Implementing the Repository Pattern

To manage our database interactions, we will be utilising a design pattern called the Repository Pattern, which is a design pattern which will help us keep our data access code less coupled from the controllers and make the ability to make changes easier in the future when we need to.

The following steps will help us build the pieces needed to setup this programming pattern.

Add Repositories Folder to Models Folder

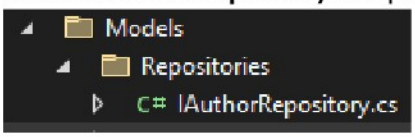
We will start by creating a new folder in the Models folder of our project to hold our repository components.

1. Right click on the Model's folder.
2. Go to the **Add > New Folder** option
3. Rename your folder **Repositories** once it is created.

Create Your First Repository Interface

Next, we will build the interface for our repository. This interface will outline the allowed method calls between the repository and our other parts of the program such as our controllers.

1. Right click on **Repositories** folder.
2. Go to the **Add > New Item** option.
3. Select the **Interface** type in the popup window.
4. Name it **IAuthorRepository** and press **Add**.



Fill Out Your Interface

Once our Interface is built, we will outline the allowed methods for the interface. This will require use to define our method signatures, including their required input parameters and return types.

1. Add your 5 main **CRUD** operations to the interface
2. The methods in this interface will outline the allowed functionality that can be requested by our repository class when requesting database interactions.

```
public interface IAuthorRepository
{
    0 references
    List<Author> GetAllAuthors();
    0 references
    Author GetAuthorById(int id);
    0 references
    void CreateAuthor(Author author);
    0 references
    void UpdateAuthor(Author author);
    0 references
    void DeleteAuthor(int id);
}
```

Create Your First Repository Class

Now we will create the class that will implement our Interface. This is where we will eventually define the logic for each of our Interface methods. Initially through we will just get it setup and we will come back add fill out the logic later.

1. Right Click on the **Repositories** folder
2. Select **Add > Class**
3. Call your class **AuthorRepository**

Setup Repository Class Inheritance and Constructor

Now that our class is created, we need to do our initial setup. We will start by making it inherit from our new Interface and then add a readonly variable and constructor to request access to our context class so we can use it to access the database.

1. Make the **AuthorRepository** class inherit **IAuthorRepository** Interface
    - Ignore red underline for now
2. Add readonly field and **Constructor** to request access to our **BookStoreDBContext** class
    - The context class should already be in the dependency injection from last session.

```
public class AuthorRepository : IAuthorRepository
{
    private readonly BookStoreDBContext _context;

    0 references
    public AuthorRepository(BookStoreDBContext context)
    {
        _context = context;
    }
```

Implement Interface Methods

You will have noticed in our last step that the Interface name gets underlined in red once we added it. This is because the class does not meet the requirements for the interface and does not have the required methods created yet to comply with the Interface.

To fix this, we just follow the following steps:

3. Right click on underlined **IAuthorRepository** interface name at top of class.
4. Select **Quick Actions and Refactorings**
5. Select **Implement Interface** option.

NOTE: *This should build a series of methods in your class with throw exception commands in them. It should also get rid of the red underline. Once this happens, we can leave the class as it is for now, we will fill these methods in properly later in the session.*

Add Repository class to Dependency Injection.

The final step we need to do to finish our repository pattern setup is to add them to the dependency injection. This will be done in the Program.cs class in the services section.

1. Open **Program.cs** class from Solution explorer.
2. Locate section where we added **Context** class last week (**builder.services** sections.)
3. Create a space in the section directly below where we added the **Context** class.
4. Add **AuthorRepository** to Program.cs using interface as the key and the repository class as the associated class.

```
builder.Services.AddDbContext<BookstoreDBContext>(options => {
    options.UseSqlServer(connnectionString);
});
//Adds our repositories to our dependency injection by also providing their
//associated interfaces. This meamns we can request them in our constructor by
//asking for something that complies with the interface, instead of the class
//name itself.
builder.Services.AddScoped<IAuthorRepository,AuthorRepository>();
```

NOTE: *By using the interface as the first property here, we can request our repository by using the interface instead of the class name. This will allow us to change the repository later if needed but not affect our other classes that use it.*

Once this is all finished, we are ready to move onto our controller for the Authors section.