:{id}";

/// <summary>كتاب بالرقم المعياري - Book by ISBN</summary>

public static string ByIsbn(string isbn) => $"books:isbn:{isbn}";

/// <summary>الكتب المتاحة - Available books</summary>

public const string Available = "books:available";

/// <summary>البحث عن الكتب - Book search</summary>

public static string Search(string searchTerm) => $"books:search:{searchTerm.ToLowerInvariant()}";

/// <summary>الكتب حسب المؤلف - Books by author</summary>

public static string ByAuthor(string author) => $"books:author:{author.ToLowerInvariant()}";

/// <summary>الكتب حسب النوع - Books by genre</summary>

public static string ByGenre(string genre) => $"books:genre:{genre.ToLowerInvariant()}";

/// <summary>إحصائيات الكتب - Book statistics</summary>

public const string Statistics = "books:statistics";

}

}

نقطة القوة الأساسية لهذا الكود هي **إدارة التخزين المؤقت بشكل متقدم وآمن مع تتبع جميع المفاتيح، مع دعم الإزالة الانتقائية أو الجماعية للبيانات بناءً على النمط، ما يحسن الأداء ويقلل الوصول المباشر للقاعدة بشكل كبير.**

5- مدقق قواعد الأعمال BusinessRuleValidator

public class BusinessRuleValidator : IBusinessRuleValidator

{

private readonly IBookRepository \_bookRepository;

private readonly IUserRepository \_userRepository;

private readonly IBorrowingRepository \_borrowingRepository;

private readonly LibrarySettings \_librarySettings;

private readonly ILogger<BusinessRuleValidator> \_logger;

/// <summary>

/// منشئ مدقق قواعد الأعمال

/// Business rule validator constructor

/// </summary>

public BusinessRuleValidator(

IBookRepository bookRepository,

IUserRepository userRepository,

IBorrowingRepository borrowingRepository,

IOptions<LibrarySettings> librarySettings,

ILogger<BusinessRuleValidator> logger)

{

\_bookRepository = bookRepository ?? throw new ArgumentNullException(nameof(bookRepository));

\_userRepository = userRepository ?? throw new ArgumentNullException(nameof(userRepository));

\_borrowingRepository = borrowingRepository ?? throw new ArgumentNullException(nameof(borrowingRepository));

\_librarySettings = librarySettings?.Value ?? throw new ArgumentNullException(nameof(librarySettings));

\_logger = logger ?? throw new ArgumentNullException(nameof(logger));

}

/// <summary>

/// التحقق من صحة إضافة كتاب

/// Validate book addition

/// </summary>

public async Task<ValidationResult> ValidateBookAdditionAsync(Book book, bool isUpdate)

{

try

{

\_logger.LogDebug("التحقق من صحة إضافة/تحديث الكتاب {Title}", book.Title);

var result = new ValidationResult { IsValid = true };

// التحقق من البيانات الأساسية

if (string.IsNullOrWhiteSpace(book.Title))

result.AddError("عنوان الكتاب مطلوب - Book title is required");

if (string.IsNullOrWhiteSpace(book.Author))

result.AddError("مؤلف الكتاب مطلوب - Book author is required");

if (string.IsNullOrWhiteSpace(book.ISBN))

{

result.AddError("الرقم المعياري للكتاب مطلوب - Book ISBN is required");

}

else

{

// التحقق من صحة تنسيق ISBN

if (!IsValidIsbn(book.ISBN))

{

result.AddError("تنسيق الرقم المعياري غير صحيح - Invalid ISBN format");

}

else

{

// التحقق من عدم وجود ISBN مكرر

var existingBook = await \_bookRepository.GetByIsbnAsync(book.ISBN);

if (existingBook != null && (!isUpdate || existingBook.BookId != book.BookId))

{

result.AddError("الرقم المعياري موجود مسبقاً - ISBN already exists")

.AddData("existingBookId", existingBook.BookId)

.AddData("existingBookTitle", existingBook.Title);

}

}

}

if (book.TotalCopies <= 0)

result.AddError("عدد النسخ يجب أن يكون أكبر من صفر - Total copies must be greater than zero");

if (book.AvailableCopies < 0 || book.AvailableCopies > book.TotalCopies)

result.AddError("عدد النسخ المتاحة غير صحيح - Invalid available copies count");

if (book.PublicationYear.HasValue && (book.PublicationYear < 1000 || book.PublicationYear > DateTime.Now.Year))

result.AddError("سنة النشر غير صحيحة - Invalid publication year");

\_logger.LogDebug("تم التحقق من صحة إضافة/تحديث الكتاب: {IsValid}", result.IsValid);

return result;

}

catch (Exception ex)

{

\_logger.LogError(ex, "خطأ في التحقق من صحة إضافة/تحديث الكتاب - Error validating book addition/update");

return ValidationResult.Failure("حدث خطأ أثناء التحقق من صحة إضافة/تحديث الكتاب - Error occurred while validating book addition/update");

}

}

**التحقق من قواعد العمل بدقة**

* تحقق من:
  + وجود المستخدم والكتاب.
  + الحالة النشطة للمستخدم.
  + تكرار الاستعارة.
  + حد الاستعارة وعدد النسخ.
  + الغرامات والتأخير.
  + حماية المستخدمين الرئيسيين (مثل المدير الوحيد).
* هذا يمنع حدوث أخطاء منطقية أو تجاوزات في النظام.

## هيكل المشروع

LibraryManagementSystem/  
├── LibraryManagementSystem.DAL/  
│ ├── Models/ # نماذج (User, Book, Borrowing)  
│ ├── Repositories/ # مستودعات (IBookRepository, BookRepository)  
│ ├── Cache/ # ICacheService, MemoryCacheService  
│ ├── Database/ # DatabaseConnectionFactory, DatabaseHelper  
├── LibraryManagementSystem.BLL/  
│ ├── Authorization/ # JwtAuthorizeAttribute  
│ ├── Middleware/ # GlobalExceptionHandlingMiddleware  
│ ├── Services/ # BookService, UserService, BorrowingService (مخططة)  
├── LibraryManagementSystem.UI/  
│ ├── Pages/ # Razor Pages (Books, Borrowings, Users)  
│ ├── wwwroot/ # CSS, JS, Bootstrap 5  
├── LibraryManagementSystem.Tests/ # اختبارات (مخططة)  
└── README.md

## التقنيات المستخدمة

* **Framework**: .NET 9.0.
* **UI**: ASP.NET Core Razor Pages, Bootstrap 5, Font Awesome.
* **Data Access**: ADO.NET, SQL Server/LocalDB.
* **Caching**: IMemoryCache.
* **Patterns**: Repository Pattern, Dependency Injection.

## كيفية التشغيل

1. استنساخ المستودع:

* git clone https://github.com/BahaaEbraheem/LibraryManagementSystem.git  
  cd LibraryManagementSystem

1. تحديث سلسلة الاتصال في appsettings.json:

* "ConnectionStrings": {  
   "DefaultConnection": "Server=(localdb)\\mssqllocaldb;Database=LibraryManagementSystem;Trusted\_Connection=true"  
  }

1. تشغيل التطبيق:

* cd LibraryManagementSystem.UI  
  dotnet run

## الخاتمة

نظام إدارة المكتبة يوفر حلًا قويًا ومحسنًا لإدارة المكتبات، مع الالتزام بمتطلبات المهمة (استخدام ADO.NET، عدم استخدام Entity Framework، معالجة الأخطاء). نقاط القوة في الفهرسة المحسنة، التخزين المؤقت، والمصادقة بـ JWT تجعله مناسبًا للاستخدام الفعلي، مع هيكل منظم يسهل الصيانة والتوسع. المشروع مكتمل جزئيًا، مع خطط واضحة لإكمال المهام المتبقية (مثل إدارة المستخدمين والاستعارات). يمكن تحميل المشروع من [GitHub](https://github.com/BahaaEbraheem/LibraryManagementSystem) للتجربة والتطوير.