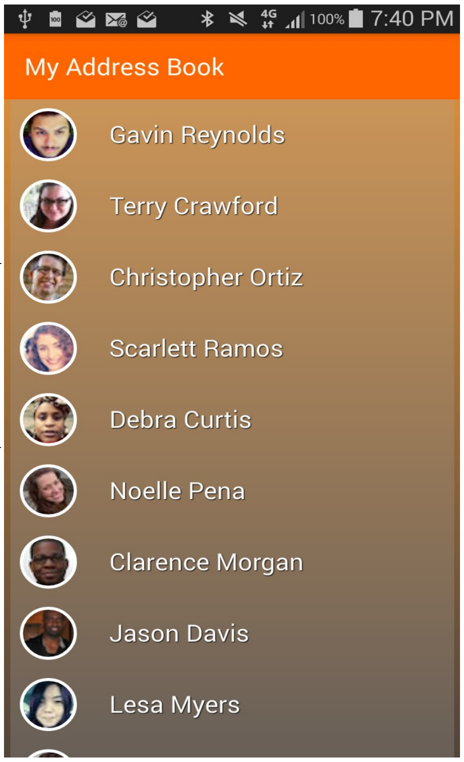
Address Book Application Guillermo Alvarez Colman

Project Github: https://github.com/cognoscola/address_book
Portfolio: http://www.bondfire.ca/portfolio#android

For this assignment I chose to create an Android application which mimics an Address book. I used a Master/Detail type of flow. If running from an Android phone, the user will first see a list of contacts (Fake contacts fetched from http://www.randomuser.me/) when starting up the app. They can click on any of these contacts to reveal details about the contact they selected.

General Flow of data



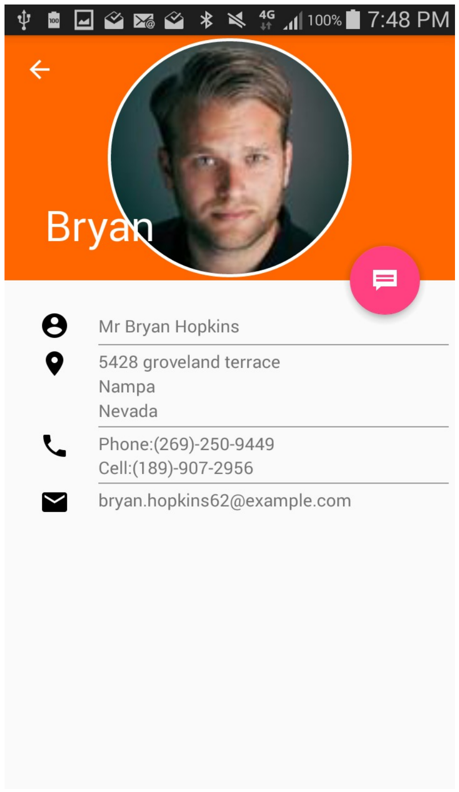
When starting up the app an activity will launch which contains a RecyclerView (which acts as a listview) and an empty container class. Data is requested from the network when the activity has been created and the Recycler view will populate once data has arrived. The RecyclerView items only display 2 pieces of information: The name of the contact and a thumbnail picture of the contact. Each item in the RecyclerView is clickable. When an item is pressed one of two things will happens depending on wether the user is on a tablet or on a phone. If on a phone, the selected item index will be passed to the next Activity via an intent where it will then be used to bring up the selected contact's details. These details will be shown on a fragment. If on a tablet (untested) the empty container class will be populated with a fragment that will display the contact's information.

The details Activity is very simple. It uses a CollapsingToolbarLayout as its ActionBar. This layout is able to expand and collapse itself when the user scrolls down or up. This layout contains a RoundedImageView that displays profile picture of the user. This picture is fetched from the newtork when we create the fragment (except when running on tablet).

The fragment displays text information about the contact such as his or her name, adress, phone number and email. There are other miscellanious information that is fetched from the network such as gender, password and username however it is possibly useless informatino to the user.

There is also logic to preseve state information when the application is destroyed unwillingly. For example, when the user changes the screen orientation while in the Contacts List Activity, fetched information (except for images) will be saved by converting to a json format, stored in a Bundle and then later converted back to java classes when onRestoreInstanceState is called. Images are cached automatically with our ImageLoader library. The android system automically saves fragment states when performing configuration changes and all that is required is to fetch instances of our UI when the fragment gets recreated.

Implemented Features:



Mr Bryan Hopkins

5428 groveland terrace
Nampa
Nevada

Phone:(269)-250-9449
Cell:(189)-907-2956

bryan.hopkins62@example.com

- Fetches and displays a list of contacts
- Displays contact information on a seperate fragment or activity
- Quick sms action button on the Details page.

Possible Features to add:

- Given more time, I would have liked to add the ability to create, edit, and delete contacts. This would probably have taken anywhere from 2 to 5+ hours depending on how we wish store information. I would have to create and set the layouts for a new Edit activity, add functionality to save to the phone's local storage and/or sync this information to a remote storage. I would have to perform validation of data as users create or edit entries. I would also have to test various modules.
- I would have add UI on the first activity that displays only when the recycler view is empty. It would have a label saying “No result found” and a button that when pressed would attempt to fetch data again. This would take less than 10 mintues.
- Ability to reorder RecyclerView entries alphabetically using first or last names. This would take no more than 20 minutes as we would have to change the layouts to accomodate UI which will allow this feature.
- Ability to search an entry. This would take no more than 30 minutes as we would have to change layouts to accomodate UI which will allow this feature as well as include layout and functionality for displaying search results.
- I would have spent more time on general appeance. In this project I tried out 2 or 3 variations of recyclerview and details view appearances and chose the best looking one. Given more time I would have tried more variations to see how it looks.

Changes that will make the project more Robust:

- There are several areas of the project I would definitely change to make it more robust. One area is the RoundedImageView. This is a neat custom view I found online and it works well except that it is very cumbersome with trying to set the layout properly. There are various cases that are unhandle (such as when using layout weights or fill_parent) where the view does not display properly. A temporary solution is to place it inside a FrameLayout.

- I would also change the way I fetch and store images from the network. I used a library called LazyList (https://github.com/thest1/LazyList) which automatically fetches and caches images for you. The downside is that it depends on having the WRITE_EXTERNAL_STORAGE permission so it can do its caching on the drive. This is a a permission that is not necessary for an app like this. One good thing about this library is that it loads on demand. It doesn't not load images that are not visible on the screen. However, once the app is closed all fetched images are destroyed.

- Fetching random users is performed using an outdated library called RandomUsersApi (https://github.com/joselufo/RandomUserApi), which is built on top of Retrofit (a library created by Square which simply handles REST calls and converts an HTTP API to Java interface. Given just a little more time I would have written this myself it is not a cumbersome task.

- The app does not save data when exiting. It would obviously be better to cache RandomUser data between usages.