How to Use this Template

1. Make a copy [File → Make a copy...]
2. Rename this file: "**Capstone_Stage1**"
3. Replace the text in green

Submission Instructions

1. After you've completed all the sections, download this document as a PDF [File → Download as PDF]
2. Create a new GitHub repo for the capstone. Name it "**Capstone Project**"
3. Add this document to your repo. Make sure it's named "**Capstone_Stage1.pdf**"

GitHub Username: github.com/Hishmad

Stockita POS

Description

The **point of sale** (**POS**) or **point of purchase** (**POP**) is the the application where a retail transaction is completed. It is the point at which a customer makes a transaction or payment to the merchant in exchange for goods or after provision of a service.
This application does the need for price tags, selling prices are linked to the product code of an item when adding Master Item, so the cashier merely needs to scan barcode to process a sale or lookup

from the dialog. If there is a price change, this can also be easily done through the detail of Item Master. Other advantages include ability to implement discounts, tax and more efficient Sales and Purchase records.

The main goal for this app is that it stores the data in the cloud, and it can be used among multi users to share the same data, that makes it easy for business owners to have more than one cashier, like one to many relationship.

Intended User

This application is intended the most for shop owner and their employee, like restaurants and cafes or mini market, but because it is a mobile app and storing the data in the cloud also it is very useful for exhibition booth.

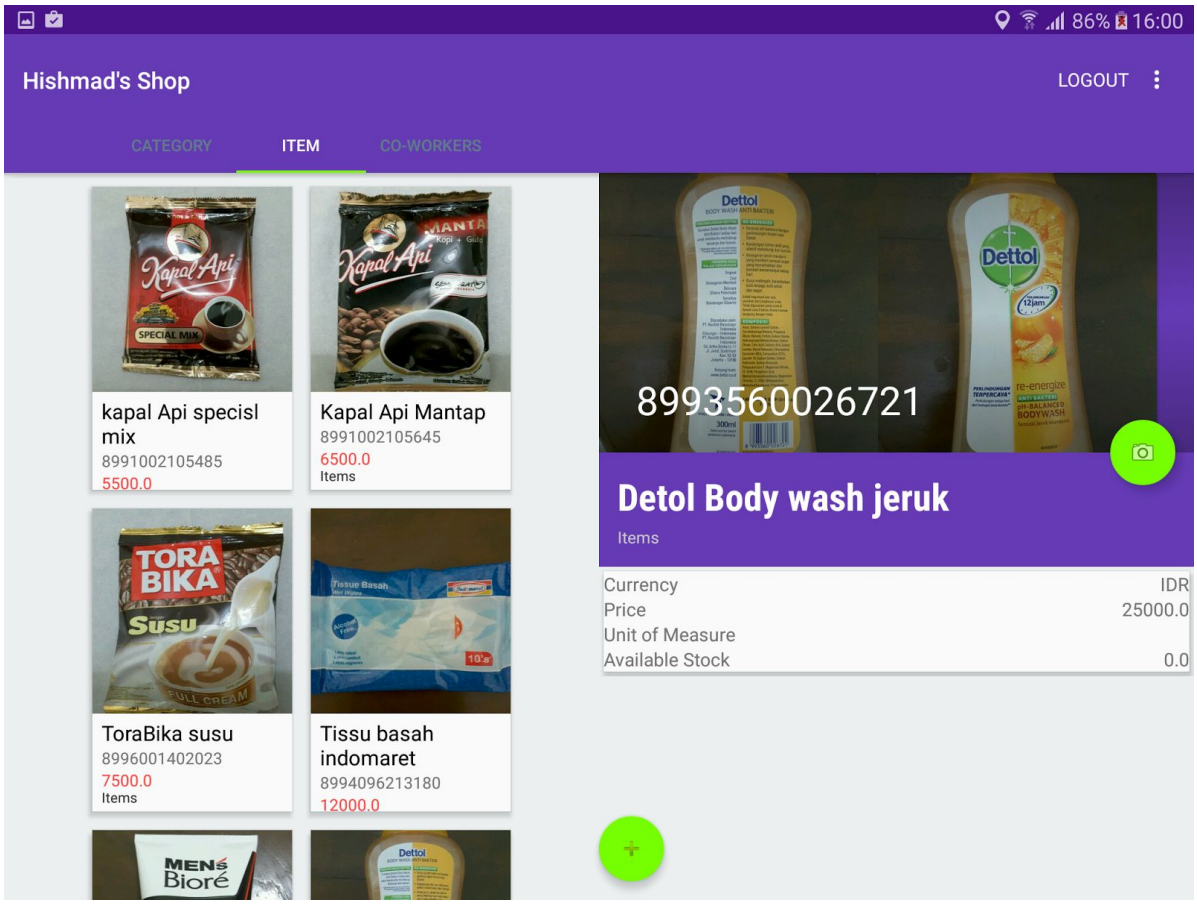
Features

List the main features of your app. For example:

- Saves product information
- Saves sales & purchase transaction
- Takes unlimited pictures of each item master
- Can have multi users
- Save the data in the cloud, Firebase.com
- Have a barcode scanner
- Login with Google account
- Support phone and tablets layout

User Interface Mocks

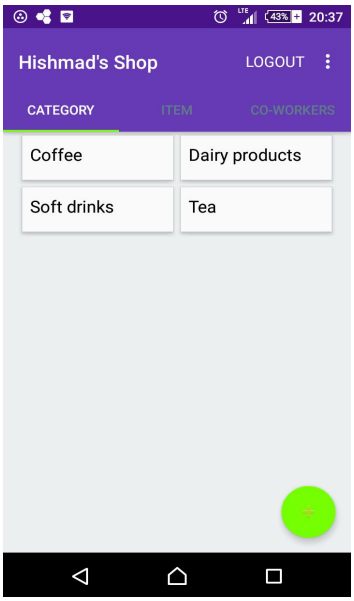
These can be created by hand (take a photo of your drawings and insert them in this flow), or using a program like Photoshop or Balsamiq.

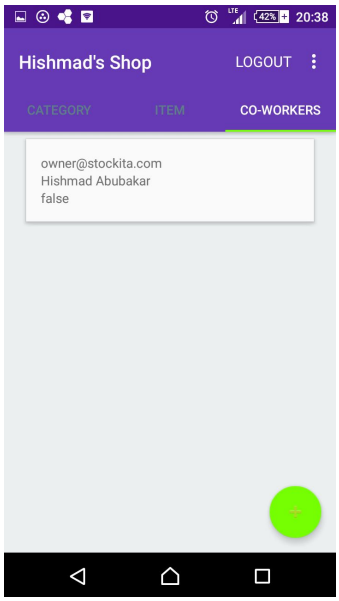


Screen 1

This screen is showing a list of Master Items in the left and the detail for each item in the right, so the user can add or edit Master Item including images taken from the gallery or take a new photo shoot.

This is a two pane mode for sw600-land, which is one activity with two fragment container.





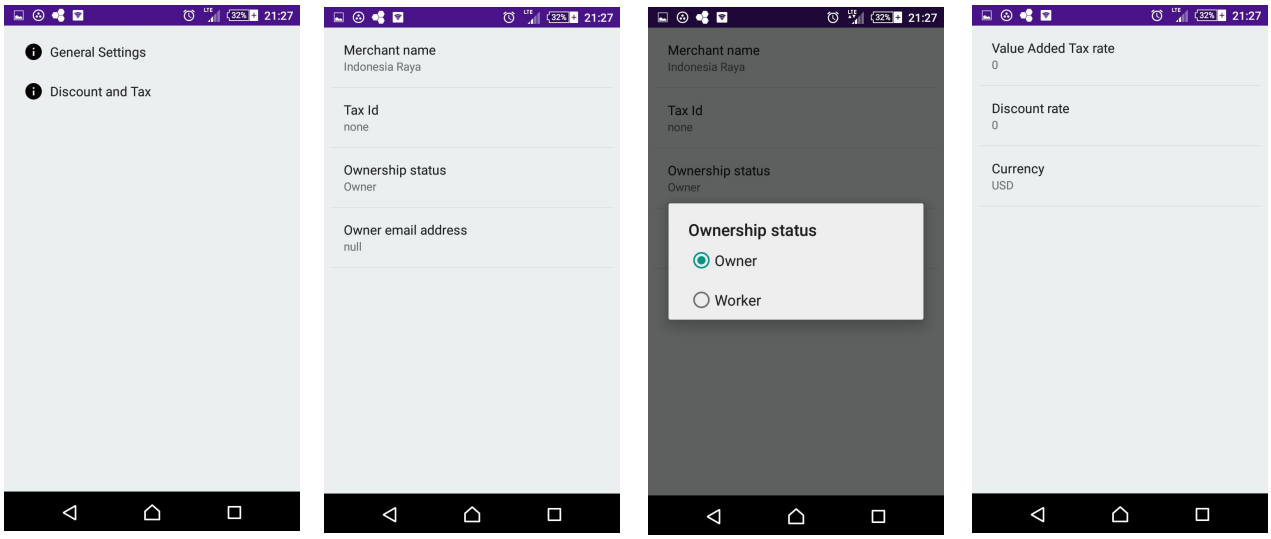
Screen 2, on the left

This is the category master, so the user can add as different categories for several product, it is like grouping.

Screen 3, on the right

The user also can have multi users, they can be his/her employee and co workers, so they can become portable cashiers, if the user click on the item in the list, it will be true as false by default, so if true then this worker can access the owner data.

As we can see that these three screens one, two, and three, are three fragments in one Activity with ViewPager API.



Screen 4, on the most left

This is the settings activity, and all the user input will be stored in shared preference, the user can make general settings such name, tax id, discount and tax rate.

Screen 5, second from the left

This screen is showing when the user clicked on the General Settings, the interesting part is the ownership status, the user can chose if this is the owner/boss or a worker/employee, if the user decided to be a worker then he/she must provide the boss email address, so they can access the boss data if authorized.

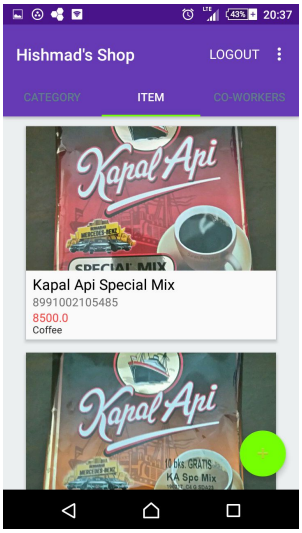
So this is like one to many relationship, one owner/boss can have many worker/employee but one employee can have only one boss.

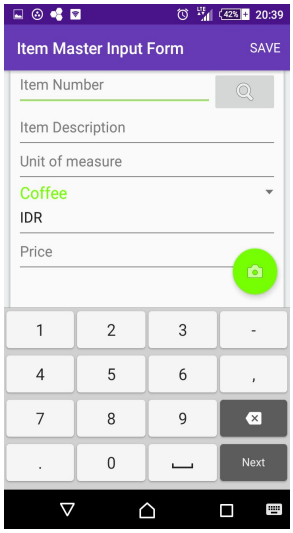
Screen 6, second from the right

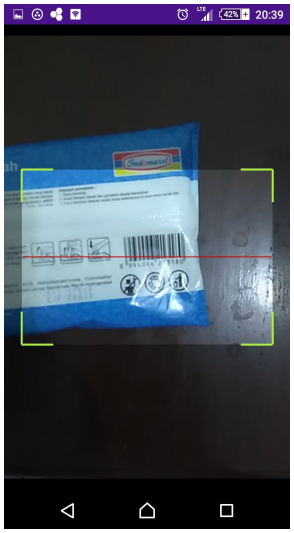
In this screen, it shows a dialog for the user to chose the ownership as explained in screen 5 above.

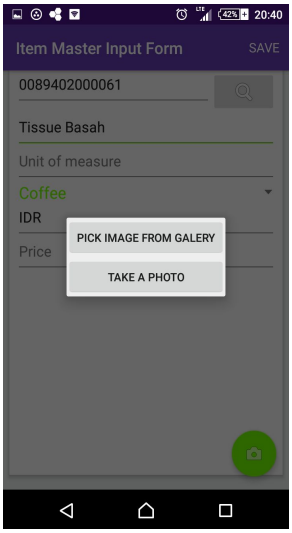
Screen 7, on the most right

This is the Discount and Tax settings, so the user can set the tax and discount rate, also the user can choose a currency.









Screen 8, on the most left

This is same as Screen 1, but it is the phone version layout, how every so if the user click on the green (+) Floating Action Button, it will take the user to screen 9 which is in different activity.

Screen 9, second from the left

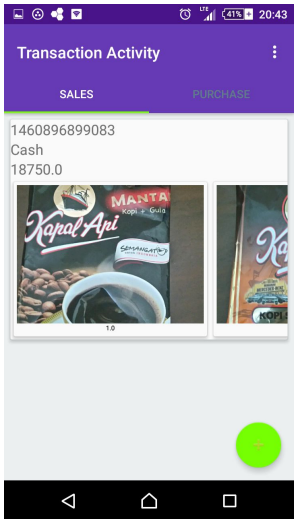
This form allow the user to input a new item, he or she can type the item number or use barcode scanner as shown in the next screen, also the user can add images of this item, he/she can take direct photo, or get images from the gallery on the phone.

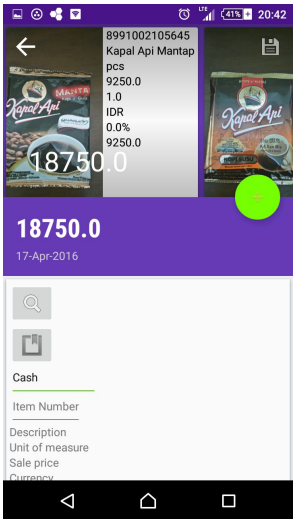
Screen 10, second from the right

The barcode scanner.

Screen 11, on the most right

If the user clicked on the green Floating Action Button with camera icon shown in screen 9, this will show a dialog so he or she can choose to take a picture or find one from the gallery.





Screen 12, on the left

Now here is the Transaction Activity, which is one activity with two sliding tab, one is Sales (POS) and the other one is Purchase (POP), actually they have a similar design so I will explain the photos from the POS.

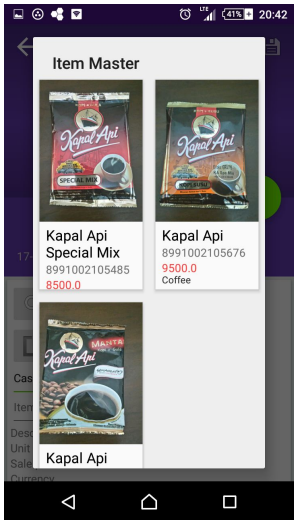
We can see a list of transaction made by the cashier, each item in this list is representing one transaction and can have images of product been sold in that transaction, so the transaction is in the vertical orientation, and the product within that transaction is in the horizontal orientation.

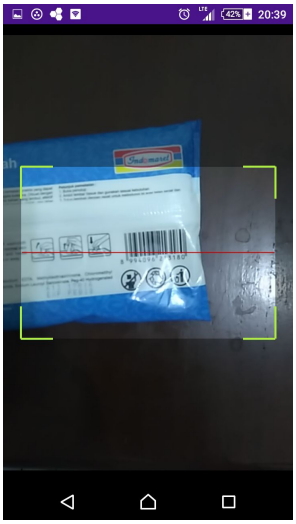
If the user clicked on the green FAB or he/she clicked on an item in the list, it will take him/her to screen 13, which is the detail transaction.

Screen 13, on the right

This screen is the detail transaction, if the user came here when he click on the FAB in screen 12 then it will show an empty form, but if he/she came here when clicked on the item in the list the it will show form with existing sales transaction so they can add or delete things and the numbers will change accordingly.

Each time the use click on the green FAB on the right of the screen it will add product for this sales transaction, by default the quantity is 1, how ever the cashier can choose to lookup from the dialog showing list of product or to scan barcode like screen 14 and 15 respectively, or he/she can type the item number manually .





Screen 14, on the left

This is the product list dialog.

Screen 15, on the right

This is the product barcode scanner.

Key Considerations

How will your app handle data persistence?

This app stores almost all of its data in the cloud using Firebase.com, how ever some small part of data such like settings are stored in the sharedPreference, and the user email and the authorised status are stored in local SQLite using Content Provider exclusively for this app.

Firebase can also have an offline mode, so it will sync once the user connected to the internet, this API is provided by Firebase so we don't have to build custom SyncAdapter or Job Scheduler or any of its kind, the Firebase API does it all in the background thread.

Describe any corner cases in the UX.

The most UX that I would like to point out for this app, is specially for the phone gadget that the user will be able to operate this app using one hand, that's including scanning barcode in the cashier desk, this will give speed to handle more customers que in the shop.

But for tablets, it is suitable most for restaurant, as it does not require speed, so the user can show an images of food and drinks from that tablet and choose to order one.

Describe any libraries you'll be using and share your reasoning for including them.

This is the best part to explain it, as I am delighted and thankful to the people who build extraordinary excellent libraries out there, especially Google.

Here are the list of libraries I am using in this app;

1. The Android support libraries, of course, AppCompat, Design, RecyclerView, CardView, the reason is to use the latest technologies that is available at present and easy to maintain in the future.
2. Google Play Services, for the Google account sign in and GCM.
3. Firebase for APIs that support storing, retrieving, syncing, and securing data.
4. Picasso by Square and Glide by BumpTech for downloading images, this saves a lot of boiled plated codes.
5. ButterKnife it's really fun.
6. Firebase-UI by Firebase community, for some simple recyclerview adapter, but for some more complex adapter, I am using RecyclerView.Adapter API then use Firebase.Child EventListener within.
7. Barcode Scanner by zBar, for scanning barcodes.

Next Steps: Required Tasks

This is the section where you can take the main features of your app (declared above) and decompose them into tangible technical tasks that you can complete incrementally until you have a finished app.

Task 1: Project Setup

1. Build.gradle, we must declare libraries that we are going to use in this app.
2. Create gradle.properties, and declare the Unique Firebase Root Url so later we can reference to it.
3. Create a class that extends android.app.Application, here we want to initialize and configure Firebase, because we are going to use it everywhere inside the app.
4. Create an abstract class named BaseActivity.java that extends AppCompatActivity and implements GoogleApiClient.OnConnectionFaildListener, this is the most all activities classes will inherit from. It implements GoogleApiClient callbacks to enable logout in all activities, and defines variables that are being shared across the app.
5. In BaseActivity class also we need to use Firebase's Auth State Listener, because the user will login with Google account into firebase, also we need to have function to handle logout, and another function to check Google Play Services API.
6. Still in BaseActivity class we are going to have function on how to handle ownership status that the user typed in the Settings, whether he or she is the owner/boss or worker, of course this information must be available if at any time he/she changed that status, or been authorised or unauthorised by the boss.
7. Create a public class named Utility.java this class will have static methods that can be resued accross the app, here we must make every single resuble method to be independent and concise.
8. Create a public class named Constant, where we will store all of the constants in our app, to be reference to here.
9. Create /res/anim folder, then create one or maybe two xml files using <set> to define interpolator to be used by some views in the app.
10. Make changes in the /res/values for colors.xml, dimens.xml, styles.xml.
11. Also create new folders /res/values-land, /res/values-sw600dp, and /res/values-sw600dp-land.
12. Create new folder /res/layout-sw600dp, and /res/layout-sw600dp-land.
13. Create LoginActivity.java to handle user login using Google Account.
14. Ongoing development maybe some changes or addition will be made here.

Task 2: Implement UI for Each Activity and Fragment

Here I will list all the Activity and their inner classes and XML files, Fragment and their inner classes and XML files, also Adapter and their XML files:

1. MainActivity.java, has activity_main, it is available in /res/layout, /res/layout-sw600dp and /res/layout-sw600dp-land
2. MainActivity.java, must have inner class SectionPagerAdapter that extends FragmentStatePagerAdapter which manage the fragments within the ViewPager API.
3. Their is three fragments attached to the FragmentStatePagerAdapter, they are CategoryRecyclerViewFragment.java, ItemMasterRecyclerViewFragment.java, and CoworkerRecyclerViewFragment.java.
4. Each of these fragments must have XML layout and adapter with XML layout also to populate data into the recyclerview.
5. Create CategoryRecyclerViewFragment.java extends Fragments and implements LoaderManager.LoaderCallbacks<Cursor> and also create /res/layout/category_recycler_view.xml.
6. Within CategoryRecyclerViewFragment.java create inner class Adapter extends RecyclerView.Adapter and create /res/layout/category_recycler_adapter.xml.
7. Now the LoaderManager will load data from local Sqlite pass it to the Adapter then the adapter will use that data to sync with Firebase database using one of Firebase's listeners.
8. The process is almost similar for the ItemMasterRecyclerViewFragment.java and CoworkerRecyclerViewFragment.java but little diffrent is that we are going to create an Adapter that extends FirebaseRecyclerAdapter which is a Firebase-UI libarary, instead of RecyclerView.Adapter.
9. I will use RecyclerView.Adapter and FirebaseRecyclerAdapter.java depends, if it is just a simple one then I will use FirebaseRecyclerAdapter.java, if it is more complex like nested recyclerview adapter, then I would use RecyclerView.Adapter class.
10. Ongoing development maybe some changes or addition will be made here.

Task 3: If the user click on the Floating Action Button that is in MainActivity

There is one FAB in the activity_main.xml that implemented in MainActivity.java, this FAB must have switch depends at which page the user currently viewing.

The purpose of this FAB is to create a new form, such as new form for Category Master, or a new form for Item Master, each of it must use intent to get into new activity.

1. CategoryMasterFormActivity.java has /res/layout/category_master_activity.xml.

2. Then we must initialize a fragment, and invoke getFragmentManager() to begin using this fragment.
3. That fragment named AddNewCategoryFormFragment.java extends Fragment and /re/layout/categoty_master_form_fragment.xml
4. In this fragment we are going to save data into Firebase database at the node for this user according to user login email.
5. When the data is stored in that node, then it will be retrieved and displayed by the CategoryRecyclerViewFragment.java explained in Task 2.
6. Also similar process for the ItemMasterFormActivity.java has res/layout/item_master_form_activity.xml.
7. Then we will initialize fragment named AddNewItemMasterFormFragment.java extends Fragment, and create /res/layout/item_master_form_fragment.xml
8. In that fragment the user will be able to add new item master, then save them into Firebase database, using the same node that he/she saved the category master.
9. Ongoing development maybe some changes or addition will be made here.

Task 4: If the user click on any item in the list

If the user click on any item in the list, it will take him/her to a detail screen, for viewing and editing, also adding images or removing images etc.

1. If the user click on an item from ItemMasterRecyclerViewFragment, here we must use a click listener within the adapter.
2. Because we are going to have two screen layout, the phone layout, and the tablet layout, so must have to possibility, if it is a phone, then we pass the data directly from the adapter using Intent to the destination activity and pass the data with that Intent.
3. But if tablets, then we are going to have an interface for callbacks method to the MainActivity.java, because this fragment is in the MainActivity ViewPager.
4. That callbacks method will commit the detail fragment into the second container as shown in Screen 1 above.
5. That detail fragment named DetailMasterItemFormFragment.java extends Fragment, will have a Firebase listeners to retrieve data, and also update and store edit/add data/images into Firebase database for this specific item.
6. If the user hold a long click on an item in the list, then it will send the node for that item selected to delete them from Firebase database.
7. This long click listener will be much simple, it is in the adapter and just invoke Firebase.setValue() to null.
8. Ongoing development maybe some changes or addition will be made here.

Task 5: Transaction activity

This activity host two fragments using ViewPager same as the one we are using in MainActivity.java, one fragment is for Sales Transaction (POS) and the other one is for Purchase Transaction (POP), as shown in screen 12, 13 , 14, and 15.

1. For POS and POP has similar functionality, the different is POS sells items out from the inventory, and POP buys items into the inventory, (at this development we don't have Inventory Control Module) .
2. The user can click on the FAB it add new transaction or he/she can click on the item in the list to edit/add data.
3. The transaction must have a barcode scanner or lookup dialog that the user can choose item from that dialog.
4. The transaction must have subtotal, discount, tax, and grand total.
5. All data will be stored and retrieved into Firebase database.
6. Under heavy development maybe some changes or addition will be made here.
7. Oh my God.

Task 6: Data Model

We must have POJO that implement Parcelable interface, these are the model that we will create:

1. Model Category Master.
2. Model Coworkers.
3. Model Item Image Master.
4. Model Item Master.
5. Model User
6. Model Sales Header.
7. Model Sales Detail.
8. Model Purchase Header.
9. Model Purchase Details.

Task 7: Supporting

We are going to create supporting class, such as

1. Content Provider for local database SQLite.
2. Contract Data for the Content Provider.
3. Barcode Scanner
4. Item Lookup Dialog

5. Image Dialog, the user can chose to take a picture or go to the gallery.
6. Setting Activity.
7. Service to upload images in the background thread.

Task 8: UI, UX and accessibility

At the final stages before I submit the project to Udacity, the app must make sure to implement all the required UI, UX and Accessibility.

1. Transition and animation in certain view.
2. All child view must implement the accessibility standard.
3. The app will use the Material Design recommended by Google, in term of margin, padding, and colors.
4. Ongoing development maybe some changes or addition will be made here.

Thank you,
Regards
Hishmad

Add as many tasks as you need to complete your app.

Submission Instructions

1. After you've completed all the sections, download this document as a PDF [File → Download as PDF]
2. Create a new GitHub repo for the capstone. Name it "**Capstone Project**"
3. Add this document to your repo. Make sure it's named "**Capstone_Stage1.pdf**"