1. Egyszerű REST-CRUD API „BookLibrary” részletes specifikáció

1. Projekt áttekintés

Cél:

Könyvek, szerzők és kategóriák kezelésére szolgáló RESTful WebAPI készítése ASP.NET Core és EF Core segítségével, PostgreSQL adatbázissal.

Fő technológiák:

- .NET 8 (vagy legfrissebb LTS) és C#
- ASP.NET Core Web API
- Entity Framework Core 8
- PostgreSQL
- Swagger/OpenAPI dokumentáció
- Git + GitHub

Időterv (1–2 hét):

Fázis	Tevékenység	Időigény
1. Tervezés	Domain modell, entitások, use case-ek	1–2 nap
2. Környezet előkészítés	.NET projekt, DB, Git repo	1 nap
3. Adatréteg (EF Core)	DbContext, entitások, migrációk	2–3 nap
4. API réteg	Controller-ek, DTO-k, CRUD műveletek	3–4 nap
5. Tesztelés & Dokumentáció	Swagger, egységtesztek	1–2 nap
6. Kiterjesztések	JWT autentikáció, szerepkörök	opcionális

---

2. Domain modell

```
Author 1---* Book *---1 Category
```

Entitás	Tulajdonságok	Relációk
Book	Id, Title, Description, PublishDate, AuthorId, CategoryId	Egy Authorhoz, egy Categoryhoz tartozik
Author	Id, FirstName, LastName, Bio	Több Book
Category	Id, Name, Description	Több Book

---

3. Projekt struktúra

```
BookLibrary/
├─ src/
│ ├─ BookLibrary.Api/ # WebAPI projekt
│ ├─ BookLibrary.Core/ # Domain modellek, DTO-k
│ └─ BookLibrary.Infrastructure/ # EF Core, DbContext, Migrációk
└─ tests/
└─ BookLibrary.Tests/ # Egység- és integrációs tesztek
```

---

4. Lépésről lépésre útmutató

4.1 Projekt létrehozása

```
mkdir BookLibrary && cd BookLibrary
mkdir src tests
cd src
# WebAPI
dotnet new webapi -n BookLibrary.Api
# Class library-k
dotnet new classlib -n BookLibrary.Core
dotnet new classlib -n BookLibrary.Infrastructure
# Solution
cd ..
dotnet new sln
dotnet sln add src/BookLibrary.Api/BookLibrary.Api.csproj \
src/BookLibrary.Core/BookLibrary.Core.csproj \
src/BookLibrary.Infrastructure/BookLibrary.Infrastructure.csproj
```

4.2 Domain modellek (BookLibrary.Core)

- Hozz létre `Entities` mappát, benne a `Book`, `Author`, `Category` osztályokat.
- Adj meg alapvető property-ket.

```
namespace  BookLibrary.Core.Entities

{

public  class  Book

{

public  int Id { get; set; }

public  string Title { get; set; }

public  string Description { get; set; }

public  DateTime PublishDate { get; set; }

public  int AuthorId { get; set; }

public  Author Author { get; set; }

public  int CategoryId { get; set; }

public  Category Category { get; set; }

}

// Author, Category hasonlóan...

}
```

4.3 Infrastructure: DbContext és migrációk

- Telepítsd a csomagokat:

```
dotnet  add  src/BookLibrary.Infrastructure  package  Npgsql.EntityFrameworkCore.PostgreSQL

dotnet  add  src/BookLibrary.Infrastructure  package  Microsoft.EntityFrameworkCore.Design
```

- Hozd létre a `BookLibraryDbContext`-et.

```
namespace BookLibrary.Infrastructure

{

public class BookLibraryDbContext : DbContext

{

public BookLibraryDbContext(DbContextOptions<BookLibraryDbContext> options)

: base(options) { }



public DbSet<Book> Books { get; set; }

public DbSet<Author> Authors { get; set; }

public DbSet<Category> Categories { get; set; }

}

}
```

- `appsettings.json` : PostgreSQL connection string.
- Migrációk létrehozása és alkalmazása:

```
cd src/BookLibrary.Infrastructure

dotnet ef migrations add InitialCreate --startup-project ../BookLibrary.Api

dotnet ef database update --startup-project ../BookLibrary.Api
```

4.4 API réteg: Controller-ek és DTO-k

- Telepítsd az Infrastructure és Core referenciákat az Api projekthez.
- DTO-k: `BookDto` , `CreateBookDto` , `UpdateBookDto` .
- Automapper használata (opcionális) a DTO ↔ Entity konverzióhoz.
- Példa `BooksController` :

```
[ApiController]

[Route("api/[controller]")]

public  class  BooksController : ControllerBase

{

private  readonly  BookLibraryDbContext _context;

public  BooksController(BookLibraryDbContext context) => _context = context;



[HttpGet]

public  async  Task<IEnumerable<BookDto>> GetBooks() { /* lekérdezés, DTO-ba térítés */ }



[HttpGet("{id}")]

public  async  Task<ActionResult<BookDto>> GetBook(int id) { /* ... */ }



[HttpPost]

public  async  Task<ActionResult<BookDto>> CreateBook(CreateBookDto dto) { /* ... */ }



[HttpPut("{id}")]

public  async  Task<IActionResult> UpdateBook(int id, UpdateBookDto dto) { /* ... */ }



[HttpDelete("{id}")]

public  async  Task<IActionResult> DeleteBook(int id) { /* ... */ }

}
```

4.5 Swagger dokumentáció

- `builder.Services.AddSwaggerGen();`
- `app.UseSwagger(); app.UseSwaggerUI();`
- Futtatás után: `https://localhost:{port}/swagger`

4.6 Tesztelés

- Egységtesztek xUnit-el a Controller-ekhez.
- InMemory EF Core használata tesztekben.

```
dotnet  new  xunit  -n  BookLibrary.Tests

dotnet  add  BookLibrary.Tests  reference  ../src/BookLibrary.Api
```

---

5. További bővítési javaslatok

1. **JWT autentikáció**: `Microsoft.AspNetCore.Authentication.JwtBearer`
2. **Szerepkör-kezelés**: admin létrehozhat, user csak olvashat.
3. **Cache**: Redis-be cache-eld a `GET /books` eredményt.
4. **CI/CD**: GitHub Actions workflow Docker build + migráció + tesztek.
