CHAPTER 1 :
INTRODUCTION

1.1 Aim of the project :

To develop an user friendly android application to know the Books present in the Library . The app should Be able to show the lists of books present in the library .On pressing the Search Button it asks the user to type any name and after entering a name it should show the books related to that particular name with Relevant cover page of book ,with Relevant author names and also a small description of books.

1.2 Overview of the project :

Mobile app development is rapidly growing. From retail, education, telecommunications and ecommerce to insurance, healthcare and government, organizations across industries must meet user expectations for real-time, convenient ways to conduct transactions and access information. The project , "BOOK LIBRARY APPLICATION" is a cutting edge and versatile Book Library application fundamentally designed to help people to connect knowledge by providing a digital experience of searching books in library and getting the details of books It is very User friendly and easy to use. This application is compatible with all the Android versions . And with said that, anyone owing their own android phone, the Book Library Application can create a huge difference in hooking them up with others provided . It is very simple to use. It just briefs with Lists of books present, authors, and book details.

1.2 Outcome of the project :

In Book Library Application project user will be able get the List of Books present in Library. The UI is very simple and Understandble to the user they can press the Search button to search whatever name they like after entering it shows or lists the Books and by pressing the particular book the user can see a particular book authors given and a Small Description of Books.

CHAPTER - 2 :

DESIGN AND IMPLEMENTATION

2.1 Propose System

Android is an operating system which is built basically for Mobile phones. It is based on the Linux Kernel and other open-source software and is developed by Google. Android is very popular nowadays among students and students are now choosing Android for their projects. It's very much important for a beginner to build baby Android apps to learn Android.

Book Library is the application of science and technology to have the information of Books present in the Library through online in a Digital Manner. App connects to the Google Books API to retrieve the list of Books for the topic searched and then displays them in a decorative Book Shelf format. Using the app, the admin of the library can upload book details and where it is located. Then the user can see the available books. In earlier days, libraries used to maintain registers and book passes for every reader. Whenever we wish to borrow a book, we first search the books manually on every shelf and then go to the library desk . The intention of developing user-friendly Book Library app is to fetch the data in the need of taking information about Books digitally .This is a simple Book Library app in android using java .In the Book Library app the user will be able to interact with UI in which user need to enter the name of the Book they need whether it is present in Library or not and it displays the Book information with relevant Image or Cover page of related Book.

2.1 Source Code :

XML CODE:

```
<?xml version="1.0" encoding="utf-8"?><!--

 ~    http://www.apache.org/licenses/LICENSE-2.0
 ~
 ~ Unless required by applicable law or agreed to in writing, software
 ~ distributed under the License is distributed on an "AS IS" BASIS,
 ~ WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 ~ See the License for the specific language governing permissions and
 ~ limitations under the License.
 -->

<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.kaushiknsanji.bookslibrary">

    <!-- Permission for access to Internet -->
    <uses-permission android:name="android.permission.INTERNET" />
    <!-- Permission to check the Network State -->
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">

        <!-- Main Activity as well as the activity that allows to search Books -->
        <activity
            android:name=".BookSearchActivity"
            android:launchMode="singleTop">

            <!-- Launch Mode is SingleTop as this is also a Searchable Activity -->
            <intent-filter>
```

```
        <action android:name="android.intent.action.MAIN" />
        <action android:name="android.intent.action.SEARCH" />

        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
    <!-- Defining the Searchable Configuration of the Searchable Activity -->
    <meta-data
        android:name="android.app.searchable"
        android:resource="@xml/searchable" />
</activity>

<!-- Activity for the Search related settings -->
<activity
    android:name=".settings.SearchSettingsActivity"
    android:label="@string/search_settings_title_str"
    android:parentActivityName=".BookSearchActivity">
    <intent-filter>
        <category android:name="android.intent.category.DEFAULT" />
    </intent-filter>
    <!-- Parent Activity meta-data for android 4.0 and lower -->
    <meta-data
        android:name="android.support.PARENT_ACTIVITY"
        android:value=".BookSearchActivity" />
</activity>

<!-- Activity that provides info on the Book selected in the Main Activity -->
<activity
    android:name=".BookDetailActivity"
    android:label="@string/book_detail_title_str"
    android:launchMode="singleTop"
    android:parentActivityName=".BookSearchActivity">

    <!-- Launch Mode is SingleTop to retain the Intent data sent by its Parent -->
    <intent-filter>
        <category android:name="android.intent.category.DEFAULT" />
    </intent-filter>
    <!-- Parent Activity meta-data for android 4.0 and lower -->
    <meta-data
        android:name="android.support.PARENT_ACTIVITY"
        android:value=".BookSearchActivity" />
</activity>
```

```
        <!-- Activity that displays a full picture of the Book from the activity
        opened for viewing Book details -->
        <activity
            android:name=".BookImageActivity"
            android:label="@string/book_image_title_str"
            android:parentActivityName=".BookDetailActivity">
            <intent-filter>
                <category android:name="android.intent.category.DEFAULT" />
            </intent-filter>
            <!-- Parent Activity meta-data for android 4.0 and lower -->
            <meta-data
                android:name="android.support.PARENT_ACTIVITY"
                android:value=".BookDetailActivity" />
        </activity>

        <!-- Activity that displays info related to the App and its developer -->
        <activity
            android:name=".AboutActivity"
            android:label="@string/about_title_str"
            android:parentActivityName=".BookSearchActivity">
            <intent-filter>
                <category android:name="android.intent.category.DEFAULT" />
            </intent-filter>
            <!-- Parent Activity meta-data for android 4.0 and lower -->
            <meta-data
                android:name="android.support.PARENT_ACTIVITY"
                android:value=".BookSearchActivity" />
        </activity>

        <!-- Provider for the Recent Search Suggestions -->
        <provider
            android:name=".providers.RecentBookSearchProvider"

android:authorities="com.example.kaushiknsanji.bookslibrary.providers.RecentBookSearchProvider
"
            android:exported="false" />

    </application>

</manifest>
```

JAVA CODE:

```
*/

package com.example.kaushiknsanji.bookslibrary;

import android.app.SearchManager;
import android.content.Context;
import android.content.Intent;
import android.content.SharedPreferences;
import android.graphics.Typeface;
import android.os.Build;
import android.os.Bundle;
import android.os.Handler;
import android.os.Message;
import android.provider.SearchRecentSuggestions;
import android.support.annotation.NonNull;
import android.support.design.widget.TabLayout;
import android.support.v4.app.FragmentManager;
import android.support.v4.app.LoaderManager;
import android.support.v4.content.Loader;
import android.support.v4.view.ViewPager;
import android.support.v7.app.AppCompatActivity;
import android.support.v7.preference.PreferenceManager;
import android.support.v7.widget.SearchView;
import android.text.Html;
import android.text.Spanned;
import android.text.TextUtils;
import android.util.Log;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.ImageButton;
import android.widget.ProgressBar;
import android.widget.ScrollView;
import android.widget.TextView;
import android.widget.Toast;
```

```
import com.example.kaushiknsanji.bookslibrary.adapters.DisplayPagerAdapter;
import com.example.kaushiknsanji.bookslibrary.adapterviews.KeywordFiltersDialogFragment;
import com.example.kaushiknsanji.bookslibrary.adapterviews.RecyclerViewFragment;
import com.example.kaushiknsanji.bookslibrary.cache.BitmapImageCache;
import com.example.kaushiknsanji.bookslibrary.dialogs.NetworkErrorDialogFragment;
import com.example.kaushiknsanji.bookslibrary.dialogs.PaginationNumberPickerDialogFragment;
import com.example.kaushiknsanji.bookslibrary.models.BookInfo;
import com.example.kaushiknsanji.bookslibrary.observers.OnAdapterItemDataSwapListener;
import com.example.kaushiknsanji.bookslibrary.observers.OnPagerFragmentVerticalScrollListener;
import com.example.kaushiknsanji.bookslibrary.providers.RecentBookSearchProvider;
import com.example.kaushiknsanji.bookslibrary.settings.SearchSettingsActivity;
import com.example.kaushiknsanji.bookslibrary.utils.PreferencesObserverUtility;
import com.example.kaushiknsanji.bookslibrary.utils.TextAppearanceUtility;
import com.example.kaushiknsanji.bookslibrary.workers.BooksLoader;

import java.lang.ref.WeakReference;
import java.util.List;

/**
 * The Main and the Searchable Activity of the app that shows the layout 'R.layout.activity_main'
 * containing a SearchView button and a ViewPager
 * which displays a list/grid of books based on the user's search
 *
 * @author Kaushik N Sanji
 */
public class BookSearchActivity
        extends AppCompatActivity
        implements KeywordFiltersDialogFragment.OnKeywordFilterSelectedListener,
        TabLayout.OnTabSelectedListener,
        LoaderManager.LoaderCallbacks<List<BookInfo>>,
        OnAdapterItemDataSwapListener, OnPagerFragmentVerticalScrollListener,
        SharedPreferences.OnSharedPreferenceChangeListener,
        OnClickListener {

    //Constant used for logs
    private static final String LOG_TAG = BookSearchActivity.class.getSimpleName();
    //Bundle Key constants used for saving/restoring the state
    private static final String KEYWORD_FILTERS_STATE_BOOL_KEY =
"KeywordFilters.State";
```

```
    private static final String SEARCH_VIEW_STATE_BOOL_KEY = "SearchView.State";
    private static final String SEARCH_VIEW_QUERY_STR_KEY = "SearchView.Query";
    private static final String SEARCH_VIEW_QUERY_IN_PROGRESS_STR_KEY =
"SearchView.QueryInProgress";
    private static final String ACTIVE_TAB_POSITION_INT_KEY = "TabLayout.ActiveTabIndex";
    private static final String VISIBLE_ITEM_VIEW_POSITION_INT_KEY =
"ViewPager.ItemPosition";
    private static final String WELCOME_PAGE_STATE_INT_KEY = "WelcomePage.State";
    private static final String PROGRESS_BAR_STATE_INT_KEY = "ProgressBar.State";
    //Instance of the Network Error Handler for displaying the Network Error Dialog
    private final NetworkErrorHandler mNetworkErrorHandler = new NetworkErrorHandler(this);
    //For Recent Search Suggestions
    private SearchRecentSuggestions mRecentSuggestions;
    //For the SearchView
    private SearchView mSearchView;
    //For the SearchView's MenuItem
    private MenuItem mSearchMenuItem;
    //For the ViewPager
    private ViewPager mViewPager;
    //For the Tabs attached to ViewPager
    private TabLayout mTabLayout;
    //For the Pagination buttons displayed at the bottom (when visible)
    private ImageButton mPageFirstButton;
    private ImageButton mPageLastButton;
    private ImageButton mPageNextButton;
    private ImageButton mPagePreviousButton;
    private ImageButton mPageMoreButton;
    //For the hidden Welcome Page
    private ScrollView mWelcomePageScrollView;
    //For the hidden No Result Page
    private ScrollView mNoResultPageScrollView;
    //For the hidden Progress Bar
    private ProgressBar mIndeterminateProgressBar;
    //Stores the state of the Dialog for the Keyword Filters,
    //defaulted to False to indicate that the Dialog is currently not shown
    private boolean mKeywordFiltersViewState = false;
    //Boolean flag that saves the state of whether the SearchView was expanded for searching
    private boolean mIsSearchViewExpanded;
    //Saves the Search Query executed by the user in SearchView
```

```
    private String mSearchQueryStr;
    //Saves the Search Query being entered by the user in SearchView, but not completed the search
    private String mSearchQueryInProgressStr;
    //Saves the first visible Adapter Item position
    private int mVisibleItemViewPosition;
    //List of Preference Keys to exclude while triggering the loader to load data
    private List<String> mKeysToExclude;
    //For the Settings SharedPreferences
    private SharedPreferences mPreferences;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        Log.d(LOG_TAG, "onCreate: Started");

        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        //Loading the default values for the Preferences on the first Initial launch after install
        PreferenceManager.setDefaultValues(this, R.xml.preferences, false);

        //Reading the List of Preference Keys to exclude while triggering the loader to load data
        mKeysToExclude = PreferencesObserverUtility.getPreferenceKeysToExclude(this);

        //Retrieving the instance of SharedPreferences
        mPreferences = PreferenceManager.getDefaultSharedPreferences(this);

        //Instantiating SearchRecentSuggestions
        mRecentSuggestions = new SearchRecentSuggestions(
                this, RecentBookSearchProvider.AUTHORITY,
RecentBookSearchProvider.DATABASE_MODE_QUERIES
        );

        //Initializing the Pagination Buttons
        mPageFirstButton = findViewById(R.id.page_first_button_id);
        mPageLastButton = findViewById(R.id.page_last_button_id);
        mPageNextButton = findViewById(R.id.page_next_button_id);
        mPagePreviousButton = findViewById(R.id.page_previous_button_id);
        mPageMoreButton = findViewById(R.id.page_more_button_id);
```

```
        //Registering click listener on the Pagination buttons
        mPageFirstButton.setOnClickListener(this);
        mPageLastButton.setOnClickListener(this);
        mPageNextButton.setOnClickListener(this);
        mPagePreviousButton.setOnClickListener(this);
        mPageMoreButton.setOnClickListener(this);

        //Preparing the Welcome Page layout
        manageWelcomePage(true, -1);

        //Preparing the No Result Page layout
        manageNoResultPage(true, -1);

        //Finding the Progress Bar
        mIndeterminateProgressBar = findViewById(R.id.progress_bar_id);

        //ViewPager for swiping through fragments
        mViewPager = findViewById(R.id.view_pager_id);

        //Adapter for ViewPager to display the correct fragment at the active position
        DisplayPagerAdapter viewPagerAdapter = new
DisplayPagerAdapter(getSupportFragmentManager(), this.getApplicationContext());

        //Binding the Adapter to ViewPager
        mViewPager.setAdapter(viewPagerAdapter);

        //Tabs to be shown for the Fragments displayed
        mTabLayout = findViewById(R.id.sliding_tabs_id);

        //Binding the TabLayout to ViewPager
        mTabLayout.setupWithViewPager(mViewPager);

        //Iterating over the tabs to set the custom view
        int noOfTabs = mTabLayout.getTabCount();
        for (int tabIndex = 0; tabIndex < noOfTabs; tabIndex++) {
            TabLayout.Tab tab = mTabLayout.getTabAt(tabIndex);
            tab.setCustomView(viewPagerAdapter.getTabView(tabIndex));
        }
```

```
        if (savedInstanceState == null) {
            //On Initial Launch of the App

            //Make the First tab as Selected so that it displays the title as well
            onTabSelected(mTabLayout.getTabAt(0));

            //Resetting the value of Page index related settings to 1, when not 1 on initial load
            resetPageIndex();

            //Displaying the Welcome Page
            manageWelcomePage(false, View.VISIBLE);
        }

    }

    /**
     * Method that manages the hidden No Result Page layout 'R.layout.no_result_page'
     *
     * @param doPrepare  Boolean to indicate whether the layout page is to be initialized or not
     *                   <br/> When <b>TRUE</b>, the layout page will be initialized.
     *                   #visibility value can be anything in this case as it will not be updated.
     *                   <br/> When <b>FALSE</b>, the layout page will NOT be re-initialized,
     *                   instead it allows to change the visibility of the layout page.
     * @param visibility is one the Integer values of the View's Visibilities,
     *                   to which the layout page's visibility is to be updated.
     */
    private void manageNoResultPage(boolean doPrepare, int visibility) {

        //Retrieving the font to be used for the Text Content
        Typeface contentTypeface = Typeface.createFromAsset(getAssets(),
"fonts/lobster_two_regular.ttf");

        if (doPrepare) {
            //When the layout page is to be prepared

            //Initializing the ScrollView having the content of No Result Page
            mNoResultPageScrollView = findViewById(R.id.no_result_page_id);

            //Loading the font for the Title Text
```

```
        TextView titleTextView =
mNoResultPageScrollView.findViewById(R.id.no_result_content_title_text_id);
        titleTextView.setTypeface(Typeface.createFromAsset(getAssets(),
"fonts/ar_hermann_medium.ttf"));

        //Loading the font for the remaining Text content
        TextView content2TextView =
mNoResultPageScrollView.findViewById(R.id.no_result_text_2_id);
        content2TextView.setTypeface(contentTypeface);
        TextView content3TextView =
mNoResultPageScrollView.findViewById(R.id.no_result_text_3_id);
        content3TextView.setTypeface(contentTypeface);

    } else {
        //When the layout page is to be shown/hidden

        if (visibility == View.VISIBLE) {
            //When the layout page needs to be visible

            //Retrieving the TextView of the first line to set its text accordingly
            TextView content1TextView =
mNoResultPageScrollView.findViewById(R.id.no_result_text_1_id);
            //Generating the Html Text with Search Query entered by the user
            Spanned htmlSpanText;
            if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.N) {
                htmlSpanText = Html.fromHtml(getString(R.string.no_result_page_textline_1,
mSearchQueryStr), Html.FROM_HTML_MODE_LEGACY);
            } else {
                htmlSpanText = Html.fromHtml(getString(R.string.no_result_page_textline_1,
mSearchQueryStr));
            }
            //Setting the above Html Text
            content1TextView.setText(htmlSpanText);
            //Setting the font
            content1TextView.setTypeface(contentTypeface);

            //Hiding the Tabs
            mTabLayout.setVisibility(View.GONE);

        } else if (visibility == View.GONE) {
```

```
            //When the layout page needs to be hidden

            //Making the Tabs Visible
            mTabLayout.setVisibility(View.VISIBLE);
        }

        //Setting the layout page to the requested visibility
        mNoResultPageScrollView.setVisibility(visibility);
    }

}

/**
 * Method that manages the hidden Welcome Page layout 'R.layout.welcome_page'
 *
 * @param doPrepare  Boolean to indicate whether the layout page is to be initialized or not
 *                   <br/> When <b>TRUE</b>, the layout page will be initialized.
 *                   #visibility value can be anything in this case as it will not be updated.
 *                   <br/> When <b>FALSE</b>, the layout page will NOT be re-initialized,
 *                   instead it allows to change the visibility of the layout page.
 * @param visibility is one the Integer values of the View's Visibilities,
 *                   to which the layout page's visibility is to be updated.
 */
private void manageWelcomePage(boolean doPrepare, int visibility) {
    if (doPrepare) {
        //When the layout page is to be prepared

        //Initializing the ScrollView having the content of Welcome Page
        mWelcomePageScrollView = findViewById(R.id.welcome_page_id);

        //Loading the font for the Title Text
        TextView titleTextView =
mWelcomePageScrollView.findViewById(R.id.welcome_text_1_id);
        titleTextView.setTypeface(Typeface.createFromAsset(getAssets(), "fonts/cambriab.ttf"));

        //Retrieving the font to be used for the Text Content
        Typeface contentTypeface = Typeface.createFromAsset(getAssets(),
"fonts/maiandra_gd_regular.ttf");
```

```
            //Loading the font for the Text Content
            TextView content1TextView =
mWelcomePageScrollView.findViewById(R.id.welcome_text_2_id);
            content1TextView.setTypeface(contentTypeface);
            TextView content2TextView =
mWelcomePageScrollView.findViewById(R.id.welcome_text_3_id);
            content2TextView.setTypeface(contentTypeface);

            //Replacing the drawable identifier string in the text content with its actual drawable
            TextAppearanceUtility.replaceTextWithImage(this, content2TextView);

        } else {
            //When the layout page is to be shown/hidden

            if (visibility == View.VISIBLE) {
                //When the layout page needs to be visible

                //Hiding the Tabs
                mTabLayout.setVisibility(View.GONE);

            } else if (visibility == View.GONE) {
                //When the layout page needs to be hidden

                //Making the Tabs Visible
                mTabLayout.setVisibility(View.VISIBLE);
            }

            //Setting the layout page to the requested visibility
            mWelcomePageScrollView.setVisibility(visibility);
        }

    }

    /**
     * Method to toggle the visibility of the Indeterminate Progress Bar
     *
     * @param visibility is one the Integer values of the View's Visibilities,
     *                   to which the ProgressBar's visibility is to be updated.
     */
```

```
    private void toggleProgressBarVisibility(int visibility) {
        mIndeterminateProgressBar.setVisibility(visibility);
    }

    //Called by the Activity after Start, to restore the activity's state from the Bundle
    //while being reinitialized
    @Override
    protected void onRestoreInstanceState(Bundle savedInstanceState) {
        super.onRestoreInstanceState(savedInstanceState);

        //Restoring the visibility state of the Welcome Page
        if (savedInstanceState.getInt(WELCOME_PAGE_STATE_INT_KEY) == View.VISIBLE) {
            manageWelcomePage(false, View.VISIBLE);
        }

        //Restoring the visibility state of the Progress Bar
        if (savedInstanceState.getInt(PROGRESS_BAR_STATE_INT_KEY) == View.VISIBLE) {
            toggleProgressBarVisibility(View.VISIBLE);
        }

        //Restoring the "Search Keyword Filters" state
        mKeywordFiltersViewState =
savedInstanceState.getBoolean(KEYWORD_FILTERS_STATE_BOOL_KEY);

        //Restoring the SearchView State
        mIsSearchViewExpanded =
savedInstanceState.getBoolean(SEARCH_VIEW_STATE_BOOL_KEY);
        mSearchQueryStr = savedInstanceState.getString(SEARCH_VIEW_QUERY_STR_KEY);
        mSearchQueryInProgressStr =
savedInstanceState.getString(SEARCH_VIEW_QUERY_IN_PROGRESS_STR_KEY);

        //Restoring the value of the position of the first Adapter item previously visible in the
ViewPager
        mVisibleItemViewPosition =
savedInstanceState.getInt(VISIBLE_ITEM_VIEW_POSITION_INT_KEY);

        //Restoring the active tab

mViewPager.setCurrentItem(savedInstanceState.getInt(ACTIVE_TAB_POSITION_INT_KEY));
        onTabSelected(mTabLayout.getTabAt(mViewPager.getCurrentItem()));
```

```
        //When the user had searched something previously
        if (!TextUtils.isEmpty(mSearchQueryStr)) {
            //Restoring the Query as the Activity Title
            setTitle(getString(R.string.searched_book_title, mSearchQueryStr));
        }

        //Restoring the state of Pagination Buttons based on the current setting
        updatePaginationButtonsState();
    }

    //Called by the Activity before Stop, to save the activity's state in the Bundle
    @Override
    protected void onSaveInstanceState(Bundle outState) {
        //Saving the "Search Keyword Filters" state
        outState.putBoolean(KEYWORD_FILTERS_STATE_BOOL_KEY,
mKeywordFiltersViewState);

        //Saving the SearchView State if inflated and present
        if (mSearchMenuItem != null) {
            outState.putBoolean(SEARCH_VIEW_STATE_BOOL_KEY,
mSearchMenuItem.isActionViewExpanded());
            outState.putString(SEARCH_VIEW_QUERY_IN_PROGRESS_STR_KEY,
mSearchView.getQuery().toString());
        }
        outState.putString(SEARCH_VIEW_QUERY_STR_KEY, mSearchQueryStr);

        //Saving the current tab position
        outState.putInt(ACTIVE_TAB_POSITION_INT_KEY, mViewPager.getCurrentItem());

        //Saving the current position of the first Adapter item visible (partially) in the ViewPager
        outState.putInt(VISIBLE_ITEM_VIEW_POSITION_INT_KEY,
getCurrentFragmentFromViewPager().getFirstVisibleItemPosition());

        //Saving the visibility state of the Welcome Page
        outState.putInt(WELCOME_PAGE_STATE_INT_KEY,
mWelcomePageScrollView.getVisibility());

        //Saving the visibility state of the Progress Bar
```

```
        outState.putInt(PROGRESS_BAR_STATE_INT_KEY,
mIndeterminateProgressBar.getVisibility());

        super.onSaveInstanceState(outState);
    }

    //Called by the Activity when it is prepared to be shown
    @Override
    protected void onResume() {
        Log.d(LOG_TAG, "onResume: Started");
        super.onResume();

        //Registering the Listener on TabLayout
        mTabLayout.addOnTabSelectedListener(this);

        //Registering the Preference Change Listener
        mPreferences.registerOnSharedPreferenceChangeListener(this);

    }

    //Called by the Activity when it loses focus
    @Override
    protected void onPause() {
        super.onPause();

        //UnRegistering the Listener on TabLayout
        mTabLayout.removeOnTabSelectedListener(this);

        if (isFinishing()) {
            //When App is exiting

            //UnRegistering the Preference Change Listener
            mPreferences.unregisterOnSharedPreferenceChangeListener(this);

            //Clearing any pending callbacks/messages sent to the Network Error Handler
            mNetworkErrorHandler.removeCallbacksAndMessages(null);
        }

    }
```

```
/**
 * Method that handles the ACTION_SEARCH Intent
 *
 * @param intent is the Intent received by this Activity
 */
private void handleIntent(Intent intent) {
    //Working based on the Intent's ACTION
    if (Intent.ACTION_SEARCH.equals(intent.getAction())) {
        //Retrieving the Search Query from EXTRA
        String currentSearchQueryStr = intent.getStringExtra(SearchManager.QUERY);

        //Hiding the Welcome Page if Visible
        if (mWelcomePageScrollView.getVisibility() == View.VISIBLE) {
            manageWelcomePage(false, View.GONE);
        }

        //Hiding the No Result Page if Visible
        if (mNoResultPageScrollView.getVisibility() == View.VISIBLE) {
            manageNoResultPage(false, View.GONE);
        }

        //Checking if the current query is new to restart the loader
        boolean isNewQuery = !TextUtils.isEmpty(mSearchQueryStr) &&
!currentSearchQueryStr.equalsIgnoreCase(mSearchQueryStr);

        //Copying the current query
        mSearchQueryStr = currentSearchQueryStr;

        //Adding the Search Query to the Recent Search Suggestions
        mRecentSuggestions.saveRecentQuery(mSearchQueryStr, null);

        //Collapse the SearchView
        collapseSearchView();

        //Set the Query searched as the Activity Title
        setTitle(getString(R.string.searched_book_title, mSearchQueryStr));

        //Performing the Book Search operation through a Loader
```

```
        Toast.makeText(this, "Searching for " + mSearchQueryStr,
Toast.LENGTH_SHORT).show();

        //Displaying the Progress Bar
        toggleProgressBarVisibility(View.VISIBLE);

        if (isNewQuery) {
            //Resetting the Adapter Item View position to 0 (First Item data in the adapter)
            scrollToItemPosition(0, true);

            //Resetting the value of Page index related settings to 1, for the new Search Query
            resetPageIndex();

            //Restarting Loader when it is a new Search query
            getSupportLoaderManager().restartLoader(BooksLoader.BOOK_SEARCH_LOADER,
null, this);

            //Clearing the Bitmap Memory Cache for the new Search done
            BitmapImageCache.clearCache();
        } else {
            //Triggering the load with the same Search Query
            getSupportLoaderManager().initLoader(BooksLoader.BOOK_SEARCH_LOADER, null,
this);
        }
    }
  }

  /**
   * Method that resets the 'startIndex' (Page to Display) setting to 1, when not 1
   * and also its related 'endIndex' setting value to that of 'startIndex'
   */
  private void resetPageIndex() {
    //Retrieving the preference key string of 'startIndex'
    String startIndexPrefKeyStr = getString(R.string.pref_page_to_display_key);
    //Retrieving the preference key string of 'endIndex'
    String endIndexPrefKeyStr = getString(R.string.pref_page_to_display_max_value_key);
    //Retrieving the preference key string for the last page index viewed
    String lastViewedPageIndexPrefKeyStr = getString(R.string.pref_last_displayed_page_key);
    //Retrieving the default value of 'startIndex' setting
```

```
        int startIndexPrefKeyDefaultValue =
getResources().getInteger(R.integer.pref_page_to_display_default_value);
        //Retrieving the current value of 'startIndex' setting
        int startIndex = mPreferences.getInt(startIndexPrefKeyStr, startIndexPrefKeyDefaultValue);
        //Retrieving the current value of 'endIndex' setting
        int endIndex = mPreferences.getInt(endIndexPrefKeyStr, startIndexPrefKeyDefaultValue);

        if (startIndex != 1) {
            //When the 'startIndex' setting value is not equal to 1

            //Adding the key to exclusion, to avoid listener from retriggering the load on data change
            PreferencesObserverUtility.addKeyToExclude(mKeysToExclude, startIndexPrefKeyStr);

            //Opening the Editor to update the value
            SharedPreferences.Editor prefEditor = mPreferences.edit();
            //Setting to its default value, which is 1
            prefEditor.putInt(startIndexPrefKeyStr, startIndexPrefKeyDefaultValue);
            //Updating the value of 'endIndex' as well, to the default value of 'startIndex'
            prefEditor.putInt(endIndexPrefKeyStr, startIndexPrefKeyDefaultValue);
            //Updating the same value to the preference of the last viewed page index
            prefEditor.putInt(lastViewedPageIndexPrefKeyStr, startIndexPrefKeyDefaultValue);
            prefEditor.apply(); //applying the changes

            //Removing the key from exclusion, to listen to the future updates on this key
            PreferencesObserverUtility.removeKeyToInclude(mKeysToExclude, startIndexPrefKeyStr);

        } else if (endIndex != startIndex) {
            //When the 'startIndex' setting value is 1, but 'endIndex' setting value is different
            //(This case occurs on a new search query entered by the user)

            //Opening the Editor to update the value
            SharedPreferences.Editor prefEditor = mPreferences.edit();
            //Setting to its default value, which is 1
            prefEditor.putInt(endIndexPrefKeyStr, startIndexPrefKeyDefaultValue);
            //Updating the same value to the preference of the last viewed page index
            prefEditor.putInt(lastViewedPageIndexPrefKeyStr, startIndexPrefKeyDefaultValue);
            prefEditor.apply(); //applying the changes

        }
```

```
}

/**
 * Method that updates the state of the Pagination Buttons
 * based on the current setting
 */
private void updatePaginationButtonsState() {

    //Retrieving the 'startIndex' (Page to Display) setting value
    int startIndex = mPreferences.getInt(getString(R.string.pref_page_to_display_key),
            getResources().getInteger(R.integer.pref_page_to_display_default_value));

    //Retrieving the 'endIndex' preference value
    int endIndex = mPreferences.getInt(getString(R.string.pref_page_to_display_max_value_key),
            startIndex);

    Log.d(LOG_TAG, "updatePaginationButtonsState: startIndex " + startIndex);
    Log.d(LOG_TAG, "updatePaginationButtonsState: endIndex " + endIndex);

    if (startIndex == endIndex && startIndex != 1) {
        //When the last page is reached

        //Disabling the page-last and page-next buttons
        mPageLastButton.setEnabled(false);
        mPageNextButton.setEnabled(false);

        //Enabling the rest
        mPageFirstButton.setEnabled(true);
        mPagePreviousButton.setEnabled(true);
        mPageMoreButton.setEnabled(true);

        Log.d(LOG_TAG, "updatePaginationButtonsState: last buttons disabled");


    }
    if (startIndex == endIndex && startIndex == 1) {
        //When the first and last page is same, and only one page is existing
```

```
            //Disabling all the buttons
            mPageLastButton.setEnabled(false);
            mPageNextButton.setEnabled(false);
            mPageFirstButton.setEnabled(false);
            mPagePreviousButton.setEnabled(false);
            mPageMoreButton.setEnabled(false);

            Log.d(LOG_TAG, "updatePaginationButtonsState: all buttons disabled");


        } else if (startIndex != endIndex && startIndex == 1) {
            //When the first page is reached, and last page is not same as first page

            //Disabling the page-first and page-previous buttons
            mPageFirstButton.setEnabled(false);
            mPagePreviousButton.setEnabled(false);

            //Enabling the rest
            mPageMoreButton.setEnabled(true);
            mPageLastButton.setEnabled(true);
            mPageNextButton.setEnabled(true);

            Log.d(LOG_TAG, "updatePaginationButtonsState: first buttons disabled");


        } else if (startIndex != endIndex) {
            //Enabling all the buttons when first and last page are different
            mPageFirstButton.setEnabled(true);
            mPagePreviousButton.setEnabled(true);
            mPageMoreButton.setEnabled(true);
            mPageLastButton.setEnabled(true);
            mPageNextButton.setEnabled(true);

            Log.d(LOG_TAG, "updatePaginationButtonsState: all buttons enabled");

        }

    }
```

```
/**
 * Method that collapses the SearchView when the Search is submitted
 */
private void collapseSearchView() {
    //Checking for Null that occurs when the device is rotated
    if (mSearchView != null) {
        //Removing Focus
        mSearchView.clearFocus();
        //Clearing the Query on SearchView
        mSearchView.setQuery("", false);
        //Making SearchView to appear as Search Icon
        mSearchView.setIconified(true);
        //Collapsing the SearchView MenuItem
        mSearchMenuItem.collapseActionView();
    }
}

/**
 * Method that is called for restoring the SearchView Content
 * on Configuration change, if the search is not submitted
 *
 * @param searchQueryStr is the Search Query that needs to be updated to SearchView
 */
private void restoreSearchViewContent(String searchQueryStr) {
    //Expanding the Search Action View
    mSearchMenuItem.expandActionView();
    if (!TextUtils.isEmpty(searchQueryStr)) {
        //Restoring the Query if any
        mSearchView.setQuery(searchQueryStr, false);
    }

    if (!mKeywordFiltersViewState) {
        //Restoring Focus on SearchView if "Search Keyword Filters" Dialog is NOT active
        mSearchView.requestFocus();
    } else {
        //Clearing Focus on SearchView if "Search Keyword Filters" Dialog is active
        mSearchView.clearFocus();
    }
}
```

```
/**
 * As this Activity is configured with launchMode as 'singleTop',
 * we are handling new Search Intents with this method
 *
 * @param intent is the Intent received by this Activity
 */
@Override
protected void onNewIntent(Intent intent) {
    setIntent(intent); //Setting the new intent received
    //Handling the new Intent
    handleIntent(intent);
}

//Method that inflates the Menu
//and initializes the SearchView for Assisted Search
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    Log.d(LOG_TAG, "onCreateOptionsMenu: Started");

    //Inflating the Menu options from menu_main.xml
    getMenuInflater().inflate(R.menu.menu_main, menu);

    //Retrieving the SearchManager Instance
    SearchManager = (SearchManager) getSystemService(Context.SEARCH_SERVICE);

    if (mSearchMenuItem == null) {
        //Retrieving the SearchView Menu Item
        mSearchMenuItem = menu.findItem(R.id.search_action_id);
        mSearchView = (SearchView) mSearchMenuItem.getActionView();
    }

    //Setting the Instance of SearchableInfo on SearchView to setup Assisted Search
    if (searchManager != null) {
        mSearchView.setSearchableInfo(searchManager.getSearchableInfo(getComponentName()));
    }

    if (mIsSearchViewExpanded) {
        //Restoring the Action View state and the Search Content if any
```

```
            restoreSearchViewContent(mSearchQueryInProgressStr);
            //Resetting the member value to blank
            mSearchQueryInProgressStr = "";
        }

        //On Initial launch, the Search Query will be empty (using this as a flag for the same)
        if (TextUtils.isEmpty(mSearchQueryStr)) {
            //Registering the OnAdapterItemDataSwapListener &
OnPagerFragmentVerticalScrollListener
            //during the Initial launch
            //(AdapterViews for the ViewPager will be available only at this point during the initial
launch)

            //Retrieving the current ViewPager position
            int position = mViewPager.getCurrentItem();
            //Retrieving the current RecyclerViewFragment
            RecyclerViewFragment fragment = getFragmentByPositionFromViewPager(position);
            //Registering the OnAdapterItemDataSwapListener
            fragment.registerItemDataSwapListener(this, position);
            //Registering the OnPagerFragmentVerticalScrollListener
            fragment.setOnPagerFragmentVerticalScrollListener(this);
        }

        return true;
    }

    /**
     * This hook is called whenever an item in your options menu is selected.
     * The default implementation simply returns false to have the normal
     * processing happen.
     *
     * @param item The menu item that was selected.
     * @return boolean Return false to allow normal menu processing to
     * proceed, true to consume it here.
     * @see #onCreateOptionsMenu
     */
    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        //Handling the Menu item selected
```

```
        switch (item.getItemId()) {
            case R.id.search_action_id:
                //"Search Books" action will never come in overflow menu, hence returning false
                return false;
            case R.id.search_settings_action_id:
                //Starting the SearchSettingsActivity when "Search Settings" is clicked
                Intent settingsIntent = new Intent(this, SearchSettingsActivity.class);
                startActivity(settingsIntent);
                return true;
            case R.id.clear_recent_search_action_id:
                //Clearing the Recent suggestion history, when "Clear Search History" is clicked
                mRecentSuggestions.clearHistory();
                //Displaying a toast for the action done
                Toast.makeText(this, R.string.search_suggestions_cleared_msg,
Toast.LENGTH_SHORT).show();
                return true;
            case R.id.keyword_filter_action_id:
                //When "Search Keyword Filters" is clicked
                handleKeywordFilterAction();
                return true;
            case R.id.about_action_id:
                //Starting the AboutActivity when "About" is clicked
                Intent aboutIntent = new Intent(this, AboutActivity.class);
                startActivity(aboutIntent);
                return true;
            default:
                return super.onOptionsItemSelected(item);
        }
    }

    /**
     * Method that handles the action related to the Overflow Menu item "Search Keyword Filters"
     */
    private void handleKeywordFilterAction() {
        if (mSearchMenuItem.isActionViewExpanded()) {
            //Clearing the focus from SearchView
            mSearchView.clearFocus();
            //Display the Keyword Filter Selection dialog only when the user is Searching for a Book
```

```
            KeywordFiltersDialogFragment.newInstance().show(getFragmentManager(),
KeywordFiltersDialogFragment.FRAGMENT_TAG);
            //Setting the Dialog state to True as the Dialog is active
            mKeywordFiltersViewState = true;
        } else {
            //Displaying a Toast Message to indicate the proper usage
            //when the user is currently not searching but clicked the menu item "Search Keyword
Filters"
            Toast.makeText(this, getString(R.string.keyword_filter_search_usage_error),
Toast.LENGTH_SHORT).show();
        }
    }

    /**
     * Method that returns the current {@link RecyclerViewFragment}
     * of the {@link ViewPager}
     *
     * @return current {@link RecyclerViewFragment} of the {@link ViewPager}
     */
    private RecyclerViewFragment getCurrentFragmentFromViewPager() {
        DisplayPagerAdapter pagerAdapter = (DisplayPagerAdapter) mViewPager.getAdapter();
        return (RecyclerViewFragment)
pagerAdapter.getRegisteredFragment(mViewPager.getCurrentItem());
    }

    /**
     * Method that returns the {@link RecyclerViewFragment}
     * of the {@link ViewPager} at the given position
     *
     * @param position is the Position in the {@link ViewPager} whose Fragment is to be retrieved
     * @return {@link RecyclerViewFragment} at the position in the {@link ViewPager}
     */
    private RecyclerViewFragment getFragmentByPositionFromViewPager(int position) {
        DisplayPagerAdapter pagerAdapter = (DisplayPagerAdapter) mViewPager.getAdapter();
        return (RecyclerViewFragment) pagerAdapter.getRegisteredFragment(position);
    }

    /**
     * Method that sets the currently viewing position in the RecyclerView to the
```

```
     * item position specified
     *
     * @param position       is the item position to which the RecyclerView needs to be positioned to
     * @param scrollImmediate is a boolean which denotes the way in which the scroll to position
     *                        needs to be handled
     *                        <br/><b>TRUE</b> if the scroll to position needs to be set immediately
     *                        without any animation
     *                        <br/><b>FALSE</b> if the scroll to position needs to be done naturally
     *                        with the default animation
     */
    private void scrollToItemPosition(int position, boolean scrollImmediate) {
        //Retrieving the current RecyclerViewFragment from the ViewPager
        RecyclerViewFragment fragment = getCurrentFragmentFromViewPager();
        //Retrieving the first visible current item's position in the RecyclerView
        mVisibleItemViewPosition = fragment.getFirstVisibleItemPosition();
        //Validating the position passed is different from the current one to update if required
        if (mVisibleItemViewPosition != position) {
            //Updating the item position reference
            mVisibleItemViewPosition = position;
            //Scrolling to the position passed if the current visible item position is different
            fragment.scrollToItemPosition(mVisibleItemViewPosition, scrollImmediate);
        }

        if (mVisibleItemViewPosition == 0) {
            //Hiding the Pagination panel when the RecyclerView is positioned to its first item
            findViewById(R.id.pagination_panel_id).setVisibility(View.GONE);
        }
    }

    /**
     * Callback Method of {@link KeywordFiltersDialogFragment}
     * that updates the Search string in SearchView
     * with the Search Keyword Filter selected for advanced searching
     *
     * @param filterValue is the Keyword String that needs to be appended to the
     *                    Search string in SearchView
     */
    @Override
    public void onKeywordFilterSelected(String filterValue) {
```

```
    //Appending the Keyword Filter selected to the Search Query: START
    String searchQuery = mSearchView.getQuery().toString();
    mSearchView.setQuery(TextUtils.concat(searchQuery, filterValue), false);
    //Appending the Keyword Filter selected to the Search Query: END

    //Acquiring the focus on SearchView
    mSearchView.requestFocus();

    //Setting the Dialog state to False as the Dialog is dismissed after selection
    mKeywordFiltersViewState = false;
}

/**
 * Called when a tab enters the selected state.
 *
 * @param tab The tab that was selected
 */
@Override
public void onTabSelected(TabLayout.Tab tab) {
    Log.d(LOG_TAG, "onTabSelected: Started");

    //Fix added to correct the Position pointed by the ViewPager: START
    //(Directly touching the tab instead of swiping can result in this)
    int newPosition = tab.getPosition();
    if (mViewPager.getCurrentItem() != newPosition) {
        //When position is incorrect, restore the position using the tab's position
        mViewPager.setCurrentItem(newPosition);
    }
    //Fix added to correct the Position pointed by the ViewPager: END

    //Retrieving the Custom View of Tab
    View customView = tab.getCustomView();
    if (customView != null) {
        //Finding the TextView to make it Visible
        TextView tabTextView = customView.findViewById(R.id.tab_text_id);
        //Making the Text Visible
        tabTextView.setVisibility(View.VISIBLE);
    }
```

```
        //When the user had searched something previously
        if (!TextUtils.isEmpty(mSearchQueryStr)) {
            //Retrieving the current RecyclerViewFragment
            RecyclerViewFragment fragment = getFragmentByPositionFromViewPager(newPosition);
            //Registering the OnAdapterItemDataSwapListener for the current tab
            fragment.registerItemDataSwapListener(this, newPosition);
            //Registering the OnPagerFragmentVerticalScrollListener for the current tab
            fragment.setOnPagerFragmentVerticalScrollListener(this);

            //Displaying the Progress Bar
            toggleProgressBarVisibility(View.VISIBLE);

            Log.d(LOG_TAG, "onTabSelected: reloading loader");
            //Reloading the result of the Search previously executed
            getSupportLoaderManager().initLoader(BooksLoader.BOOK_SEARCH_LOADER, null,
this);
        }

    }

    /**
     * Called when a tab exits the selected state.
     *
     * @param tab The tab that was unselected
     */
    @Override
    public void onTabUnselected(TabLayout.Tab tab) {
        //Retrieving the tab position
        int oldPosition = tab.getPosition();
        //Retrieving the Custom View of Tab
        View customView = tab.getCustomView();
        if (customView != null) {
            //Finding the TextView to Hide
            TextView tabTextView = customView.findViewById(R.id.tab_text_id);
            //Hiding the Text
            tabTextView.setVisibility(View.GONE);
        }

        //Saving the value of the position of the first Adapter item, last visible in the tab unselected
```

```
    RecyclerViewFragment fragment = getFragmentByPositionFromViewPager(oldPosition);
    mVisibleItemViewPosition = fragment.getFirstVisibleItemPosition();

    //Unregistering the OnAdapterItemDataSwapListener on this tab unselected
    fragment.clearItemDataSwapListener(oldPosition);
    //Unregistering the OnPagerFragmentVerticalScrollListener on this tab unselected
    fragment.clearOnPagerFragmentVerticalScrollListener();
  }

  /**
   * Called when a tab that is already selected is chosen again by the user. Some applications
   * may use this action to return to the top level of a category.
   *
   * @param tab The tab that was reselected.
   */
  @Override
  public void onTabReselected(TabLayout.Tab tab) {
    //Scroll to Top when the current tab is reselected
    scrollToItemPosition(0, false);
  }

  /**
   * Instantiate and return a new Loader for the given ID.
   *
   * @param id   The ID whose loader is to be created.
   * @param args Any arguments supplied by the caller.
   * @return Return a new Loader instance that is ready to start loading.
   */
  @NonNull
  @Override
  public Loader<List<BookInfo>> onCreateLoader(int id, Bundle args) {
    return new BooksLoader(this, mSearchQueryStr);
  }

  /**
   * Called when a previously created loader has finished its load.
   * This is where we display the Book volumes extracted for the Search done.
   *
   * @param loader    The Loader that has finished.
```

```
    * @param bookInfos List of {@link BookInfo} objects representing the book volumes extracted
    *                  by the Loader.
    */
   @Override
   public void onLoadFinished(@NonNull Loader<List<BookInfo>> loader, List<BookInfo>
bookInfos) {
      switch (loader.getId()) {
         case BooksLoader.BOOK_SEARCH_LOADER:
            if (bookInfos != null && bookInfos.size() > 0) {
               //Loading the data to the RecyclerViewFragment when present
               RecyclerViewFragment fragment = getCurrentFragmentFromViewPager();
               fragment.loadNewData(bookInfos);
               Log.d(LOG_TAG, "onLoadFinished: data loaded by - " + fragment);
            } else {
               //When the data returned is NULL or Empty
               BooksLoader = (BooksLoader) loader;

               if (!booksLoader.getNetworkConnectivityStatus()) {
                  //Reporting Network Failure when False

                  //Clearing the data displayed by the current RecyclerViewFragment
                  RecyclerViewFragment fragment = getCurrentFragmentFromViewPager();
                  fragment.clearData();

                  //Hiding the Progress Bar
                  toggleProgressBarVisibility(View.GONE);

                  //Sending the Network Error message to the handler to display the dialog

mNetworkErrorHandler.sendEmptyMessage(NetworkErrorHandler.NW_ERR_DIALOG);

               } else {
                  //When there is NO network issue and the current page has no data to be shown

                  //Retrieving the preference key string of 'Page to Display' setting, that is, the
'startIndex'
                  String startIndexPrefKeyStr = getString(R.string.pref_page_to_display_key);
```

```
                //Retrieving the preference key string of the Max value of 'Page to Display' setting,
that is, the 'endIndex'
                String endIndexPrefKeyStr =
getString(R.string.pref_page_to_display_max_value_key);

                //Retrieving the 'startIndex' (Page to Display) setting value
                int startIndex = mPreferences.getInt(startIndexPrefKeyStr,
                    getResources().getInteger(R.integer.pref_page_to_display_default_value));

                //Retrieving the 'endIndex' preference value
                int endIndex =
mPreferences.getInt(getString(R.string.pref_page_to_display_max_value_key),
                    startIndex);

                //Retrieving the value of preference for the last viewed page index
                int lastViewedPageIndex =
mPreferences.getInt(getString(R.string.pref_last_displayed_page_key),
                    startIndex);

                if (startIndex == 1 && endIndex == startIndex && lastViewedPageIndex ==
startIndex) {
                    Log.d(LOG_TAG, "onLoadFinished: No data, All index is currently at " +
startIndex);
                    //When all page index says 1, it means no result has been generated for the new
query executed

                    //Displaying the No Page Result in such cases
                    manageNoResultPage(false, View.VISIBLE);

                    //Clearing the data displayed by the current RecyclerViewFragment
                    RecyclerViewFragment fragment = getCurrentFragmentFromViewPager();
                    fragment.clearData();

                    //Hiding the Progress Bar
                    toggleProgressBarVisibility(View.GONE);

                } else {
                    //Restoring the last viewed page index to the 'Page to Display' settings
                    //when this is not a new query and other page index requested, reports no result
```

```
                    if (lastViewedPageIndex == startIndex && lastViewedPageIndex > 1) {
                        //Moving back to previous page index when the 'startIndex' setting
                        //is as same as the last viewed page index
                        lastViewedPageIndex -= 1;
                    } else {
                        //(Avoiding to display the toast multiple times when both 'startIndex'
                        //setting is as same as the last viewed page index)

                        //Displaying a message on restoring the last viewed page
                        Toast.makeText(this, getString(R.string.restoring_page_msg,
lastViewedPageIndex, startIndex), Toast.LENGTH_SHORT).show();
                    }

                    Log.d(LOG_TAG, "onLoadFinished: Restoring page " + lastViewedPageIndex + "
from " + startIndex);

                    //Opening the Editor to restore the value of last viewed page index to 'Page to
Display' setting
                    SharedPreferences.Editor prefEditor = mPreferences.edit();

                    //Restoring the value of the last viewed page to 'startIndex' and 'endIndex' setting
                    prefEditor.putInt(startIndexPrefKeyStr, lastViewedPageIndex);
                    prefEditor.putInt(endIndexPrefKeyStr, lastViewedPageIndex);
                    prefEditor.apply(); //Applying the changes
                }

            }
        }
        break;
    }
}

/**
 * Called when a previously created loader is being reset, and thus
 * making its data unavailable.  The application should at this point
 * remove any references it has to the Loader's data.
 *
 * @param loader The Loader that is being reset.
 */
```

```
@Override
public void onLoaderReset(@NonNull Loader<List<BookInfo>> loader) {
    switch (loader.getId()) {
        case BooksLoader.BOOK_SEARCH_LOADER:
            //Clearing the data from the Adapter hosted by the RecyclerViewFragment
            getCurrentFragmentFromViewPager().clearData();
            break;
    }
}

/**
 * Method invoked when the data on the RecyclerView's Adapter has been swapped successfully
 */
@Override
public void onItemDataSwapped() {
    //Restoring the current position to the last viewed Adapter item position on the Fragment
    scrollToItemPosition(mVisibleItemViewPosition, false);

    //Updating the state of Pagination Buttons after the data swap
    updatePaginationButtonsState();

    //Hiding the Progress Bar
    toggleProgressBarVisibility(View.GONE);
}

/**
 * Called when a shared preference is changed, added, or removed. This
 * may be called even if a preference is set to its existing value.
 *
 * @param sharedPreferences The {@link SharedPreferences} that received
 *                          the change.
 * @param key               The key of the preference that was changed, added, or
 *                          removed
 */
public void onSharedPreferenceChanged(SharedPreferences sharedPreferences, String key) {
    if (!mKeysToExclude.contains(key)) {
        //Resetting the Adapter Item View position reference to 0 (First Item data in the adapter)
        //on Preference change
        mVisibleItemViewPosition = 0;
```

```
        //Get the active loader and trigger content change for data reload
        Loader<List<BookInfo>> loader =
getSupportLoaderManager().getLoader(BooksLoader.BOOK_SEARCH_LOADER);
        if (loader != null) {
            //Displaying the Progress Bar
            toggleProgressBarVisibility(View.VISIBLE);

            BooksLoader = (BooksLoader) loader;
            booksLoader.onContentChanged(); //Signalling the content change on the loader
        }
    }

}

/**
 * Method invoked when the ViewPager's scroll has reached
 * the last three items in its Fragment
 * {@link RecyclerViewFragment}
 *
 * @param verticalScrollAmount is the amount of vertical scroll.
 *                             If >0 then scroll is moving towards the bottom;
 *                             If <0 then scroll is moving towards the top
 */
@Override
public void onBottomReached(int verticalScrollAmount) {
    Log.d(LOG_TAG, "onBottomReached: verticalScrollAmount: " + verticalScrollAmount);
    View paginationPanelView = findViewById(R.id.pagination_panel_id);
    if (verticalScrollAmount > 0) {
        //Displaying the Pagination Panel when the scroll
        //reaches the last three items in its Fragment
        paginationPanelView.setVisibility(View.VISIBLE);
    } else {
        //Hiding the Pagination Panel when the scroll
        //moves away from the last three items in its Fragment
        paginationPanelView.setVisibility(View.GONE);
    }
}
```

```
/**
 * Called when a view has been clicked.
 *
 * @param view The view that was clicked.
 */
@Override
public void onClick(View view) {

    //Retrieving the preference key string of 'Page to Display' setting, that is, the 'startIndex'
    String startIndexPrefKeyStr = getString(R.string.pref_page_to_display_key);
    //Retrieving the 'startIndex' (Page to Display) setting value
    int startIndex = mPreferences.getInt(startIndexPrefKeyStr,
            getResources().getInteger(R.integer.pref_page_to_display_default_value));
    //Opening the Editor to update the value
    SharedPreferences.Editor prefEditor = mPreferences.edit();

    //Updating the current page index to the preference of the last viewed page index
    prefEditor.putInt(getString(R.string.pref_last_displayed_page_key), startIndex);

    //Executing the click action based on the view's id
    switch (view.getId()) {
        case R.id.page_first_button_id:
            //On Page First action, updating the 'Page to Display' setting to 1
            prefEditor.putInt(startIndexPrefKeyStr, 1);
            prefEditor.apply(); //applying the changes
            //Displaying a Toast Message
            Toast.makeText(this, getString(R.string.navigate_page_first_msg),
Toast.LENGTH_SHORT).show();
            break;

        case R.id.page_previous_button_id:
            //On Page Previous action, updating the 'Page to Display' setting
            //to a value less than itself by 1
            startIndex = startIndex - 1;
            prefEditor.putInt(startIndexPrefKeyStr, startIndex);
            prefEditor.apply(); //applying the changes
            //Displaying a Toast Message
            Toast.makeText(this, getString(R.string.navigate_page_x_msg, startIndex),
Toast.LENGTH_SHORT).show();
```

```
            break;

        case R.id.page_more_button_id:
            //On Page More action, displaying a Number Picker Dialog
            //to allow the user to make the choice of viewing a random page

            //Retrieving the Minimum and Maximum values for the NumberPicker
            int minValue = getResources().getInteger(R.integer.pref_page_to_display_default_value);
            int maxValue =
mPreferences.getInt(getString(R.string.pref_page_to_display_max_value_key),
                    minValue);

            //Creating the DialogFragment Instance
            PaginationNumberPickerDialogFragment numberPickerDialogFragment
                    = PaginationNumberPickerDialogFragment.newInstance(minValue, maxValue);
            //Displaying the DialogFragment
            numberPickerDialogFragment.show(getSupportFragmentManager(),
                    PaginationNumberPickerDialogFragment.DIALOG_FRAGMENT_TAG);
            break;

        case R.id.page_next_button_id:
            //On Page Next action, updating the 'Page to Display' setting
            //to a value greater than itself by 1
            startIndex = startIndex + 1;
            prefEditor.putInt(startIndexPrefKeyStr, startIndex);
            prefEditor.apply(); //applying the changes
            //Displaying a Toast Message
            Toast.makeText(this, getString(R.string.navigate_page_x_msg, startIndex),
Toast.LENGTH_SHORT).show();
            break;

        case R.id.page_last_button_id:
            //On Page Last action, updating the 'Page to Display' setting to
            //a value equal to that of the predetermined 'endIndex' preference value
            prefEditor.putInt(startIndexPrefKeyStr,
                    mPreferences.getInt(getString(R.string.pref_page_to_display_max_value_key),
                            startIndex));
            prefEditor.apply(); //applying the changes
            //Displaying a Toast Message
```

```
            Toast.makeText(this, getString(R.string.navigate_page_last_msg),
Toast.LENGTH_SHORT).show();
            break;

        }

    }

    /**
     * Method that displays the Network Error Dialog
     */
    private void showNetworkErrorDialog() {

        //Retrieving the FragmentManager
        FragmentManager = getSupportFragmentManager();
        //Retrieving the instance of the Dialog to be shown through the FragmentManager
        NetworkErrorDialogFragment dialogFragment =
            (NetworkErrorDialogFragment)
fragmentManager.findFragmentByTag(NetworkErrorDialogFragment.DIALOG_FRAGMENT_TA
G);

        if (dialogFragment == null) {
            //When there is no instance attached, that is the dialog is not active

            //Creating an instance of the Network Error dialog and displaying the same
            dialogFragment = NetworkErrorDialogFragment.newInstance();
            dialogFragment.show(fragmentManager,
NetworkErrorDialogFragment.DIALOG_FRAGMENT_TAG);
        }

    }

    /**
     * Custom {@link Handler} class for displaying the Network Error when it occurs
     * during requesting data
     */
    private static class NetworkErrorHandler extends Handler {
        //Constant set for handling the Network Error Message
        private static final int NW_ERR_DIALOG = 503;
```

```
    //Storing weak reference to the outer activity
    private final WeakReference<BookSearchActivity> mActivity;

    /**
     * Constructor of this {@link NetworkErrorHandler}
     *
     * @param activity is the Activity instantiating this Handler
     */
    NetworkErrorHandler(BookSearchActivity activity) {
      mActivity = new WeakReference<>(activity);
    }
    @Override
    public void handleMessage(Message msg) {
      BookSearchActivity = mActivity.get();
      if (bookSearchActivity != null
            && msg.what == NW_ERR_DIALOG) {
        //When the attached activity is alive
        //and the message is for Network Error Message

        //Display the Network Error Dialog
        bookSearchActivity.showNetworkErrorDialog();

      } else {
        //For all else, propagating the call to super
        super.handleMessage(msg);
      }
    }
  }

}
```

CHAPTER - 3 :

RESULT ANALYSIS

3.1 Snapshots :

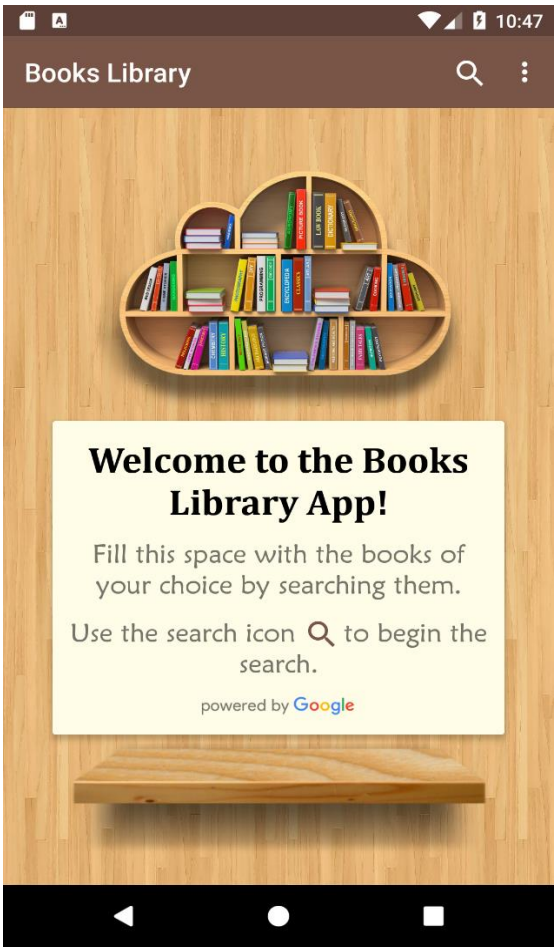

Fig . 3.1.1

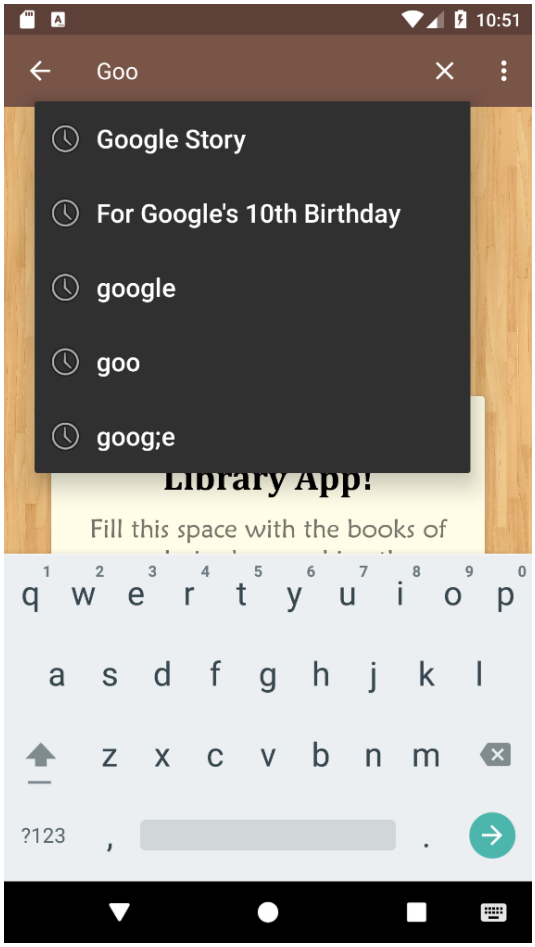

Fig 3.1.2

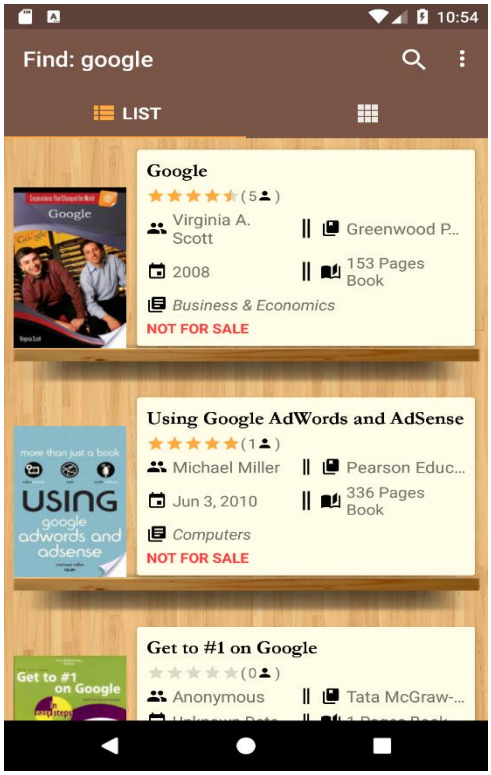

Fig 3.1.3

Fig 3.1.4

Fig 3.1.5

Fig 3.1.6

Fig 3.1.7

Fig 3.1.8

3.2 RESULTS:

- **Fig 3.1.1 : The Welcome screen layout**

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

- **Fig 3.1.2 : Assisted Search**

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

- **Fig 3.1.3 :**

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 as shown below.

- **Fig 3.1.4 :**

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 success of this action, a toast will be displayed.

- **Fig 3.1.5 and Fig 3.1.6:**

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 the LIST Tab. In this the results are displayed like in a 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 (of 2 columns).

- **Fig 3.1.7 and Fig 3.1.8 :**

Clicking on an item in the List Tab View or the Grid Tab View, opens up the Details page activity_book_detail.xml inflated by the activity BookDetailActivity. 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.

CHAPTER - 4 :

CONCLUSION AND FUTURE WORK

4.1 Conclusion:

The development of the Book Library application is not an easy task. In this project we present the main steps in development of application of Book Library using the external Link or Url for android. By this system Book Library Application generation becomes easy. Less chances of malfunctioning are there. The Application has reached a steady state but still improvements are to be made. The Application is operated at a high level of efficiency and all the work and user associated with the Application understand its advantage. It was intended to solve as requirement specification. In future this Application can be implemented to all over the world and will be designed for cross platform.

4.2 Future Enhancement :

There is always a room for improvements in any software package, however good and efficient it may be done. But the most important thing is it should be flexible to accept further modifications. Right now we are just dealing with the Grid and listing representation . Further we would be including the Database to download it as a PDF and many other features to make it go with the new and trending technology. Thus implementing the further enhancements will make the project more flexible and also ease for the users.

CHAPTER - 5 :

REFERENCES

- https://www.udacity.com
- https://github.com
- https://www.geeksforgeeks.org
- https://www.youtube.com
- https://developer.android.com