**Digital Library Wall**

App Connects to the API to retrieve the list of books for the topic searched and then displays them in a decorative Bookshelf format.

**Steps followed in the project:**

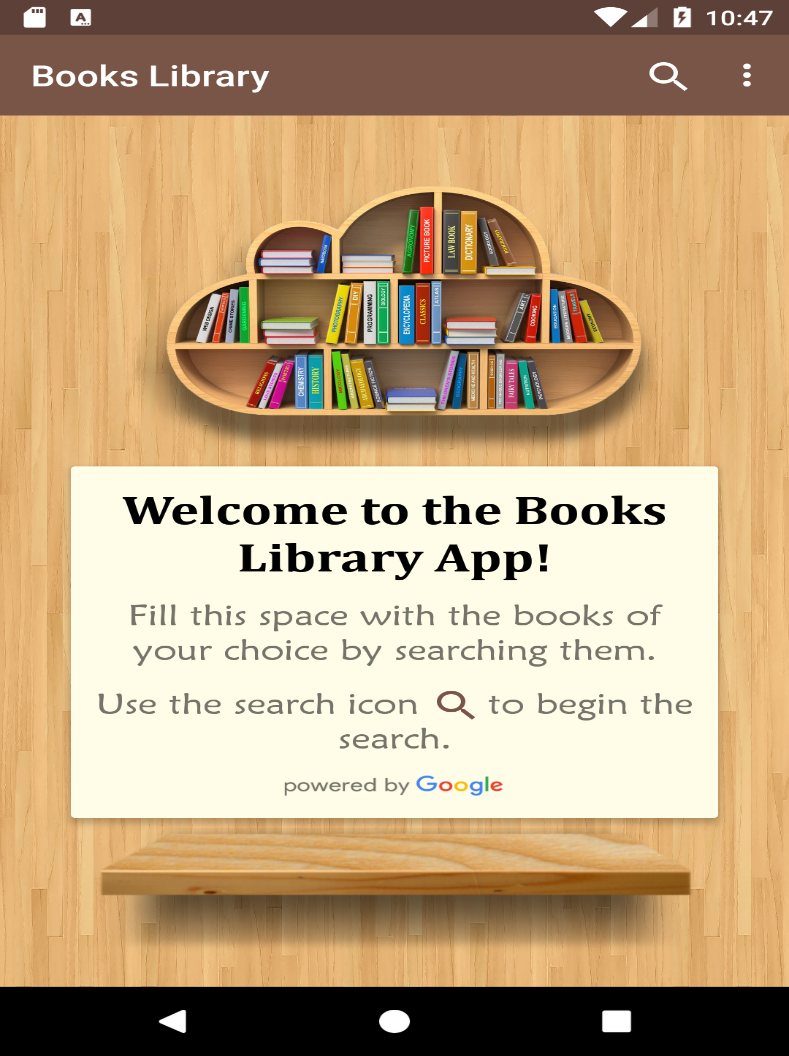
* Contains a List View that is populated with list items having information displayed with proper text wrapping.
* List items displays the author and title information.
* ListViews to be populated with properly parsed information, from the JSON Response.
* For the user to search the desired book, the screen has a search box (SearchView) that accepts a word or phrase.
* App should fetch the query related books via an HTTP request from Google Books API, using a class such as HttpUriRequest or HttpURLConnection.
* All network calls should occur off the UI Thread by making use of AsyncTask or similar threading object.
* On device rotation
* The layout remains scrollable.
* The app saves state and restores the list back to the previously scrolled position.
* The UI properly adjusts so that all contents of each list item is still visible and not truncated/overlapped.
* The Search button remains visible on the screen even after the device is rotated.
* Checks whether connected to internet or not. Also, validates for any occurrence of bad server response or lack of response.
* No external libraries to be used.
* A Default message is shown when there is no data to display for the search query made.
* A Default text is shown for guiding the user, on what to enter while searching.
* Assisted Search Implementation with SearchView.
* Used RecyclerView in place of ListView and GridView for its advantages in performance and easy placeholders for custom item decoration.
* Custom [RecyclerView.ItemDecoration](https://github.com/kaushiknsanji/Books_Library_App/blob/0c15b06877ca29523a588b67f30431f4acfaed37/app/src/main/java/com/example/kaushiknsanji/bookslibrary/adapterviews/RecyclerViewFragment.java#L362-L445) for decorating each of the items in List/Grid with the Book shelf decoration.
* Explored [FragmentStatePagerAdapter](https://github.com/kaushiknsanji/Books_Library_App/blob/udacity/app/src/main/java/com/example/kaushiknsanji/bookslibrary/adapters/DisplayPagerAdapter.java) that displays the Fragments (retaining their state) for the ViewPager.
* Implemented android.support.v7.preference.Preference Preferences for the Settings.
* No external libraries are used for communicating with the REST API and for loading the images. AsyncTaskLoader has been used for downloading the data and images in the background thread. Images are downloaded using a Headless [Fragment](https://github.com/kaushiknsanji/Books_Library_App/blob/udacity/app/src/main/java/com/example/kaushiknsanji/bookslibrary/workers/ImageDownloader.java).
* Developed [BitmapImageCache](https://github.com/kaushiknsanji/Books_Library_App/blob/udacity/app/src/main/java/com/example/kaushiknsanji/bookslibrary/cache/BitmapImageCache.java) utility that uses android.util.LruCache to cache the recent Bitmap Images downloaded.
* Most layouts are designed using ConstraintLayout to flatten the layout hierarchy as far as possible.
* Indeterminate progress bar is implemented with animation-list / AnimationDrawable.
* [TextAppearanceUtility](https://github.com/kaushiknsanji/Books_Library_App/blob/udacity/app/src/main/java/com/example/kaushiknsanji/bookslibrary/utils/TextAppearanceUtility.java) for decorating TextViews using Spannables, for strikethrough, image within text, selective text coloring and relative text resize.
* Custom Fonts for TextViews.
* CardView for displaying the information of a Book.

**Design and Implementation of the App:**

**Welcome Screen Layout:**

The first screen displayed when the app is launched, is the welcome page screen.

This layout is inflated by the BookSearchActivity and is included in the activity\_main.xml. This is visible only for the first time when the app is launched and loaded into memory. The Screen basically tells the user on how to search for a book.

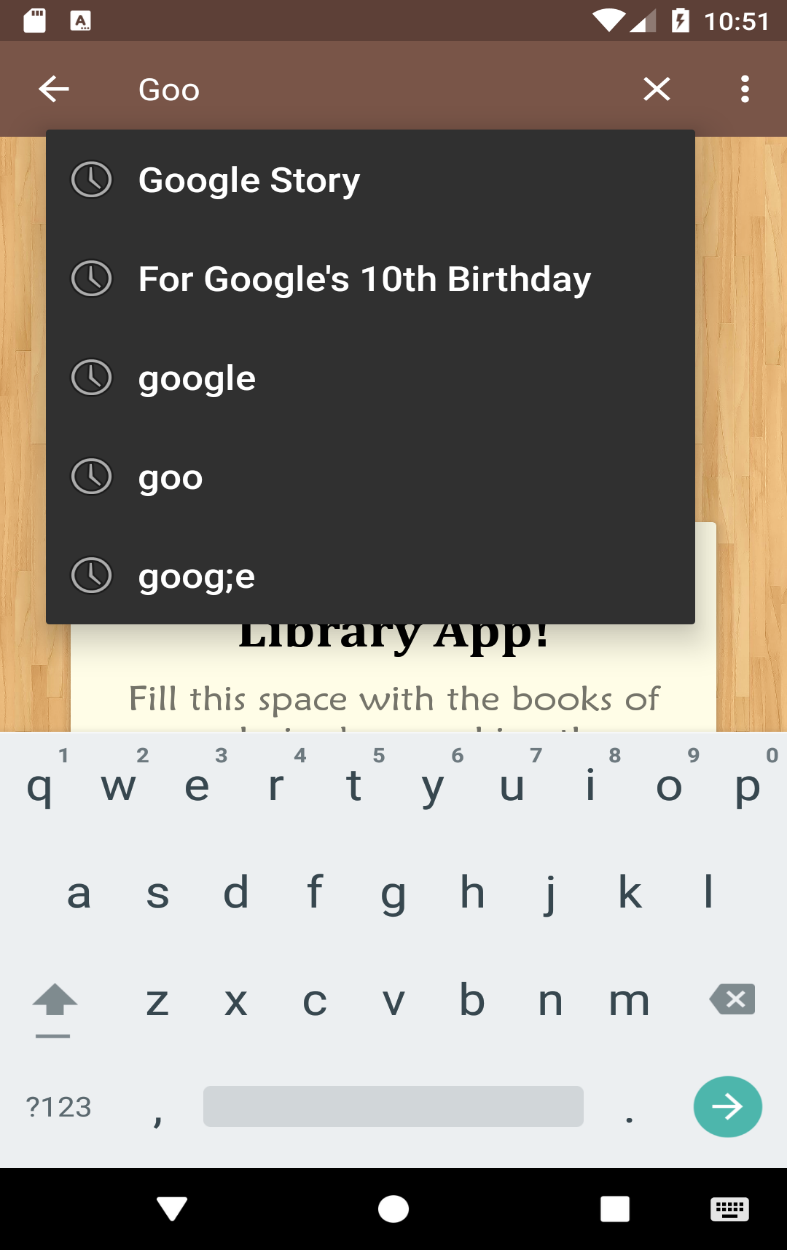


**Assisted Search:**

The SearchView in the action menu is implemented with assisted search. Hence this activity also becomes the searchable activity with the searchable activity with the searchable configuration as defined. As seen in the Searchable configuration, a Recent Search Suggestions Provider is also implemented and is displayed when the user types in atleast 3 character in the search to show the corresponding match, provided if there were any recent search with those 3 characters

When the SearchView is expanded, user can opt to search by Voice as well. But this will not allow the user to make any modifications as the transcribed text is directly fed to the search box and executed in one shot. Any search that is done, is updated to the title.

One can clear the recent search history by simply going to the overflow menu at the top and select the menu option that says, “Clear Search History”. On the Success of the action, a toast will be displayed.



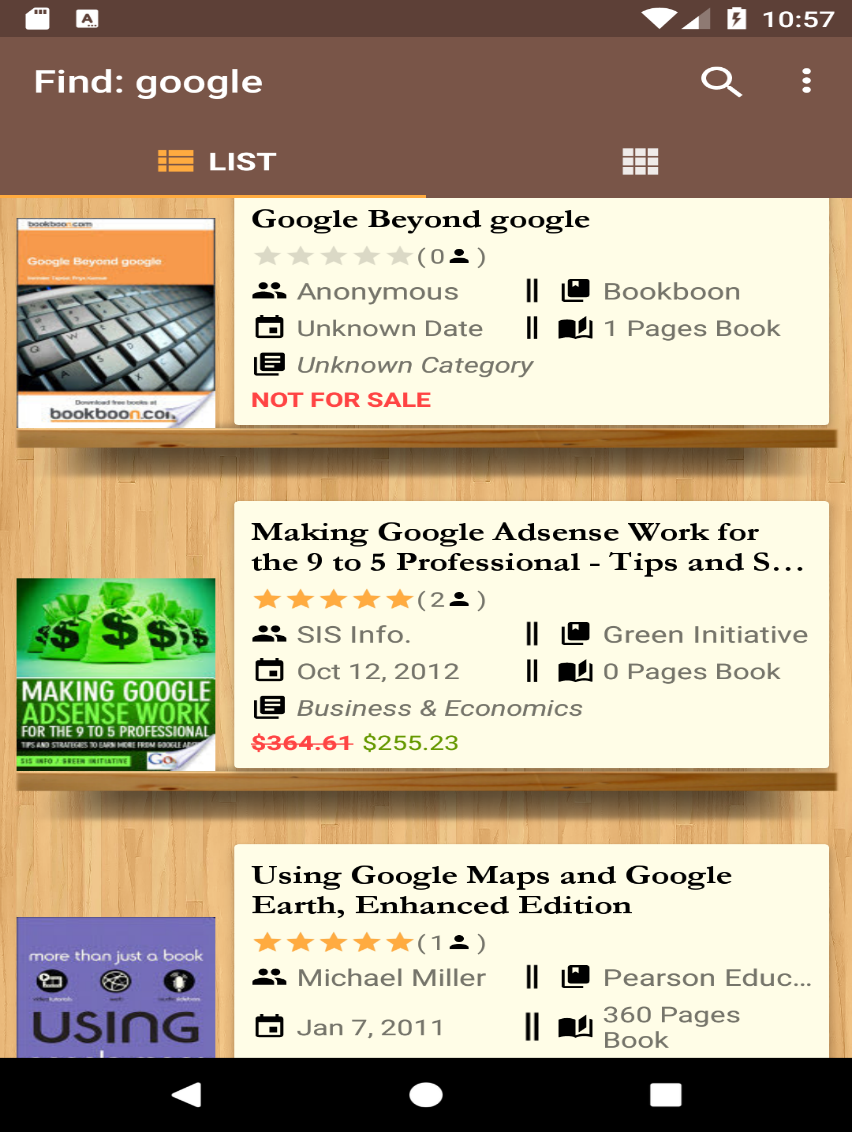
**Displaying Results:**

Once the Search is entered by the user, the welcome screen if active is replaced with a ViewPager and its Fragments to display the results. The ViewPager will be set to show its first tab by default, which is LIST Tab. In this the results, are displayed like in the vertical list with list of books related to the search. The other tab is the GRID Tab which allows the user to view the results in a Staggered Grid View (in 2 Columns).

Adapter used by the PageViewer is an extension of FragmentStatePagerAdapter whose fragments for list & grid view is given by the RecyclerViewFragment, which uses RecyclerView instead of ListView or GridView.

For List View:

The RecyclerView uses a LinearLayoutManager as its layout manager and an adapter [RecyclerListAdapter](https://github.com/kaushiknsanji/Books_Library_App/blob/udacity/app/src/main/java/com/example/kaushiknsanji/bookslibrary/adapters/RecyclerListAdapter.java) to populate the List.



For Grid View:

The RecyclerView uses a StaggeredGridLayoutManager as its layout manager and an adapter [RecyclerGridAdapter](https://github.com/kaushiknsanji/Books_Library_App/blob/udacity/app/src/main/java/com/example/kaushiknsanji/bookslibrary/adapters/RecyclerGridAdapter.java) to populate the Grid.



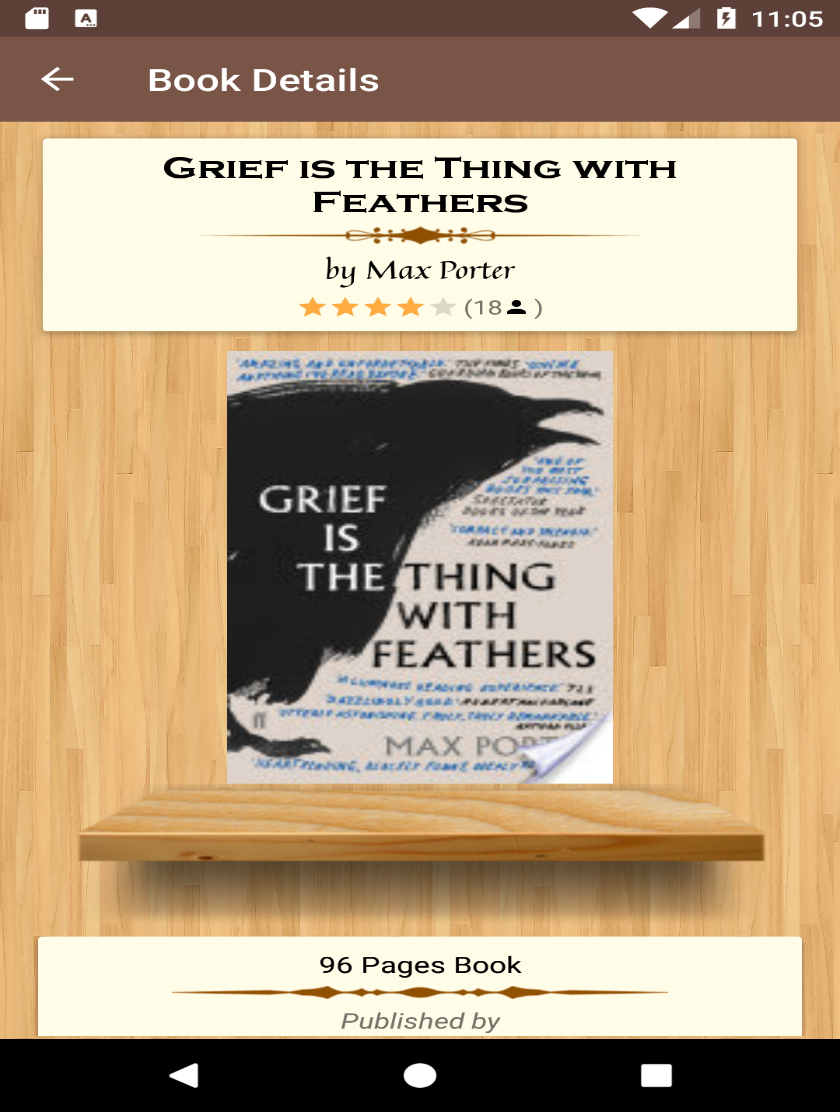
For both of the layout managers used, an item decoration is added defined by the [BookShelfItemDecoration](https://github.com/kaushiknsanji/Books_Library_App/blob/0c15b06877ca29523a588b67f30431f4acfaed37/app/src/main/java/com/example/kaushiknsanji/bookslibrary/adapterviews/RecyclerViewFragment.java#L362-L445). This basically adds a bookshelf decoration just below the item. Both the above adapters registers an item click listener whose interface callbacks are defined by [OnAdapterItemClickListener](https://github.com/kaushiknsanji/Books_Library_App/blob/udacity/app/src/main/java/com/example/kaushiknsanji/bookslibrary/observers/OnAdapterItemClickListener.java).

The RecyclerViewFragment registers an OnScrollListener to propagate the corresponding event to the BookSearchActivity. This is implemented by extending RecyclerView. OnScrollListener which is done by the class [BaseRecyclerViewScrollListener](https://github.com/kaushiknsanji/Books_Library_App/blob/udacity/app/src/main/java/com/example/kaushiknsanji/bookslibrary/observers/BaseRecyclerViewScrollListener.java). This is an abstract class which provides an event callback when the scroll reaches the last 3 items in the list, to reveal the pagination buttons for the user to navigate to different pages in the current search, as shown below. The RecyclerViewFragment registers a concrete implementation of this class which is [RecyclerViewScrollListener](https://github.com/kaushiknsanji/Books_Library_App/blob/0c15b06877ca29523a588b67f30431f4acfaed37/app/src/main/java/com/example/kaushiknsanji/bookslibrary/adapterviews/RecyclerViewFragment.java#L447-L466) to propagate the event to the BookSearchActivity, which reveals the hidden pagination panel (R.id.pagination\_panel\_id).

**The Details Page:**

Clicking on an item in the List Tab View or the Grid Tab View, opens up the Details page [activity\_book\_detail.xml](https://github.com/kaushiknsanji/Books_Library_App/blob/udacity/app/src/main/res/layout/activity_book_detail.xml) inflated by the activity [BookDetailActivity](https://github.com/kaushiknsanji/Books_Library_App/blob/udacity/app/src/main/java/com/example/kaushiknsanji/bookslibrary/BookDetailActivity.java). This displays additional information such as

* The book description
* Sample previews of the Book, that basically takes the user to a link via an Intent to the browser.
* Information link when no previews present and
* A button that takes the user to a page for buying a book if the book is saleable in the user's region.



**The Book Image Page:**

If the Image of the Book is loaded in the Details page activity\_book\_detail.xml, then the user can click on the image to open a larger image of the same, which is displayed by the activity [BookImageActivity.](https://github.com/kaushiknsanji/Books_Library_App/blob/udacity/app/src/main/java/com/example/kaushiknsanji/bookslibrary/BookImageActivity.java)

**Controlling the Search Results:**

Search Results can be controlled by varying certain parameters embedded in the search query. This is defined by the [Google Books API](https://developers.google.com/books/docs/v1/using#WorkingVolumes) which this application is based on and is acting as a client that receives the data for the search executed.

Most of these parameters can be altered by using the **"Search Settings"** menu item in the BookSearchActivity. The **"Search Settings"** menu item, opens up the preferences [preferences.xml](https://github.com/kaushiknsanji/Books_Library_App/blob/udacity/app/src/main/res/xml/preferences.xml) loaded by the activity [SearchSettingsActivity](https://github.com/kaushiknsanji/Books_Library_App/blob/udacity/app/src/main/java/com/example/kaushiknsanji/bookslibrary/settings/SearchSettingsActivity.java). These preferences provide options to override the search parameters used in the query, and the following are the ones that can changed -

* filter parameter which restricts the results returned to the kind of books we want like Partially viewable/Free eBooks/Paid eBooks and so on.
* printType parameter which restricts the results returned to specific print or publication type.
* orderBy parameter that defines the sorting order of the results.
* Pagination parameters like
* startIndex parameter that tells the page index to be shown.
* maxResults parameters that tells the max number of results to be shown per page.

The search settings implement the androidx. preference. Preference Preferences. As such all the values are stored in the default SharedPreferences. For the Pagination parameters, custom [NumberPicker](https://github.com/kaushiknsanji/Books_Library_App/blob/udacity/app/src/main/java/com/example/kaushiknsanji/bookslibrary/settings/NumberPickerPreference.java) Preference has been implemented by extending the DialogPreference.

There is also another preference setting provided, which resets all the preferences (used in the search) to their default values. This is implemented by [ConfirmationPreference](https://github.com/kaushiknsanji/Books_Library_App/blob/udacity/app/src/main/java/com/example/kaushiknsanji/bookslibrary/settings/ConfirmationPreference.java)

When the user returns to the BookSearchActivity after making the required changes in the **"Search Settings"**, the search query will be rebuilt, and the results will be loaded accordingly. For this to happen, BookSearchActivity implements the Listener SharedPreferences.OnSharedPreferenceChangeListener.

While searching, one can also filter the results by entering keyword prefixes into the search box, specifically to search only in that field of the output. Following are the parameters that cater to this category –

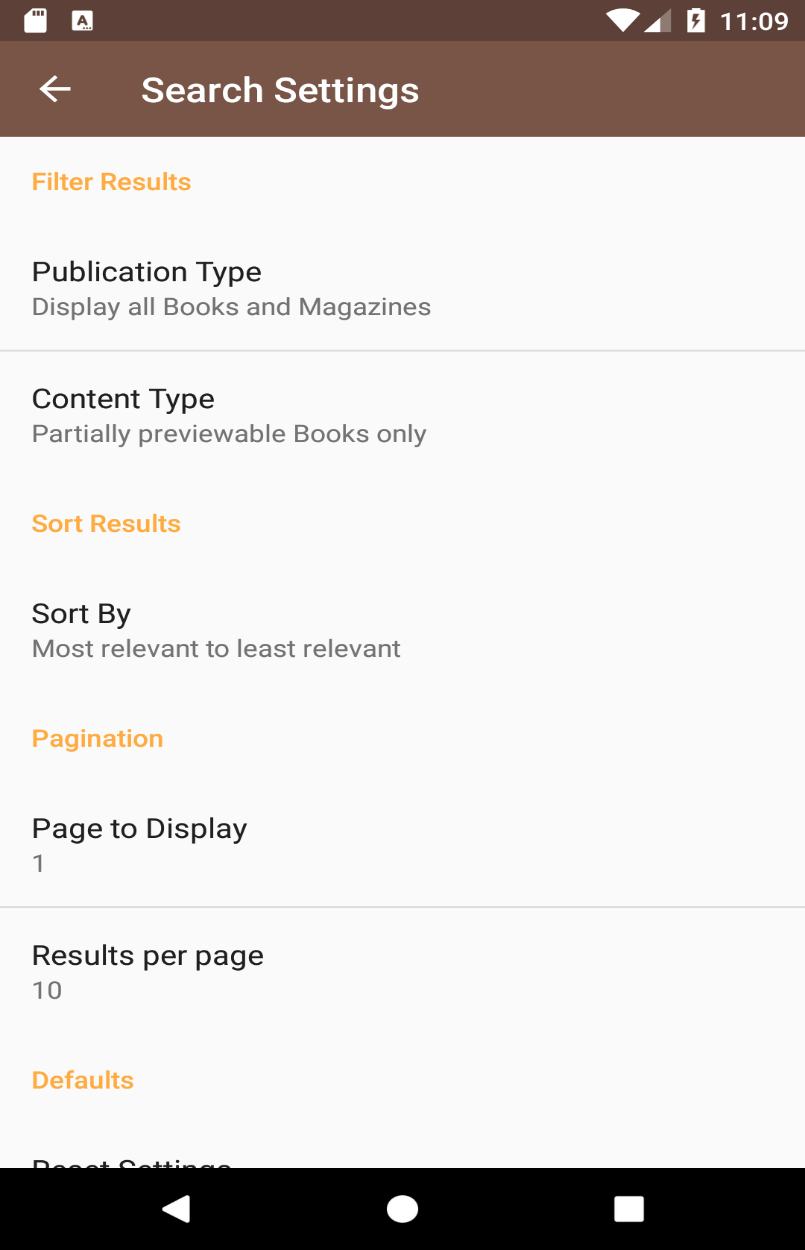
intitle - Returns results where the text following this keyword is found in the title.

inauthor - Returns results where the text following this keyword is found in the author.

inpublisher - Returns results where the text following this keyword is found in the publisher.

subject - Returns results where the text following this keyword is listed in the category list of the volume.

The above is **NOT** provided by the **"Search Settings"** menu item, instead it can be accessed by another menu item that says, **"Search Keyword Filters"**. This shows a dialog with a list of cards specific to each of the above mentioned prefixes, which is displayed by the DialogFragment [KeywordFiltersDialogFragment](https://github.com/kaushiknsanji/Books_Library_App/blob/udacity/app/src/main/java/com/example/kaushiknsanji/bookslibrary/adapterviews/KeywordFiltersDialogFragment.java). Selecting any of these cards, will insert the prefix into the search box. Any word/phrase typed after this, will be searched in that field of the results.



**Loading of Data:**

Loading of Data for the Book Search done is carried out in a background thread using [BooksLoader](https://github.com/kaushiknsanji/Books_Library_App/blob/udacity/app/src/main/java/com/example/kaushiknsanji/bookslibrary/workers/BooksLoader.java) that extends an AsyncTaskLoader. Request and response is carried out via a REST call to the Google Books API. The data format used for communication is the JSON format. Talking to the REST API and parsing the response is done by the utility code [BookClientUtility](https://github.com/kaushiknsanji/Books_Library_App/blob/udacity/app/src/main/java/com/example/kaushiknsanji/bookslibrary/utils/BookClientUtility.java).

**Loading of Images:**

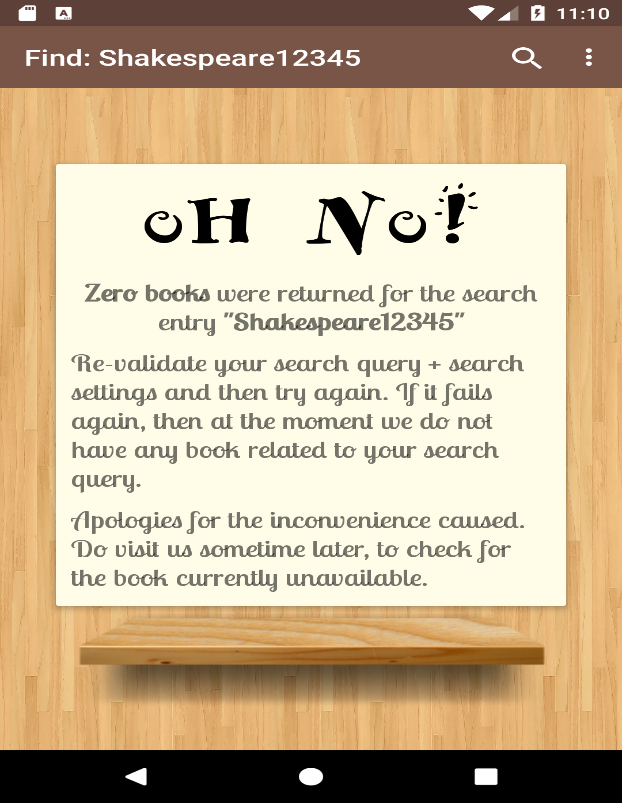
Loading of Images for each of the items in the list/grid views, in the Details page and the Book Image page is carried out in a background thread through a viewless Fragment [ImageDownloaderFragment](https://github.com/kaushiknsanji/Books_Library_App/blob/udacity/app/src/main/java/com/example/kaushiknsanji/bookslibrary/workers/ImageDownloaderFragment.java). Functioning of this fragment is as follows -

It first checks whether the image to be loaded is present in the Bitmap Cache, implemented by [BitmapImageCache](https://github.com/kaushiknsanji/Books_Library_App/blob/udacity/app/src/main/java/com/example/kaushiknsanji/bookslibrary/cache/BitmapImageCache.java)

If present in the cache, it updates the image to the corresponding ImageView passed.

If not present in cache, then the image is downloaded in a background thread using [ImageDownloader](https://github.com/kaushiknsanji/Books_Library_App/blob/udacity/app/src/main/java/com/example/kaushiknsanji/bookslibrary/workers/ImageDownloader.java) that extends an AsyncTaskLoader. Once successfully downloaded, it updates the image to the corresponding ImageView passed.

The identifier for the Fragment and its Loader is maintained to be unique to the item being displayed in the List/Grid View such that each item displays the correct image to be shown, without resulting in any duplication. If duplication still happens, the loader will take care of loading the correct one when the item is being displayed the second time in the screen.

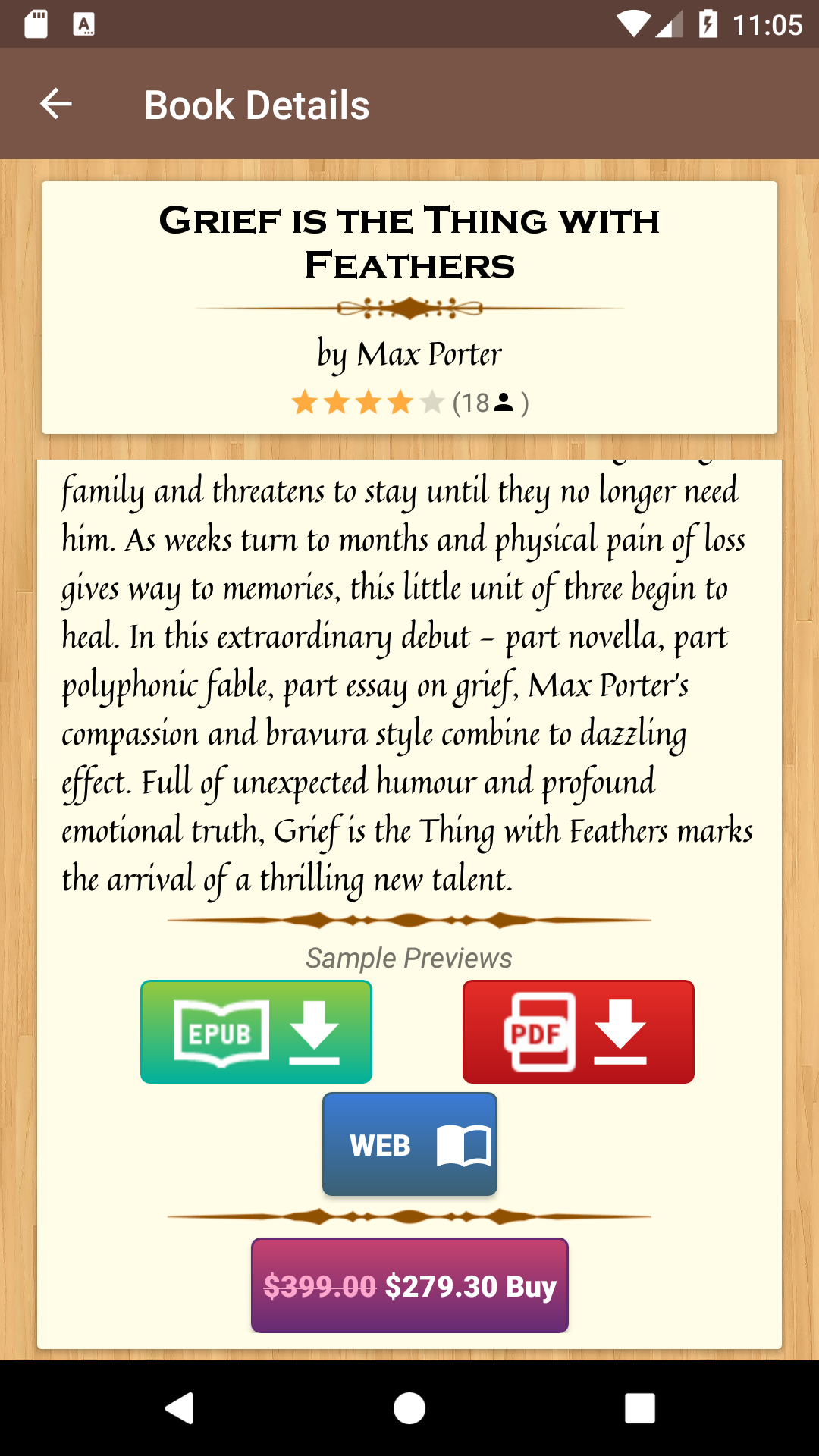


**Information in general, on the entire app:**

After scrolling to some extent in the List Tab View or the Grid Tab View:

* If the user swipes across the tabs, then the fragment being displayed will scroll to the item last shown in the previous tab.
* If the user selects the active tab again, user will be taken to the top item in the fragment.

If the search entered by the user does not yield any result, then the BookSearchActivity will display the hidden embedded page no\_result\_page.xml.



If during search, that is while initiating a request to the REST API, a network connectivity issue is encountered, then a Network Error Dialog is displayed for the user to taken accordingly. The dialog is implemented by the DialogFragment NetowkErrorDialogFragment that inflates the layout network\_error\_dialog.xml.

If the response fails with any HTTP error, then this will be consumed silently, and the search will not happen. Instead, the previous search results continue to be displayed.

During the loading of books data, a custom [progress bar](https://github.com/kaushiknsanji/Books_Library_App/blob/udacity/app/src/main/res/drawable/progress_bar_indeterminate.xml) will be displayed in the BookSearchActivity. This is implemented using the animation-list / AnimationDrawable.

**The About Page:**

This can be viewed by going into the menu item **"About"** in the overflow menu of the BookSearchActivity. This page describes in brief about the app and has links to my bio and the course details hosted by Udacity. This is shown by the activity [AboutActivity](https://github.com/kaushiknsanji/Books_Library_App/blob/udacity/app/src/main/java/com/example/kaushiknsanji/bookslibrary/AboutActivity.java) that inflates the layout [activity\_about.xml](https://github.com/kaushiknsanji/Books_Library_App/blob/udacity/app/src/main/res/layout/activity_about.xml).