Book Search Documentation

Client-side Homework

Woodroffe, Jena
FNA1A3
13 May 2024

Table of Contents

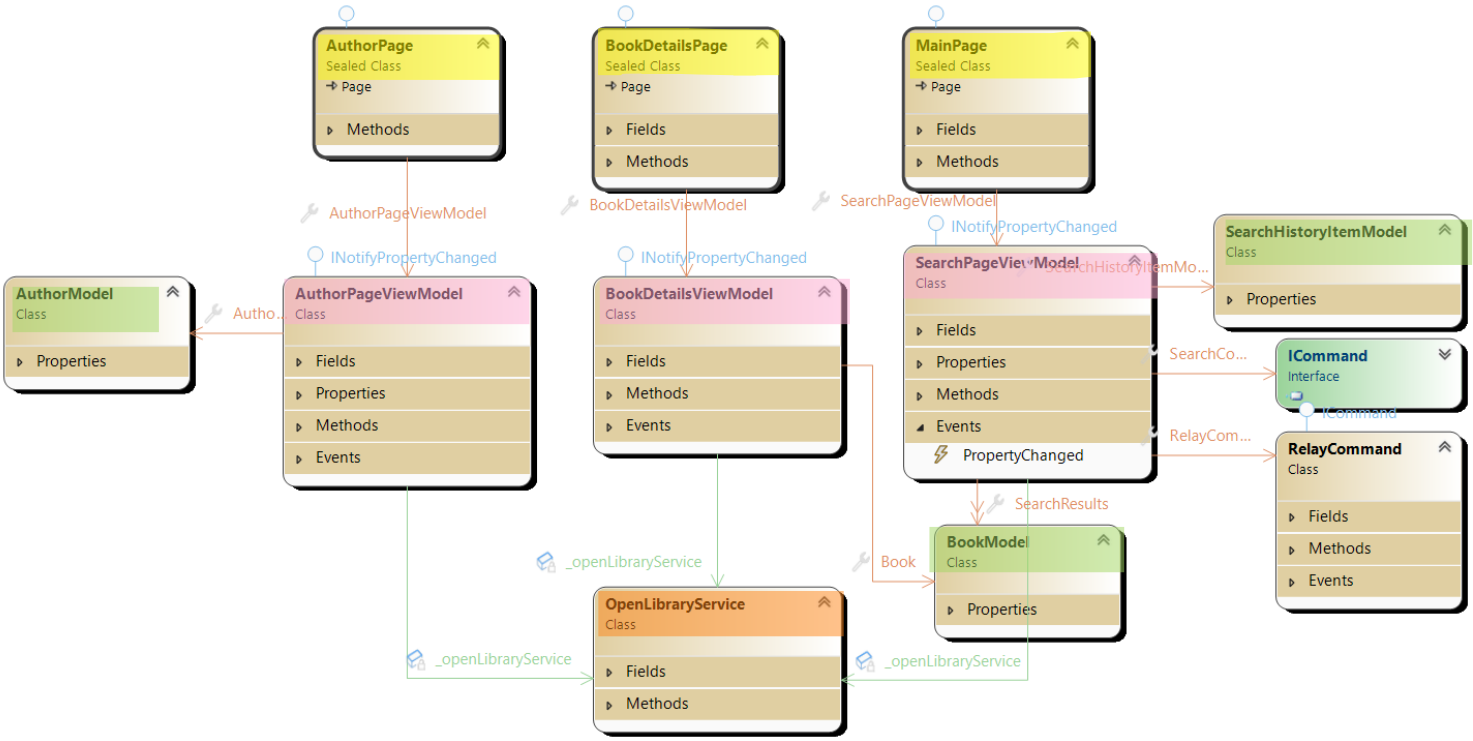
Introduction

The Book Search application is designed to search for books using the Open Library API. The users can search by Title, Author, Subject or with no selected criteria and will be given the matching results retrieved from the API. Clicking on a book from the search results will open a new window with more information about the book – including publishing date, description, and author's name. The author's name can be clicked to view more details about the author (such as their date of birth, website link and their biography if they have one). The application also makes use of isolated storage to store the user's search history. The search history is then used to reload previous results when returning to the Search Page. All the pages can be easily navigated between using back buttons.

Architecture

This application is created using the MVVM design pattern and the main components are separated into Views (yellow), ViewModels (pink), Models (green) and Services (orange).

Figure:



Main Components:

The Views (SearchPage, BookPage and AuthorPage) display the UI and trigger events in the ViewModels (all views are data-bound to their ViewModel). The ViewModels invokes methods in the OpenLibraryService class to send API requests. The Views are then updated based on the ViewModel changes.

Class Design

Views

1. SearchPage
   This is the loading page of the application that allows the user to enter their text and criteria and then search for books accordingly. The books are displayed in a list with their cover image, name and author and can be clicked on for more information.

2. BookDetailsPage
   This page is opened when a user selects a book from the SearchPage. More data is requested and then displayed about the book (such as summary and publication date). The author's name is also clickable for more information.

3. AuthorPage
   Clicking on the author's name will lead you to this page. Here, more information is displayed about the author (such as providing their website link and biography), depending on what is available.

ViewModels

1. SearchPageViewModel
   This class manages and controls the SearchPage. When the user clicks the search button, this class will call the methods of OpenLibraryService to get the books, and the view will be populated with them.

2. BookDetailViewModel
   This class manages and controls the BookDetailsPage. When a user clicks on a book in SearchPage, the respective BookModel is passed to this class, whereby it will use OpenLibraryService to extract more information about the book and then display it.

3. AuthorPageViewModel
   This class manages and controls the AuthorPage. When an author's name is clicked, the AuthorKey is passed to this class and then given to OpenLibraryService to retrieve more information about the author and populate the page with the data.

Models

1. BookModel
   Defines the format for Books that is used both in SearchPageViewModel to hold an observable list of Books and then by BookDetailsViewModel to get relevant information about a selected book.

2. AuthorModel

Used in AuthorPageViewModel to store data about an author and then display that in the AuthorPage.

3. SearchHistoryItemModel
   Used in SearchPageViewModel to store the information about a search (the search text, criteria and search time) and write it to local storage.

Services

OpenLibraryService

Responsible for all of the API communication. Executes three main queries: 1) query books based on a user search 2) query a select book's information and 3) querying information about an author. Returns the data back to the ViewModel in the correct Model format.

API logic

All the ViewModels in this project rely on the ***OpenLibraryService*** class to facilitate communication between the UWP app and the Open Library API. The service is able to search for books, retrieve book descriptions, and get author details.

- The service uses HttpClient class to send HTTP requests and receive HTTP responses from the Open Library API

- Search URLs are formatted depending on the method and then searched using the HttpClient

```
searchUrl =
$"https://openlibrary.org/search.json?q={Uri.EscapeDataString(searchText)}";
HttpResponseMessage response = await _httpClient.GetAsync(searchUrl);
```

- The response's content is then read and converted to a string consisting of data in Json format

```
string responseJson = await response.Content.ReadAsStringAsync();
```

- The Json response is then deserialised into dynamic objects using Newtonsoft.Json.JsonConvert

```
dynamic searchData = JsonConvert.DeserializeObject<dynamic>(responseJson);
```

- The necessary information is then extracted from these dynamic objects and either assigned to Models or necessary properties.

```
var docs = searchData.docs;
foreach( var doc in docs ) {
searchResults.Add(new BookModel
{
          Title = doc.title, ...
```

API call example

Below is a detailed description of how I query the data of an author when the user wishes to open their author page. The ViewModel receives the “author key” on initialisation, which is handed to the OpenLibraryService class to query the data and return it as an AuthorModel.

Step 1: The “author key” field is updated in the View, triggering FetchAuthorData() in the AuthorPageViewModel. The author key is passed to OpenLibraryService as follows:

```
AuthorModel author = await _openLibraryService.GetAuthorDetailsAsync(_authorKey);
```

Step 2: Inside GetAuthorDetailsAsync() – create the API URL according to the Open Library API’s documentation

```
string apiUrl = $"https://openlibrary.org/authors/{authorKey}.json";
```

Step 3: Send the HTTP GET request

```
HttpResponseMessage response = await client.GetAsync(apiUrl);
```

Step 4: Handling the response – check if the response.IsSuccessSatusCode() and continues if true

Step 5: Reading and deserialising Json into a JObject representing the author’s details:

```
string jsonResponse = await response.Content.ReadAsStringAsync();
JObject authorObject = JObject.Parse(jsonResponse);
```

Step 6: Extracting author details from the JObject. Some information is simple to get, such as Name, Birthdate and Bio, but many of the author Json responses contain different fields, so checks are done to see if this author’s information contains a “links” tag and if so, to extract the URL from that potential array. All of the information extracted is then added to an AuthorModel object and returned to the AuthorPageViewModel.

```
...
author.Name = (string)authorObject["name"];
author.BirthDate = (string)authorObject["birth_date"];
author.Bio = (string)authorObject["bio"];

return author;
```

Step 7: AuthorPageViewModel displaying new data:

```
if (author!= null)
{
	AuthorName = author.Name;
	BirthDate = author.BirthDate;
	Bio = author.Bio;
	AuthorUrl = author.Url;
	Debug.Write("url in vm: " + AuthorUrl);
```

```
}
else
{
       Bio = "Couldn't retrieve author's information.";
}
```

Since each of the fields listed above are data-bound to either textboxes/hyperlinks in the View, upon updating the fields in the ViewModel, the author's data will be updated in the View.