CS290.02
Midterm Take-Home Project

Important 1: This is not a team exercise. You should work on this mid-term individually**.**
Important 2: This is due Thursday, February 24 at 5pm. This way you will still be able to go out Thursday night, while also still attending lecture!
Important 3: The frustration of a compiled program is that if it's not compiling, you cannot get anything done. The point of this class it to teach you a thing or two, not to give you a 0 if you couldn't get something working because you did not understand the error. *Please start this project early enough* so that if you hit a major roadblock, you can reach out to me on the Ed Forum or jrp96@cs.duke.edu. I will not be staying up until midnight the night before just in case you have questions. :-)

Deep Breath. OK.

Let's make an app that allows you to track your reading list from a pre-populated list of books. To help you decide which books to add to your list, you'll need to fetch the summary for each book from OpenLibrary with an API call.

I've got a starter project for you that contains:
- A TabContainer that lets you switch back and forth between the BookList and the ReadingList. Those are empty views right now.
- All the model data you need. The Book model exists along with the books data. And the OpenLibraryResponse has been modeled for you as well. All that's going to come back is a book's summary.
- The APIClient and the RestAPIClient.
- The DataStore. Yep, I'm even providing that.

There is a brief video of the app included in the midterm project folder so you can see what you'll be building. It's a little annoying because the taps are subtle, but you'll get the idea. Someday I'll get better screen recording software.

Functionality
You have two lists, a list of books and a reading list. When you tap on a book, you can see detail about that book: the book cover, title, author, and the summary fetched from the API. There is also a button in the upper right corner. If the book is not on the reading list, it says "Add". If the book is on the reading list, it says "Remove". And, clearly, that button adds and removes books on the reading list. You can view the reading list by tapping on its tab button. The books on the reading list are not click-able, but you can swipe them to remove them from the reading list.

Book Model
The book model includes an id, title, the author's name, and a coverUrl for the image. You'll want to have the app fetch the cover image and display it; that url is not for you to somehow manually download it and include it in the app assets.

API
OpenLibrary documentation is here:
https://openlibrary.org/dev/docs/api/books

You'll want to use the JSON version of the "Works" API. Note that you can just test it out in the browser to see it working. Again, I supplied the model for that call, so you won't have to decipher the actual JSON to get what you want. Just make sure you are seeing JSON when you provide the id, like https://openlibrary.org/works/OL45883W.json. You will need to construct the URL to make the call. To use the api response model I designed … to get the actual summary of the book … you'll want to reference like so: response.summaryContainer.summary.

Structure

The important thing to realize is that, in this app, the books are not changing. The only thing that is changing is whether they are on the reading list or not.

I'm betting you'll end up with the following files that have the following responsibilities:

DataStore - This holds and publishes the pre-supplied Books as well as the ReadingList. It will also be the DataStore's job to actually add and remove books on the readingList. I don't think you need to do anything to the provided file.

BookList - Displays the list of books! And navigates to a detail view.

BookListVM - Subscribes to the dataStore's list of books and provides them to the view. I agree that the book list is not changing, but it's still the dataStore's responsibility to provide it.

ReadingList - Displays the current reading list. Also has a delete swipe action to facilitate remove items from the reading list.

ReadingListVM - Subscribes to the dataStore's readingList and provides those items to the view

BookDetail - Displays the book details. It also has a toolbar with a button that lets you Add and Remove a book from the readingList. And it tells the viewModel to go get the book summary and displays an error if there was an API error. This view is a little different than what you've done with an API call in that only a portion of the page is populated with what you're getting from the API. Make sure to show a ProgressView() for the summary until it is loaded.

BookDetailVM - Hmmm … this one has a little complexity. The BookDetailVM needs to handle both fetching the OpenLibrary summary of the book AND getting the reading list from the datastore. Since the Book Detail View is going to have that button to add and remove the book from the readingList, the BookDetailVM needs to always have the most recent version of the readingList from the DataStore. So:

- You probably need to subscribe to the reading list
- You probably need to support making the API call
- You'll need to tell the dataStore to add or remove a book from the reading list
- Here's the kicker: how are you going to publish a variable that tells the view whether or not the book is currently on the reading list? The viewModel needs to tell the view the exact status of whether the book is on the reading list so that the view will redraw when that status changes. In other words, we want the Add/Remove button text to update when the book is added or removed, so we need to have something published from the viewModel that drives that. I would suggest a Bool that indicates whether or not the book is on the current readingList. You'd probably want to set that Bool every time you get an updated readingList from the dataStore. Since you are subscribing to the reading list, when the reading list changes, the store will tell you. You'll then want figure out if your book is on the

current version of the reading list and update that boolean. To spell it out for you further, you want to do that figuring out “inside” the “sink” closure.

OpenLibraryEndpoint - This will construct the URL to make the API call

OpenLibraryAPIService - This will actually make the API call.

Other things to not overlook

- I’ve included NavigationView wrappers around the list views in the TabContainer. That way you don’t forget them, but also, you won’t want to include them again. We only need them once, there. I also included them in the Preview where necessary
- The BookDetailVM is going to be making those API calls on a background thread. You probably want to have some property wrapper in that viewModel that tells it to make non async functions run on the main thread.

So, what’s hard here? Managing the add/remove reading list functionality and the API call functionality. If it were me, I wouldn’t try to do everything at once. I’d get one of those two things working first, and then the other one. And then I’d make the views look nice. And along the way I’d make sure my code was sensibly organized and presented.

Grading Rubric

The mid-term is worth 20% of your grade. This grade is determined by a 20 point scale as follows:

BASICS
Appropriately designed book rows and book detail: 3 points
Neat and sensible organization and presentation of code: 3 points.

DATASTORE
Populating the list views from the DataStore properly: 4 points

API
Designing and making the API call properly: 2 points
Display the results of the API call and handling loading states: 2 points
Display API error for “Book with Bad Id”: 2 points

READING LIST FUNCTIONALITY
BookDetail view properly handles the changing of reading list state when the button is tapped: 4 points