In this document

Web Application Development Tutorial - Part 2: The Book List Page

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 (this part)**
- 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

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

Video Tutorial

This part is also recorded as a video tutorial and **published on YouTube**.

Dynamic JavaScript Proxies

It's common to call the HTTP API endpoints via AJAX from the **JavaScript** side. You can use `$.ajax` or another tool to call the endpoints. However, ABP offers a better way.

ABP **dynamically** creates **JavaScript Proxies** for all API endpoints. So, you can use any **endpoint** just like calling a **JavaScript function**.

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

Testing in the Developer Console

Share on :

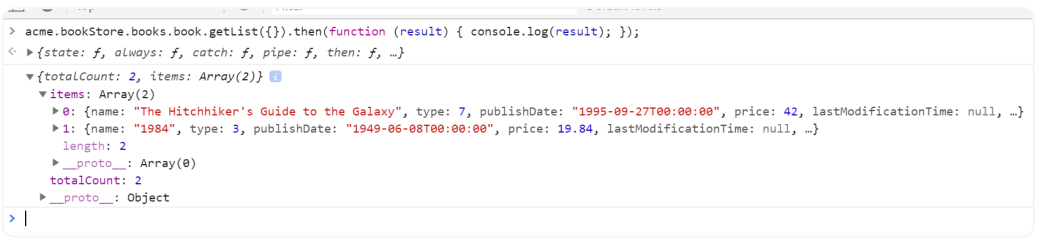
In this document

You can easily test the JavaScript proxies using your favorite browser's **Developer Console**. Run the application, open your browser's **developer tools** (*shortcut is generally F12*), switch to the **Console** tab, type the following code and press enter:

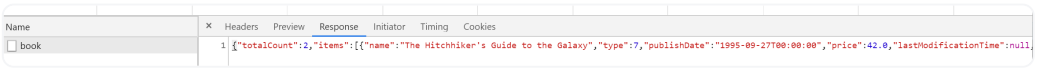
```
acme.bookStore.books.book.getList({}).done(function (re
```

- `acme.bookStore.books` is the namespace of the `BookAppService` converted to camelCase.
- `book` is the conventional name for the `BookAppService` (removed `AppService` postfix and converted to camelCase).
- `getList` is the conventional name for the `GetListAsync` method defined in the `CrudAppService` base class (removed `Async` postfix and converted to camelCase).
- `{}` argument is used to send an empty object to the `GetListAsync` method which normally expects an object of type `PagedAndSortedResultRequestDto` that is used to send paging and sorting options to the server (all properties are optional with default values, so you can send an empty object).
- `getList` function returns a `promise`. You can pass a callback to the `then` (or `done`) function to get the result returned from the server.

Running this code produces the following output:



You can see the **book list** returned from the server. You can also check the **network** tab of the developer tools to see the client to server communication:



Let's **create a new book** using the `create` function:

```
acme.bookStore.books.book.create({
        name: 'Foundation',
        type: 7,
        publishDate: '1951-05-24',
        price: 21.5
    }).then(function (result) {
        console.log('successfully created the book with
    });
```

You should see a message in the console something like that:

```
successfully created the book with id: 439b0ea8-923e-8e
```

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

Check the `Books` table in the database to see the new book row. You can try `get`, `update` and `delete` functions yourself.

We will use these dynamic proxy functions in the next sections to communicate to the server.

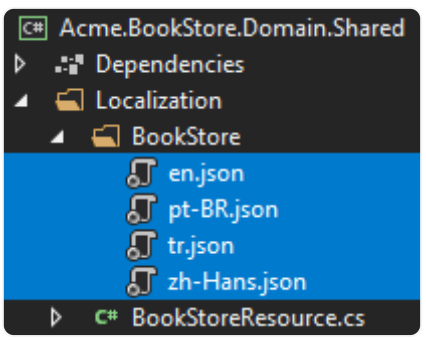
Share on :

In this document

Localization

Before starting to the UI development, we first want to prepare the localization texts (you normally do this when needed while developing your application).

Localization texts are located under the `Localization/BookStore` folder of the `Acme.BookStore.Domain.Shared` project:



Open the `en.json` (*the English translations*) file and change the content as below:

```
{
  "Culture": "en",
  "Texts": {
    "Menu:Home": "Home",
    "Welcome": "Welcome",
    "LongWelcomeMessage": "Welcome to the application.
    "Menu:BookStore": "Book Store",
    "Menu:Books": "Books",
    "Actions": "Actions",
    "Close": "Close",
    "Delete": "Delete",
    "Edit": "Edit",
    "PublishDate": "Publish date",
    "NewBook": "New book",
    "Name": "Name",
    "Type": "Type",
    "Price": "Price",
    "CreationTime": "Creation time",
    "AreYouSure": "Are you sure?",
    "AreYouSureToDelete": "Are you sure you want to del
    "Enum:BookType:0": "Undefined",
    "Enum:BookType:1": "Adventure",
    "Enum:BookType:2": "Biography",
    "Enum:BookType:3": "Dystopia",
    "Enum:BookType:4": "Fantastic",
    "Enum:BookType:5": "Horror",
    "Enum:BookType:6": "Science",
    "Enum:BookType:7": "Science fiction",
    "Enum:BookType:8": "Poetry"
  }
}
```

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

Share on :

In this document

- Localization key names are arbitrary. You can set any name. We prefer some conventions for specific text types;
  - Add `Menu:` prefix for menu items.
  - Use `Enum:<enum-type>:<enum-value>` naming convention to localize the enum members. When you do it like that, ABP can automatically localize the enums in some proper cases.

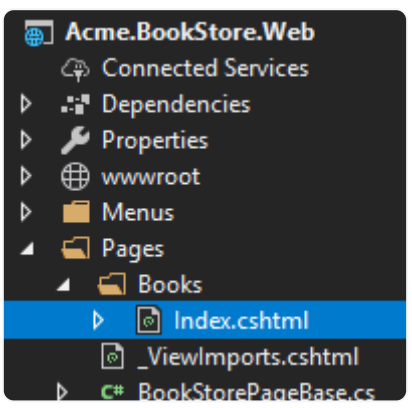
If a text is not defined in the localization file, it **fallbacks** to the localization key (as ASP.NET Core's standard behavior).

> ABP's localization system is built on ASP.NET Core's standard localization system and extends it in many ways. See the localization document for details.

Create a Books Page

It's time to create something visible and usable! Instead of classic MVC, we will use the Razor Pages UI approach which is recommended by Microsoft.

Create `Books` folder under the `Pages` folder of the `Acme.BookStore.Web` project. Add a new Razor Page by right clicking the Books folder then selecting **Add > Razor Page** menu item. Name it as `Index` :



Open the `Index.cshtml` and change the whole content as shown below:

```
@page
@using Acme.BookStore.Web.Pages.Books
@model IndexModel

<h2>Books</h2>
```

`Index.cshtml.cs` content should be like that:

```
using Microsoft.AspNetCore.Mvc.RazorPages;

namespace Acme.BookStore.Web.Pages.Books
{
    public class IndexModel : PageModel
    {
        public void OnGet()
        {

        }
    }
}
```

4.1 (latest) English

Filter topics

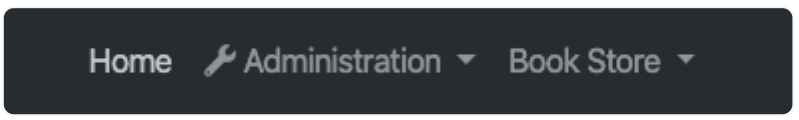
- Getting Started
- Startup Templates
- Tutorials
  - Web Application Development
    - 1: Creating the Server Side
    - 2: The Book List Page
    - 3: Creating, Updating and Deleting Books
    - 4: Integration Tests
    - 5: Authorization
    - 6: Authors: Domain layer
    - 7: Authors: Database Integration
    - 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

Add Books Page to the Main Menu

Open the `BookStoreMenuContributor` class in the `Menus` folder and add the following code to the end of the `ConfigureMainMenuAsync` method:

```
context.Menu.AddItem(
    new ApplicationMenuItem(
        "BooksStore",
        l["Menu:BookStore"],
        icon: "fa fa-book"
    ).AddItem(
        new ApplicationMenuItem(
            "BooksStore.Books",
            l["Menu:Books"],
            url: "/Books"
        )
    )
);
```

Run the project, login to the application with the username `admin` and the password `1q2w3E*` and see the new menu item has been added to the main menu:



When you click to the Books menu item under the Book Store parent, you are being redirected to the new empty Books Page.

Book List

We will use the Datatables.net jQuery library to show the book list. Datatables library completely work via AJAX, it is fast, popular and provides a good user experience.

> Datatables library is configured in the startup template, so you can directly use it in any page without including any style or script file to your page.

Index.cshtml

Change the `Pages/Books/Index.cshtml` as following:

Share on :

In this document

4.1 (latest) English

Filter topics

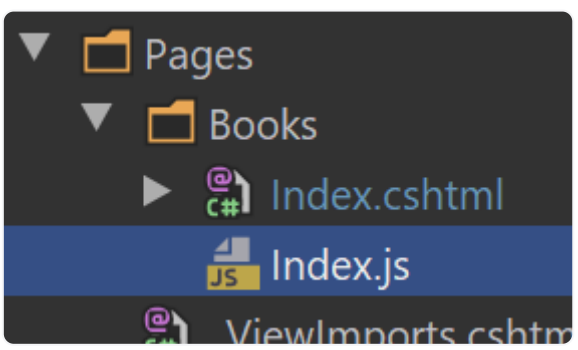
- Getting Started
- Startup Templates
- Tutorials
  - Web Application Development
    - 1: Creating the Server Side
    - 2: The Book List Page
    - 3: Creating, Updating and Deleting Books
    - 4: Integration Tests
    - 5: Authorization
    - 6: Authors: Domain layer
    - 7: Authors: Database Integration
    - 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

```
@page
@using Acme.BookStore.Localization
@using Acme.BookStore.Web.Pages.Books
@using Microsoft.Extensions.Localization
@model IndexModel
@inject IStringLocalizer<BookStoreResource> L
@section scripts
{
    <abp-script src="/Pages/Books/Index.js" />
}
<abp-card>
    <abp-card-header>
        <h2>@L["Books"]</h2>
    </abp-card-header>
    <abp-card-body>
        <abp-table striped-rows="true" id="BooksTable">
    </abp-card-body>
</abp-card>
```

- `abp-script` tag helper is used to add external **scripts** to the page. It has many additional features compared to standard `script` tag. It handles **minification** and **versioning**. See the bundling & minification document for details.
- `abp-card` is a tag helper for Twitter Bootstrap's card component. There are other useful tag helpers provided by the ABP Framework to easily use most of the bootstrap components. You could use the regular HTML tags instead of these tag helpers, but using tag helpers reduces HTML code and prevents errors by help the of IntelliSense and compile time type checking. Further information, see the tag helpers document.

Index.js

Create an `Index.js` file under the `Pages/Books` folder:



The content of the file is shown below:

Share on :

In this document

```
$(function () {
    var l = abp.localization.getResource('BookStore');

    var dataTable = $('#BooksTable').DataTable(
        abp.libs.datatables.normalizeConfiguration({
            serverSide: true,
            paging: true,
            order: [[1, "asc"]],
            searching: false,
            scrollX: true,
            ajax: abp.libs.datatables.createAjax(acme.b
            columnDefs: [
                {
                    title: l('Name'),
                    data: "name"
                },
                {
                    title: l('Type'),
                    data: "type",
                    render: function (data) {
                        return l('Enum:BookType:' + dat
                    }
                },
                {
                    title: l('PublishDate'),
                    data: "publishDate",
                    render: function (data) {
                        return luxon
                            .DateTime
                            .fromISO(data, {
                                locale: abp.localizatio
                            }).toLocaleString();
                    }
                },
                {
                    title: l('Price'),
                    data: "price"
                },
                {
                    title: l('CreationTime'), data: "cr
                    render: function (data) {
                        return luxon
                            .DateTime
                            .fromISO(data, {
                                locale: abp.localizatio
                            }).toLocaleString(luxon.Dat
                    }
                }
            ]
        })
    );
});
```

- `abp.localization.getResource` gets a function that is used to localize text using the same JSON file defined in the server side. In this way, you can share the localization values with the client side.
- `abp.libs.datatables.normalizeConfiguration` is a helper function defined by the ABP Framework. There's no requirement to use it, but it simplifies the Datatables configuration by providing conventional default values for missing options.

In this document

4.1 (latest) English

Filter topics

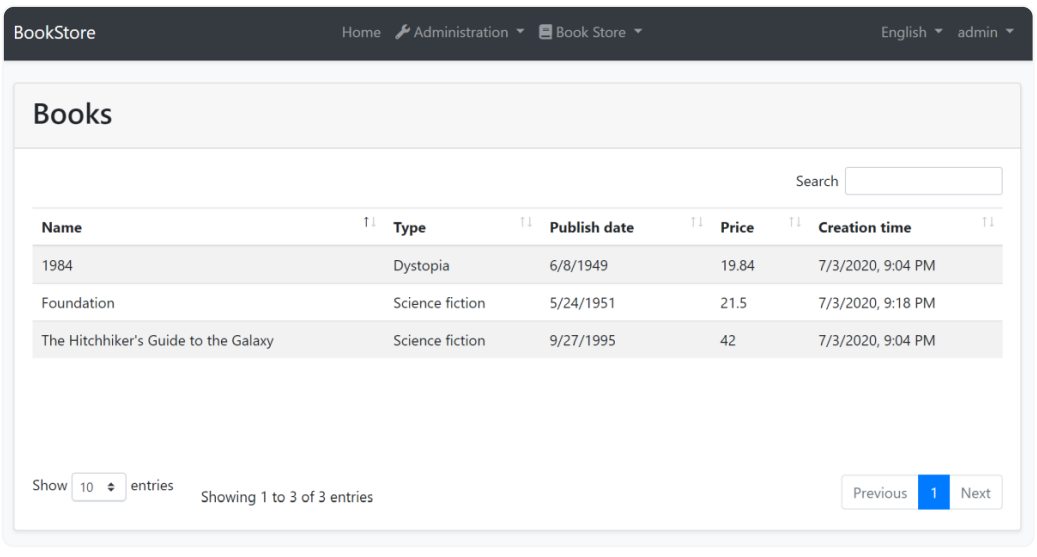
- Getting Started
- Startup Templates
- Tutorials
  - Web Application Development
    - 1: Creating the Server Side
    - 2: The Book List Page
    - 3: Creating, Updating and Deleting Books
    - 4: Integration Tests
    - 5: Authorization
    - 6: Authors: Domain layer
    - 7: Authors: Database Integration
    - 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

- `abp.libs.datatables.createAjax` is another helper function to adapt ABP's dynamic JavaScript API proxies to Datatable's expected parameter format
- `acme.bookStore.books.book.getList` is the dynamic JavaScript proxy function introduced before.
- luxon library is also a standard library that is pre-configured in the solution, so you can use to perform date/time operations easily.

> See Datatables documentation for all configuration options.

Run the Final Application

You can run the application! The final UI of this part is shown below:



This is a fully working, server side paged, sorted and localized table of books.

The Next Part

See the next part of this tutorial.

Share on :

In this document