وظيفة مقرر **WMB** للفصل F24

مشروع المكتبة الإلكترونية

E_Library

الدكتور المشرف

د. محمد مازن المصطفى

المعيار 1	المعيار 2	المعيار 3	المعيار 4	المعيار 5	المعيار 6
تصميم وبناء قاعدة بيانات	نشر قاعدة المعطيات	بناء تطبيق يعمل على فلتر وينفذ جميع المهام	استخدام إجراءات التحقق ومظهر جيد للتطبيق	تقرير جيد يعكس خطوات العمل بالتفصيل + تسجيل فيديو لعملية فحص مخرجات المنتج	استخدام API
10	10	35	10	20	15

إعداد الطلاب

1. MWS_WMB _HW_F24_ C1_bahaa_314468
2. MWS_WDC_HW_F24_C1_ Jilan_339202
3. MWS_WDC_HW_F24_C2_ Khaldoun_332508
4. MWS_WDC_HW_F24_C1_ mohanad_307209
5. MWS_WDC_HW_F24_C2_ ynall_220639

الفهرس

بيانات الدخول

المستخدم الإداري:

اسم المستخدم : admin

كلمة المرور : admin123

نظرة عامة على المشروع

مشروع المكتبة الإلكترونية هو نظام متكامل لإدارة الكتب والمؤلفين والناشرين، يتكون من واجهة خلفية (Backend)مبنية باستخدام ASP.NET Core وواجهة أمامية (Frontend) مبنية باستخدام .Flutter يهدف المشروع إلى توفير منصة سهلة الاستخدام لإدارة محتوى المكتبة وعرضه للمستخدمين، مع توفير آليات للبحث والتصفح والإدارة.

المتطلبات المنجزة

1. نظام المستخدمين والأمان

تسجيل الدخول والمصادقة:

- نظام تسجيل دخول آمن للمستخدمين العاديين والإداريين
- استخدام JWT (JSON Web Tokens) للمصادقة وإدارة الجلسات
- تخزين كلمات المرور بشكل آمن (في النسخة النهائية)
- تسجيل الخروج وإدارة انتهاء صلاحية الجلسات

إدارة الصلاحيات:

- تمييز واضح بين صلاحيات المستخدم العادي والإداري
- تقييد الوصول إلى وظائف الإدارة (إضافة، تعديل، حذف) للإداريين فقط
- حماية نقاط النهاية (Endpoints) باستخدام سياسات التفويض(Authorization Policies)
- تطبيق مبدأ الحد الأدنى من الامتيازات(Principle of Least Privilege)

2. إدارة الكتب

عرض وتصفح الكتب:

- عرض قائمة الكتب مع خيارات التصفية والترتيب
- عرض تفاصيل الكتاب الكاملة (العنوان، النوع، السعر، المؤلف، الناشر)
- تصفح الكتب حسب المؤلف أو الناشر أو النوع
- عرض الكتب المرتبطة بمؤلف أو ناشر معين

البحث في الكتب:

- البحث بالعنوان أو اسم المؤلف أو اسم الناشر
- البحث المتقدم مع خيارات تصفية متعددة
- عرض نتائج البحث بشكل منظم وقابل للترتيب

إدارة الكتب (للإداريين):

- إضافة كتب جديدة مع جميع البيانات المطلوبة
- تعديل بيانات الكتب الموجودة
- حذف الكتب مع التحقق من عدم وجود تعارضات

- ربط الكتب بالمؤلفين والناشرين

3. إدارة المؤلفين

عرض وتصفح المؤلفين:

- عرض قائمة المؤلفين مع معلوماتهم الأساسية
- عرض صفحة تفاصيل المؤلف مع سيرته وقائمة كتبه
- تصفح المؤلفين حسب البلد أو المدينة

البحث عن المؤلفين:

- البحث باسم المؤلف (الأول أو الأخير أو الاسم الكامل)
- البحث حسب البلد أو المدينة
- عرض نتائج البحث بشكل منظم

إدارة المؤلفين (للإداريين):

- إضافة مؤلفين جدد مع جميع البيانات المطلوبة
- تعديل بيانات المؤلفين الموجودين
- حذف المؤلفين مع التحقق من عدم وجود كتب مرتبطة
- إدارة العلاقة بين المؤلفين والكتب

4. إدارة الناشرين

عرض وتصفح الناشرين:

- عرض قائمة الناشرين مع معلوماتهم الأساسية
- عرض صفحة تفاصيل الناشر مع معلوماته وقائمة الكتب المنشورة
- تصفح الناشرين حسب البلد أو المدينة

البحث عن الناشرين:

- البحث باسم الناشر
- البحث حسب البلد أو المدينة
- عرض نتائج البحث بشكل منظم

إدارة الناشرين (للإداريين):

- إضافة ناشرين جدد مع جميع البيانات المطلوبة
- تعديل بيانات الناشرين الموجودين
- حذف الناشرين مع التحقق من عدم وجود كتب مرتبطة
- إدارة العلاقة بين الناشرين والكتب

التقنيات المستخدمة

الواجهة الخلفية(Backend)

1. **إطار العمل والبنية الأساسية**

- **ASP.NET Core 7.0**: إطار عمل متكامل لبناء واجهات برمجة التطبيقات (API) بأداء عالي
- **Minimal API**: نهج حديث لبناء واجهات برمجة التطبيقات بشكل أكثر كفاءة وأقل تعقيد<|im_start|>
- **C# 11**: أحدث إصدار من لغة البرمجة C# مع ميزات متقدمة
- **Dependency Injection**: حقن التبعيات المدمج في ASP.NET Core لتحسين قابلية الاختبار والصيانة

2. **قاعدة البيانات وإدارة البيانات**

- **Entity Framework Core 7.0**: نظام ORM متطور لإدارة قاعدة البيانات
- **SQL Server**: قاعدة البيانات العلائقية لتخزين البيانات
- **LINQ**: لغة استعلام متكاملة للتعامل مع البيانات بشكل برمجي
- **Repository Pattern**: نمط تصميم لفصل منطق الوصول للبيانات عن بقية التطبيق
- **Unit of Work Pattern**: نمط تصميم لإدارة التعاملات مع قاعدة البيانات

3. **الأمان والمصادقة**

- **JWT Authentication**: نظام مصادقة باستخدام JSON Web Tokens
- **Authorization Policies**: سياسات تفويض مخصصة لإدارة الصلاحيات
- **CORS**: تكوين مشاركة الموارد عبر المنشأ للسماح بالاتصال من الواجهة الأمامية
- **Data Validation**: التحقق من صحة البيانات المدخلة لمنع الهجمات

4. **أدوات وتقنيات إضافية**

- **Swagger/OpenAPI**: توثيق تلقائي لواجهة برمجة التطبيقات
- **Logging**: نظام تسجيل الأحداث لتتبع الأخطاء والأداء

- **Async/Await**: برمجة غير متزامنة لتحسين الأداء والاستجابة
- **DTOs (Data Transfer Objects)**: كائنات نقل البيانات لفصل نماذج قاعدة البيانات عن واجهةAPI

الواجهة الأمامية(Frontend)

1. **إطار العمل والبنية الأساسية**
   - **Flutter 3.10**: إطار عمل متعدد المنصات لبناء تطبيقات الموبايل والويب
   - **Dart 3.0**: لغة البرمجة المستخدمة فيFlutter
   - **Material Design 3**: نظام تصميم متكامل للواجهات
   - **Responsive Design**: تصميم متجاوب يعمل على مختلف أحجام الشاشات
2. **إدارة الحالة وهيكلة التطبيق**
   - **Bloc Pattern**: نمط إدارة الحالة في تطبيقاتFlutter
   - **Repository Pattern**: فصل منطق الوصول للبيانات
   - **Clean Architecture**: هيكلة نظيفة للتطبيق تفصل بين طبقات العرض والمنطق والبيانات
   - **Dependency Injection**: حقن التبعيات لتسهيل الاختبار والصيانة
3. **الاتصال بالخادم وإدارة البيانات**
   - **Dio**: مكتبة HTTP متقدمة للاتصال بواجهة برمجة التطبيقات
   - **JSON Serialization**: تحويل البيانات من وإلىJSON
   - **Interceptors**: اعتراض طلبات HTTP لإضافة رؤوس المصادقة وإدارة الأخطاء
   - **Caching**: تخزين مؤقت للبيانات لتحسين الأداء وتقليل الاتصال بالخادم
4. **واجهة المستخدم والتجربة**
   - **Custom Widgets**: مكونات واجهة مستخدم مخصصة
   - **Animations**: حركات وانتقالات سلسة لتحسين تجربة المستخدم
   - **RTL Support**: دعم كامل للغة العربية والكتابة من اليمين لليسار
   - **Theming**: نظام سمات قابل للتخصيص
   - **Form Validation**: التحقق من صحة النماذج في واجهة المستخدم

نقاط القوة

1. **الهيكلة والتصميم المعماري**

هيكلة نظيفة متعددة الطبقات:

- طبقة العرض :(Presentation Layer) واجهة المستخدم وتفاعلاته
- طبقة المنطق :(Business Logic Layer) قواعد العمل ومعالجة البيانات
- طبقة الوصول للبيانات :(Data Access Layer) التعامل مع قاعدة البيانات
- طبقة الكيانات :(Entity Layer) نماذج البيانات الأساسية

تطبيق أنماط التصميم:

- Repository Pattern:لفصل منطق الوصول للبيانات
- Dependency Injection:لتقليل الاقتران بين المكونات
- DTO Pattern:لنقل البيانات بين الطبقات
- Factory Pattern:لإنشاء الكائنات بطريقة موحدة
- Singleton Pattern:للمكونات التي تحتاج إلى نسخة واحدة فقط

تصميم قاعدة بيانات متماسك:

- علاقات واضحة بين الكيانات (الكتب، المؤلفين، الناشرين)
- استخدام المفاتيح الأجنبية والقيود لضمان تكامل البيانات
- تطبيق القواعد المعيارية (Normalization) لتقليل التكرار
- استخدام الفهارس (Indexes) لتحسين أداء الاستعلامات

2. **الأمان والحماية**

نظام مصادقة متكامل:

- استخدام JWT (JSON Web Tokens) للمصادقة
- تشفير كلمات المرور باستخدام خوارزميات آمنة
- تجديد الرموز (Token Refresh) لتحسين الأمان
- التحقق من صلاحية الرموز في كل طلب

تحكم دقيق بالصلاحيات:

- تقسيم المستخدمين إلى مجموعات (إداري، مستخدم عادي)
- تطبيق سياسات التفويض على مستوى نقاط النهاية(Endpoints)

- التحقق من الصلاحيات قبل تنفيذ العمليات الحساسة
- تسجيل محاولات الوصول غير المصرح بها

حماية البيانات:

- التحقق من صحة البيانات المدخلة(Validation)
- الحماية من هجمات حقن(SQL Injection) SQL
- الحماية من هجمات(Cross-Site Scripting) XSS
- تطبيق (Cross-Origin Resource Sharing) CORS بشكل آمن

3. الأداء والكفاءة

استعلامات مُحسّنة:

- استخدام Include و Eager Loading لتحميل البيانات المرتبطة بكفاءة
- تنفيذ الاستعلامات بشكل غير متزامن (Async) لتحسين الاستجابة
- استخدام التخزين المؤقت (Caching) للبيانات المتكررة

تحسين أداء الواجهة الأمامية:

- تحميل البيانات بشكل تدريجي(Lazy Loading)
- استخدام التخزين المؤقت للصور والبيانات
- تحسين أداء القوائم الطويلة باستخدامListView.builder

المراجع

مراجع الواجهة الخلفية(Backend)

- **ASP.NET Core Documentation**
- **مقالات ومدونات**
- MSDN Magazine: https://docs.microsoft.com/en-us/archive/msdn-magazine
- .NET Blog: https://devblogs.microsoft.com/dotnet
- Andrew Lock's Blog: https://andrewlock.net

مراجع الواجهة الأمامية(Frontend)

- **Flutter Documentation**
- الموقع الرسمي https://flutter.dev/docs :
- دليل Flutter للمبتدئين https://flutter.dev/docs/get-started :
- دليل Dart: https://dart.dev/guides
- **Flutter Dev**
- مجتمع Flutter: https://flutter.dev/community
- Flutter YouTube Channel: https://www.youtube.com/c/flutterdev
- Flutter Medium Publication: https://medium.com/flutter

الخلاصة

مشروع المكتبة الإلكترونية يمثل نظام <|im_start|>متكاملاً لإدارة الكتب والمؤلفين والناشرين، مبني باستخدام أحدث التقنيات في تطوير الواجهة الخلفية (ASP.NET Core) والواجهة الأمامية (Flutter). يتميز المشروع بهيكلة نظيفة، وأداء عالي، وتجربة مستخدم سلسة، مع التركيز على الأمان وقابلية التوسع.

تم تصميم النظام ليكون سهل الاستخدام للمستخدمين العاديين، مع توفير أدوات إدارية قوية للمسؤولين. كما يوفر النظام دعماً كاملاً للغة العربية، مما يجعله مناسباً للاستخدام في المكتبات والمؤسسات التعليمية في العالم العربي. بفضل التصميم المرن والهيكلة النظيفة، يمكن توسيع النظام بسهولة ليشمل ميزات إضافية مثل إدارة الاستعارة، ونظام التقييم والمراجعات، والتكامل مع أنظمة الدفع الإلكتروني، مما يجعله قاعدة صلبة لبناء نظام مكتبة إلكترونية متكامل.

تم إضافة بيانات أولية للعمل بها Seed Data

```
114         // التحقق من وجود المستخدمين
115         if (!context.Users.Any())
116         {
117             Console.WriteLine("إضافة بيانات المستخدمين...");
118             context.Users.Add(new e_library_backend.Models.User
119             {
120                 Username = "admin",
121                 Password = "admin123",
122                 FName = "Admin",
123                 LName = "User",
124                 IsAdmin = true
125             });
126             anyChanges = true;
127         }
128 
129         // التحقق من وجود المؤلفين
130         if (!context.Authors.Any())
131         {
132             Console.WriteLine("إضافة بيانات المؤلفين...");
133             context.Authors.AddRange(
134                 new e_library_backend.Models.Author
135                 {
136                     FName = "نجيب",
137                     LName = "محفوظ",
138                     Country = "مصر",
139                     City = "القاهرة"
140                 },
141                 new e_library_backend.Models.Author
142                 {
143                     FName = "غسان",
144                     LName = "كنفاني",
145                     Country = "فلسطين",
```

أهم أكواد المشروع للمتطلبات الأساسية

1- نظام المصادقة والتفويض(Authentication & Authorization)

```
33
34  // Add JWT authentication
35  builder.Services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme)
36      .AddJwtBearer(options =>
37      {
38          options.TokenValidationParameters = new TokenValidationParameters
39          {
40              ValidateIssuerSigningKey = true,
41              IssuerSigningKey = new SymmetricSecurityKey(
42                  Encoding.ASCII.GetBytes(builder.Configuration["Jwt:Key"] ?? "YourSecretKeyHere12345678901234567890")),
43              ValidateIssuer = false,
44              ValidateAudience = false,
45              ValidateLifetime = true,
46              ClockSkew = TimeSpan.Zero
47          };
48      });
49
50  // Add authorization policies
51  builder.Services.AddAuthorization(options =>
52  {
53      options.AddPolicy("AdminOnly", policy =>
54          policy.RequireRole("Admin"));
55  });
56
57  // Add CORS
58  builder.Services.AddCors(options =>
59  {
60      options.AddPolicy("AllowAll", builder =>
61      {
62          builder.AllowAnyOrigin()
63                 .AllowAnyMethod()
64                 .AllowAnyHeader();
65      });
66  });
```

الشرح :يقوم هذا الكود بإعداد نظام المصادقة باستخدام .(JSON Web Tokens) JWT يتم تكوين خيارات التحقق من صحة الرمز المميز مثل مفتاح التوقيع والتحقق من صلاحية الرمز. كما يتم إضافة سياسة تفويض "AdminOnly"التي تتطلب أن يكون المستخدم في دور."Admin"

2- إدارة قاعدة البيانات(Database Context)

```
4
5   namespace e_library_backend.Data;
6
    17 references
7   public class LibraryDbContext : DbContext
8   {
        0 references
9       public LibraryDbContext(DbContextOptions<LibraryDbContext> options) : base(options) { }
10
        8 references
11      public DbSet<User> Users => Set<User>();
        12 references
12      public DbSet<Author> Authors => Set<Author>();
        12 references
13      public DbSet<Publisher> Publishers => Set<Publisher>();
        14 references
14      public DbSet<Book> Books => Set<Book>();
15
        0 references
16      protected override void OnModelCreating(ModelBuilder modelBuilder)
17      {
18          // Configure relationships
19          modelBuilder.Entity<Book>()
20              .HasOne(b => b.Author)
21              .WithMany(a => a.Books)
22              .HasForeignKey(b => b.AuthorId);
23
24          modelBuilder.Entity<Book>()
25              .HasOne(b => b.Publisher)
26              .WithMany(p => p.Books)
27              .HasForeignKey(b => b.PublisherId);
28
29
30      }
```

الشرح :يعرف هذا الكود سياق قاعدة البيانات (DbContext) الذي يمثل جلسة مع قاعدة البيانات. يحتوي على مجموعات DbSet لكل كيان .(Users, Authors, Publishers, Books) كما يتم تكوين العلاقات بين الكيانات في طريقةOnModelCreating ، مثل العلاقة بين الكتاب والمؤلف والناشر.

3- نموذج البيانات(Data Models)

```
namespace e_library_backend.Models;

12 references
public class Book
{
    7 references
    public int Id { get; set; }
    11 references
    public string Title { get; set; } = string.Empty;
    10 references
    public string Type { get; set; } = string.Empty;
    10 references
    public decimal Price { get; set; }

    12 references
    public int PublisherId { get; set; }
    11 references
    public Publisher? Publisher { get; set; }

    13 references
    public int AuthorId { get; set; }
    16 references
    public Author? Author { get; set; }
}
```

الشرح :يمثل هذا النموذج كيان الكتاب في النظام. يحتوي على خصائص أساسية مثل العنوان والنوع والسعر، بالإضافة إلى علاقات مع كيانات أخرى مثل المؤلف والناشر. تستخدم هذه النماذج لتمثيل البيانات في قاعدة البيانات وعبر طبقات التطبيق.

4- واجهات برمجة التطبيقات(API Endpoints)

```
1 reference
public static void MapBookEndpoints(this IEndpointRouteBuilder routes)
{
    var group = routes.MapGroup("/api");

    // Get all books
    group.MapGet("/books", async (IBookRepository bookRepository) =>
        await bookRepository.GetAllBooksAsync());

    // Search books by title
    group.MapGet("/books/search", async (string title, IBookRepository bookRepository) =>
        await bookRepository.SearchBooksByTitleAsync(title));

    // Create new book (admin only)
    group.MapPost("/books", async (HttpRequest request, IBookRepository bookRepository) =>···
     .RequireAuthorization("AdminOnly");

    // Get book by ID
    group.MapGet("/books/{id}", async (int id, IBookRepository bookRepository) =>···

    // Get books by author ID
    group.MapGet("/books/author/{authorId}", async (int authorId, IBookRepository bookRepository) =>···

    // Get books by publisher ID
    group.MapGet("/books/publisher/{publisherId}", async (int publisherId, IBookRepository bookRepository) =>···

    // Update book (admin only)
    group.MapPut("/books/{id}", async (int id, HttpRequest request, IBookRepository bookRepository) =>···
     .RequireAuthorization("AdminOnly");

    // Delete book (admin only)
    group.MapDelete("/books/{id}", async (int id, IBookRepository bookRepository) =>···
     .RequireAuthorization("AdminOnly");
}
```

الشرح :يعرف هذا الكود نقاط النهاية (Endpoints) للتعامل مع الكتب في النظام. يتضمن عمليات مثل الحصول على جميع الكتب، البحث عن كتاب بالمعرف أو العنوان، الحصول على كتب مؤلف معين، وإنشاء كتاب جديد. يلاحظ أن بعض العمليات مثل إنشاء كتاب جديد تتطلب تفويضًا خاصًا.(AdminOnly)

5- مستودعات البيانات(Repositories)

```
2 references
public class BookRepository : IBookRepository
{
    14 references
    private readonly LibraryDbContext _context;

    0 references
    public BookRepository(LibraryDbContext context)
    {
        _context = context;
    }

    2 references
    public async Task<IEnumerable<BookDto>> GetAllBooksAsync()
    {
        var books = await _context.Books
            .Include(b => b.Author)
            .Include(b => b.Publisher)
            .ToListAsync();

        return books.Select(b => new BookDto
        {
            Id = b.Id,
            Title = b.Title,
            Type = b.Type,
            Price = b.Price,
            PublisherId = b.PublisherId,
            PublisherName = b.Publisher?.PName ?? string.Empty,
            AuthorId = b.AuthorId,
            AuthorFullName = $"{b.Author?.FName} {b.Author?.LName}".Trim()
        });
    }
```

```
    }
    2 references
    public async Task<IEnumerable<BookDto>> SearchBooksByTitleAsync(string title)
    {
        var books = await _context.Books
            .Include(b => b.Author)
            .Include(b => b.Publisher)
            .Where(b => b.Title.Contains(title))
            .ToListAsync();
        return books.Select(b => new BookDto
        {
            Id = b.Id,
            Title = b.Title,
            Type = b.Type,
            Price = b.Price,
            PublisherId = b.PublisherId,
            PublisherName = b.Publisher?.PName ?? string.Empty,
            AuthorId = b.AuthorId,
            AuthorFullName = $"{b.Author?.FName} {b.Author?.LName}".Trim()
        });
    }
    2 references
    public async Task<Book> CreateBookAsync(CreateBookDto bookDto)
    {
        var book = new Book
        {
            Title = bookDto.Title,
            Type = bookDto.Type,
            Price = bookDto.Price,
            PublisherId = bookDto.PublisherId,
            AuthorId = bookDto.AuthorId
        };
        _context.Books.Add(book);
        await _context.SaveChangesAsync();
        return book;
```

```
8   public class BookRepository : IBookRepository
68      public async Task<BookDto?> GetBookByIdAsync(int id)
69      {
70          var book = await _context.Books
71              .Include(b => b.Author)
72              .Include(b => b.Publisher)
73              .FirstOrDefaultAsync(b => b.Id == id);
74
75          if (book == null)
76              return null;
77
78          return new BookDto
79          {
80              Id = book.Id,
81              Title = book.Title,
82              Type = book.Type,
83              Price = book.Price,
84              PublisherId = book.PublisherId,
85              PublisherName = book.Publisher?.PName ?? string.Empty,
86              AuthorId = book.AuthorId,
87              AuthorFullName = $"{book.Author?.FName} {book.Author?.LName}".Trim()
88          };
89      }
90
        2 references
91      public async Task<IEnumerable<BookDto>> GetBooksByAuthorIdAsync(int authorId)
92      {
93          var books = await _context.Books
94              .Include(b => b.Author)
95              .Include(b => b.Publisher)
96              .Where(b => b.AuthorId == authorId)
97              .ToListAsync();
98
99          return books.Select(b => new BookDto
100         {
101             Id = b.Id,
```

```
8   public class BookRepository : IBookRepository
91      public async Task<IEnumerable<BookDto>> GetBooksByAuthorIdAsync(int authorId)
92      {
93          var books = await _context.Books
94              .Include(b => b.Author)
95              .Include(b => b.Publisher)
96              .Where(b => b.AuthorId == authorId)
97              .ToListAsync();
98
99          return books.Select(b => new BookDto
100         {
101             Id = b.Id,
102             Title = b.Title,
103             Type = b.Type,
104             Price = b.Price,
105             PublisherId = b.PublisherId,
106             PublisherName = b.Publisher?.PName ?? string.Empty,
107             AuthorId = b.AuthorId,
108             AuthorFullName = $"{b.Author?.FName} {b.Author?.LName}".Trim()
109         });
110     }
111
        2 references
112     public async Task<IEnumerable<BookDto>> GetBooksByPublisherIdAsync(int publisherId)
113     {
114         var books = await _context.Books
115             .Include(b => b.Author)
116             .Include(b => b.Publisher)
117             .Where(b => b.PublisherId == publisherId)
118             .ToListAsync();
119
120         return books.Select(b => new BookDto
121         {
122             Id = b.Id,
123             Title = b.Title,
124             Type = b.Type,
```

```
131     }
132
        2 references
133     public async Task<bool> UpdateBookAsync(int id, CreateBookDto bookDto)
134     {
135         var book = await _context.Books.FindAsync(id);
136         if (book == null)
137             return false;
138
139         // تحديث بيانات الكتاب
140         book.Title = bookDto.Title;
141         book.Type = bookDto.Type;
142         book.Price = bookDto.Price;
143         book.AuthorId = bookDto.AuthorId;
144         book.PublisherId = bookDto.PublisherId;
145
146         _context.Books.Update(book);
147         await _context.SaveChangesAsync();
148         return true;
149     }
150
        2 references
151     public async Task<bool> DeleteBookAsync(int id)
152     {
153         var book = await _context.Books.FindAsync(id);
154         if (book == null)
155             return false;
156
157         _context.Books.Remove(book);
158         await _context.SaveChangesAsync();
159         return true;
160     }
```

الشرح :يمثل هذا الكود مستودع البيانات للكتب. يوفر طرقًا للوصول إلى بيانات الكتب وتنفيذ عمليات مثل الحصول على جميع الكتب والبحث عن كتب بالعنوان. يستخدم Entity Framework Core للتعامل مع قاعدة البيانات ويقوم بتحويل نماذج الكيانات إلى نماذج نقل البيانات (DTOs) لاستخدامها في واجهة API.

6- خدمة API في الواجهة الأمامية(Frontend API Service)

```dart
import 'package:dio/io.dart' show IOHttpClientAdapter;
import 'package:flutter/foundation.dart';
import 'package:dio/dio.dart';
import 'package:http/http.dart' as http;

class ApiService {
  final Dio _dio = Dio();
  late String baseUrl;
  List<String>? _possibleIps;

  ApiService() {
    if (Platform.isAndroid) {
      // تخزين قائمة العناوين البديلة
      _possibleIps = [
        '10.0.2.2', // عنوان خاص بمحاكي الأندرويد (يشير إلى localhost على الكمبيوتر المضيف)
        '192.168.42.10',
        '10.2.0.2',
        '192.168.43.230',
        '192.168.43.1',
        '192.168.42.129',
      ];

      // تعيين عنوان URL الأساسي مبدئ<|im_start|> إلى عنوان المحاكي
      baseUrl = 'http://10.0.2.2:5298/api';
    } else if (Platform.isIOS) {
      baseUrl = 'http://localhost:5298/api'; // استخدم localhost لمحاكي iOS
    } else {
      baseUrl = 'http://localhost:5298/api';
    }

    // إعدادات Dio
    _dio.options.connectTimeout = const Duration(seconds: 15);
    _dio.options.receiveTimeout = const Duration(seconds: 15);
     dio.options.sendTimeout = const Duration(seconds: 15);


    // إضافة interceptor للتصحيح - تعطيل في الإنتاج
    if (kDebugMode) {
      _dio.interceptors.add(
        LogInterceptor(
          requestBody: true,
          responseBody: true,
          // تقليل حجم السجلات
          requestHeader: false,
          responseHeader: false,
        ),
      );
    }

    // تعطيل التحقق من شهادة SSL
    (_dio.httpClientAdapter as IOHttpClientAdapter).createHttpClient = () {
      final client = HttpClient();
      client.badCertificateCallback = (cert, host, port) => true;
      return client;
    };

    debugPrint('تم تهيئة ApiService مع عنوان: $baseUrl');
  }

  // Authentication methods
  Future<Map<String, dynamic>> login(String username, String password) async {
    try {
      final response = await _dio.post(
        '$baseUrl/login',
        data: {'username': username, 'password': password},
      );
      return response.data;
    } on DioException catch (e) {
```

```dart
    debugPrint('تم تهيئة ApiService مع عنوان: $baseUrl');
  }

  // Authentication methods
  Future<Map<String, dynamic>> login(String username, String password) async {
    try {
      final response = await _dio.post(
        '$baseUrl/login',
        data: {'username': username, 'password': password},
      );
      return response.data;
    } on DioException catch (e) {
      if (e.type == DioExceptionType.connectionTimeout ||
          e.type == DioExceptionType.receiveTimeout ||
          e.type == DioExceptionType.sendTimeout) {
        throw Exception(
          'فشل الاتصال بالخادم. يرجى التحقق من اتصال الإنترنت والمحاولة مرة أخرى.',
        );
      } else if (e.type == DioExceptionType.badCertificate) {
        throw Exception(
          'مشكلة في شهادة الأمان. يرجى التحقق من إعدادات الاتصال.',
        );
      } else if (e.type == DioExceptionType.connectionError) {
        throw Exception(
          'خطأ في الاتصال. تأكد من تشغيل الخادم ومن اتصالك بالإنترنت. تحقق من عنوان IP وإعدادات الشبكة.',
        );
      }
      debugPrint('خطأ في تسجيل الدخول: $e');
      throw Exception('فشل تسجيل الدخول: ${e.toString()}');
    } catch (e) {
      debugPrint('خطأ في تسجيل الدخول: $e');
      throw Exception('فشل تسجيل الدخول: ${e.toString()}');
```

الشرح :يمثل هذا الكود خدمة API في تطبيق Flutter الأمامي. يقوم بإعداد عنوان URL الأساسي بناءً على نظام التشغيل ويوفر طرقًا للتفاعل مع واجهة API الخلفية، مثل تسجيل الدخول والحصول على الكتب والبحث عنها. يستخدم مكتبة Dio لإجراء طلبات HTTP ويتعامل مع الاستجابات والأخطاء.

7- تهيئة قاعدة البيانات وتعبئة البيانات الأولية

```
// تهيئة قاعدة البيانات وتعبئة البيانات الأولية
using (var scope = app.Services.CreateScope())
{
    var services = scope.ServiceProvider;
    try
    {
        var context = services.GetRequiredService<LibraryDbContext>();

        // التحقق من إمكانية الاتصال بقاعدة البيانات
        context.Database.CanConnect();
        Console.WriteLine("تم الاتصال بقاعدة البيانات بنجاح");

        // تطبيق الهجرات لإنشاء الجداول إذا لم تكن موجودة
        context.Database.Migrate();
        Console.WriteLine("تم تطبيق الهجرات بنجاح");

        // التحقق من وجود البيانات وإضافتها إذا لم تكن موجودة
        SeedInitialData(context);
        Console.WriteLine("تم التحقق من البيانات الأولية");
    }
    catch (Exception ex)
    {
        Console.WriteLine($"حدث خطأ أثناء تهيئة قاعدة البيانات: {ex.Message}");
    }
}

// Configure middleware
if (app.Environment.IsDevelopment())
{
    app.UseSwagger();
    app.UseSwaggerUI(c =>
    {
        c.SwaggerEndpoint("/swagger/v1/swagger.json", "E-Library API V1");
        c.RoutePrefix = "swagger";
```

الشرح :يقوم هذا الكود بتهيئة قاعدة البيانات عند بدء تشغيل التطبيق. يتحقق من إمكانية الاتصال بقاعدة البيانات، ويطبق الهجرات لإنشاء الجداول، ويقوم بتعبئة البيانات الأولية إذا لم تكن موجودة. هذا يضمن أن التطبيق يبدأ بحالة متسقة وجاهزة للاستخدام.

8- البحث عن الناشرين وعرض كتبهم

```
    public async Task<IEnumerable<Publisher>> SearchPublishersByNameAsync(string name)
    {
        return await _context.Publishers
            .Where(p => p.PName.Contains(name))
            .ToListAsync();
    }
```

الشرح :يوفر هذا الكود وظيفة البحث عن الناشرين بجزء من اسمهم. يستخدم LINQ للاستعلام عن قاعدة البيانات ويعيد قائمة بالناشرين الذين يتطابق اسمهم مع معيار البحث. هذه الوظيفة أساسية لتمكين المستخدمين من العثور على ناشر معين.

إضافة كتاب مع اختيار الناشر والمؤلف في الواجهة الأمامية

9- شاشة إضافة الكتاب(AddBookScreen)

```dart
class AddBookScreen extends StatefulWidget {
  const AddBookScreen({super.key});

  @override
  State<AddBookScreen> createState() => _AddBookScreenState();
}

class _AddBookScreenState extends State<AddBookScreen> {
  final _formKey = GlobalKey<FormState>();
  final _titleController = TextEditingController();
  final _typeController = TextEditingController();
  final _priceController = TextEditingController();

  final ApiService _apiService = ApiService();

  List<Author> _authors = [];
  List<Publisher> _publishers = [];

  int? _selectedAuthorId;
  int? _selectedPublisherId;

  bool _isLoading = false;
  bool _isLoadingData = true;
  String? _error;

  // دالة لتحويل الأرقام العربية إلى أرقام إنجليزية
  String _convertArabicToEnglishNumbers(String input) {
    const arabic = ['٩','٨','٧','٦','٥','٤','٣','٢','١','٠'];
    const english = ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9'];

    for (int i = 0; i < arabic.length; i++) {
      input = input.replaceAll(arabic[i], english[i]);
    }

    return input;
  }
```

الشرح :هذا الكود يمثل جزءًا من حالة شاشة إضافة الكتاب. يقوم بتهيئة متحكمات النص لإدخال بيانات الكتاب، وتحميل قوائم المؤلفين والناشرين من الخادم، وإرسال بيانات الكتاب الجديد عند تقديم النموذج. يستخدم BooksBloc لإدارة حالة إضافة الكتاب والتعامل مع الاستجابات.

10- واجهة المستخدم لإضافة الكتاب

الشرح :هذا الجزء من الكود يبني واجهة المستخدم لشاشة إضافة الكتاب. يتضمن حقول إدخال لعنوان الكتاب ونوعه وسعره، وقوائم منسدلة لاختيار المؤلف والناشر. يتم التحقق من صحة المدخلات باستخدامvalidators ، ويتم عرض مؤشر التحميل أثناء معالجة الطلب. كما يوفر زرًا لإضافة الكتاب وإمكانية إضافة مؤلف أو ناشر جديد.

11- حدث إضافة الكتاب(AddBookEvent)

```dart

class LoadBooksEvent extends BooksEvent {}

class SearchBooksEvent extends BooksEvent {
  final String query;

  SearchBooksEvent({required this.query});
}

class AddBookEvent extends BooksEvent {
  final String token;
  final String title;
  final String type;
  final double price;
  final int publisherId;
  final int authorId;

  AddBookEvent({
    required this.token,
    required this.title,
    required this.type,
    required this.price,
    required this.publisherId,
    required this.authorId,
  });
}

class DeleteBookEvent extends BooksEvent {
  final String token;
  final int bookId;

  DeleteBookEvent({required this.token, required this.bookId});
}

class UpdateBookEvent extends BooksEvent {
```

الشرح :هذا الكود يعرف حدث إضافة الكتاب الذي يتم إرساله إلى BooksBloc. يحتوي على جميع البيانات اللازمة لإنشاء كتاب جديد، بما في ذلك رمز المصادقة (token) وعنوان الكتاب ونوعه وسعره ومعرفات المؤلف والناشر.

12- معالجة حدث إضافة الكتاب فيBooksBloc

```dart
  }
}

Future<void> _onAddBook(AddBookEvent event, Emitter<BooksState> emit) async {
  emit(BooksLoading());
  try {
    // طباعة بيانات الكتاب للتشخيص
    debugPrint('محاولة إضافة كتاب جديد:');
    debugPrint('العنوان: ${event.title}');
    debugPrint('النوع: ${event.type}');
    debugPrint('السعر: ${event.price}');
    debugPrint('معرف الناشر: ${event.publisherId}');
    debugPrint('معرف المؤلف: ${event.authorId}');

    await _apiService.addBook(
      {'token': event.token},
      {
        'title': event.title,
        'type': event.type,
        'price': event.price,
        'publisherId': event.publisherId,
        'authorId': event.authorId,
      },
    );

    // بعد الإضافة، قم بتحميل الكتب مرة أخرى
    add(LoadBooksEvent());
  } catch (e) {
    debugPrint('خطأ في إضافة الكتاب: $e');
    emit(BooksError(message: e.toString()));
  }
}
```

الشرح :هذا الكود يمثل طريقة معالجة حدث إضافة الكتاب في .BooksBloc يقوم بتغيير حالة Bloc إلى BooksLoading، ثم يستدعي خدمة API لإضافة الكتاب باستخدام البيانات المقدمة في الحدث. بعد نجاح العملية، يقوم بإضافة حدث LoadBooksEvent لتحديث قائمة الكتب. في حالة حدوث خطأ، يتم تغيير الحالة إلى BooksErrorمع رسالة الخطأ.

13- طلب API لإضافة الكتاب

```dart
Future<Map<String, dynamic>> addBook(
  Map<String, dynamic> tokenMap,
  Map<String, dynamic> bookData,
) async {
  try {
    final String token = tokenMap['token'];

    // Make sure token is properly formatted
    final options = Options(
      headers: {
        'Authorization': 'Bearer $token',
        'Content-Type': 'application/json',
      },
    );

    // Format data to match backend expectations
    final Map<String, dynamic> formattedData = {
      'Title': bookData['title'],
      'Type': bookData['type'],
      'Price': bookData['price'],
      'PublisherId': bookData['publisherId'],
      'AuthorId': bookData['authorId'],
    };
    debugPrint('Sending formatted data: $formattedData');
    // Send request
    final response = await _dio.post(
      '$baseUrl/books',
      data: formattedData,
      options: options,
    );

    return response.data;
  } catch (e) {
    if (e is DioException && e.response?.statusCode == 403) {
      debugPrint('Authorization error: 403 Forbidden');
```

الشرح :هذا الكود يمثل طريقة إضافة كتاب في خدمة API. يقوم بتنسيق البيانات لتتوافق مع توقعات الخادم الخلفي، ويضيف رأس التفويض (Authorization header) باستخدام رمزJWT ، ثم يرسل طلب POST إلى نقطة نهاية API المناسبة. يتم التعامل مع الاستجابة وإعادتها، أو رمي استثناء في حالة حدوث خطأ.

14- معالجة طلب إضافة الكتاب في الخادم الخلفي

```
22
23          // Create new book (admin only)
24          group.MapPost("/books", async (HttpRequest request, IBookRepository bookRepository) =>···
60           .RequireAuthorization("AdminOnly");
61
```

الشرح :هذا الكود يعرف نقطة نهاية API لإنشاء كتاب جديد في الخادم الخلفي. يقرأ بيانات الطلب، ويحولها إلى كائن CreateBookDto، ثم يستدعي مستودع الكتب لإنشاء الكتاب في قاعدة البيانات. يتطلب تفويضًا من نوع "AdminOnly"للتأكد من أن المستخدم لديه صلاحيات كافية. يعيد استجابة Created مع معرف الكتاب الجديد في حالة النجاح، أو BadRequest مع رسالة الخطأ في حالة الفشل.

15- إنشاء الكتاب في مستودع الكتب

```
53      }
        2 references
54      public async Task<Book> CreateBookAsync(CreateBookDto bookDto)
55      {
56          var book = new Book
57          {
58              Title = bookDto.Title,
59              Type = bookDto.Type,
60              Price = bookDto.Price,
61              PublisherId = bookDto.PublisherId,
62              AuthorId = bookDto.AuthorId
63          };
64          _context.Books.Add(book);
65          await _context.SaveChangesAsync();
66          return book;
67      }
```

الشرح :هذا الكود يمثل طريقة إنشاء كتاب جديد في مستودع الكتب. يقوم بتحويل كائن CreateBookDto إلى كائن Book، ثم يضيفه إلى سياق قاعدة البيانات ويحفظ التغييرات. يعيد كائن الكتاب المنشأ الذي يتضمن المعرف المولد تلقائيًا.

هذه الأكواد توضح كيفية تنفيذ وظيفة إضافة كتاب مع اختيار الناشر والمؤلف في النظام، بدءًا من واجهة المستخدم في التطبيق الأمامي وحتى حفظ البيانات في قاعدة البيانات في الخادم الخلفي.