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 3: Creating, Updating and Deleting Books

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 (this part)**
- 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:

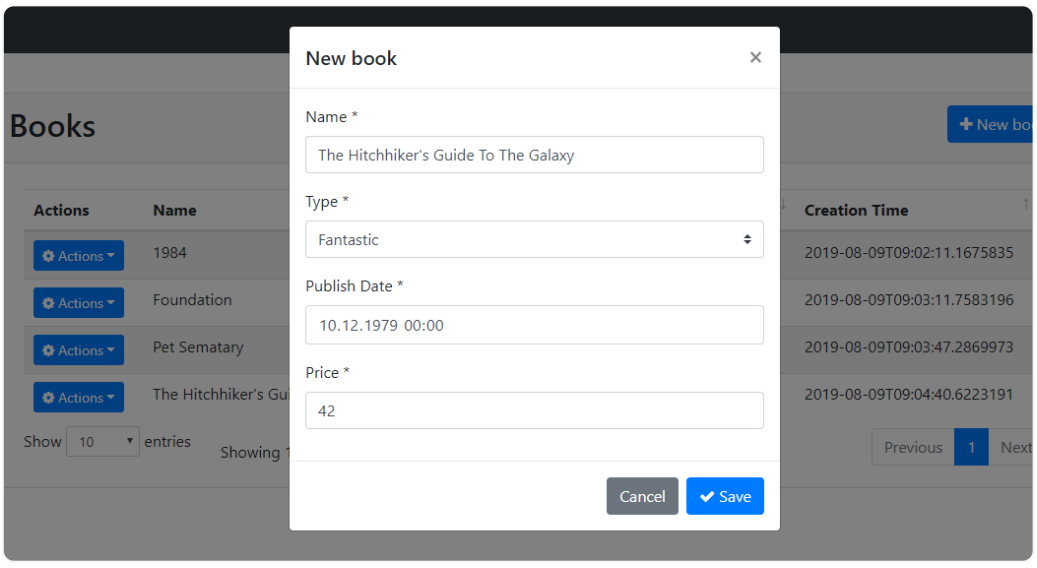
- MVC (Razor Pages) UI with EF Core
- Blazor UI with EF Core
- Angular UI with MongoDB

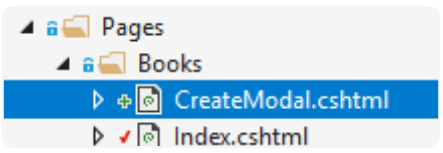
Video Tutorial

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

Creating a New Book

In this section, you will learn how to create a new modal dialog form to create a new book. The modal dialog will look like in the image below:





Create the Modal Form

Create a new razor page, named `CreateModal.cshtml` under the `Pages/Books` folder of the `Acme.BookStore.Web` project.

CreateModal.cshtml.cs

Open the `CreateModal.cshtml.cs` file (`CreateModalModel` class) and replace with the following code:

```
using System.Threading.Tasks;
using Acme.BookStore.Books;
using Microsoft.AspNetCore.Mvc;

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

        private readonly IBookAppService _bookAppServic

        public CreateModalModel(IBookAppService bookApp
        {
            _bookAppService = bookAppService;
        }

        public void OnGet()
        {
            Book = new CreateUpdateBookDto();
        }

        public async Task<IActionResult> OnPostAsync()
        {
            await _bookAppService.CreateAsync(Book);
            return NoContent();
        }
    }
}
```


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

- This class is derived from the `BookStorePageModel` instead of standard `PageModel`. `BookStorePageModel` indirectly inherits the `PageModel` and adds some common properties & methods that can be shared in your page model classes.
- `[BindProperty]` attribute on the `Book` property binds post request data to this property.
- This class simply injects the `IBookAppService` in the constructor and calls the `CreateAsync` method in the `OnPostAsync` handler.
- It creates a new `CreateUpdateBookDto` object in the `OnGet` method. ASP.NET Core can work without creating a new instance like that. However, it doesn't create an instance for you and if your class has some default value assignments or code execution in the class constructor, they won't work. For this case, we set default values for some of the `CreateUpdateBookDto` properties.

CreateModal.cshtml

Open the `CreateModal.cshtml` file and paste the code below:

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

- This modal uses `abp-dynamic-form` tag helper to automatically create the form from the `CreateBookViewModel` model class.
- `abp-model` attribute indicates the model object where it's the `Book` property in this case.
- `abp-form-content` tag helper is a placeholder to render the form controls (it is optional and needed only if you have added some other content in the `abp-dynamic-form` tag, just like in this page).

> Tip: `Layout` should be `null` just as done in this example since we don't want to include all the layout for the modals when they are loaded via AJAX.

Add the "New book" Button

Open the `Pages/Books/Index.cshtml` and set the content of `abp-card-header` tag as below:

Share on :

In this document

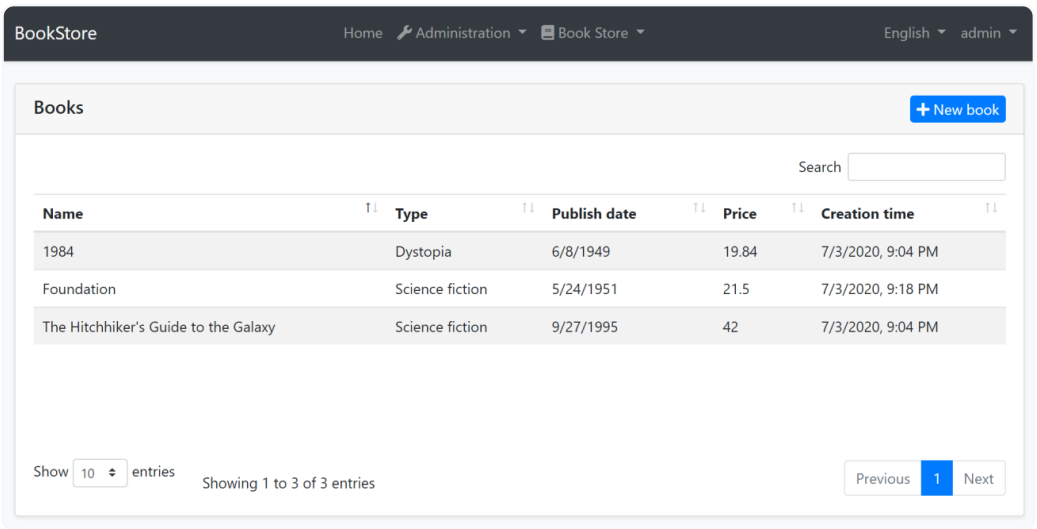
```
<abp-card-header>
    <abp-row>
        <abp-column size-md="_6">
            <abp-card-title>@L["Books"]</abp-card-title
        </abp-column>
        <abp-column size-md="_6" class="text-right">
            <abp-button id="NewBookButton"
                        text="@L["NewBook"].Value"
                        icon="plus"
                        button-type="Primary"/>
        </abp-column>
    </abp-row>
</abp-card-header>
```

The final content of the `Index.cshtml` is shown below:

```
@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>
        <abp-row>
            <abp-column size-md="_6">
                <abp-card-title>@L["Books"]</abp-card-t
            </abp-column>
            <abp-column size-md="_6" class="text-right"
                <abp-button id="NewBookButton"
                            text="@L["NewBook"].Value"
                            icon="plus"
                            button-type="Primary"/>
            </abp-column>
        </abp-row>
    </abp-card-header>
    <abp-card-body>
        <abp-table striped-rows="true" id="BooksTable">
    </abp-card-body>
</abp-card>
```

This adds a new button called **New book** to the **top-right** of the table:

In this document



Open the `Pages/Books/Index.js` and add the following code just after the `Datatable` configuration:

```
var createModal = new abp.ModalManager(abp.appPath + 'B

createModal.onResult(function () {
    dataTable.ajax.reload();
});

$('#NewBookButton').click(function (e) {
    e.preventDefault();
    createModal.open();
});
```

- `abp.ModalManager` is a helper class to manage modals in the client side. It internally uses Twitter Bootstrap's standard modal, but abstracts many details by providing a simple API.
- `createModal.onResult(...)` used to refresh the data table after creating a new book.
- `createModal.open();` is used to open the model to create a new book.

The final content of the `Index.js` should be like that:

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

```
$(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
                    }
                }
            ]
        })
    );

    var createModal = new abp.ModalManager(abp.appPath

    createModal.onResult(function () {
        dataTable.ajax.reload();
    });

    $('#NewBookButton').click(function (e) {
        e.preventDefault();
        createModal.open();
```

Share on :

In this document

```
    });
});
```

Now, you can **run the application** and add some new books using the new modal form.

Updating a Book

Create a new razor page, named `EditModal.cshtml` under the `Pages/Books` folder of the `Acme.BookStore.Web` project:

EditModal.cshtml.cs

Open the `EditModal.cshtml.cs` file (`EditModalModel` class) and replace with the following code:

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

```
using System;
using System.Threading.Tasks;
using Acme.BookStore.Books;
using Microsoft.AspNetCore.Mvc;

namespace Acme.BookStore.Web.Pages.Books
{
    public class EditModalModel : BookStorePageModel
    {
        [HiddenInput]
        [BindProperty(SupportsGet = true)]
        public Guid Id { get; set; }

        [BindProperty]
        public CreateUpdateBookDto Book { get; set; }

        private readonly IBookAppService _bookAppServic

        public EditModalModel(IBookAppService bookAppSe
        {
            _bookAppService = bookAppService;
        }

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

        public async Task<IActionResult> OnPostAsync()
        {
            await _bookAppService.UpdateAsync(Id, Book)
            return NoContent();
        }
    }
}
```

- `[HiddenInput]` and `[BindProperty]` are standard ASP.NET Core MVC attributes. `SupportsGet` is used to be able to get `Id` value from query string parameter of the request.
- In the `OnGetAsync` method, we get `BookDto` from the `BookAppService` and this is being mapped to the DTO object `CreateUpdateBookDto`.
- The `OnPostAsync` uses `BookAppService.UpdateAsync(...)` to update the entity.

Mapping from BookDto to CreateUpdateBookDto

To be able to map the `BookDto` to `CreateUpdateBookDto`, configure a new mapping. To do this, open the `BookStoreWebAutoMapperProfile.cs` in the `Acme.BookStore.Web` project and change it as shown below:

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

```
using AutoMapper;

namespace Acme.BookStore.Web
{
    public class BookStoreWebAutoMapperProfile : Profil
    {
        public BookStoreWebAutoMapperProfile()
        {
            CreateMap<BookDto, CreateUpdateBookDto>();
        }
    }
}
```

- We have just added `CreateMap<BookDto, CreateUpdateBookDto>();` to define this mapping.

> Notice that we do the mapping definition in the web layer as a best practice since it is only needed in this layer.

EditModal.cshtml

Replace `EditModal.cshtml` content with the following content:

```
@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-input asp-for="Id" />
            <abp-form-content />
        </abp-modal-body>
        <abp-modal-footer buttons="@(AbpModalButtons.Ca
    </abp-modal>
</abp-dynamic-form>
```

This page is very similar to the `CreateModal.cshtml`, except:

- It includes an `abp-input` for the `Id` property to store `Id` of the editing book (which is a hidden input).
- It uses `Books/EditModal` as the post URL.

Add "Actions" Dropdown to the Table

We will add a dropdown button to the table named *Actions*.

Open the `Pages/Books/Index.js` and replace the content as below:

Share on :

In this document

```
$(function () {
    var l = abp.localization.getResource('BookStore');
    var createModal = new abp.ModalManager(abp.appPath
    var editModal = new abp.ModalManager(abp.appPath +

    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('Actions'),
                    rowAction: {
                        items:
                            [
                                {
                                    text: l('Edit'),
                                    action: function (d
                                        editModal.open(
                                    }
                                }
                            ]
                    }
                },
                {
                    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
```

Share on :

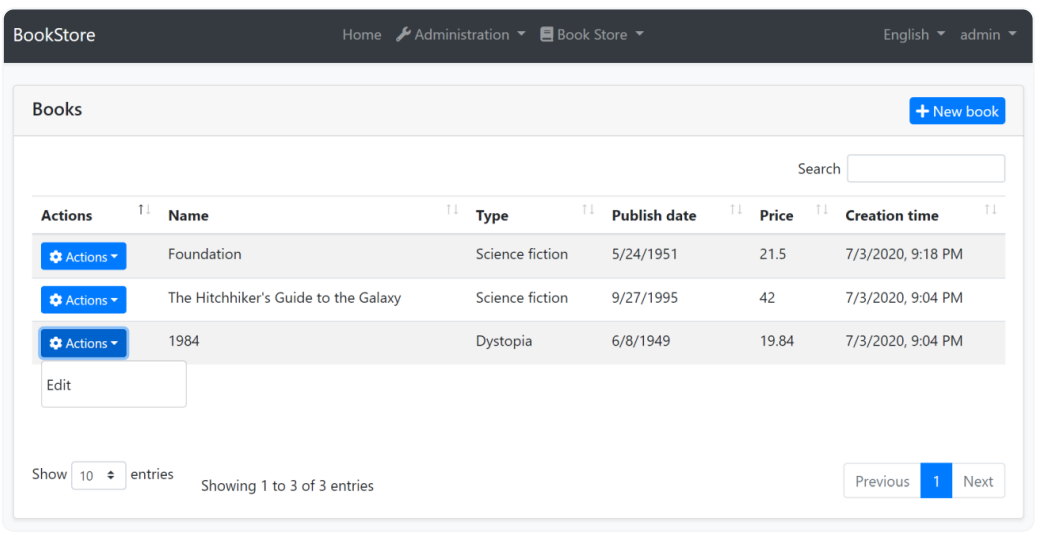
In this document

```
                        }).toLocaleString(luxon.Dat
                    }
                }
            ]
        })
    );

    createModal.onResult(function () {
        dataTable.ajax.reload();
    });

    editModal.onResult(function () {
        dataTable.ajax.reload();
    });

    $('#NewBookButton').click(function (e) {
        e.preventDefault();
        createModal.open();
    });
});
```


- Added a new `ModalManager` named `editModal` to open the edit modal dialog.
- Added a new column at the beginning of the `columnDefs` section. This column is used for the "*Actions*" dropdown button.
- "*Edit*" action simply calls `editModal.open()` to open the edit dialog.
- `editModal.onResult(...)` callback refreshes the data table when you close the edit modal.

You can run the application and edit any book by selecting the edit action on a book.

The final UI looks as below:



Deleting a Book

Open the `Pages/Books/Index.js` and add a new item to the `rowAction items`:

```
{
    text: l('Delete'),
    confirmMessage: function (data) {
        return l('BookDeletionConfirmationMessage', dat
    },
    action: function (data) {
        acme.bookStore.books.book
            .delete(data.record.id)
            .then(function() {
                abp.notify.info(l('SuccessfullyDeleted'
                dataTable.ajax.reload();
            });
    }
}
```

- `confirmMessage` option is used to ask a confirmation question before executing the `action`.
- `acme.bookStore.books.book.delete(...)` method makes an AJAX request to the server to delete a book.
- `abp.notify.info()` shows a notification after the delete operation.

Since we've used two new localization texts (`BookDeletionConfirmationMessage` and `SuccessfullyDeleted`) you need to add these to the localization file (`en.json` under the `Localization/BookStore` folder of the `Acme.BookStore.Domain.Shared` project):

```
"BookDeletionConfirmationMessage": "Are you sure to del
"SuccessfullyDeleted": "Successfully deleted!"
```

The final `Index.js` content is shown below:

In this document

```
$(function () {
    var l = abp.localization.getResource('BookStore');
    var createModal = new abp.ModalManager(abp.appPath
    var editModal = new abp.ModalManager(abp.appPath +

    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('Actions'),
                    rowAction: {
                        items:
                            [
                                {
                                    text: l('Edit'),
                                    action: function (d
                                        editModal.open(
                                    }
                                },
                                {
                                    text: l('Delete'),
                                    confirmMessage: fun
                                        return l(
                                            'BookDeleti
                                            data.record
                                        );
                                    },
                                    action: function (d
                                        acme.bookStore.
                                            .delete(dat
                                            .then(funct
                                                abp.not
                                                    l('
                                                );
                                                dataTab
                                            });
                                    }
                                }
                            ]
                    }
                },
                {
                    title: l('Name'),
                    data: "name"
                },
                {
                    title: l('Type'),
                    data: "type",
                    render: function (data) {
                        return l('Enum:BookType:' + dat
                    }
                },
                {
                    title: l('PublishDate'),
                    data: "publishDate",
```


In this document

```
            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
            }
                    }
                ]
            })
    );

    createModal.onResult(function () {
        dataTable.ajax.reload();
    });

    editModal.onResult(function () {
        dataTable.ajax.reload();
    });

    $('#NewBookButton').click(function (e) {
        e.preventDefault();
        createModal.open();
    });
});
```

You can run the application and try to delete a book.

The Next Part

See the next part of this tutorial.

In this document