Adding Data From a JSON File To Our Web Application

Adding Data To Our Web Application (From JSON File)

We can use an existing JSON file that contains data for our application.
Create "data" subfolder inside wwwroot and place the json file inside it.

```
"author": "Chinua Achebe",
"country": "Nigeria",
"imageLink": "images/things-fall-apart.jpg",
"language": "English",
"link": "https://en.wikipedia.org/wiki/Things_Fall_Apart\n",
"pages": 209,
"title": "Things Fall Apart",
"year": 1958
```

Create a Models folder inside our project.
Add a new class (this will be the Book class which will map to the data in JSON file).
Add → Class → Book (we typically use the singular for a class name).

Now we define the properties of the Book class (basing it on the data in the JSON file).

If we want to use a different property name to the JSON file we can use JsonPropertyName attribute (like a post-it note which tells us what the property is really called.)

```
using System.Text.Json.Serialization;
```

```
0 references
public class Book
{
    0 references
    public string Author { get; set; }

    0 references
    public string Country { get; set; }

    [JsonPropertyName("imageLink")]
    0 references
    public string Image { get; set; }

    0 references
    public string Language { get; set; }

    0 references
    public string Link { get; set; }

    0 references
    public int Pages { get; set; }

    0 references
    public string Title { get; set; }

    0 references
    public int Year { get; set; }

    0 references
    public int[] Ratings { get; set; }
}
```

We should always create our own ToString() method (like we do for any classes we create).

In this example we can use the JsonSerializer.
We need using System.Text.Json;

It takes the data from the object and converts it into the text that will be in the JSON file.
Takes the books one after the other (with their properties) and puts them into a string.

```
public override string ToString()
{
    JsonSerializer.Serialize<Book>(this);
}
```

Because this method has a single line of code, we can change this to shorthand.

```
public override string ToString() => JsonSerializer.Serialize<Book>(this);
```

Create A Service (Like A Waiter)

We should use a service to display the data. Think of a service like a waiter in a restaurant. You ask the waiter for what you want, he goes and does it and brings it back to you.

Create a Services folder inside our project.
Add a new class (this will be the Book class which will map to the data in JSON file).
Add → Class → JsonBookService

```
0 references
public JsonBookService(IWebHostEnvironment webHostEnvironment)
{
    WebHostEnvironment = webHostEnvironment;
}

2 references
public IWebHostEnvironment WebHostEnvironment { get; }
```

Web applications live in a host (they are hosted).

Our constructor is basically getting a host environment (and keeping it as a property).

```
1 reference
private string JsonFileName
{
    get { return Path.Combine(WebHostEnvironment.WebRootPath, "data", "books.json"); }
}
```

When we want to access the JSON file, we don't want to hardcode the folder/path structure. This property combines the web root from the host, with the subfolder and the JSON file.

```
public IEnumerable<Book> GetBooks()
{
    using (var jsonFileReader = File.OpenText(JsonFileName))
    {
        return JsonSerializer.Deserialize<Book[]>(jsonFileReader.ReadToEnd(),
            new JsonSerializerOptions
            {
                PropertyNameCaseInsensitive = true
            });
    }
}
```

We have retried the JSON file and have to convert that text into the book objects we defined. Here we are deserialising (taking the JSON and converting it into the Book object shape).

Making an array of books.
Read the JSON file to the end (so we get all books),
Pass optional stuff (ignore the case).
IEnumerable is the great great grandfather of lists.
Things that you can foreach over (are things you can numerate).

We now have a service that has one job – to give us a "list" of books.

Telling The Application About The Service

We have a waiter, but we have to tell the restaurant that we have a waiter.
Otherwise the restaurant won't know he exists, and won't use him.

So we need to go to Startup.cs and write some code there to tell the application that we have a service it can use. It's mostly boilerplate code

Remember, an ASP.NET Web Application is still a console application.
e.g. Program.cs has a main (like all our console apps) but it is a main that makes a host.
And Program.cs says that we should use Startup clas.

A special file that has some methods.

The Startup class has a ConfigureServices method which tells the application about the services. We need to tell the application about our JsonBookService.

We can choose from Singleton (only one) or Transient (comes and goes as needed).

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddRazorPages();
    services.AddTransient<JsonBookService>();
}
```

We created the service in a different namespace so we need to add a using statement.

Now the service we made is a member of this collection of services (an advertisement).

Display The Data on Our Web Application

Index.cshtml is a Razor page.
It all runs server-side and allows you to write C# and HTML together.
Index.cshtml has a code-behind friend (this is the page model). The friend is Index.cshtml.cs

We can look at this file and see stuff, e.g. a logger model.
And we can see the constructor with some stuff as an argument.
Logging is a service. Log this to a file, or anywhere. You ask for one in the constructor.

And we want to add the JsonBookService service to this page.
So we can add this to the constructor (add the using) and give it a name.
We can basically add as many services as we need in this page.
And we already created our own service to get data from the Json file.
We told the application about the service in Startup.cs and now any page can use it.
We do not clutter the code in our page (makes it easier to reuse code).

```
public class IndexModel : PageModel
{
    private readonly ILogger<IndexModel> _logger;
    public JsonBookService BookService;
    2 references
    public IEnumerable<Book> Books { get; private set; }

    0 references
    public IndexModel(ILogger<IndexModel> logger, JsonBookService bookService)
    {
        _logger = logger;
        BookService = bookService;
    }

    0 references
    public void OnGet()
    {
        Books = BookService.GetBooks();
    }
}
```

Index.cshtml.cs

We add the JsonBookService object into our constructer (and give it a name).
Declaring to ASP.Net "I need some stuff now go and get it"

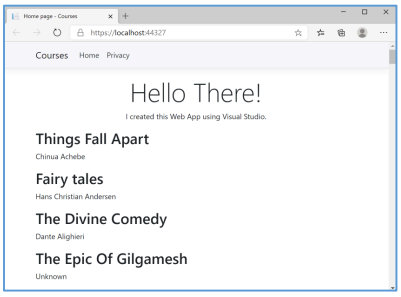
Like we take the logger and save it in a variable, we must do this with the JsonBookService.

Our page should display a list of books (we already created a method for this in our service).
So we must ensure the model contains a list of books. Private set (only this class can set it).
Razor pages have special OnGet() - what do we want it to do when someone gets the page?
We want it to get the list of books. So we use the GetBooks() method of the BookService.

So now the Index page knows about Books.
So we go to Index.cshtml we can just do a foreach book and list the title and author.

We combine HTML and C# in our Razor page to do this.

```
@foreach (var book in Model.Books)
{
    <h2>@book.Title</h2>
    <p>@book.Author</p>
}
```

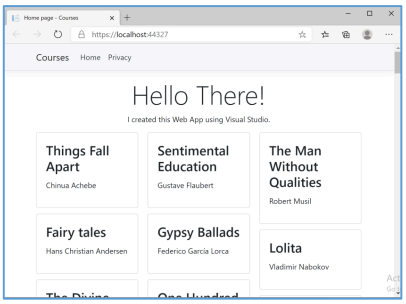


Styling Our Razor Pages

Displaying the data on our page is not enough – now we need to make the page look good.

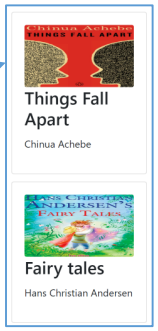
We can use Bootstrap classes (as this is already included).
So we need to make sure that we understand Boostrap!

```
<div class="card-columns">
    @foreach (var book in Model.Books)
    {
        <div class="card">
            <div class="card-body">
                <h3 class="card-title">@book.Title</h3>
                <p>@book.Author</p>
            </div>

        </div>
    }
</div>
```



Now I want to add an image.
I'll put my images in an images subfolder inside wwwroot (as my JSON file says this).

```
<div class="card-columns">
    @foreach (var book in Model.Books)
    {
        <div class="card">
            <div class="card-body">
                <img class="card-img" src="@book.Image" width="100" height="100" alt="@book.Title" />
                <h3 class="card-title">@book.Title</h3>
                <p>@book.Author</p>
            </div>

        </div>
    }
</div>
```



Important Note:

The last two pages of these notes shows how to display the data from the JSON file on the Index page, but you should try to create your own page to display it.

A new page displaying the data is a requirement of your Lab Exercise.