Search in documents Contributors Edit Last edit: 11/19/2020 Share on :

In this document

This document has multiple versions. Select the options best fit for you.

UI MVC / Ra Database Er

Web Application Development Tutorial - Part 10: Book to Author Relation

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
- Part 8: Authors: Application Layer
- Part 9: Authors: User Interface
- **Part 10: Book to Author Relation (this part)**

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

We have created `Book` and `Author` functionalities for the book store application. However, currently there is no relation between these entities.

In this tutorial, we will establish a **1 to N** relation between the `Author` and the `Book` entities.

Add Relation to The Book Entity

Open the `Books/Book.cs` in the `Acme.BookStore.Domain` project and add the following property to the `Book` entity:

```
public Guid AuthorId { get; set; }
```

> In this tutorial, we preferred to not add a **navigation property** to the `Author` entity from the `Book` class (like `public Author Author { get; set; }`). This is due to follow the DDD best practices (rule: refer to other aggregates only by id). However, you can add such a navigation property and configure it for the EF Core. In this way, you don't need to write join queries while getting books with their authors (like we will done below) which makes your application code simpler.

Database & Data Migration

Added a new, required `AuthorId` property to the `Book` entity. But, **what about the existing books** on the database? They currently don't have `AuthorId` s and this will be a problem when we try to run the application.

This is a **typical migration problem** and the decision depends on your case;

- If you haven't published your application to the production yet, you can just delete existing books in the database, or you can even delete the entire database in your development environment.
- You can update the existing data programmatically on data migration or seed phase.
- You can manually handle it on the database.

We prefer to **delete the database** (you can run the `Drop-Database` in the *Package Manager Console*) since this is just an example project and data loss is not important. Since this topic is not related to the ABP Framework, we don't go deeper for all the scenarios.

Update the EF Core Mapping

Open the `BookStoreDbContextModelCreatingExtensions` class under the `EntityFrameworkCore` folder of the `Acme.BookStore.EntityFrameworkCore` project and change the `builder.Entity<Book>` part as shown below:

```
builder.Entity<Book>(b =>
{
    b.ToTable(BookStoreConsts.DbTablePrefix + "Books",
    b.ConfigureByConvention(); //auto configure for the
    b.Property(x => x.Name).IsRequired().HasMaxLength(1

    // ADD THE MAPPING FOR THE RELATION
    b.HasOne<Author>().WithMany().HasForeignKey(x => x.
});
```

Add New EF Core Migration

Run the following command in the Package Manager Console (of the Visual Studio) to add a new database migration:

```
Add-Migration "Added_AuthorId_To_Book"
```

> Ensure that the `Acme.BookStore.EntityFrameworkCore.DbMigrations` is the Default project and the `Acme.BookStore.DbMigrator` is the startup project, as always.

This should create a new migration class with the following code in its `Up` method:

```
migrationBuilder.AddColumn<Guid>(
    name: "AuthorId",
    table: "AppBooks",
    nullable: false,
    defaultValue: new Guid("00000000-0000-0000-0000-000

migrationBuilder.CreateIndex(
    name: "IX_AppBooks_AuthorId",
    table: "AppBooks",
    column: "AuthorId");

migrationBuilder.AddForeignKey(
    name: "FK_AppBooks_AppAuthors_AuthorId",
    table: "AppBooks",
    column: "AuthorId",
    principalTable: "AppAuthors",
    principalColumn: "Id",
    onDelete: ReferentialAction.Cascade);
```

- Adds an `AuthorId` field to the `AppBooks` table.
- Creates an index on the `AuthorId` field.
- Declares the foreign key to the `AppAuthors` table.

Change the Data Seeder

Since the `AuthorId` is a required property of the `Book` entity, current data seeder code can not work. Open the `BookStoreDataSeederContributor` in the `Acme.BookStore.Domain` project and change as the following:

```
using System;
using System.Threading.Tasks;
using Acme.BookStore.Authors;
using Acme.BookStore.Books;
using Volo.Abp.Data;
using Volo.Abp.DependencyInjection;
using Volo.Abp.Domain.Repositories;

namespace Acme.BookStore
{
    public class BookStoreDataSeederContributor
        : IDataSeedContributor, ITransientDependency
    {
        private readonly IRepository<Book, Guid> _bookR
        private readonly IAuthorRepository _authorRepos
        private readonly AuthorManager _authorManager;

        public BookStoreDataSeederContributor(
            IRepository<Book, Guid> bookRepository,
            IAuthorRepository authorRepository,
            AuthorManager authorManager)
        {
            _bookRepository = bookRepository;
            _authorRepository = authorRepository;
            _authorManager = authorManager;
        }

        public async Task SeedAsync(DataSeedContext con
        {
            if (await _bookRepository.GetCountAsync() >
            {
                return;
            }

            var orwell = await _authorRepository.Insert
                await _authorManager.CreateAsync(
                    "George Orwell",
                    new DateTime(1903, 06, 25),
                    "Orwell produced literary criticism
                )
            );

            var douglas = await _authorRepository.Inser
                await _authorManager.CreateAsync(
                    "Douglas Adams",
                    new DateTime(1952, 03, 11),
                    "Douglas Adams was an English autho
                )
            );

            await _bookRepository.InsertAsync(
                new Book
                {
                    AuthorId = orwell.Id, // SET THE AU
                    Name = "1984",
                    Type = BookType.Dystopia,
                    PublishDate = new DateTime(1949, 6,
                    Price = 19.84f
                },
                autoSave: true
            );
```

Share on :

In this document

```
                await _bookRepository.InsertAsync(
                    new Book
                    {
                        AuthorId = douglas.Id, // SET THE A
                        Name = "The Hitchhiker's Guide to t
                        Type = BookType.ScienceFiction,
                        PublishDate = new DateTime(1995, 9,
                        Price = 42.0f
                    },
                    autoSave: true
                );
            }
        }
    }
```

The only change is that we set the `AuthorId` properties of the `Book` entities.

> Delete existing books or delete the database before executing the `DbMigrator`. See the *Database & Data Migration* section above for more info.

You can now run the `.DbMigrator` console application to **migrate** the **database schema** and **seed** the initial data.

Application Layer

We will change the `BookAppService` to support the Author relation.

Data Transfer Objects

Let's begin from the DTOs.

BookDto

Open the `BookDto` class in the `Books` folder of the `Acme.BookStore.Application.Contracts` project and add the following properties:

```
public Guid AuthorId { get; set; }
public string AuthorName { get; set; }
```

The final `BookDto` class should be following:

Share on :

In this document

```
using System;
using Volo.Abp.Application.Dtos;

namespace Acme.BookStore.Books
{
    public class BookDto : AuditedEntityDto<Guid>
    {
        public Guid AuthorId { get; set; }

        public string AuthorName { get; set; }

        public string Name { get; set; }

        public BookType Type { get; set; }

        public DateTime PublishDate { get; set; }

        public float Price { get; set; }
    }
}
```

CreateUpdateBookDto

Open the `CreateUpdateBookDto` class in the `Books` folder of the `Acme.BookStore.Application.Contracts` project and add an `AuthorId` property as shown:

```
public Guid AuthorId { get; set; }
```

AuthorLookupDto

Create a new class, `AuthorLookupDto`, inside the `Books` folder of the `Acme.BookStore.Application.Contracts` project:

```
using System;
using Volo.Abp.Application.Dtos;

namespace Acme.BookStore.Books
{
    public class AuthorLookupDto : EntityDto<Guid>
    {
        public string Name { get; set; }
    }
}
```

This will be used in a new method that will be added to the `IBookAppService`.

IBookAppService

Open the `IBookAppService` interface in the `Books` folder of the `Acme.BookStore.Application.Contracts` project and add a new method, named `GetAuthorLookupAsync`, as shown below:

Share on :

In this document

```
using System;
using System.Threading.Tasks;
using Volo.Abp.Application.Dtos;
using Volo.Abp.Application.Services;

namespace Acme.BookStore.Books
{
    public interface IBookAppService :
        ICrudAppService< //Defines CRUD methods
            BookDto, //Used to show books
            Guid, //Primary key of the book entity
            PagedAndSortedResultRequestDto, //Used for
            CreateUpdateBookDto> //Used to create/updat
    {
        // ADD the NEW METHOD
        Task<ListResultDto<AuthorLookupDto>> GetAuthorL
    }
}
```

This new method will be used from the UI to get a list of authors and fill a dropdown list to select the author of a book.

BookAppService

Open the `BookAppService` interface in the `Books` folder of the `Acme.BookStore.Application` project and replace the file content with the following code:

Share on :

In this document

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Acme.BookStore.Authors;
using Acme.BookStore.Permissions;
using Microsoft.AspNetCore.Authorization;
using Volo.Abp.Application.Dtos;
using Volo.Abp.Application.Services;
using Volo.Abp.Domain.Entities;
using Volo.Abp.Domain.Repositories;

namespace Acme.BookStore.Books
{
    [Authorize(BookStorePermissions.Books.Default)]
    public class BookAppService :
        CrudAppService<
            Book, //The Book entity
            BookDto, //Used to show books
            Guid, //Primary key of the book entity
            PagedAndSortedResultRequestDto, //Used for
            CreateUpdateBookDto>, //Used to create/upda
        IBookAppService //implement the IBookAppService
    {
        private readonly IAuthorRepository _authorRepos

        public BookAppService(
            IRepository<Book, Guid> repository,
            IAuthorRepository authorRepository)
            : base(repository)
        {
            _authorRepository = authorRepository;
            GetPolicyName = BookStorePermissions.Books.
            GetListPolicyName = BookStorePermissions.Bo
            CreatePolicyName = BookStorePermissions.Boo
            UpdatePolicyName = BookStorePermissions.Boo
            DeletePolicyName = BookStorePermissions.Boo
        }

        public override async Task<BookDto> GetAsync(Gu
        {
            await CheckGetPolicyAsync();

            //Prepare a query to join books and authors
            var query = from book in Repository
                join author in _authorRepository on boo
                where book.Id == id
                select new { book, author };

            //Execute the query and get the book with a
            var queryResult = await AsyncExecuter.First
            if (queryResult == null)
            {
                throw new EntityNotFoundException(typeo
            }

            var bookDto = ObjectMapper.Map<Book, BookDt
            bookDto.AuthorName = queryResult.author.Nam
            return bookDto;
        }
```

Share on :

In this document

```
        public override async Task<PagedResultDto<BookD
            PagedAndSortedResultRequestDto input)
        {
            await CheckGetListPolicyAsync();

            //Prepare a query to join books and authors
            var query = from book in Repository
                join author in _authorRepository on boo
                orderby input.Sorting
                select new {book, author};

            query = query
                .Skip(input.SkipCount)
                .Take(input.MaxResultCount);

            //Execute the query and get a list
            var queryResult = await AsyncExecuter.ToLis

            //Convert the query result to a list of Boo
            var bookDtos = queryResult.Select(x =>
            {
                var bookDto = ObjectMapper.Map<Book, Bo
                bookDto.AuthorName = x.author.Name;
                return bookDto;
            }).ToList();

            //Get the total count with another query
            var totalCount = await Repository.GetCountA

            return new PagedResultDto<BookDto>(
                totalCount,
                bookDtos
            );
        }

        public async Task<ListResultDto<AuthorLookupDto
        {
            var authors = await _authorRepository.GetLi

            return new ListResultDto<AuthorLookupDto>(
                ObjectMapper.Map<List<Author>, List<Aut
            );
        }
    }
}
```

Share on :

In this document

Let's see the changes we've done:

- Added `[Authorize(BookStorePermissions.Books.Default)]` to authorize the methods we've newly added/overrode (remember, authorize attribute is valid for all the methods of the class when it is declared for a class).
- Injected `IAuthorRepository` to query from the authors.
- Overrode the `GetAsync` method of the base `CrudAppService`, which returns a single `BookDto` object with the given `id`.
  - Used a simple LINQ expression to join books and authors and query them together for the given book id.
  - Used `AsyncExecuter.FirstOrDefaultAsync(...)` to execute the query and get a result. `AsyncExecuter` was previously used in the `AuthorAppService`. Check the repository documentation to understand why we've used it.

  - Throws an `EntityNotFoundException` which results an `HTTP 404` (not found) result if requested book was not present in the database.
  - Finally, created a `BookDto` object using the `ObjectMapper`, then assigning the `AuthorName` manually.
- Overrode the `GetListAsync` method of the base `CrudAppService`, which returns a list of books. The logic is similar to the previous method, so you can easily understand the code.
- Created a new method: `GetAuthorLookupAsync`. This simple gets all the authors. The UI uses this method to fill a dropdown list and select and author while creating/editing books.

Object to Object Mapping Configuration

Introduced the `AuthorLookupDto` class and used object mapping inside the `GetAuthorLookupAsync` method. So, we need to add a new mapping definition inside the `BookStoreApplicationAutoMapperProfile.cs` file of the `Acme.BookStore.Application` project:

```
CreateMap<Author, AuthorLookupDto>();
```

Unit Tests

Some of the unit tests will fail since we made some changed on the `AuthorAppService`. Open the `BookAppService_Tests` in the `Books` folder of the `Acme.BookStore.Application.Tests` project and change the content as the following:

Share on :

In this document

```
using System;
using System.Linq;
using System.Threading.Tasks;
using Acme.BookStore.Authors;
using Shouldly;
using Volo.Abp.Application.Dtos;
using Volo.Abp.Validation;
using Xunit;

namespace Acme.BookStore.Books
{
    public class BookAppService_Tests : BookStoreApplic
    {
        private readonly IBookAppService _bookAppServic
        private readonly IAuthorAppService _authorAppSe

        public BookAppService_Tests()
        {
            _bookAppService = GetRequiredService<IBookA
            _authorAppService = GetRequiredService<IAut
        }

        [Fact]
        public async Task Should_Get_List_Of_Books()
        {
            //Act
            var result = await _bookAppService.GetListA
                new PagedAndSortedResultRequestDto()
            );

            //Assert
            result.TotalCount.ShouldBeGreaterThan(0);
            result.Items.ShouldContain(b => b.Name == "
                                        b.AuthorName ==
        }

        [Fact]
        public async Task Should_Create_A_Valid_Book()
        {
            var authors = await _authorAppService.GetLi
            var firstAuthor = authors.Items.First();

            //Act
            var result = await _bookAppService.CreateAs
                new CreateUpdateBookDto
                {
                    AuthorId = firstAuthor.Id,
                    Name = "New test book 42",
                    Price = 10,
                    PublishDate = System.DateTime.Now,
                    Type = BookType.ScienceFiction
                }
            );

            //Assert
            result.Id.ShouldNotBe(Guid.Empty);
            result.Name.ShouldBe("New test book 42");
        }

        [Fact]
        public async Task Should_Not_Create_A_Book_With
```

Share on :

In this document

```
        {
            var exception = await Assert.ThrowsAsync<Ab
            {
                await _bookAppService.CreateAsync(
                    new CreateUpdateBookDto
                    {
                        Name = "",
                        Price = 10,
                        PublishDate = DateTime.Now,
                        Type = BookType.ScienceFiction
                    }
                );
            });

            exception.ValidationErrors
                .ShouldContain(err => err.MemberNames.A
        }
    }
}
```

- Changed the assertion condition in the `Should_Get_List_Of_Books` from `b => b.Name == "1984"` to `b => b.Name == "1984" && b.AuthorName == "George Orwell"` to check if the author name was filled.
- Changed the `Should_Create_A_Valid_Book` method to set the `AuthorId` while creating a new book, since it is required anymore.

The User Interface

The Book List

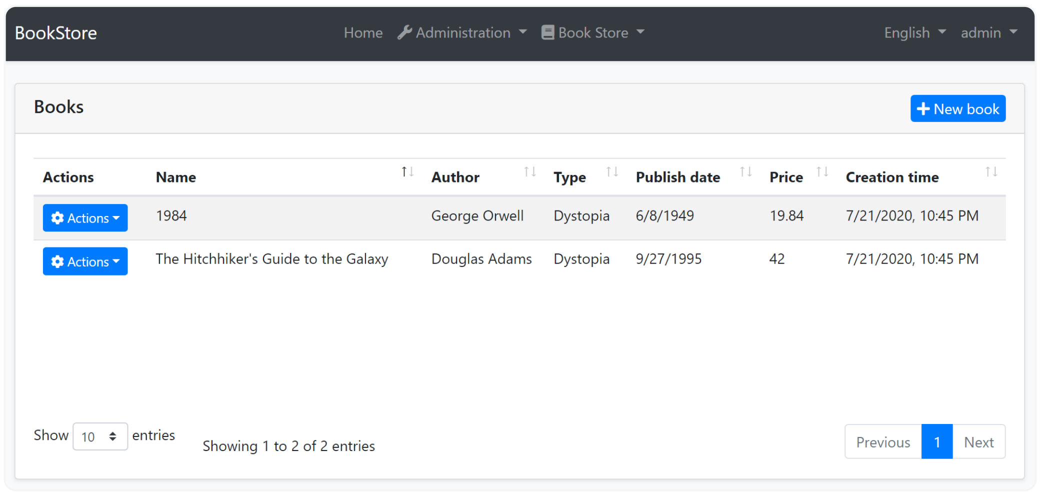
Book list page change is trivial. Open the `Pages/Books/Index.js` in the `Acme.BookStore.Web` project and add the following column definition between the `name` and `type` columns:

```
...
{
    title: l('Name'),
    data: "name"
},

// ADDED the NEW AUTHOR NAME COLUMN
{
    title: l('Author'),
    data: "authorName"
},

{
    title: l('Type'),
    data: "type",
    render: function (data) {
        return l('Enum:BookType:' + data);
    }
},
...
```

Share on :

In this document

When you run the application, you can see the *Author* column on the table:



Actions	Name	Author	Type	Publish date	Price	Creation time
Actions	1984	George Orwell	Dystopia	6/8/1949	19.84	7/21/2020, 10:45 PM
Actions	The Hitchhiker's Guide to the Galaxy	Douglas Adams	Dystopia	9/27/1995	42	7/21/2020, 10:45 PM

Showing 1 to 2 of 2 entries

Create Modal

Open the `Pages/Books/CreateModal.cshtml.cs` in the `Acme.BookStore.Web` project and change the file content as shown below:

Share on :

In this document

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.ComponentModel.DataAnnotations;
using System.Linq;
using System.Threading.Tasks;
using Acme.BookStore.Books;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.Rendering;
using Volo.Abp.AspNetCore.Mvc.UI.Bootstrap.TagHelpers.F

namespace Acme.BookStore.Web.Pages.Books
{
    public class CreateModalModel : BookStorePageModel
    {
        [BindProperty]
        public CreateBookViewModel Book { get; set; }

        public List<SelectListItem> Authors { get; set;

        private readonly IBookAppService _bookAppServic

        public CreateModalModel(
            IBookAppService bookAppService)
        {
            _bookAppService = bookAppService;
        }

        public async Task OnGetAsync()
        {
            Book = new CreateBookViewModel();

            var authorLookup = await _bookAppService.Ge
            Authors = authorLookup.Items
                .Select(x => new SelectListItem(x.Name,
                .ToList();
        }

        public async Task<IActionResult> OnPostAsync()
        {
            await _bookAppService.CreateAsync(
                ObjectMapper.Map<CreateBookViewModel, C
                );
            return NoContent();
        }

        public class CreateBookViewModel
        {
            [SelectItems(nameof(Authors))]
            [DisplayName("Author")]
            public Guid AuthorId { get; set; }

            [Required]
            [StringLength(128)]
            public string Name { get; set; }

            [Required]
            public BookType Type { get; set; } = BookTy

            [Required]
            [DataType(DataType.Date)]
```

Share on :

In this document

```
            public DateTime PublishDate { get; set; } =

            [Required]
            public float Price { get; set; }
        }
    }
}
```

- Changed type of the `Book` property from `CreateUpdateBookDto` to the new `CreateBookViewModel` class defined in this file. The main motivation of this change to customize the model class based on the User Interface (UI) requirements. We didn't want to use UI-related `[SelectItems(nameof(Authors))]` and `[DisplayName("Author")]` attributes inside the `CreateUpdateBookDto` class.
- Added `Authors` property that is filled inside the `OnGetAsync` method using the `IBookAppService.GetAuthorLookupAsync` method defined before.
- Changed the `OnPostAsync` method to map `CreateBookViewModel` object to a `CreateUpdateBookDto` object since `IBookAppService.CreateAsync` expects a parameter of this type.

Edit Modal

Open the `Pages/Books/EditModal.cshtml.cs` in the `Acme.BookStore.Web` project and change the file content as shown below:

Share on :

In this document

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.ComponentModel.DataAnnotations;
using System.Linq;
using System.Threading.Tasks;
using Acme.BookStore.Books;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.Rendering;
using Volo.Abp.AspNetCore.Mvc.UI.Bootstrap.TagHelpers.F

namespace Acme.BookStore.Web.Pages.Books
{
    public class EditModalModel : BookStorePageModel
    {
        [BindProperty]
        public EditBookViewModel Book { get; set; }

        public List<SelectListItem> Authors { get; set;

        private readonly IBookAppService _bookAppServic

        public EditModalModel(IBookAppService bookAppSe
        {
            _bookAppService = bookAppService;
        }

        public async Task OnGetAsync(Guid id)
        {
            var bookDto = await _bookAppService.GetAsyn
            Book = ObjectMapper.Map<BookDto, EditBookVi

            var authorLookup = await _bookAppService.Ge
            Authors = authorLookup.Items
                .Select(x => new SelectListItem(x.Name,
                .ToList();
        }

        public async Task<IActionResult> OnPostAsync()
        {
            await _bookAppService.UpdateAsync(
                Book.Id,
                ObjectMapper.Map<EditBookViewModel, Cre
            );

            return NoContent();
        }

        public class EditBookViewModel
        {
            [HiddenInput]
            public Guid Id { get; set; }

            [SelectItems(nameof(Authors))]
            [DisplayName("Author")]
            public Guid AuthorId { get; set; }

            [Required]
            [StringLength(128)]
            public string Name { get; set; }
```

Share on :

In this document

Share on :

In this document

```
            [Required]
            public BookType Type { get; set; } = BookTy

            [Required]
            [DataType(DataType.Date)]
            public DateTime PublishDate { get; set; } =

            [Required]
            public float Price { get; set; }
        }
    }
}
```

- Changed type of the `Book` property from `CreateUpdateBookDto` to the new `EditBookViewModel` class defined in this file, just like done before for the create modal above.
- Moved the `Id` property inside the new `EditBookViewModel` class.
- Added `Authors` property that is filled inside the `OnGetAsync` method using the `IBookAppService.GetAuthorLookupAsync` method.
- Changed the `OnPostAsync` method to map `EditBookViewModel` object to a `CreateUpdateBookDto` object since `IBookAppService.UpdateAsync` expects a parameter of this type.

These changes require a small change in the `EditModal.cshtml`. Remove the `<abp-input asp-for="Id" />` tag since we no longer need to it (since moved it to the `EditBookViewModel`). The final content of the `EditModal.cshtml` should be following:

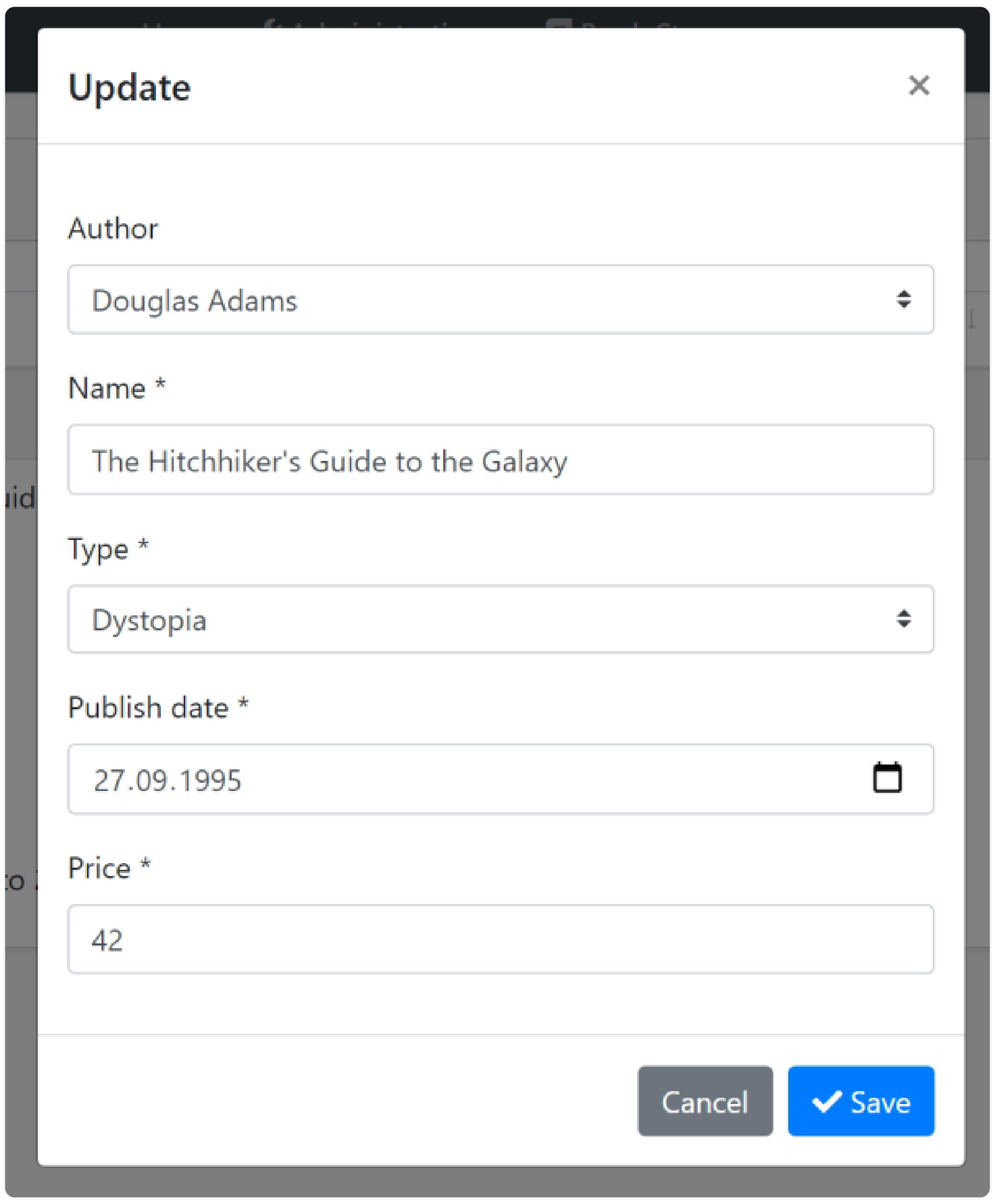
```
@page
@using Acme.BookStore.Localization
@using Acme.BookStore.Web.Pages.Books
@using Microsoft.Extensions.Localization
@using Volo.Abp.AspNetCore.Mvc.UI.Bootstrap.TagHelpers.
@model EditModalModel
@inject IStringLocalizer<BookStoreResource> L
@{
    Layout = null;
}
<abp-dynamic-form abp-model="Book" asp-page="/Books/Edi
    <abp-modal>
        <abp-modal-header title="@L["Update"].Value"></
        <abp-modal-body>
            <abp-form-content />
        </abp-modal-body>
        <abp-modal-footer buttons="@(AbpModalButtons.Ca
    </abp-modal>
</abp-dynamic-form>
```

Object to Object Mapping Configuration

The changes above requires to define some object to object mappings. Open the `BookStoreWebAutoMapperProfile.cs` in the `Acme.BookStore.Web` project and add the following mapping definitions inside the constructor:

```
CreateMap<Pages.Books.CreateModalModel.CreateBookViewMo
CreateMap<BookDto, Pages.Books.EditModalModel.EditBookV
CreateMap<Pages.Books.EditModalModel.EditBookViewModel,
```

You can run the application and try to create a new book or update an existing book. You will see a drop down list on the create/update form to select the author of the book:



Update

Author: Douglas Adams

Name *: The Hitchhiker's Guide to the Galaxy

Type *: Dystopia

Publish date *: 27.09.1995

Price *: 42

Share on :

In this document