

وظيفة مقرر WMB للفصل F24

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_HW1_F24_C2_ Jilan_339202
3. MWS_WDC_HW1_F24_C2_ Khaldoun_332508
4.
5.

تقرير مشروع المكتبة الإلكترونية

E_Library

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 (JSON Web Tokens) JWT 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 التصنيف 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 تصنيف 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 تعقيداً

- **C# 11:** أحدث إصدار من لغة البرمجة C# مع ميزات متقدمة
- **Dependency Injection:** حقن التبعية 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 واجهة API

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 في Flutter
- **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 تطبيقات Flutter
- **Repository Pattern:** فصل منطق الوصول للبيانات
- **Clean Architecture:** هيكلة نظيفة للتطبيق تفصل بين طبقات العرض والمنطق والبيانات
- **Dependency Injection:** حقن التبعية لتسهيل الاختبار والصيانة

3. الاتصال بالخادم وإدارة البيانات

- **Dio:** مكتبة HTTP متقدمة للاتصال بواجهة برمجة التطبيقات

- **JSON Serialization:** تحويل البيانات من وإلى JSON
- **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 سلسلة 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. الهيكل والتصميم المعماري

هيكل 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 (JSON Web Tokens) JWT 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 مُحسَّنة:

- 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 باستخدام ListView.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 بهيكلية 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 والهيكلية 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 (JWT (JSON Web Tokens). 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 namespace e_library_backend.Data;
5
6
7 public class LibraryDbContext : DbContext
8 {
9     public LibraryDbContext(DbContextOptions<LibraryDbContext> options) : base(options) { }
10
11     public DbSet<User> Users => Set<User>();
12     public DbSet<Author> Authors => Set<Author>();
13     public DbSet<Publisher> Publishers => Set<Publisher>();
14     public DbSet<Book> Books => Set<Book>();
15
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
31
```

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 طريقة OnModelCreating ، مثل العلاقة بين الكتاب والمؤلف والناشر.

3- نموذج البيانات (Data Models)

```
1 namespace e_library_backend.Models;
2
3 public class Book
4 {
5     public int Id { get; set; }
6     public string Title { get; set; } = string.Empty;
7     public string Type { get; set; } = string.Empty;
8     public decimal Price { get; set; }
9
10    public int PublisherId { get; set; }
11    public Publisher? Publisher { get; set; }
12
13    public int AuthorId { get; set; }
14    public Author? Author { get; set; }
15 }
```

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)

```
10 1
11 1 reference
12 public static void MapBookEndpoints(this IEndpointRouteBuilder routes)
13 {
14     var group = routes.MapGroup("/api");
15
16     // Get all books
17     group.MapGet("/books", async (IBookRepository bookRepository) =>
18         await bookRepository.GetAllBooksAsync());
19
20     // Search books by title
21     group.MapGet("/books/search", async (string title, IBookRepository bookRepository) =>
22         await bookRepository.SearchBooksByTitleAsync(title));
23
24     // Create new book (admin only)
25     group.MapPost("/books", async (HttpRequest request, IBookRepository bookRepository) =>...
60         .RequireAuthorization("AdminOnly");
61
62     // Get book by ID
63     group.MapGet("/books/{id}", async (int id, IBookRepository bookRepository) =>...
71
72     // Get books by author ID
73     group.MapGet("/books/author/{authorId}", async (int authorId, IBookRepository bookRepository) =>...
78
79     // Get books by publisher ID
80     group.MapGet("/books/publisher/{publisherId}", async (int publisherId, IBookRepository bookRepository) =>...
85
86     // Update book (admin only)
87     group.MapPut("/books/{id}", async (int id, HttpRequest request, IBookRepository bookRepository) =>...
127         .RequireAuthorization("AdminOnly");
128
129     // Delete book (admin only)
130     group.MapDelete("/books/{id}", async (int id, IBookRepository bookRepository) =>...
150         .RequireAuthorization("AdminOnly");
151 }
152
```

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

```
8 public class BookRepository : IBookRepository
9 {
10     14 references
11     private readonly LibraryDbContext _context;
12
13     0 references
14     public BookRepository(LibraryDbContext context)
15     {
16         _context = context;
17
18     2 references
19     public async Task<IEnumerable<BookDto>> GetAllBooksAsync()
20     {
21         var books = await _context.Books
22             .Include(b => b.Author)
23             .Include(b => b.Publisher)
24             .ToListAsync();
25
26         return books.Select(b => new BookDto
27         {
28             Id = b.Id,
29             Title = b.Title,
30             Type = b.Type,
31             Price = b.Price,
32             PublisherId = b.PublisherId,
33             PublisherName = b.Publisher?.PName ?? string.Empty,
34             AuthorId = b.AuthorId,
35             AuthorFullName = $"{b.Author?.FName} {b.Author?.LName}".Trim()
36         });
37     }
```

```

34     }
35     2 references
36     public async Task<IEnumerable<BookDto>> SearchBooksByTitleAsync(string title)
37     {
38         var books = await _context.Books
39             .Include(b => b.Author)
40             .Include(b => b.Publisher)
41             .Where(b => b.Title.Contains(title))
42             .ToListAsync();
43         return books.Select(b => new BookDto
44         {
45             Id = b.Id,
46             Title = b.Title,
47             Type = b.Type,
48             Price = b.Price,
49             PublisherId = b.PublisherId,
50             PublisherName = b.Publisher?.PName ?? string.Empty,
51             AuthorId = b.AuthorId,
52             AuthorFullName = $"{b.Author?.FName} {b.Author?.LName}".Trim()
53         });
54     }
55     2 references
56     public async Task<Book> CreateBookAsync(CreateBookDto bookDto)
57     {
58         var book = new Book
59         {
60             Title = bookDto.Title,
61             Type = bookDto.Type,
62             Price = bookDto.Price,
63             PublisherId = bookDto.PublisherId,
64             AuthorId = bookDto.AuthorId
65         };
66         _context.Books.Add(book);
67         await _context.SaveChangesAsync();
68         return book;
69     }
70     public class BookRepository : IBookRepository
71     {
72         public async Task<BookDto?> GetBookByIdAsync(int id)
73         {
74             var book = await _context.Books
75                 .Include(b => b.Author)
76                 .Include(b => b.Publisher)
77                 .FirstOrDefaultAsync(b => b.Id == id);
78
79             if (book == null)
80                 return null;
81
82             return new BookDto
83             {
84                 Id = book.Id,
85                 Title = book.Title,
86                 Type = book.Type,
87                 Price = book.Price,
88                 PublisherId = book.PublisherId,
89                 PublisherName = book.Publisher?.PName ?? string.Empty,
90                 AuthorId = book.AuthorId,
91                 AuthorFullName = $"{book.Author?.FName} {book.Author?.LName}".Trim()
92             };
93         }
94     }
95     2 references
96     public async Task<IEnumerable<BookDto>> GetBooksByAuthorIdAsync(int authorId)
97     {
98         var books = await _context.Books
99             .Include(b => b.Author)
100             .Include(b => b.Publisher)
101             .Where(b => b.AuthorId == authorId)
102             .ToListAsync();
103         return books.Select(b => new BookDto
104         {
105             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
112 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
133     2 references
134     public async Task<bool> UpdateBookAsync(int id, CreateBookDto bookDto)
135     {
136         var book = await _context.Books.FindAsync(id);
137         if (book == null)
138             return false;
139
140         // تحديث بيانات الكتاب
141         book.Title = bookDto.Title;
142         book.Type = bookDto.Type;
143         book.Price = bookDto.Price;
144         book.AuthorId = bookDto.AuthorId;
145         book.PublisherId = bookDto.PublisherId;
146
147         _context.Books.Update(book);
148         await _context.SaveChangesAsync();
149         return true;
150     }
151
152     2 references
153     public async Task<bool> DeleteBookAsync(int id)
154     {
155         var book = await _context.Books.FindAsync(id);
156         if (book == null)
157             return false;
158
159         _context.Books.Remove(book);
160         await _context.SaveChangesAsync();
161         return true;
162     }

```

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)

```
3 import 'package:dio/io.dart' show IOHttpClientAdapter;
4 import 'package:flutter/foundation.dart';
5 import 'package:dio/dio.dart';
6 import 'package:http/http.dart' as http;
7
8 class ApiService {
9   final Dio _dio = Dio();
10  late String baseUrl;
11  List<String>? _possibleIps;
12
13  ApiService() {
14    if (Platform.isAndroid) {
15      // تخزين قائمة العناوين البديلة
16      _possibleIps = [
17        '10.0.2.2', // عنوان خاص بمحاكي الأندرويد (على الكمبيوتر المضيف localhost يشير إلى)
18        '192.168.42.10',
19        '10.2.0.2',
20        '192.168.43.230',
21        '192.168.43.1',
22        '192.168.42.129',
23      ];
24
25      // إلى عنوان المحاكى <|im_start|>الأساسي مبدئ URL تعيين عنوان
26      baseUrl = 'http://10.0.2.2:5298/api';
27    } else if (Platform.isIOS) {
28      baseUrl = 'http://localhost:5298/api'; // localhost استخدم للـ iOS للمحاكي
29    } else {
30      baseUrl = 'http://localhost:5298/api';
31    }
32
33    // إعدادات Dio
34    _dio.options.connectTimeout = const Duration(seconds: 15);
35    _dio.options.receiveTimeout = const Duration(seconds: 15);
36    _dio.options.sendTimeout = const Duration(seconds: 15);
37
38    // للتصحيح - تعطيل في الإنتاج interceptor إضافة
39    if (kDebugMode) {
40      _dio.interceptors.add(
41        LogInterceptor(
42          requestBody: true,
43          responseBody: true,
44          // تقليل حجم السجلات
45          requestHeader: false,
46          responseHeader: false,
47        ),
48      );
49    }
50
51    // تعطيل التحقق من شهادة SSL
52    (_dio.httpClientAdapter as IOHttpClientAdapter).createHttpClient = () {
53      final client = HttpClient();
54      client.badCertificateCallback = (cert, host, port) => true;
55      return client;
56    };
57
58    debugPrint('مع عنوان ApiService تم تهيئة $baseUrl');
59  }
60
61  // Authentication methods
62  Future<Map<String, dynamic>> login(String username, String password) async {
63    try {
64      final response = await _dio.post(
65        '$baseUrl/login',
66        data: {'username': username, 'password': password},
67      );
68      return response.data;
69    } on DioException catch (e) {
70      //
71    }
72  }
```

```

58 | debugPrint('مع عنوان ApiService تم تهيئة $baseUrl');
59 | }
60 |
61 | // Authentication methods
62 | Future<Map<String, dynamic>> login(String username, String password) async {
63 |   try {
64 |     final response = await _dio.post(
65 |       '$baseUrl/login',
66 |       data: {'username': username, 'password': password},
67 |     );
68 |     return response.data;
69 |   } on DioException catch (e) {
70 |     if (e.type == DioExceptionType.connectionTimeout ||
71 |         e.type == DioExceptionType.receiveTimeout ||
72 |         e.type == DioExceptionType.sendTimeout) {
73 |       throw Exception(
74 |         'فشل الاتصال بالخادم. يرجى التحقق من اتصال الإنترنت والمحاولة مرة أخرى',
75 |       );
76 |     } else if (e.type == DioExceptionType.badCertificate) {
77 |       throw Exception(
78 |         'مشكلة في شهادة الأمان. يرجى التحقق من إعدادات الاتصال',
79 |       );
80 |     } else if (e.type == DioExceptionType.connectionError) {
81 |       throw Exception(
82 |         'وإعدادات الشبكة IP خطأ في الاتصال. تأكد من تشغيل الخادم ومن اتصالاتك بالإنترنت. تحقق من عنوان',
83 |       );
84 |     }
85 |     debugPrint('خطأ في تسجيل الدخول: $e');
86 |     throw Exception('فشل تسجيل الدخول: ${e.toString()}');
87 |   } catch (e) {
88 |     debugPrint('خطأ في تسجيل الدخول: $e');
89 |     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 طرقاً 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

```
72 // تهيئة قاعدة البيانات وتعبئة البيانات الأولية
73 using (var scope = app.Services.CreateScope())
74 {
75     var services = scope.ServiceProvider;
76     try
77     {
78         var context = services.GetRequiredService<LibraryDbContext>();
79
80         // التحقق من إمكانية الاتصال بقاعدة البيانات
81         context.Database.CanConnect();
82         Console.WriteLine("تم الاتصال بقاعدة البيانات بنجاح");
83
84         // تطبيق الهجرات لإنشاء الجداول إذا لم تكن موجودة
85         context.Database.Migrate();
86         Console.WriteLine("تم تطبيق الهجرات بنجاح");
87
88         // التحقق من وجود البيانات وإضافتها إذا لم تكن موجودة
89         SeedInitialData(context);
90         Console.WriteLine("تم التحقق من البيانات الأولية");
91     }
92     catch (Exception ex)
93     {
94         Console.WriteLine($"حدث خطأ أثناء تهيئة قاعدة البيانات: {ex.Message}");
95     }
96 }
97
98 // Configure middleware
99 if (app.Environment.IsDevelopment())
100 {
101     app.UseSwagger();
102     app.UseSwaggerUI(c =>
103     {
104         c.SwaggerEndpoint("/swagger/v1/swagger.json", "E-Library API V1");
105         c.RoutePrefix = "swagger";
106     });
107 }
```

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

```
21 public async Task<IEnumerable<Publisher>> SearchPublishersByNameAsync(string name)
22 {
23     return await _context.Publishers
24         .Where(p => p.PName.Contains(name))
25         .ToListAsync();
26 }
27
```

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)

```
11
12 class AddBookScreen extends StatefulWidget {
13   const AddBookScreen({super.key});
14
15   @override
16   State<AddBookScreen> createState() => _AddBookScreenState();
17 }
18
19 class _AddBookScreenState extends State<AddBookScreen> {
20   final _formKey = GlobalKey<FormState>();
21   final _titleController = TextEditingController();
22   final _typeController = TextEditingController();
23   final _priceController = TextEditingController();
24
25   final ApiService _apiService = ApiService();
26
27   List<Author> _authors = [];
28   List<Publisher> _publishers = [];
29
30   int? _selectedAuthorId;
31   int? _selectedPublisherId;
32
33   bool _isLoading = false;
34   bool _isLoadingData = true;
35   String? _error;
36
37   // دالة لتحويل الأرقام العربية إلى أرقام إنجليزية
38   String _convertArabicToEnglishNumbers(String input) {
39     const arabic = ['٠', '١', '٢', '٣', '٤', '٥', '٦', '٧', '٨', '٩'];
40     const english = ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9'];
41
42     for (int i = 0; i < arabic.length; i++) {
43       input = input.replaceAll(arabic[i], english[i]);
44     }
45
46     return input;
47   }
```

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.

```

e_library_frontend / lib / screens / add_book_screen.dart / _AddBookScreenState / build
19 class _AddBookScreenState extends State<AddBookScreen> {
151
152 @override
153 Widget build(BuildContext context) {
154   return Scaffold(
155     appBar: AppBar(title: const Text('إضافة كتاب'), centerTitle: true),
156     body:
157       _isLoadingData
158       ? const Center(child: CircularProgressIndicator())
159       : _error != null
160       ? Center(child: Text('خطأ: $_error'))
161       : SingleChildScrollView(
162         padding: const EdgeInsets.all(16.0),
163         child: Form(
164           key: _formKey,
165           child: Column(
166             crossAxisAlignment: CrossAxisAlignment.stretch,
167             children: [
168               TextFormField(
169                 controller: _titleController,
170                 decoration: const InputDecoration(
171                   labelText: 'عنوان الكتاب',
172                   border: OutlineInputBorder(),
173                 ), // InputDecoration
174                 textDirection:
175                   TextDirection.rtl, // Add this for RTL text
176                 textAlign:
177                   TextAlign.right, // Add this for RTL alignment
178                 validator: (value) {
179                   if (value == null || value.isEmpty) {
180                     return 'الرجاء إدخال عنوان الكتاب';
181                   }
182                   return null;
183                 },
184                 // تمكين إدخال النص من لوحة المفاتيح
185                 autofocus: true,

```

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 باستخدام validators، ويتم عرض مؤشر التحميل أثناء معالجة الطلب. كما يوفر زرًا لإضافة الكتاب وإمكانية إضافة مؤلف أو ناشر جديد.

11- حدث إضافة الكتاب (AddBookEvent)

```
2
3 class LoadBooksEvent extends BooksEvent {}
4
5 class SearchBooksEvent extends BooksEvent {
6     final String query;
7
8     SearchBooksEvent({required this.query});
9 }
10
11 class AddBookEvent extends BooksEvent {
12     final String token;
13     final String title;
14     final String type;
15     final double price;
16     final int publisherId;
17     final int authorId;
18
19     AddBookEvent({
20         required this.token,
21         required this.title,
22         required this.type,
23         required this.price,
24         required this.publisherId,
25         required this.authorId,
26     });
27 }
28
29 class DeleteBookEvent extends BooksEvent {
30     final String token;
31     final int bookId;
32
33     DeleteBookEvent({required this.token, required this.bookId});
34 }
35
36 class UpdateBookEvent extends BooksEvent {
37     final String token;
38     final int bookId;
39     final String title;
40     final String type;
41     final double price;
42     final int publisherId;
43     final int authorId;
44
45     UpdateBookEvent({
46         required this.token,
47         required this.bookId,
48         required this.title,
49         required this.type,
50         required this.price,
51         required this.publisherId,
52         required this.authorId,
53     });
54 }
```

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.

```

85 | }
86 | }
87
88 Future<void> _onAddBook(AddBookEvent event, Emitter<BooksState> emit) async {
89   emit(BooksLoading());
90   try {
91     // طباعة بيانات الكتاب للتشخيص
92     debugPrint('محاولة إضافة كتاب جديد');
93     debugPrint('العنوان: ${event.title}');
94     debugPrint('النوع: ${event.type}');
95     debugPrint('السعر: ${event.price}');
96     debugPrint('معرف الناشر: ${event.publisherId}');
97     debugPrint('معرف المؤلف: ${event.authorId}');
98
99     await _apiService.addBook(
100       {'token': event.token},
101       {
102         'title': event.title,
103         'type': event.type,
104         'price': event.price,
105         'publisherId': event.publisherId,
106         'authorId': event.authorId,
107       },
108     );
109
110     // بعد الإضافة، قم بتحميل الكتب مرة أخرى
111     add(LoadBooksEvent());
112   } catch (e) {
113     debugPrint('خطأ في إضافة الكتاب: $e');
114     emit(BooksError(message: e.toString()));
115   }
116 }

```

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 BooksError مع رسالة الخطأ.


```

141 Future<Map<String, dynamic>> addBook(
142   Map<String, dynamic> tokenMap,
143   Map<String, dynamic> bookData,
144 ) async {
145   try {
146     final String token = tokenMap['token'];
147
148     // Make sure token is properly formatted
149     final options = Options(
150       headers: {
151         'Authorization': 'Bearer $token',
152         'Content-Type': 'application/json',
153       },
154     );
155
156     // Format data to match backend expectations
157     final Map<String, dynamic> formattedData = {
158       'Title': bookData['title'],
159       'Type': bookData['type'],
160       'Price': bookData['price'],
161       'PublisherId': bookData['publisherId'],
162       'AuthorId': bookData['authorId'],
163     };
164     debugPrint('Sending formatted data: $formattedData');
165     // Send request
166     final response = await _dio.post(
167       '$baseUrl/books',
168       data: formattedData,
169       options: options,
170     );
171
172     return response.data;
173   } catch (e) {
174     if (e is DioException && e.response?.statusCode == 403) {
175       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 رمز JWT، ثم يرسل طلب POST إلى نقطة نهاية API المناسبة. يتم التعامل مع الاستجابة وإعادة تنفيذها، 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 | }  
54 | 2 references  
55 | public async Task<Book> CreateBookAsync(CreateBookDto bookDto)  
56 | {  
57 |     var book = new Book  
58 |     {  
59 |         Title = bookDto.Title,  
60 |         Type = bookDto.Type,  
61 |         Price = bookDto.Price,  
62 |         PublisherId = bookDto.PublisherId,  
63 |         AuthorId = bookDto.AuthorId  
64 |     };  
65 |     _context.Books.Add(book);  
66 |     await _context.SaveChangesAsync();  
67 |     return book;  
68 | }
```

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 الأممي وحتى حفظ البيانات في قاعدة البيانات في الخادم الخلفي.