Application understanding -- Book API

**Requirement** : We are preparing this API Application for a Library Collection.

Here the requirement is to collect list of books,we have to provide the access to the users to

* Add a new book
* Update a book
* Delete a book
* Get the list of books

We will start this with the Entity Framework, where once we are good with the entire testing then we will push the completed data structure to the Sql Server and do the Database Transactions.

Initially, to work with EF we have to import the below Nuget Package.

Microsoft.Entityframeworkcore.Inmemory

How to start with the Development.

1. We need to start with the Database structure of the requirement
2. Create Database Diagram so that we can understand the complete relation of the tables.
3. Open VS Studio --> Create a new Core API Project with the name --> BookAPI

API Development

1. Create a Folder named Models
2. Models --> create a class file with the name --> Books.cs
3. Under this create the below Attributes

|  |
| --- |
| **public class GenGuid**  **{**  **[Key]**  **public Guid bookid { get; set; } /\* this will be the primary key\*/**  **}**  **public class Books:GenGuid**  **{**  **public string bookname { get; set; }**  **public int price { get; set; }**  **public string author { get; set; }**  **}**  **public class NewBook**  **{**  **public string bookname { get; set; }**  **public int price { get; set; }**  **public string author { get; set; }**  **}** |

1. Create a new folder with then name Data--> here we will create a class file -BooksAPIDBContext.cs using this file we will create Entity Framework Database Context object, we will inject this dbcontext into our services for Dependency Injection using the Controller that’s talk to the Database to do the transactions

|  |
| --- |
| public class BooksAPIDBContext : DbContext  {  public BooksAPIDBContext(DbContextOptions options) : base(options)  {  }  /\* this property acts as tables for efcore \*/  public DbSet<Books> PropBooks { get; set; }  } |

DBContext is to be inherited from Microsoft.EntityFrameworkCore

Then right click on the DBContext and select to generate constructor with the options, with options will allow us to work with the DbContextoptions.

Now we will create a property of Books this will help us in create the table in sql server and do the transactions.

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

1. Now right click on the Controller folder and click Add-->select New Item --> MVCController Template-->name it as Bookscontroller.cs.
2. Now will annotate the controller class with

|  |
| --- |
| [ApiController] /\* data annotations\*/  [Route("api/[controller]")] |

From here we will write our Http Verb functionalities and our business logic.

|  |
| --- |
| Right click on the BooksController --> or press Ctrl+. on the keyboard  Select Generate constructor. |

Explanation:

Here we generate this constructor, in order to use the Dependency Injection of the DBContext object, this will implement Dependency Inversion Principle of Solid Principles, meaning the instance creation of objects is not happening here instead a private readonly property is used and assigned the DBContext object value and the private property is being used through the methods instead of a new instance.

With, this we are injecting the dependency of an instance to a private property.

We need to declare the private property

|  |
| --- |
| private readonly BooksAPIDBContext \_dbContext; |

In the generated constructor we need to write the below code

|  |
| --- |
| public BooksController(BooksAPIDBContext dbContext) --> DBContext object to be appended into the constructor  {  this.\_dbContext = dbContext; --> here we assign the object to the property  } |

Now with the given Methods in the application you can execute the swagger and call the api methods to test.