In this document

Web Application Development Tutorial - Part 7: Authors: Database Integration

About This Tutorial

In this tutorial series, you will build an ABP based web application named `Acme.BookStore` . This application is used to manage a list of books and their authors. It is developed using the following technologies:

- **Entity Framework Core** as the ORM provider.
- **MVC / Razor Pages** as the UI Framework.

This tutorial is organized as the following parts;

- Part 1: Creating the server side
- Part 2: The book list page
- Part 3: Creating, updating and deleting books
- Part 4: Integration tests
- Part 5: Authorization
- Part 6: Authors: Domain layer
- **Part 7: Authors: Database Integration (this part)**
- Part 8: Authors: Application Layer
- Part 9: Authors: User Interface
- Part 10: Book to Author Relation

Download the Source Code

This tutorial has multiple versions based on your **UI** and **Database** preferences. We've prepared a few combinations of the source code to be downloaded:

- MVC (Razor Pages) UI with EF Core
- Blazor UI with EF Core
- Angular UI with MongoDB

Introduction

This part explains how to configure the database integration for the `Author` entity introduced in the previous part.

DB Context

Open the `BookStoreDbContext` in the `Acme.BookStore.EntityFrameworkCore` project and add the following `DbSet` property:

4.1 (latest) English

Filter topics

- Getting Started
- Startup Templates
- Tutorials
  - Web Application Development
    - 1: Creating the Server Side
    - 2: The Book List Page
    - 3: Creating, Updating and Deleting Books
    - 4: Integration Tests
    - 5: Authorization
    - 6: Authors: Domain layer
    - 7: Authors: Database Integration
    - 8: Authors: Application Layer
    - 9: Authors: User Interface
    - 10: Book to Author Relation
  - Community Articles
  - Migrating from the ASP.NET Boilerplate
- Fundamentals
- Infrastructure
- Architecture
- API
- User Interface
- Data Access
- Real Time
- Testing
- Samples
- Application Modules
- Release Information
- Reference
- Contribution Guide

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

Then open the `BookStoreDbContextModelCreatingExtensions` class in the same project and add the following lines to the end of the `ConfigureBookStore` method:

```
builder.Entity<Author>(b =>
{
    b.ToTable(BookStoreConsts.DbTablePrefix + "Authors"
        BookStoreConsts.DbSchema);

    b.ConfigureByConvention();

    b.Property(x => x.Name)
        .IsRequired()
        .HasMaxLength(AuthorConsts.MaxNameLength);

    b.HasIndex(x => x.Name);
});
```

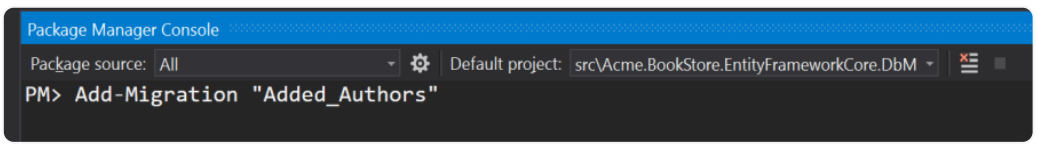
This is just like done for the `Book` entity before, so no need to explain again.

Create a new Database Migration

Open the **Package Manager Console** on Visual Studio and ensure that the **Default project** is `Acme.BookStore.EntityFrameworkCore.DbMigrations` in the Package Manager Console, as shown on the picture below. Also, set the `Acme.BookStore.Web` (or `Acme.BookStore.HttpApi.Host`, depending on your solution) as the **startup project** (right click it on the solution explorer and click to "Set as Startup Project").

Run the following command to create a new database migration:

```
Add-Migration "Added_Authors"
```



This will create a new migration class. Then run the `Update-Database` command to create the table on the database.

> See the Microsoft's documentation for more about the EF Core database migrations.

Implementing the IAuthorRepository

Share on :

In this document

4.1 (latest) English

Filter topics

- Getting Started
- Startup Templates
- Tutorials
  - Web Application Development
    - 1: Creating the Server Side
    - 2: The Book List Page
    - 3: Creating, Updating and Deleting Books
    - 4: Integration Tests
    - 5: Authorization
    - 6: Authors: Domain layer
    - 7: Authors: Database Integration
    - 8: Authors: Application Layer
    - 9: Authors: User Interface
    - 10: Book to Author Relation
  - Community Articles
  - Migrating from the ASP.NET Boilerplate
- Fundamentals
- Infrastructure
- Architecture
- API
- User Interface
- Data Access
- Real Time
- Testing
- Samples
- Application Modules
- Release Information
- Reference
- Contribution Guide

Create a new class, named `EfCoreAuthorRepository` inside the `Acme.BookStore.EntityFrameworkCore` project (in the `Authors` folder) and paste the following code:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Linq.Dynamic.Core;
using System.Threading.Tasks;
using Acme.BookStore.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore;
using Volo.Abp.Domain.Repositories.EntityFrameworkCore;
using Volo.Abp.EntityFrameworkCore;

namespace Acme.BookStore.Authors
{
    public class EfCoreAuthorRepository
        : EfCoreRepository<BookStoreDbContext, Author,
            IAuthorRepository
    {
        public EfCoreAuthorRepository(
            IDbContextProvider<BookStoreDbContext> dbCo
            : base(dbContextProvider)
        {
        }

        public async Task<Author> FindByNameAsync(strin
        {
            return await DbSet.FirstOrDefaultAsync(auth
        }

        public async Task<List<Author>> GetListAsync(
            int skipCount,
            int maxResultCount,
            string sorting,
            string filter = null)
        {
            return await DbSet
                .WhereIf(
                    !filter.IsNullOrWhiteSpace(),
                    author => author.Name.Contains(filt
                )
                .OrderBy(sorting)
                .Skip(skipCount)
                .Take(maxResultCount)
                .ToListAsync();
        }
    }
}
```

- Inherited from the `EfCoreRepository`, so it inherits the standard repository method implementations.
- `WhereIf` is a shortcut extension method of the ABP Framework. It adds the `Where` condition only if the first condition meets (it filters by name, only if the filter was provided). You could do the same yourself, but these type of shortcut methods makes our life easier.
- `sorting` can be a string like `Name`, `Name ASC` or `Name DESC`. It is possible by using the System.Linq.Dynamic.Core NuGet package.

Share on :

In this document

4.1 (latest) English

Filter topics

- Getting Started
- Startup Templates
- Tutorials
  - Web Application Development
    - 1: Creating the Server Side
    - 2: The Book List Page
    - 3: Creating, Updating and Deleting Books
    - 4: Integration Tests
    - 5: Authorization
    - 6: Authors: Domain layer
    - 7: Authors: Database Integration
    - 8: Authors: Application Layer
    - 9: Authors: User Interface
    - 10: Book to Author Relation
  - Community Articles
  - Migrating from the ASP.NET Boilerplate
- Fundamentals
- Infrastructure
- Architecture
- API
- User Interface
- Data Access
- Real Time
- Testing
- Samples
- Application Modules
- Release Information
- Reference
- Contribution Guide

> See the EF Core Integration document for more information on the EF Core based repositories.

The Next Part

See the next part of this tutorial.

Share on :

In this document