Homework 9
Android App for eBay Catalog Search

Prof. Marco Papa
Summer 2020 Semester

Developed and Designed by Rushabh Kapadia

1. Objectives

- Become familiar with Java, JSON, Android Lifecycle and Android Studio for Android app development.
- Build a good-looking Android app.
- Learn the essentials of Google's Material design rules for designing Android apps
- Learn to use the eBay Finding APIs and the Android SDK.
- Get familiar with third party libraries like Picasso, Glide and Volley.

The objective is to create an Android application as specified in the document below and in the video available here on YouTube.

2. Background

2.1 Android Studio

Android Studio is the official Integrated Development Environment (IDE) for Android application development, based on IntelliJ IDEA - a powerful Java IDE. On top of the capabilities you expect from IntelliJ, Android Studio offers:

- Flexible Gradle - based build system.
- Build variants and multiple apk file generation.
- Code templates to help you build common app features.
- Rich layout editor with support for drag and drop theme editing.
- Lint tools to catch performance, usability, version compatibility, and other problems.
- ProGuard and app-signing capabilities.
- Built-in support for Google Cloud Platform, making it easy to integrate Google Cloud Messaging and App Engine.

More information about Android Studio can be found at:
http://developer.android.com/tools/studio/index.html

2.2 Android

Android is a mobile operating system initially developed by Android Inc. a firm purchased by Google in 2005. Android is based upon a modified version of the Linux kernel. As of Nov 2018, Android was the number 1 mobile OS, in unit sales, surpassing iOS, while iOS was still the most profitable platform.

The Official Android home page is located at:
http://www.android.com/

The Official Android Developer home page is located at:
http://developer.android.com/

3. Prerequisites

This homework requires the use of the following components:

- Download and install Android Studio. Technically, you may use any IDE other than Android Studio such as Eclipse, but the latest SDKs may not be supported with Eclipse. We will not be providing any help on problems arising due to your choice of alternate IDEs.
- You must use the **emulator**. Everything should just work out of the box.

- If you are new to Android Development, the Hints are going to be your best friend!

4. High Level Design

This homework is a mobile app extended version of Homework 6 and Homework 8.
In this exercise, you will develop an Android application, which allows users to search through the eBay items catalog and look at the detailed information about them. The App contains 5 screens: Initial Form, Item Catalog, Product Information screen, Seller Information screen and Shipping Information screen.

This homework uses 2 eBay API calls. The *findItemsAdvanced* API call is the same as in HW8. So, you can use the same Node.js backend as HW8 with an additional call added as a new endpoint. In case you need to change something in

Node, make sure you do not break your Angular assignment (or deploy a separate copy) as the grading for homework will not be finished at least until 1 week after the due date.

To ensure the same level of effort to all students, use of Java is required. Kotlin is not allowed.

PS: This app has been designed and implemented in a Pixel 2XL emulator by using SDK API 29. It is highly recommended that you use the same virtual device and API to ensure consistency in the grading.

Demo will be on an Android emulator, using Zoom Remote Control, see the rules:

https://csci571.com/courseinfo.html#homeworks

5. Implementation

5.1 App Icon and Splash Screen

In order to get the app icon/image, please see the **hints** section. The app begins with a welcome screen (see **Figure 2**) which displays the icon provided in the hint section.

This screen is called Splash Screen and can be implemented using many different methods. The simplest method consists of creating a resource file for launcher screen

and adding it as a style to *AppTheme.Launcher* (see the **hints**). This image is also the app icon as shown in **Figure 1**.

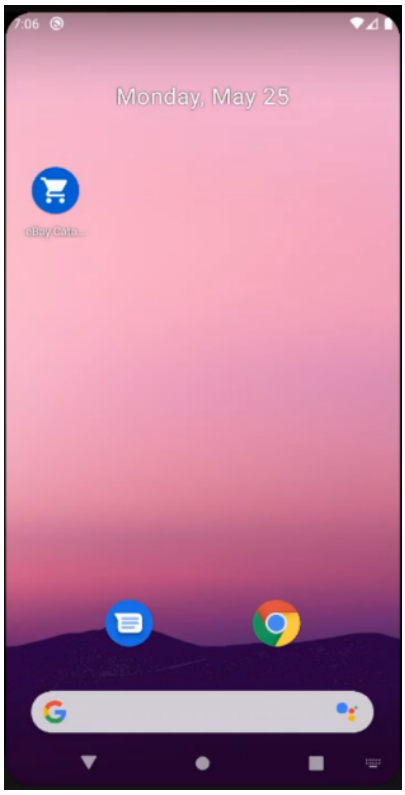

Figure 1: App Icon

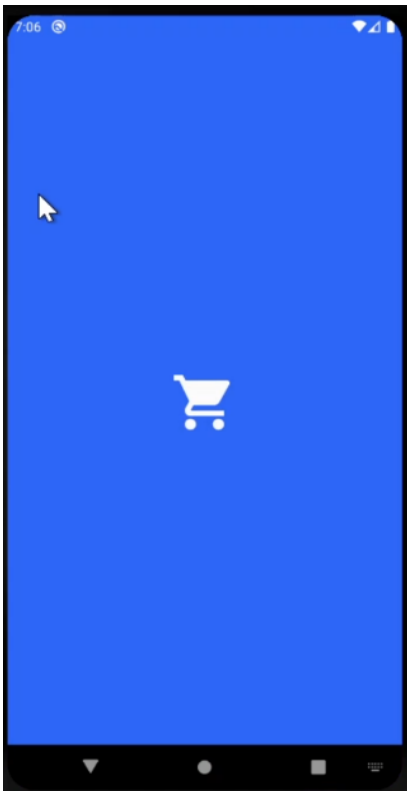

Figure 2: Splash Screen

5.2 Initial Search Form

When you open the app, there will be an initial form displayed on the screen as shown in Figure 3. The initial form contains the same filters that were used in previous homework assignmwnta. The fields are:

- Keyword field – Mandatory
- Price Range – Minimum and Maximum Price fields
- Condition – 3 checkboxes, for *New, Used* and *Unspecified*
- Sort By criteria – 4 options, name *Best Match, Price: Highest first, Price + Shipping: Highest first, Price + Shipping: Lowest first*

The form also performs the following validations:

- **Keyword** field should not be empty
- **Price Range** fields should not allow negative values. Minimum price should be less than Maximum price. Both the fields are optional and can be applied independent of the other as well. (Same as homework #6)
- **Condition** fields are optional. They can be checked or not checked
- **Sort By** dropdown should have **Best Match** value as default

The fields can be implemented using the **EditText, TextView, CheckBox and Spinner UI** components available in android SDK. The keypad for entering price fields should be displayed as shown in the right-most image of **Figure 3**. Also note the **toast** message at the bottom in case of errors in the form.

The **Search** button triggers the field checking and fetching of the results using the *findItemsAdvanced* eBay API. The **Clear** button restores the form to the initial (default) state.

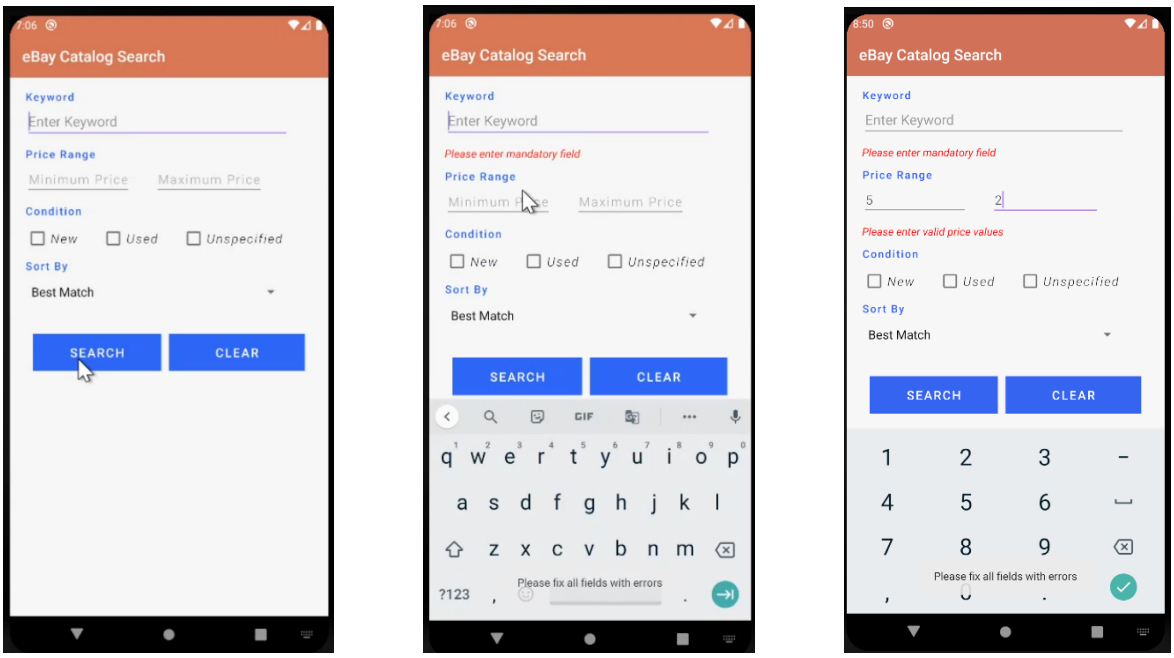

Figure 3: Search Form and Error validation conditions

5.3 Item Catalog Screen

On submitting the form, the loading symbol should be displayed while the results are being fetched (see **Figure 4**). If there are results available, a **maximum of 50 listings** should be displayed in a card format as shown in **Figure 5**. If less than 50 items are retrieved, display only that many items. Notice the items retrieved count at the top for the given keyword. The count should be equal to the number of items received from the Node server.

The top action bar should have the 'Search Results' title and the **back button** to go back to the search form screen (which has the filter values that were used for the current search). In case no items are returned, the screen should show **No Records** centered on the screen and a **toast** as well (see Figure 6).

The search catalog should also **refresh** when vertically swiped. This must be done by making a new REST call to the node server and fetching the products again (Refer to the video to understand the functionality).

The components used to create this screen are *FrameLayout, ProgressBar, TextView, ScrollView* to make the list scrollable*, SwipeRefreshLayout* (covered in hints), *CardView* (for each item card), *RecyclerView* to show the grid list and toast. The text inside the *TextView* elements should be formatted using HTML formatting applied to the *TextView* content.

The API response for the given API Endpoint is the **same as Homework 8.**

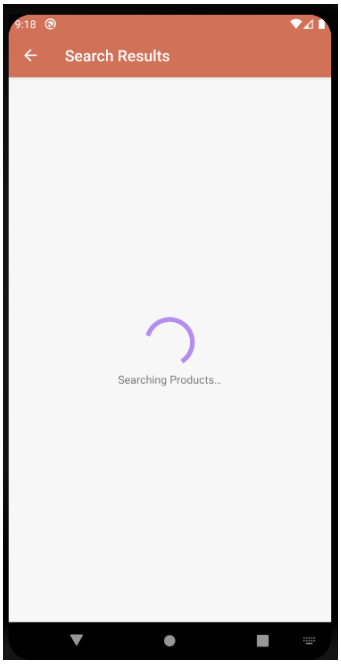

Figure 4: Loading Screen

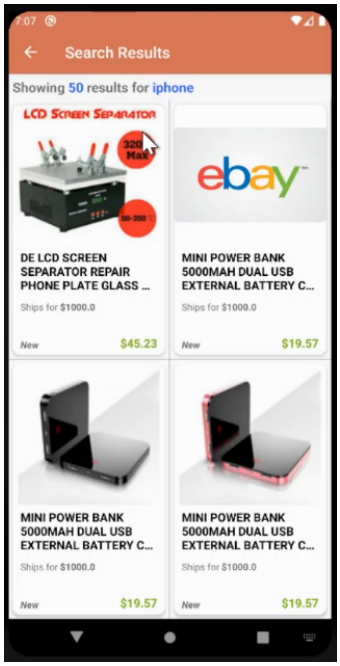

Figure 5: Catalog Screen

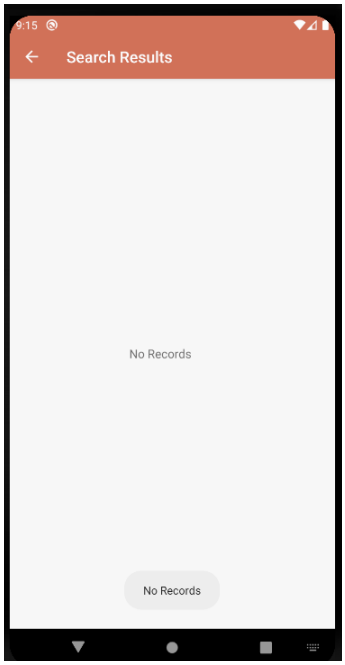

Figure 6: No Result Screen

5.3.1 Item Card

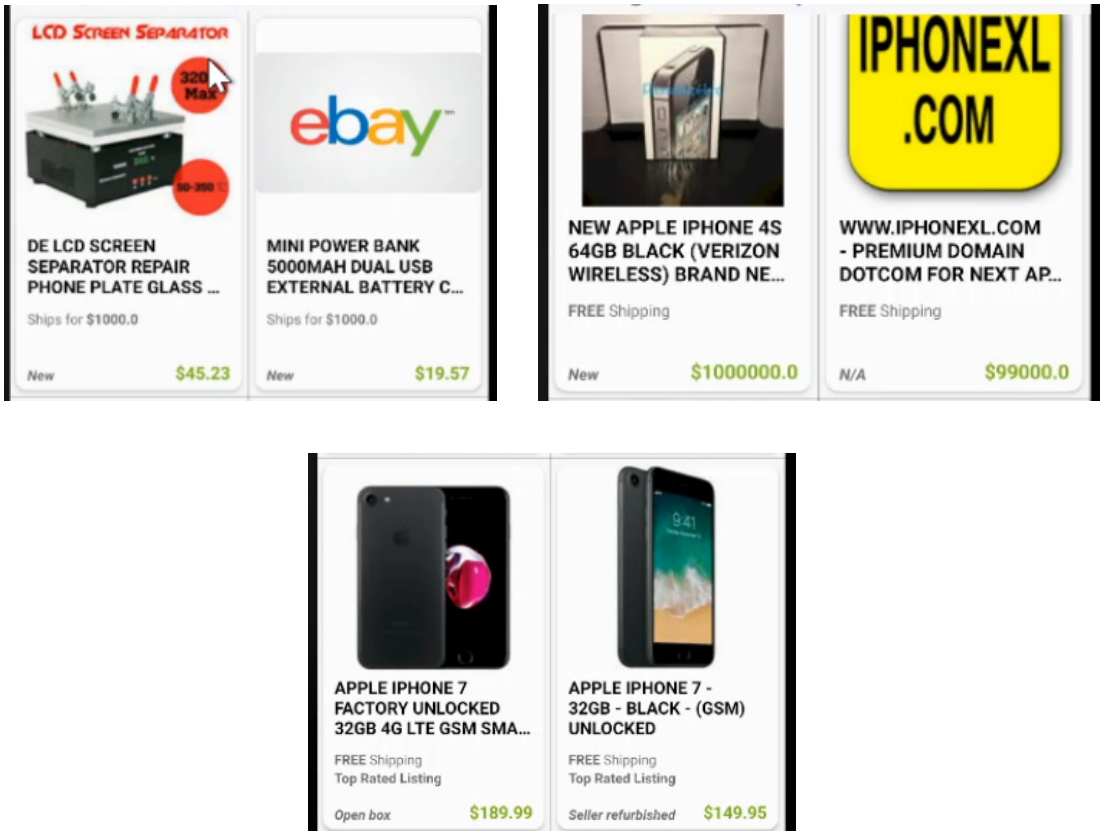

Figure 7: Item cards

An item card uses *ImageView and TextViews* for displaying information. The images should use *fitCenter* scale type to crop the image correctly. The CardView should display rounded corners as shown in **Figure 7**. The item card displayed on the catalog screen uses the following fields from the JSON response:

Item Information in the Result Card	**Tags in eBay JSON response**
Item image URL	*searchResult->item->galleryURL (Use the default image if the image has the URL -* https://thumbs1.ebaystatic.com/ pict/04040_0.jpg*.)*
Item Title	*searchResult->item->title (*The title should only span a maximum of 3 lines. If it exceeds 3 lines, ellipsize the title string. Refer hints section for help*)*
Condition of the item	*searchResult->item->condition->conditionDisplayName (If the category field is empty, display* ***N/A.*** *If any other value is available, display that condition)*

Item Top Rated Image (If the current listing is top rated for the given item)	if the value of *searchResult-->item->topRatedListing* is true, write "**Top Rated Listing**" below the shipping information.
Free Shipping available?	If the value of the tag *searchResult-->item->shippingInfo->shippingServiceCost* is equal to 0.0, write "**Free** Shipping" as show in Figure 7. Otherwise, write "Ships for **$XY**". (Refer video for styling)
Price	The price value is read from *searchResult-> item->sellingStatus-> convertedCurrentPrice*

Table 1: JSON Fields to Card Mapping

5.4 Detailed Product Screen

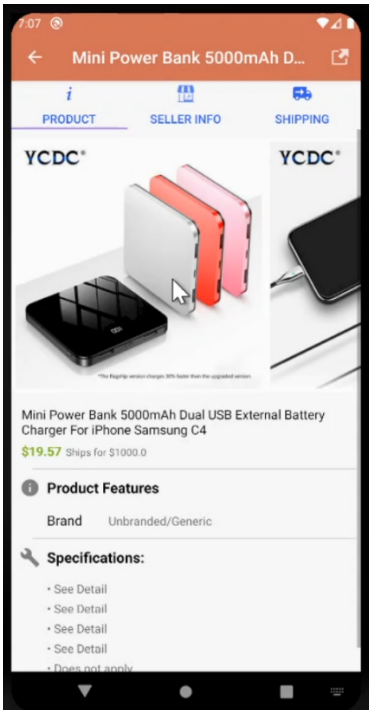

Figure 8: Product Summary Tab

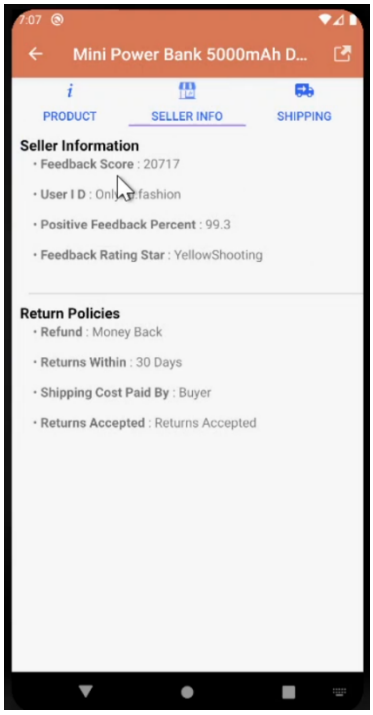

Figure 9: Seller Information Tab

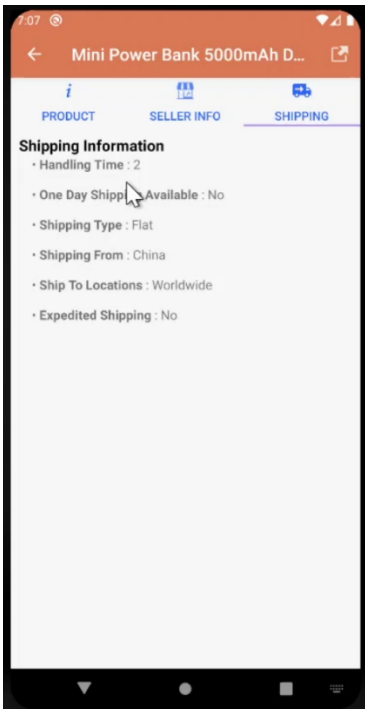

Figure 10: Shipping Information Tab

On clicking the Item card, a new *Android Activity* should be opened and a call to the detailed item API (*GetSingleItem* eBay API) should be made via the node server. While fetching the data, the screen should show a **progress bar** similar to **Figure 4**. After fetching the data, 3 tabs will be displayed on the screen: *PRODUCT, SELLER INFO and SHIPPING.* To implement this functionality, the *ViewPager* and *TabLayout* components

should be used. All the 3 tabs are vertically scrollable (use *ScrollView* functionality). The top action bar should have a **back button** which goes back to the product catalog screen (works like a back press), the **title** of the product and a **redirect** icon which opens the product link on the eBay website (see the hints section for help) in Google Chrome on the emulator. In case the *viewItemURL* is invalid, the default URL link (eBay home page) should be used.

To fetch the details about the product, the *GetSingleItem* API is used. The eBay API endpoint URL will only have 1 field (**productID**) that has to be provided by the Android app to the node server, as shown below. **YourAppKey** is your eBay key.

http://open.api.ebay.com/shopping?callname=GetSingleItem&responseencoding=JSON&appid=**YourAppKey**&siteid=0&version=967&ItemID=**productID**&IncludeSelector=Description,Details,ItemSpecifics

For all the products, all the other fields used in the above URL will stay the same.
The **productID** field is the *searchResult->item->itemId* field from the *getItemsAdvanced* response for the clicked *item*.

A sample JSON response for a given ItemID is shown in **Figure 11**.

http://open.api.ebay.com/shopping?callname=GetSingleItem&responseencoding=JSON&appid=**YourAppKey**&siteid=0&version=967&ItemID=254603172743&IncludeSelector=Description,Details,ItemSpecifics

Response –


```
{
   Timestamp: "2020-06-13T06:17:11.125Z",
   Ack: "Success",
   Build: "E1141_CORE_APILW_19170841_R1",
   Version: "1141",
 - Item: {
      BestOfferEnabled: false,
      Description: "<p>Figure is complete and like new! Kept on display from a kid free home.</p>",
      ItemID: "254603172743",
      EndTime: "2020-05-22T10:32:42.000Z",
      StartTime: "2020-05-21T01:29:22.000Z",
      ViewItemURLForNaturalSearch: "https://www.ebay.com/itm/Harry-Potter-Wizarding-World-Ron-Weasley-Jointed-Doll-Deboxed-Complete-/254603172743",
      ListingType: "FixedPriceItem",
      Location: "Manchaca, Texas",
    + PaymentMethods: […],
      GalleryURL: "https://thumbs4.ebaystatic.com/pict/2546031727438080_1.jpg",
    + PictureURL: […],
      PostalCode: "786**",
      PrimaryCategoryID: "29798",
      PrimaryCategoryName: "Collectibles:Fantasy, Mythical & Magic:Harry Potter",
      Quantity: 1,
    + Seller: {…},
      BidCount: 0,
    + ConvertedCurrentPrice: {…},
    + CurrentPrice: {…},
    + HighBidder: {…},
      ListingStatus: "Completed",
      QuantitySold: 1,
    + ShipToLocations: […],
      Site: "US",
      TimeLeft: "PT0S",
      Title: "Harry Potter Wizarding World Ron Weasley Jointed Doll Deboxed Complete",
    + ItemSpecifics: {…},
      HitCount: 43,
      PrimaryCategoryIDPath: "1:10860:29798",
    + Storefront: {…},
      Country: "US",
    + ReturnPolicy: {…},
      AutoPay: true,
      PaymentAllowedSite: [ ],
      IntegratedMerchantCreditCardEnabled: false,
      HandlingTime: 2,
      ConditionID: 3000,
      ConditionDisplayName: "Used",
      GlobalShipping: false,
      QuantitySoldByPickupInStore: 0,
      NewBestOffer: false
   }
}
```

Figure 11: GetSingleItem JSON response

5.4.1 Product Summary Tab

When the data is fetched, the default tab is the Product tab which gives a highlight of the product's features (see **Figure 7**). The tab contains an image gallery of the product at the top (use *HorizontalScrollView* and multiple *ImageView* components dynamically appended into *LinearLayout* component). The images displayed are from the PictureURL field in the JSON response (see **Figure 11**).

The **title**, **price** and **shipping information** fields are obtained from the *GetItemsAdvanced* JSON response for this item (the JSON object which was used for displaying Item cards on Item Catalog Screen). Refer to the video to see the styling for these fields. The styling is different on the product summary tab for these fields from the Item Card.

After these fields, there is a horizontal divider to mark the start of new section. The **Product Features** section heading should be as shown in **Figure 7**. This section should be visible on the screen only if the *Subtitle* and/or the *Brand* fields are present (if both the fields are present, *Subtitle* should be first). If both fields are missing, **do not display this section.** The Subtitle is available in the JSON response (see **Figure 11**). The Brand field should be available in the *ItemSpecifics* field of the JSON response (see **Figure 11**). An example of the Brand field is shown in **Figure 12**.

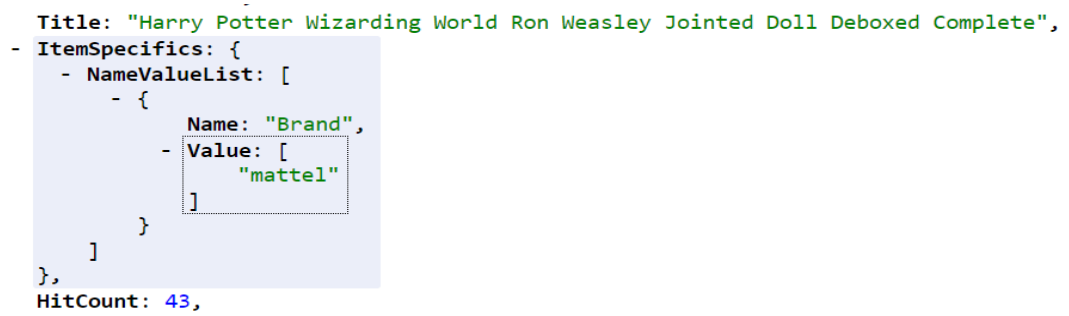

Figure 12: ItemSpecifics Field in JSON response

After the **Product Features** section, there is a horizontal divider to mark the start of new section. The **Specifications** section heading should be as shown in **Figure 7**. The Specifications section will display the Name, Value fields of each object in the *ItemSpecifics->NameValueList* array (except the object containing Name = Brand since it is displayed as part of Product Features section). If the *NameValueList* array is absent or empty, this section should not be visible. A maximum of 5 objects in the *NameValueList* should be displayed (5 objects which do not include the object

containing Name = Brand). Notice the **bullet points** for these fields in **Figure 7**. Use HTML formatting in the *TextView* component to display the bullet points.

Note: All the Images needed to display the section heading icons as well as the 3 tabs are provided in the zip file for Homework #9.

5.4.2 Product Seller Information Tab

This tab contains 2 sections separated by a horizontal divider. The ***Seller Information*** section uses all the key:value pairs from the JSON object under the “Seller" key in the JSON response from *GetSingleItem* API (Figure 11). The ***Return Policies*** section uses all the keys from the JSON Object under the “ReturnPolicy” key in the JSON response from the *GetSingleItem* API. The keys in the *Seller and ReturnPolicy keys* have to be separated using at words. For example, the “ReturnsWithin” key will be displayed as “Returns Within” (break the words in the key at every capital letter). The styling should be as shown in **Figure 8**. Use HTML formatting in TextView to style the text correctly.

5.4.3 Product Shipping Information Tab

This tab contains a single section of ***Shipping information***. This section uses all the key:value pairs from the JSON object under the “shippingInfo” key from the JSON response of *GetItemsAdvanced* API (the JSON object for the item retrieved on the catalog screen). The styling should be as shown in **Figure 10** and the video.

Note: For the detailed product description using 3 tabs, the data from both the JSON responses (GetItemsAdvanced and GetSingleItem) has to be used. Make sure you send the data from the item catalog screen to the product description screen rather than making a new API call to the GetItemsAdvanced endpoint.

Note: The product listing should only be removed if any of the fields related to price, title, shipping and seller information is missing. The other cases have been covered in the respective sections and the listing should not be removed.

5.5 Swipe Refresh

The **Item Catalog** screen is the place where pull to refresh functionality is to be implemented as shown in **Figure 13**.

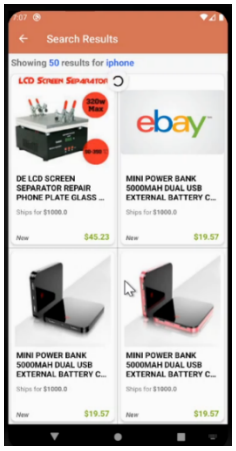

Figure 13: Pull to refresh

The idea is to **re-fetch item listings** and update the catalog in the Recycler View on the current tab whenever the pull to refresh is used. Refer to the video for a better understanding.

5.6 Progress bar

Every time the user has to wait before they can see the data, you must display a progress bar as shown in **Figure 4**. The progress bar needs to be present across **Catalog screen, Detailed Product screen** and just says “Searching Products…”. One possibility would be to display the progress bar by default (where needed) and then hide it as soon as the data is received/view is prepared. See the hints for progress bar related ideas and styling.

Note: Based on your implementation, you might need to put progress bars on different places. During the demo, there should NOT be any screen with default/dummy/ placeholder values or an empty screen.

5.7 Summary of detailing and error handling

1. Make sure there are no conditions under which the app crashes.
2. Make sure all icons and text are correctly positioned as in the video/screenshots.
3. Make sure the screens and toasts are correctly displayed.
4. Make sure there is a progress bar every time there is nothing to show as data is being retrieved.
5. All API calls need to be made using Node.js backend.

5.8 Additional Info

For things not specified in the document, grading guidelines, or the video, you can make your own decisions. But keep in mind about the following points:

- Always display a proper message and don't crash if an error happens.
- You can only make HTTP/HTTPS requests to your backend Node.js on GAE.
- All HTTP requests should be asynchronous and should not block the main UI thread. You can use third party libraries like Volley to achieve this in a simple manner.

6. Implementation Hints

6.1 Icons

6.1.1 Icons

The images used in this homework are available in the ZIP file mentioned in the Piazza post for Homework #9. You can choose to work with xml/png/jpg versions. We recommend using xml as it is easy to modify colors by setting the Fill Colors.

6.2 Third party libraries

Sometimes using 3rd party libraries can make your implementation much easier and quicker. Some libraries you may have to use are listed in the following sections.

6.2.1 Volley HTTP requests

Volley can be helpful with asynchronous http requests to load data. You can also use Volley network ImageView to load photos in Google tab. You can learn more about them here:

https://developer.android.com/training/volley/index.html

6.2.2 Picasso

Picasso is a powerful image downloading and caching library for Android. See:

http://square.github.io/picasso/

If you decide to use RecycleView to display the photos with Picasso Please use version 2.5.2 since latest version does not support RecycleView well. See:

https://github.com/codepath/android_guides/wiki/Displaying-Images-with-the-Picasso-Library

6.2.3 Glide

Glide is also powerful image downloading and caching library for Android. It is similar to Picasso. You can also use Glide to load photos in Google tab. See:

https://bumptech.github.io/glide/

6.3 Working with action bars and menus

https://developer.android.com/training/appbar/setting-up
https://stackoverflow.com/questions/38195522/what-is-oncreateoptionsmenumenu-menu

6.4 Displaying Progress Bars

https://stackoverflow.com/a/28561589
https://stackoverflow.com/questions/5337613/how-to-change-color-in-circular-progress-bar

6.5 Implementing Splash Screen

There are many ways to implement a splash screen. This blog highlights almost all of them with examples. See:

https://android.jlelse.eu/the-complete-android-splash-screen-guide-c7db82bce565

6.6 Adding the App Icon

https://dev.to/sfarias051/how-to-create-adaptive-icons-for-android-using-android-studio-459h

6.7 Adding ellipsis to long strings

https://stackoverflow.com/questions/6393487/how-can-i-show-ellipses-on-my-textview-if-it-is-greater-than-the-1-line

6.8 Adding tabs in Android

https://www.spaceotechnologies.com/create-multiple-tabs-using-android-tab-layout/

6.9 Adding a button to ActionBar

https://developer.android.com/training/appbar/actions
https://stackoverflow.com/questions/12070744/add-back-button-to-action-bar

6.10 Implementing a Recycler view in android

https://developer.android.com/guide/topics/ui/layout/recyclerview
https://stackoverflow.com/a/40217754
https://abhiandroid.com/materialdesign/recyclerview-gridview.html

6.11 Adding Toasts

https://stackoverflow.com/questions/3500197/how-to-display-toast-in-android

6.12 To implement pull to refresh feature

https://medium.com/@houdayer.corentin/how-to-use-swipe-to-refresh-swiperefreshlayout-99abd674c907

6.13 Passing variables to intent

https://stackoverflow.com/questions/2405120/how-to-start-an-intent-by-passing-some-parameters-to-it

6.14 Spinner Dropdown for Sort By criteria

https://www.javatpoint.com/android-spinner-example
https://developer.android.com/guide/topics/ui/controls/spinner

6.15 Formatting Text using HTML in TextView

https://stackoverflow.com/a/27462961

6.16 Image Gallery to display Product Images

https://www.tutlane.com/tutorial/android/android-scrollview-horizontal-vertical-with-examples
https://www.tutorialspoint.com/how-to-implement-horizontalscrollview-like-gallery-in-android

6.17 Open Product Link in browser

https://www.tutorialkart.com/kotlin-android/android-open-url-in-browser-activity/

6.18 Back press behavior on Back button

https://stackoverflow.com/a/27807976

7. Files to Submit

You should also ZIP all your source code (the java/ and res/ directories excluding the vector drawables that were downloaded) and submit the resulting ZIP file only if requested by the course staff.

Unlike other semesters, you will have to demo your submission using **Zoom Remote Control** during a special grading session. Details and logistics for the project demonstration will be provided in class, on the Announcement page and on Piazza. **The Final Project Demonstration is done a laptop/ notebook/MacBook or Windows PC using the emulator, and not a physical mobile device.**

****IMPORTANT****

All videos are part of the homework description. All discussions and explanations on Piazza related to this homework are part of the homework description and will be accounted into grading. So please review all Piazza threads before finishing the assignment.