

H APP

Complete Documentation

CS201 - Principles of Software Development

Prepared by Aditya Aggarwal, David Goodfellow, Ivan Chen,
Ray Jungho Lim, and WoongHee Lee

Table of Contents

Overall Concept	3
High-Level Requirements	4
Login Logic	5
Receiver Logic	8
Giver Logic	10
Receiver Details	11
Database/Backend Logic	12
Donation Listings Logic	13
Requests Listings Logic	14
Technical Specifications	15
Login Logic/GUI	15
Database & Client Communication Logic	18
Database Structure	21
Detailed Design	23
Server	
Login	23
After Login	26
Client	
Login UI	23
User Activities UI	27
UI Interaction UML	30
Technical Requirements	34
Testing	35
Deployment	46

Overall Concept

There are many homeless people, or many who have been struck with some hard times, using their phones and require food, water, clothes, or medical supplies. Our goal is to make an app that help people provide these necessities to those less fortunate. By using maps api, we can direct those in need to locations set by individuals willing enough to contribute what they have. People will have incentive to help others, by donating leftover foods or overstocked goods.

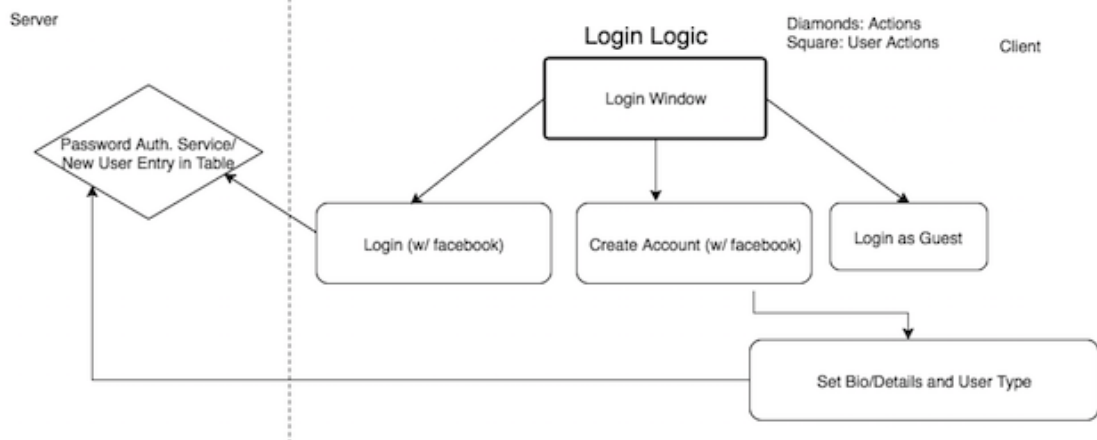
High Level Requirement

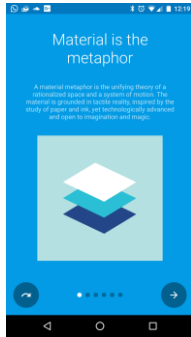
There are two different users , donors and requesters. Donors and requesters interact with different user interfaces, because they access different features. Nonetheless, the overall structure and layout is the same.

You can also interact as Guest. Guests are allowed to look through the contents, however, they do not have access to any special feature that donors or requesters have.

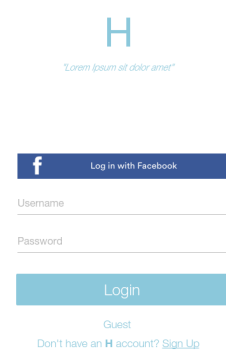
Login Logic

Here is a diagram showing the all the components involved in login feature, which determines account authentication and identification.



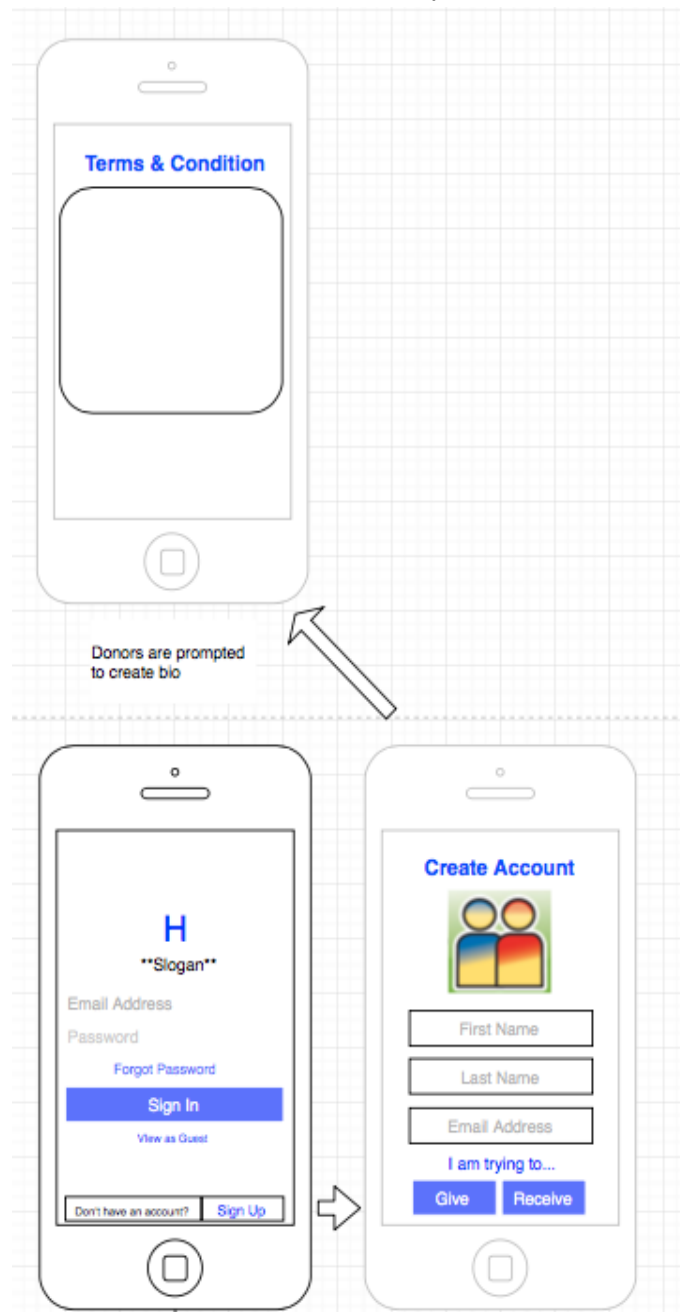


The first page is the the login page. For first time users, there is an onboarding process that we will quickly create with 4 images and a slider.

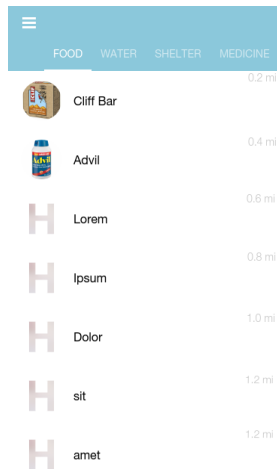


The login page will have our logo and an area that prompts you to input your password and username. A message should appear if you can't login for any reason.

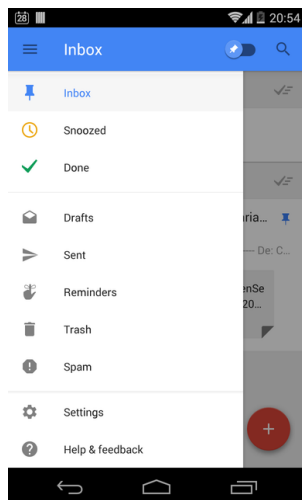
Here is a image of rough mockups of what the rest of login will look like:



Receiver Logic



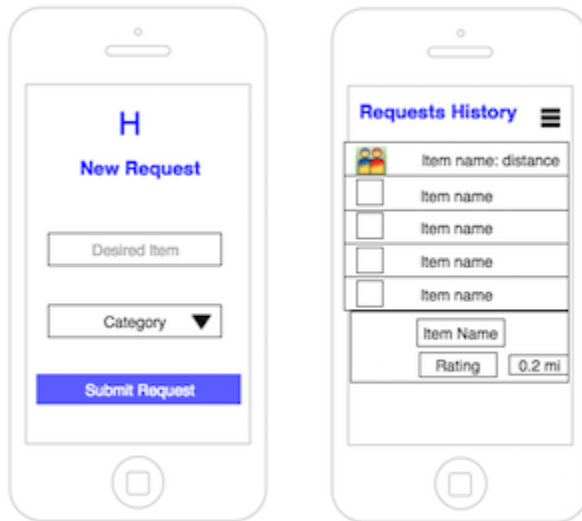
After the login process, depending on your preference, you're relocated to a screen that fits your preference. If you are in need, your screen will contain items people are willing to donate around you.



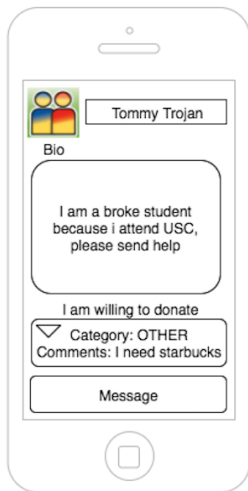
There exists a hamburger menu (following the guidelines of Android Material Design) that will allow you to navigate between different screens that display different features.

Other features we will include for receivers are:

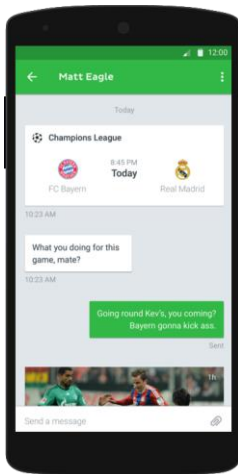
- An option to make a custom request if the receiver wants a specific item
- A history of items requested and their status
- Your own profile and settings, which shows the user's information.



Giver Logic



The donor will receive the Receivers bio when the receiver requests an object that the donor is willing to give or when the Donor wants to fulfill a listing for a person who has requested something custom/has a wish of their own.

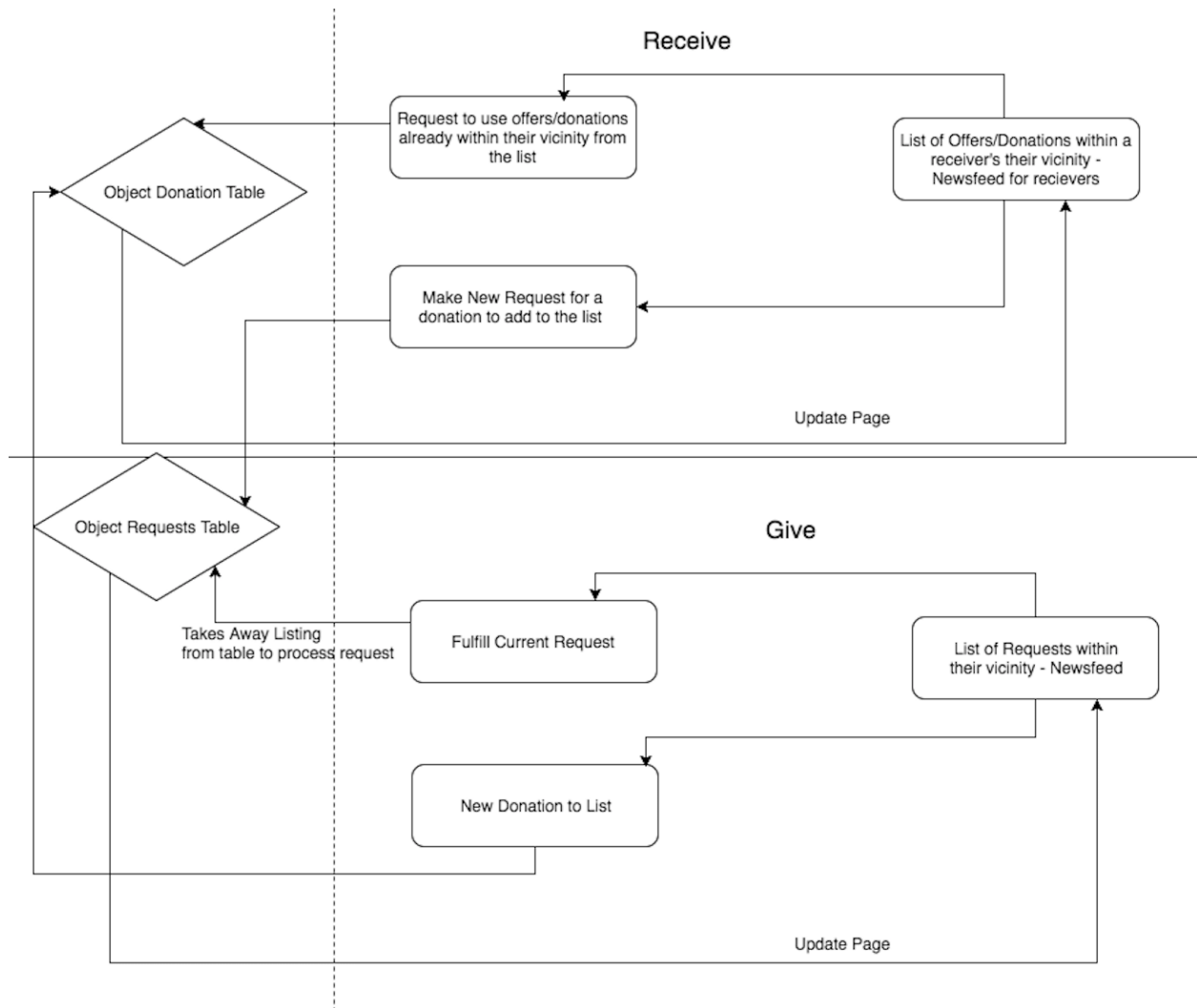


We will add a chat feature to help these two users complete transactions thoroughly. This will not be the main multithreading component of our app (rather that is the request and donations listing logic below). Therefore, we will be using a 3rd party API to implement this feature, efficiently called [Layer](#).

An image of Layer's premade Atlas GUI kit, which we will be using to efficiently complete this feature, is added to the right.

It will contain your image and an image of the person you're talking too, and display a chat box containing everything you typed thus far to that specific person.

Database/Backend Logic



Donations Listings Logic

Upon logging in/creating an account as a receiver, the user is directed to a news feed list. This news feed list consists of individual listings of current open donations within the determined vicinity of the program. Each individual listing has an optional picture of the good, the name of the good, the category of the good, and the distance in terms of miles from the user. It will not specify the exact location of the donation.

The user will be able to scroll through the listings and see all that is currently being offered. Through a filter button, the user will be able to sort the listings by category or by time. If the user sees a listing that fulfills his needs, he can select on that listing which will bring up an option pane that will prompt the user to either request this offering or to go back. If he/she selects to request this offering, it will alert the donor who listed the offering to either accept or deny the request. If accepted on the donor's end, it will connect the donor with the receiver through a messaging portal so they can plan the donation. At this point, the listing will be removed from the donations listings table in the database and no longer render in the news feeds of other users looking to receive. If denied from the donor, it will alert the receiver that their request was denied. If the donor denies the request, the app will ask the donor if he would like to take down his donation listing or keep it up. The database would be updated accordingly.

If the user does not see any donation listing that fits his/her needs, they will have the option to submit a new request. If this button is clicked, the user would now be prompted for a page to fill out more information about their request such as an optional picture, the category it should be under, and what product specifically it is. On this page there will be a confirmation button and a cancel button. The cancel button would direct them back to the news feed. The confirmation button would go and create a new requests listing in the request list table in the database which will render on the newsfeed of donors in the vicinity. Upon success from the database, the user will get a confirmation message and be returned to the listings page.

Requests Listings Logic

Upon logging in/creating an account as a donor, the user is prompted with a news feed list. This news feed list consists of individual listings of current open donation requests within the determined vicinity of the program from those trying to receive a donation. Each individual listing has an optional picture of the good, the name of the good, the category of the good, and the distance in terms of miles from the user. It will not specify the exact location of the donation.

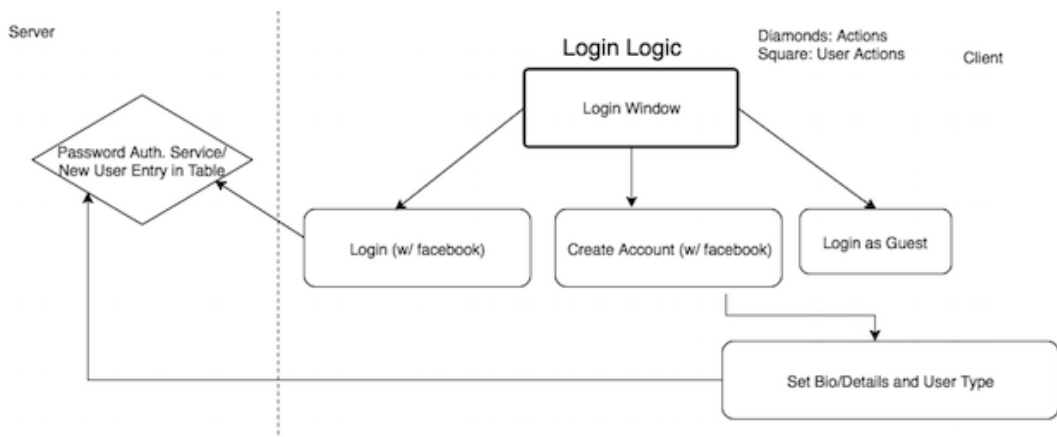
The user will be able to scroll through the listings and see all that is currently being requested. Through a filter button, the user will be able to sort the listings by category or by time. If the user sees a request that he can fulfill, he can select on that listing which will bring up an option pane that will prompt the user to either complete this request or to go back. If he/she selects to request this offering, it will alert the receiver who listed the request to either accept or deny the donation as he/she may no longer be available now. If accepted on the receiver's end, it will connect the donor with the receiver through a messaging portal so they can plan the donation. At this point, the listing will be removed from the requests listings table in the database and no longer render in the news feeds of other users looking to donate to this particular request. If denied from the receiver, it will alert the donor that their request was denied. If the receiver denies the request, the app will ask the donor if he would like to take down his request listing or keep it up. The database would be updated accordingly.

If the user does not see any request listing that he/she can fulfill, they will have the option to submit a new donation listing. If this button is clicked, the user would now be prompted for a page to fill out more information about their request such as an optional picture, the category it should be under, and what product specifically it is. On this page there will be a confirmation button and a cancel button. The cancel button would direct them back to the news feed. The confirmation button would go and create a new donation listing in the donation list table in the database and the new donation listing will render on the newsfeed of receivers in the vicinity. Upon success from the database, the user will get a confirmation message and be returned to the listings page.

Technical Specifications

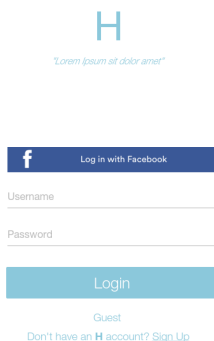
Login Logic

Here is a diagram showing all the components involved in the login process, which will be implemented through the following three classes.



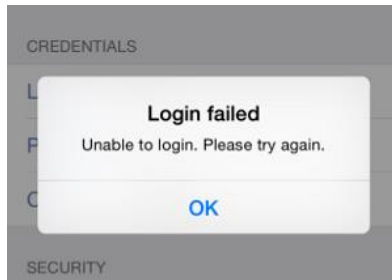
Login GUI

The Factory Server should first display a GUI that allows Users to login via Facebook or via writing their password and email address.

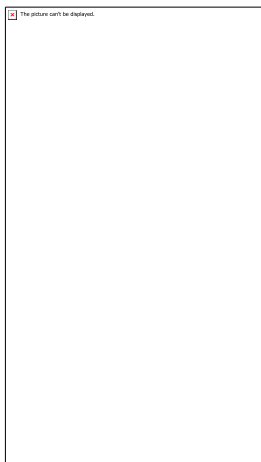


The login page will have our logo and an area that prompts the user to input his/her password and email address. An error message should appear if the user can't login for any reason.

It should allow for people to sign up with facebook to make their account. We'll store the email address, name, and their picture if they login from facebook and the rest will be manual input requested by the app from the user. When the Login button is clicked, if the value entered is not a valid account, display an error message that says, "Unable to login, invalid credentials". Here is a sample GUI showing the error message which is similar to this:



We will determine whether the password is accurate by comparing the typed password to the saved hashed password for the account to make password data more secure. This is what will be stored in our database.



When the Sign Up button is clicked, It will direct you to another screen that allows for you to enter required information about yourself. The first page is the the login page. For first time users, there is an onboarding process that we will quickly create with 4 images and a slider.

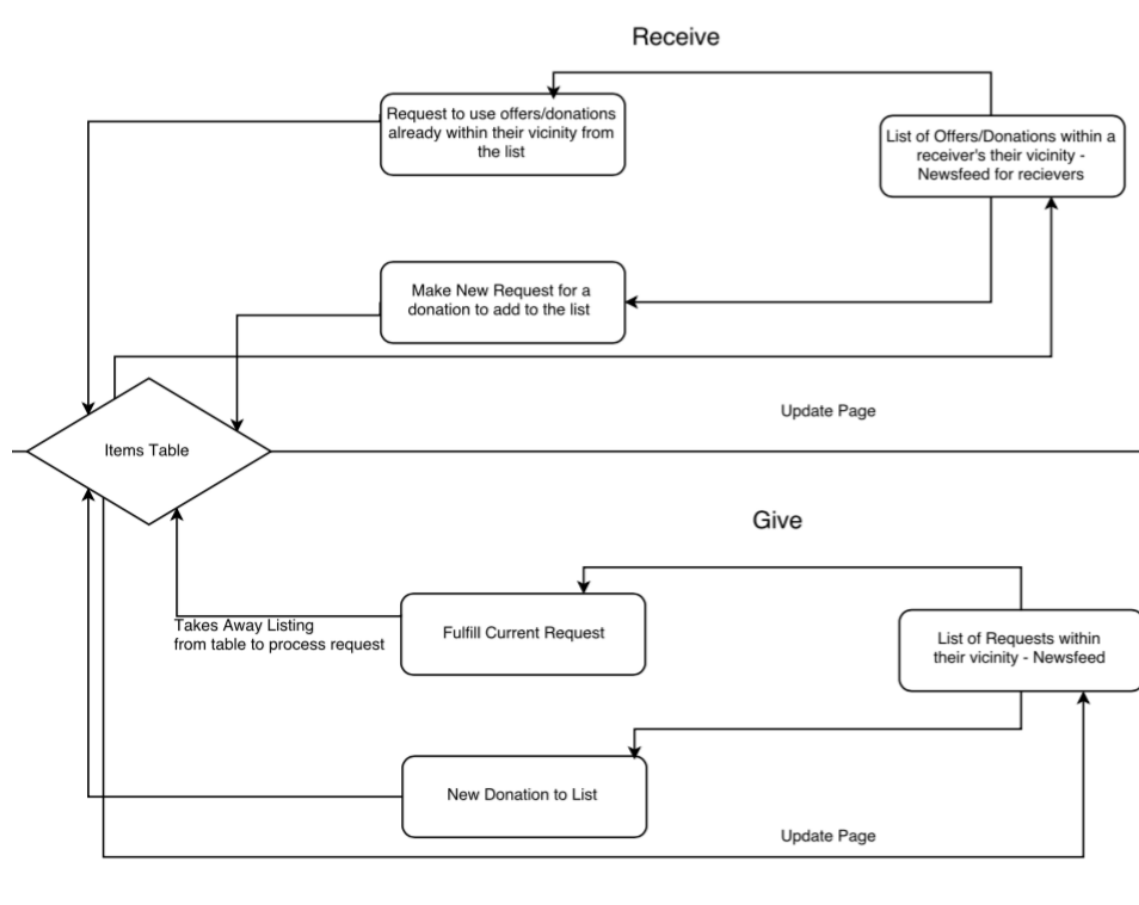
Look at controller below
for slider to understand
context of image

When the "View as Guest" button is clicked, it will direct you to Listing Window (Receiver Side).



People will be able to navigate this app through a side menu bar that you can toggle using the top left hamburger menu bar icon.

Database & Client Communication Logic Here is a diagram showing the logic of the app after the login process, which will be implemented through all the classes and database described below.



Web Server

There is a controller class of a server application on a web domain. When a certain URL is connected to and receives the port of the user, it runs a function that implements either a GET, POST, UPDATE, or DELETE request from the database, depending on how the URL is structured (the goal is to build a REST interface) and returns the data as string.

Client

On the client side upon hitting the server, the client will be connected to a designated URL which connects to an application on a port. Once connected, the client will be able to interact with the REST API and thus the whole system. Based on their user's actions, methods will be invoked to have the Controller class GET, POST, UPDATE, or DELETE instances in the database. If it is a query, it will receive the data back as a string, we will parse it into a JSON Object that gets parsed into our own custom classes. Then, we will display that data to the user accordingly.

The picture can't be displayed.

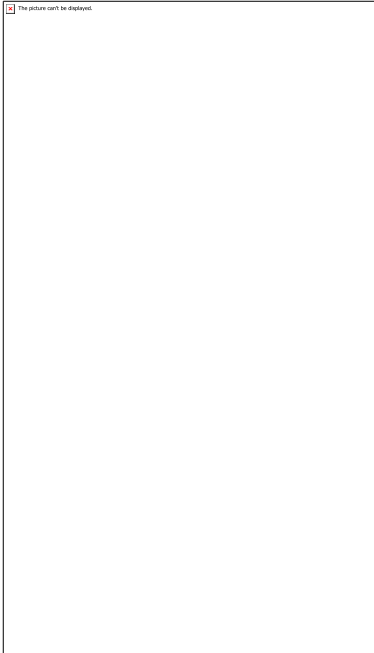
NewsFeedActivity GUI Abstract Class This abstract class will be the parent of both the GUI shown to the giver and the receiver. Both news feeds have the same layout but just differ in what listings are shown and what happens when you click on a listing to request. Due to this, the layout methods will be implemented in this class but the methods to upload listings and what to prompt the user with when you click a listing to request will be implemented by the inheriting classes.

GiverActivity GUI This class will inherit from the abstract class. It will implement a method to upload the listings of current donation requests and prompt the user to confirm he/she would like to request to fulfill this donation when a listing is clicked on. If they select “Yes” then it will alert the receiver and ask for a confirmation.

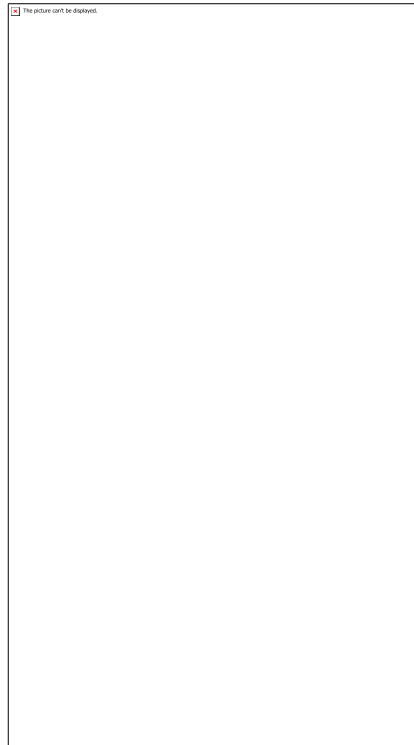
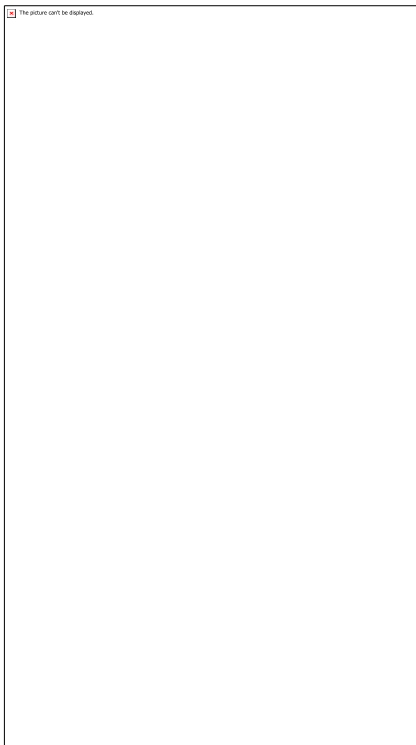
ReceiverActivity GUI This class will inherit from the Newsfeed GUI abstract class. It will implement a method to upload the listings of current donation listings and prompt the user to confirm he/she would like to request this donation when a listing is clicked on. If they select “Yes” then it will alert the giver and ask for a confirmation.

The picture can't be displayed.

Request/DonateItemActivity - If someone wants to request a custom item or give an item they can do it using this activity. Each item should have a photo, name, and be tagged with a category



HistoryActivity- The data in this activity has a history of your requests or the items you gave. When you click on any of the items the **DetailsActivity** appears.



Details Activity GUI

This abstract class will be the parent of the Giver and Receiver Details Activity GUI.

When a user's listing is selected and requested (when someone hits **fulfill request** or **locate item** button), it will alert the user through a notification saying their listing was requested.

When you click on the notification or open the app, it will display this details activity about the listing.

This page will show details about the listing that was

requested, the bio of the requester, the distance from your current location, and a complete transaction button. Hitting the complete transaction button will allow you to say that you've successfully found/gave the item, failed to find the item/give the item, or give you the option to cancel which will put the listing back in the listings page (newsfeed).

Database Structure

The database will consist of three tables: one that holds all the items, one that holds the users, and one that holds the categories for items.

Items Table	Type	Notes
Item Type	Boolean	Request = true, Donation Listing = false
Location	VARCHAR(255)	(latitude, longitude) - parse into two floats
Name	VARCHAR(255)	
Image	Text	Url (upload to a hosting space)
ID	INT	
User Profile ID	INT	The user that posted the listing.
Completed User ID	INT	The user that requests/fulfills the listing.
Completed	SMALLINT	0 = open listing, 1 = listing being processed, 2 = completed listing
Category ID	INT	Foreign key to link to what category

The items table holds all items that have been listed and their respective information. Each item listing will have a unique ID which will act as the primary key for the table. It will also hold three foreign keys. Two of the foreign keys will link to specific users involved with this item, one for the user who listed the item and one for the other that requested/fulfilled the listing. The last foreign key will be for the category which will be used for filtration purposes. The record will also have a boolean value called Item Type. This is used to indicate whether the listing was for a new request listing (true) or a new donation listing (false). The record will also hold a small int called completed. This will indicate whether the listing is currently active for those to select, being processed between two users, or completed. This will be used to render only the active listings and historical records with each user. Lastly, each record in this table will also hold basic information including the name and location of the listing, and an optional image.

User Profile Table	Type	Notes
ID	Int	Used for key in database
Email	VARCHAR(255)	hashed
Password	VARCHAR(255)	hashed
Bio	TEXT	
userType	BOOLEAN	True - giver and false - receiver
Picture	Text	Url (upload to a hosting space)
First Name	VARCHAR(255)	
Last Name	VARCHAR(255)	

The user profile table will hold basic information such as a hashed email, a hashed password, a bio, a first name, a last name, and an optional picture. It will also hold a unique ID for each user as a primary key for the table. Lastly, it will contain a boolean called

userType that will be true if they are registered as a donor and false if they are registered as a receiver.

Category Table	Type	Notes
Category ID	INT	
Category Name	VARCHAR(255)	
Color	VARCHAR(255)	Hardcoded

The last, and smallest, table in the database will be the category table. It will be a preset table that the user cannot manipulate. This is because the user should only be able to have listings under specific categories that are used most frequently in the market such as food, water, clothing etc. It also needs to be preset and a limited amount to ensure the filtration functionality is effective. In the table it will hold a category ID to act as the primary key. It will also hold the name of the category. Lastly, it will hold a hardcoded color to be associated with the category for GUI purposes.

Tasks

Week 1 - User Interface Programming, Database Setup, Detailed Design Doc - 4hrs per person (20 hrs)

Week 2 - Complete UI Programming, Testing Documents - 2hrs per person (10 hrs)

Week 3 - Server Programming and Client Connection, Deployment Documents, Unit testing - 6hrs per person (30 hrs)

Week 4 - System testing. Final touches and improvements. - 4hrs per person (20 hrs)

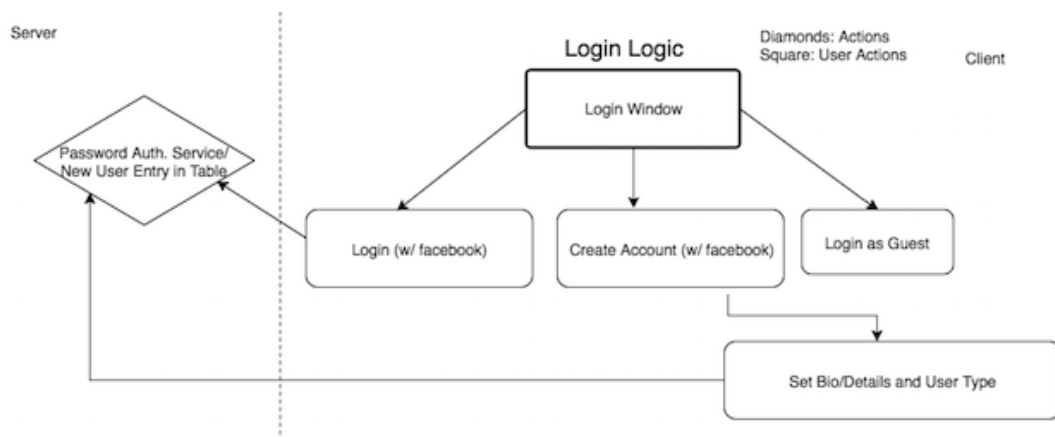
Task Leads for Week 1

Server and Database Set Up:	Lead: David	Support: Ivan, Aditya
Login GUI:	Lead: Ivan	Support: David
Listings GUI:	Lead: WoongHee	Support: Ray, Aditya
User History GUI:	Lead: Ray	Support: Aditya, Brian
Detail Activity GUI:	Lead: Aditya	Support: Brian, Ray
New Listing GUI:	Lead: Ivan	Support: David

Detailed Design

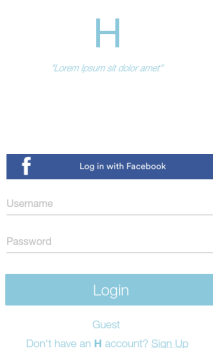
Login Logic

Here is a diagram showing all the components involved in the login process, which will be implemented through the following three classes.



LoginActivity.java

The Factory Server should first display a GUI that allows Users to login via Facebook or via writing their password and email address.



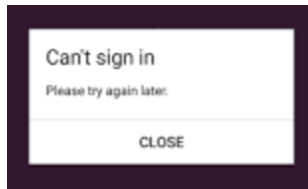
The login page will have our logo and an area that prompts the user to input his/her password and email address. An error message should appear if the user can't login for any reason.

We will use styling and `EditText` and `Button` classes to implement this.

To implement the following we will have a `login()` function that hashes a username and password passed in. Then it will make a `GET` request to the server using the Volley library which will return the appropriate `User` object. This object will then be sent as a JSON string through an intent and passed to the next activity that will parse it into a `User` Object or `JSONObject`.

The Login with facebook functionality checks if the user has been registered with the facebook H app, and if it has, it logs into our database of users using a username and the identity token returned by facebook app.

When the Login button is clicked, if the `GET` request to users does not return a user, display an error message that says, "Can't sign in. Invalid credentials". Here is a sample GUI showing the error message which is similar to this:



When the “View as Guest” button is clicked, it will direct you to `NewsFeedActivity` (Receiver Side), and grab data from your current location. Nonetheless, there will be no functionality other than the ability to navigate through the UI. All the buttons will be disabled or just prompt the user to login (using a Dialog) when clicked.

If the user clicks a sign-up link at the bottom. An `Intent` directs them to the sign-up page below:

SignUpActivity

We will also allow people to sign up with facebook to make their account. We'll run through the authentication (and app authorization) process detailed by the Facebook Accounts API, and we'll store the email address, name, Facebook app-scoped identity/user id (as a password), and their picture into our database if they login from facebook and the rest will be manual input requested by the app from the user. Before creating an account, we will run a `GET` request to determine if the user already exists. This functionality will be in a `validateIfExists()` function. If the account does not yet exist, we use an `Intent` to transition to an enter Bio screen, upload the image to a link (or use a facebook link if from facebook), and create a JSON user object and pass that as a string through the `Intent`.

EnterBioActivity

When the Sign Up button is clicked, It will direct you to another screen that allows for you to enter a bio about yourself. This will be an `EditText`. Afterwards, when the user hits NEXT, we will make a `PUT` request (that registers the user into our database) to our server at `users/UUID` (the UUID will be their User ID -- this is to ensure more security over an auto generated incremented user ID) to create a new user. When the user logs in, the get request returns the UUID as a part of the user object which is how the user can update her/his information once logged in through future update `PUT` requests. This will be encapsulated in a `createUser()` method. (We will also hide the action bar for this screen).



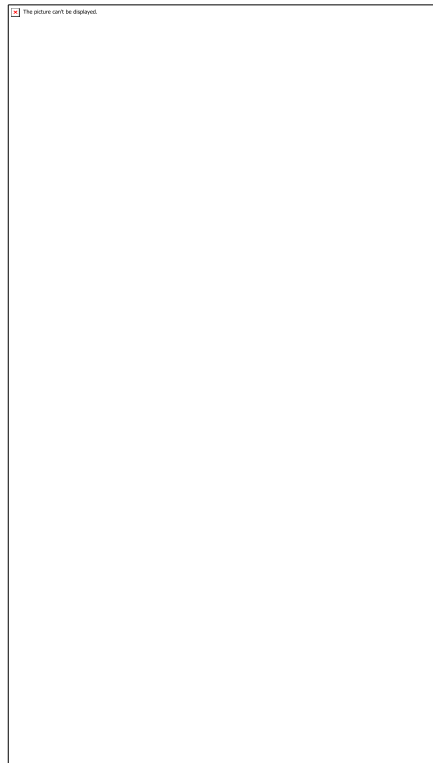
The Intent in the Enter Bio Activity will direct you to an onboarding process, in **OnBoardingActivity** with a ViewPager that we will quickly create with 4 images and a slider (using this tutorial:

<http://blog.grafixartist.com/onboarding-android-viewpager-google-way/>)

When you click finish. The **OnBoardingActivity** will direct you to the **NewsFeedFragment** and pass the user object as a JSON String in an intent.

If you login instead of sign up, the intent will skip this activity and direct you straight to the **NewsFeedFragment** detailed below (after navigation).

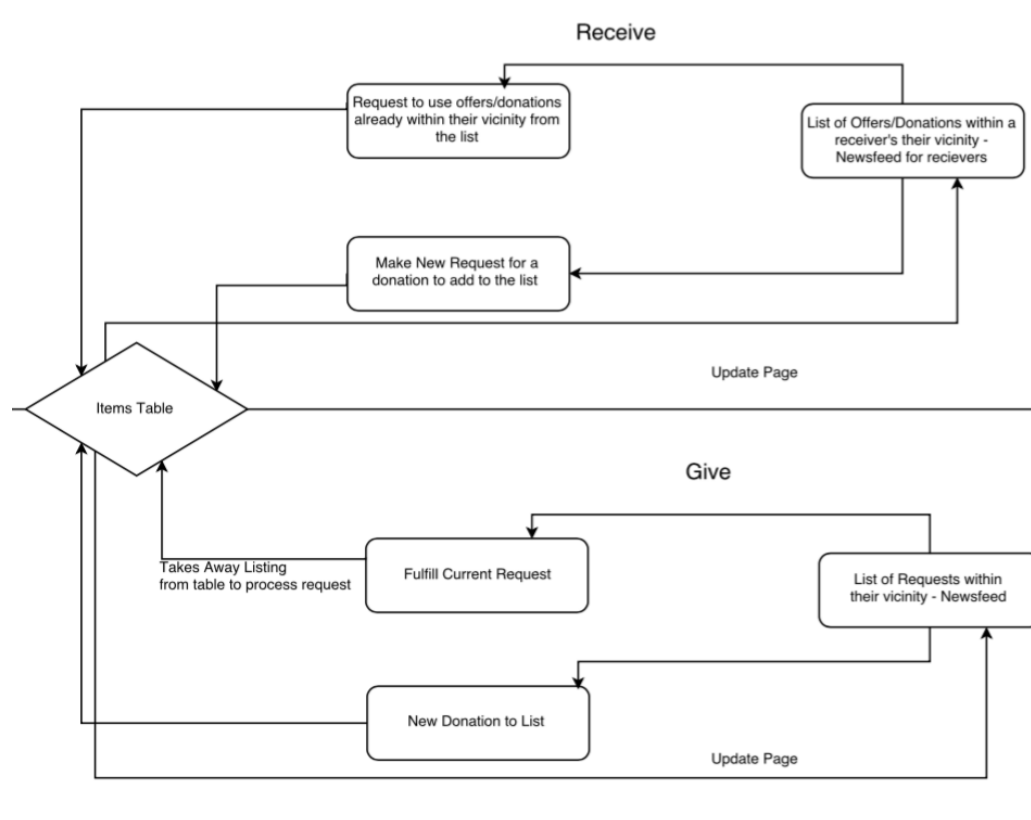
Look at controller below
for slider to understand
context of image



People will be able to navigate this app through a side menu bar that you can toggle using the top left hamburger menu bar icon. This will be created through a **DrawerLayout** on the left side of our activity. We will follow the tutorial below to create this left side navigation drawer. To save time we will use the following third party library:
<https://github.com/mikepenz/MaterialDrawer>

Before We detail the user screens of the app; it is important to get a high level overview of the data flow between the server and the client.

Database & Client Communication logic Here is a diagram showing the logic of the app after the log in process, which will be implemented through all the classes and database described below.



Web Server

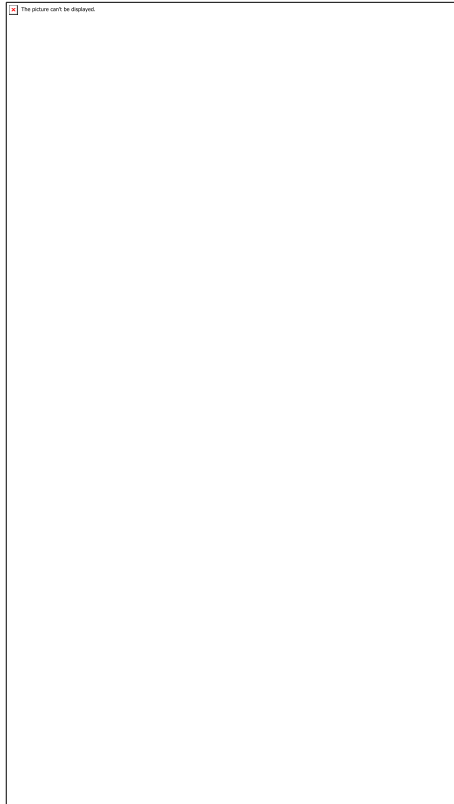
There is a controller class of a server application on a web domain. When a certain URL is connected to and receives the port of the user, it runs a function that implements either a GET, PUT, or DELETE request from the database, depending on how the URL is structured (the goal is to build a REST interface) and returns the data as string.

Client

On the client side upon hitting the server, the client will be connected to a designated URL which connects to an application on a port. Once connected, the client will be able to interact with the REST API and thus the whole system. Based on their user's actions, methods will be invoked to have the Controller class GET, PUT, or DELETE instances in the database. If it is a query, it will receive the data back as a string, we will parse it into a JSON Object that gets parsed into our own customs classes. Then, we will display that data to the user accordingly.

UserActivity - this activity will contain the navigation drawer and all the fragments detailed below (as the navigation drawer will be used to navigate between the different fragments).

To allow for flexibility of content. All the User Activity will contain is the Navigation Drawer and the `FrameLayout` to contain all the different types of fragments.



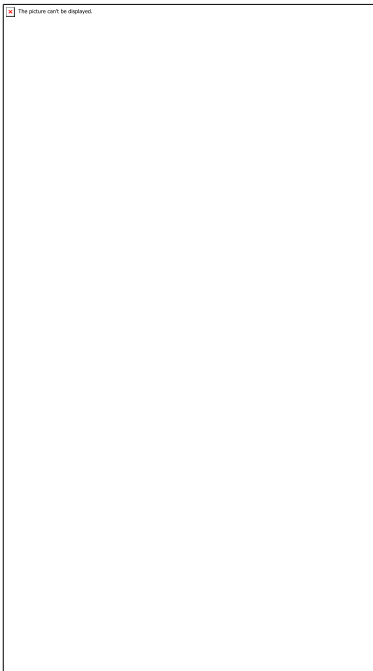
NewsFeedFragment This class will be the GUI shown to the giver and the receiver. Both news feeds have the same layout but just differ in what listings are shown and what happens when you click on a listing to request. Due to this, the layout, data integration, the methods to retrieve the listings, and what to prompt the user with when you click a listing (the item click listener) will be implemented by this classes. The class will contain a method called `getListings()`. It will run a GET Request with a location parameter and Item type parameter. (The URL will look like this `location= "(x, y)" & ItemType= "True/False"`) that returns the listings of current donation requests or Donations depending on whether the user is giver or receiver. The class will contain a function `transformData(JSONArray array)` that sorts listings by location and stores a `JSONArray` of items into a hash map that contains `ArrayList`s. The `HashMap<String, ArrayList<Items>>` will have categories as keys and items as objects.

The layout's data integration into the UI will be implemented by a `FragmentPagerAdapter` which contains auto-generated custom `ListFragments` with a specific `NewsfeedListAdapter`. Each `NewsfeedListAdapter` is tagged with a category and we will use this tag to access the appropriate `ArrayList` in the `HashMap`.

Also the class will implement a correct item click listener, which will prompt the user to confirm he/she would like to request to fulfill this donation when a listing is clicked on. If they select "Yes" then it will update item in the database with a `PUT` request to the Items table (mark listing as being processed) (ideally we also want to implement a push notification using Google Cloud Messaging, although this is unlikely to happen because of the limited time we have).



Request/DonateItemFragment - If someone wants to request a custom item or give an item they can do it using this activity. Each item should have a photo, name, and be tagged with a category. When submit is clicked a function `addListing()` will be called, that makes a `PUT` request that passes JSON to the server identified again by an auto-generated UUID. The type of item will be determined by the user Type. The UI will be implemented through an `EditText`, `ImageView`, `Spinner`, and `Button`.



HistoryFragment- The data in this activity has a history of your requests or the items you gave. This will be retrieved by a `GET` Request that filters items by user profile id or completed profile id. The URL will be structured as such: `/items?completedprofileid= "u-u-i-d/userid of user (1210f-121920-129e029-1920129) etc."` When you click on any of the items the **DetailsActivity** appears, through an Intent that also transfers the data about the object you clicked (likely as `JSONObject` or a `Parcelable Items Object`).



Details Activity GUI

(When a user's listing is selected and requested (when someone hits **fullfill request** or **locate item** button), it will alert the user through a notification saying their listing was requested.

When you click on the notification or open the app, it will display this details activity about the listing. --this is ideal although we may not have time to implement this)

This page will show details about the listing that was requested, the bio of the requester, the location where

the item is left, an option to message the giver, and a complete transaction button.

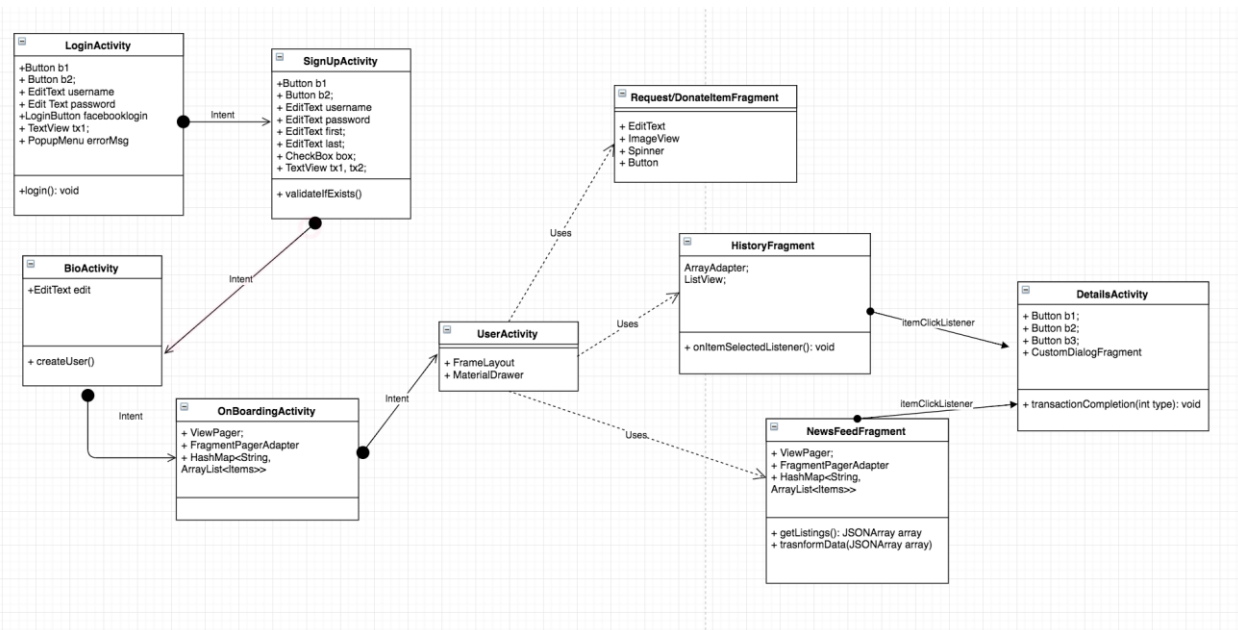
The data about the item will be retrieved from the Intent that starts the `DetailsActivity`. To get the user information we'll run a `GET` request using the "Completed User Profile ID" or the "User Profile ID" of the Item, depending on whether the user is a giver or receiver.

The `LocateItem` feature will display a map to the item of interest using an Intent to Google Maps.

Hitting the complete transaction button will allow you to say that you've successfully found/gave the item, failed to find the item/give the item, or give you the option to cancel which will put the listing back in the listings page (newsfeed). The `SUCCESS`, `CANCEL`, and `FAILURE` buttons update the item through a `PUT` Request. (Another `PUT Request` is also used to update the users rating--although this will be ideal, we may not implement this because of our time constraints).

(We will implement the message feature through the Layer API with its library called Atlas (<https://atlas.layer.com/android>)--we may not be able to implement this because of limited time but we will try)

This is a UML Diagram of How the UI interacts Together



Database Structure

The database will consist of three tables: one that holds all the items, one that holds the users, and one that holds the categories for items.

Items Table	Type	Notes
Item Type	Boolean	Request = true, Donation Listing = false
Location	VARCHAR(255)	(latitude, longitude) - parse into two floats
Name	VARCHAR(255)	
Image	Text	Url (upload to a hosting space)
ID	INT	
User Profile ID	INT	The user that posted the listing.
Completed User ID	INT	The user that requests/fulfills the listing.
Completed	SMALLINT	0 = open listing, 1 = listing being processed, 2 = completed listing
Category ID	INT	Foreign key to link to what category

The items table holds all items that have been listed and their respective information. Each item listing will have a unique ID which will act as the primary key for the table. It will also hold three foreign keys. Two of the foreign keys will link to specific users involved with this item, one for the user who listed the item and one for the other that requested/fulfilled the listing. The last foreign key will be for the category which will be used for filtration purposes. The record will also have a boolean value called Item Type. This is used to indicate whether the listing was for a new request listing (true) or a new donation listing (false). The record will also hold a small int called completed. This will indicate whether the listing is currently active for those to select, being processed between two users, or completed. This will be used to render only the active listings and historical records with each user. Lastly, each record in this table will also hold basic information including the name and location of the listing, and an optional image.

User Profile Table	Type	Notes
ID	Int	Used for key in database
Email	VARCHAR(255)	hashed
Password	VARCHAR(255)	hashed
Bio	TEXT	
userType	BOOLEAN	True - giver and false - receiver
Picture	Text	Url (upload to a hosting space)
First Name	VARCHAR(255)	
Last Name	VARCHAR(255)	

The user profile table will hold basic information such as a hashed email, a hashed password, a bio, a first name, a last name, and an optional picture. It will also hold a unique ID for each user as a primary key for the table. Lastly, it will contain a boolean called

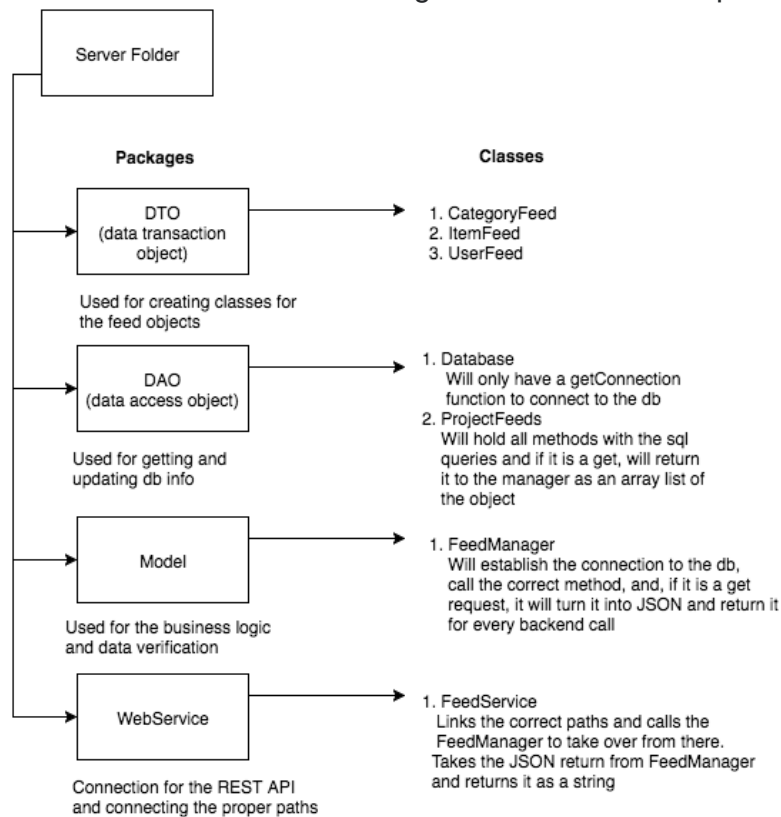
userType that will be true if they are registered as a donor and false if they are registered as a receiver.

Category Table	Type	Notes
Category ID	INT	
Category Name	VARCHAR(255)	
Color	VARCHAR(255)	Hardcoded

The last, and smallest, table in the database will be the category table. It will be a preset table that the user cannot manipulate. This is because the user should only be able to have listings under specific categories that are used most frequently in the market such as food, water, clothing etc. It also needs to be preset and a limited amount to ensure the filtration functionality is effective. In the table it will hold a category ID to act as the primary key. It will also hold the name of the category. Lastly, it will hold a hardcoded color to be associated with the category for GUI purposes.

Server Design

Below is the hierarchy of the classes on the server side. For our client to server communication we will be using a web server that implements the REST API.



We decided to go with a REST API because REST demands the use of hypertext, which scales very well since the client and server are very loosely coupled. The client needs only

know the initial URI, and subsequently chooses from server-supplied choices to navigate or perform actions. A server may download code to the client which aids in navigation and state representation.

To perform the desired action, the client will navigate to the url for the specific task. Upon this the requester will receive a string version of whatever data they requested in JSON form. The JSON form is the serialization standard so that both the client and the server communicate properly.

We will be using four packages, each containing one or multiple classes, to implement the REST API. These four packages are the DTO package, DAO package, Model package, and the WebService package.

Here is how the logic of this will work:

1. The user will direct itself to the proper URL. Upon arriving at a proper project url path, the FeedService class in the WebService package will scan for the path selected in its program.
2. Once it finds the right path, it will call the proper method in the FeedManager class in the Model package. The FeedManager class is used to first create the connection with the database. This connection will be done simply through a method called getConnection() that is in the Database class.
3. If it is a GET request to the database, it will create an arraylist and make a function call to process the request and receive the arraylist back. The object of the arraylist is whatever data transaction object is being requested for. The three options for this object is a category object, an item object, or a user object which are all classes inside the DTO package. If it is not a GET request, it will just make the correct update/delete/add call to the proper method in the ProjectFeeds class.
4. The ProjectFeeds class holds all the methods to run sql queries on the database. It creates the prepared statement, runs it, and returns an array of the objects if it is a GET call back to the FeedManager.
5. The FeedManager then takes this array and turns the returned content into JSON formatting and returns that to the FeedService.

Hardware Requirements

- Samsung Galaxy S3 or Nexus 4 (or any android phone with same/higher spec)
- CPU: Quad Core 1.4GHz
- Internal Memory: 512mb
- RAM: 1gb

Software Requirements

- midSkdVersion: 21 (Android 5.0)
- Android Studio 2.2

Tasks

Week 1 - User Interface Programming, Database Setup, Detailed Design Doc - 4hrs per person (*20 hrs*)

Week 2 - Complete UI Programming, Testing Documents - 2hrs per person (*10 hrs*)

Week 3 - Server Programming and Client Connection, Deployment Documents, Unit testing - 6hrs per person (*30 hrs*)

Week 4 - System testing. Final touches and improvements. - 4hrs per person (*20 hrs*)

Task Leads

Server and Database Set Up:	Lead: David	Support: Ivan, Aditya
Login GUI:	Lead: Ivan	Support: David
Listings GUI:	Lead: WoongHee	Support: Ray, Aditya
User History GUI:	Lead: Ray	Support: Aditya, Brian
Detail Activity GUI:	Lead: Aditya	Support: Brian, Ray
New Listing GUI:	Lead: Ivan	Support: David

Testing

LoginActivity

- 1) Pressing login button without email or password causes nothing to happen
- 2) Pressing login button with wrong password/email combination display “Invalid Login” as a dialog
- 3) Pressing login button with valid inputs opens UserActivity
- 4) Pressing sign up button opens SignUpActivity
- 5) Pressing “Log in With Facebook” opens Facebook Dialog

SignUpActivity

- 1) Pressing Sign Up button when the email exists in the database opens up a dialog saying the email already exists
- 2) Pressing Sign Up button with one of the fields missing does not work (dialog appears)
- 3) Pressing Sign Up button when you have not yet specified whether you are a donor or a receiver does not work (dialog appears)
- 4) Pressing Sign Up button with valid inputs transitioning to the CreateBioActivity
- 5) Pressing “Sign Up with Facebook” opens up FacebookDialog

CreateBioActivity

- 1) Next Button updates bio information in database
- 2) Next Button goes to OnBoardingActivity
- 3) Next Button with empty BioTextField does not work (dialog appears)

Facebook Dialog

- 1) Logging in with facebook populates data in the NewsFeedActivity (done by the Facebook API)

OnBoardingActivity

- 1) ViewPager works (can slide between images)
- 2) Finish button that transitions to UserActivity works

HistoryFragment

- 1) When one of the items is pressed, DetailActivity will open with a correct information.

- 2) When the fragment is pulled down or fragment is revisited, the data is refreshed

NewsFeedFragment

- 1) When one of the items is pressed, DetailActivity will open with a correct information
- 2) When one of the tabs is pressed, it will switch to a correct fragment
- 3) When the user tries to slide the page, it will display a new correct fragment
- 4) When the fragment is pulled down data is refreshed
- 5) If an item has already be claimed and user tries to claim it dialog appears and data in fragment is refreshed

DetailActivity

Complete Transaction Button

- 1) When pressed, it successfully opens the Complete Transaction Dialog

Message Giver Button

- 1) When pressed, it opens up a MessageActivity using the Layer API and Atlas GUI Library

Locate Item Button

- 1) when pressed, it opens up directions to location using Google Maps

Complete Transaction Dialog

- 1) Success: marks the transaction as completed
- 2) Cancel: Puts the item back on market place
- 3) Failure (We will Likely rename this to Report): removes the item from market place

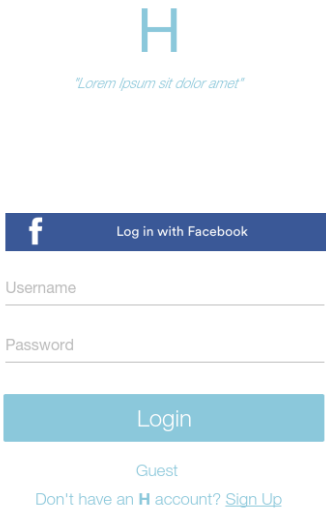
DrawerLayout (Side Menu)

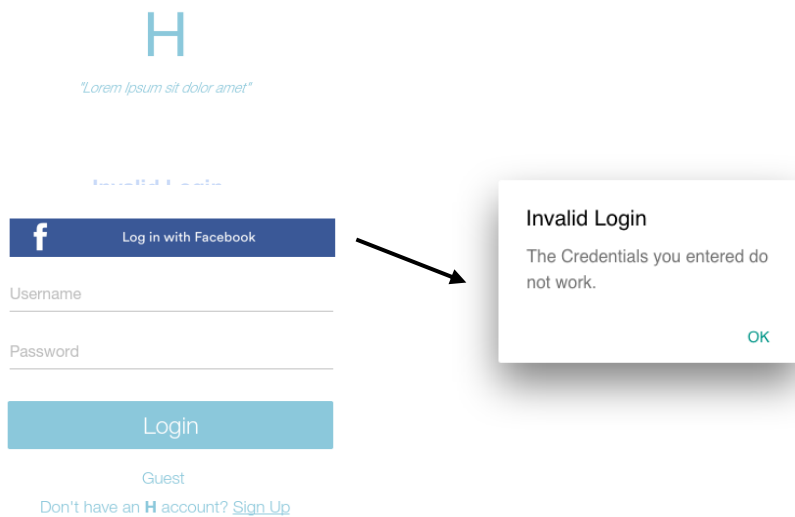
- 1) Clicking the hamburger icon successfully opens up the navigation drawer
- 2) Clicking one of the menu items successfully opens up the correct Activity

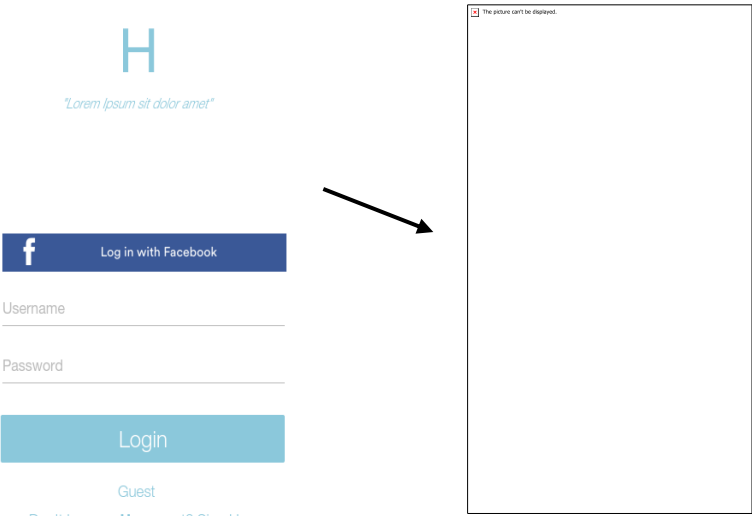
Below are some visuals in test charts

LoginActivity

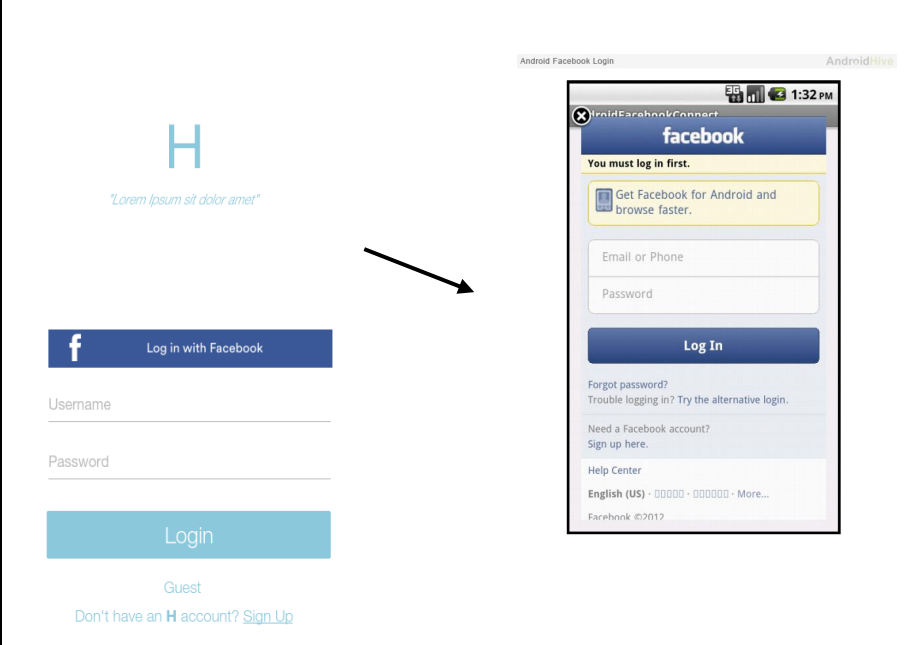
Test #	01
--------	----

Test Description	Press login button with either email or password input box empty
Steps to run test	1. Click the login button with email or password input box empty
Expected Result	Nothing happens
Actual Result	Nothing happens
Screenshot	 <p>The screenshot shows a login interface. At the top is a large blue 'H' logo with the text "Lorem Ipsum sit dolor amet" below it. Below the logo is a dark blue button with a white Facebook 'f' icon and the text "Log in with Facebook". Underneath are two input fields labeled "Username" and "Password". Below these is a light blue "Login" button. At the bottom, there is a "Guest" link and a "Don't have an H account? Sign Up" link.</p>

Test	02
Test Description	Press login button with invalid username and password combination
Steps to run test	<ol style="list-style-type: none"> 1. Input a username 2. Input an invalid username 3. Click login button
Expected Result	Dialog displays "Invalid Login"
Actual Result	<p>Dialog displays "Invalid Login"</p>  <p>The screenshot shows a login interface for a website. At the top, there is a large blue 'H' logo and a line of placeholder text. Below this is a navigation bar with a Facebook logo and a 'Log in with Facebook' button. The main login area contains 'Username' and 'Password' input fields, a blue 'Login' button, a 'Guest' link, and a 'Sign Up' link. An 'Invalid Login' dialog box is overlaid on the right side of the page, with an arrow pointing from the 'Login' button to it. The dialog box contains the text 'Invalid Login' and 'The Credentials you entered do not work.', with an 'OK' button at the bottom right.</p>
Screenshot	

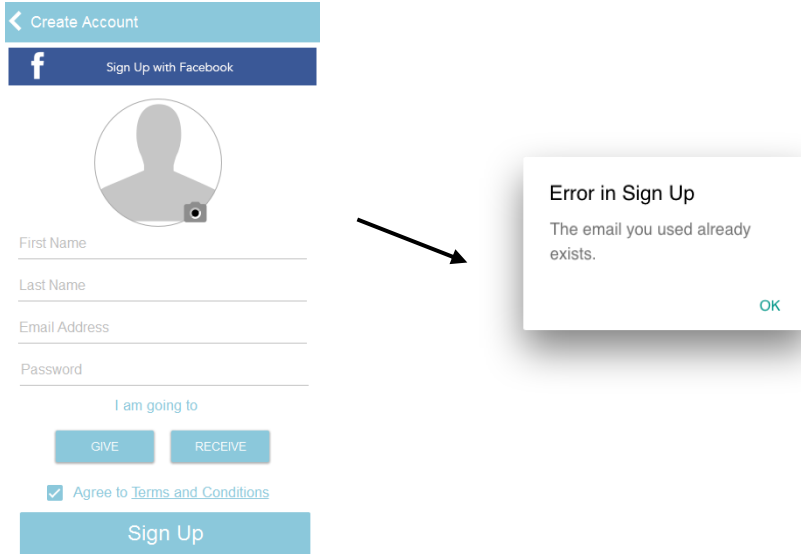
Test	03
Test Description	Pressing login with correct username and password pair
Steps to run test	<ol style="list-style-type: none"> 1. Input correct username and password pair 2. Click login button
Expected Result	Open a new Activity (UserActivity with NewsFeedFragment)
Actual Result	Open a new Activity (UserActivity with NewsFeedFragment)
Screenshot	

Test	04
Test Description	Pressing sign up button
Steps to run test	1) Press a sign up button
Expected Result	Open SignUpActivity
Actual Result	Open SignUpActivity
Screenshot	

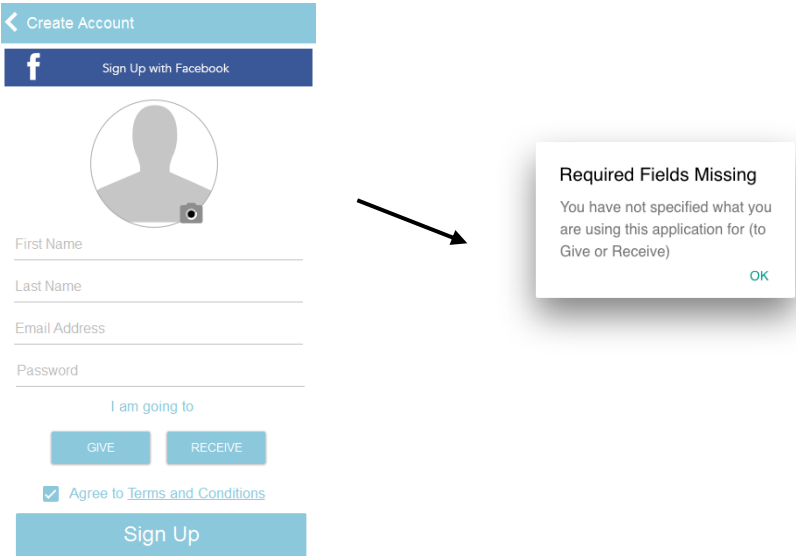
Test	05
Test Description	Pressing “Log in With Facebook” opens Facebook Dialog
Steps to run test	1) Press Log in With Facebook button
Expected Result	Open Facebook Dialog
Actual Result	Open Facebook Dialog
Screenshot	 <p>The screenshot illustrates the test execution. On the left, a placeholder login screen for 'H' is shown with a large blue 'H' logo, a placeholder text 'Lorem ipsum sit dolor amet', and a blue button labeled 'Log in with Facebook'. Below this are fields for 'Username' and 'Password', a blue 'Login' button, and links for 'Guest' and 'Sign Up'. An arrow points from the 'Log in with Facebook' button to a secondary screenshot on the right. This secondary screenshot shows a mobile device screen displaying the actual Facebook login dialog. The dialog has a blue header with the Facebook logo, a message 'You must log in first.', a yellow banner for the Facebook app, input fields for 'Email or Phone' and 'Password', a blue 'Log In' button, and links for 'Forgot password?', 'Need a Facebook account?', and 'Help Center'. The status bar at the top of the device screen shows the time as 1:32 PM.</p>

SignUpActivity

Test #	01
--------	----

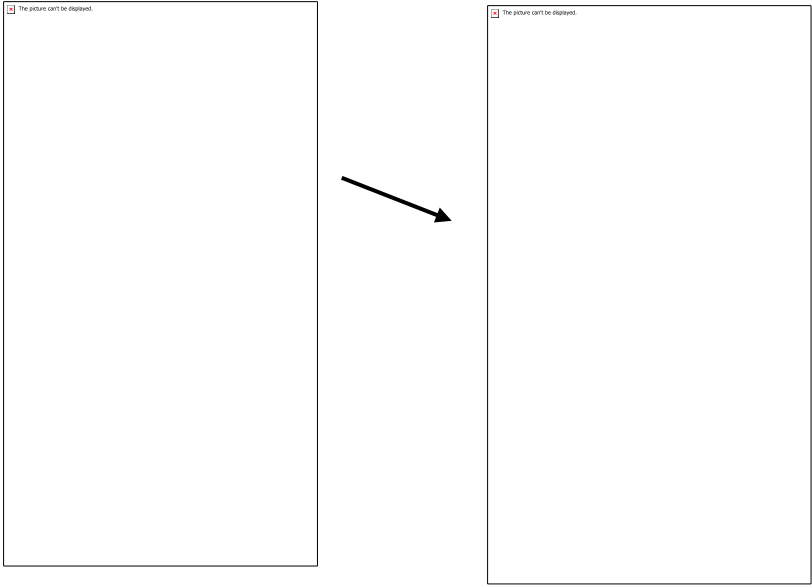
Test Description	Pressing Sign Up button when the email exists in the database
Steps to run test	<ol style="list-style-type: none"> 1. Fill out the blank areas 2. Click Sign Up
Expected Result	Opens up a dialog saying, the email already exists
Actual Result	Opens up a dialog saying, the email already exists
Screenshot	 <p>The screenshot shows a sign-up form on the left and an error dialog on the right. The form includes a back arrow and 'Create Account' link, a 'Sign Up with Facebook' button, a profile picture placeholder, and input fields for First Name, Last Name, Email Address, and Password. Below these are links for 'I am going to', 'GIVE', and 'RECEIVE' buttons, a checked checkbox for 'Agree to Terms and Conditions', and a 'Sign Up' button. An arrow points from the 'Sign Up' button to the error dialog. The error dialog is titled 'Error in Sign Up' and contains the message 'The email you used already exists.' with an 'OK' button.</p>

Test #	01
--------	----

Test Description	Pressing Sign Up button when you have not yet specified whether you are a donor or a receiver
Steps to run test	1. Fill out the blank areas but do not click either “GIVE” or “RECEIVE”
Expected Result	Opens up a dialog saying, the user has not specified
Actual Result	Opens up a dialog saying, the user has not specified
Screenshot	 <p>The screenshot shows a 'Create Account' screen. At the top, there is a blue header with a back arrow and the text 'Create Account'. Below this is a dark blue button with a white Facebook 'f' logo and the text 'Sign Up with Facebook'. Underneath is a circular profile picture placeholder. Below the profile picture are four text input fields labeled 'First Name', 'Last Name', 'Email Address', and 'Password'. Below these fields is a link that says 'I am going to' followed by two buttons: 'GIVE' and 'RECEIVE'. Below the buttons is a checked checkbox followed by the text 'Agree to Terms and Conditions'. At the bottom is a large blue 'Sign Up' button. A white dialog box with a grey border is overlaid on the right side of the screen. The dialog box has the title 'Required Fields Missing' and the text 'You have not specified what you are using this application for (to Give or Receive)'. There is an 'OK' button in the bottom right corner of the dialog box. An arrow points from the 'Sign Up' button to the dialog box.</p>

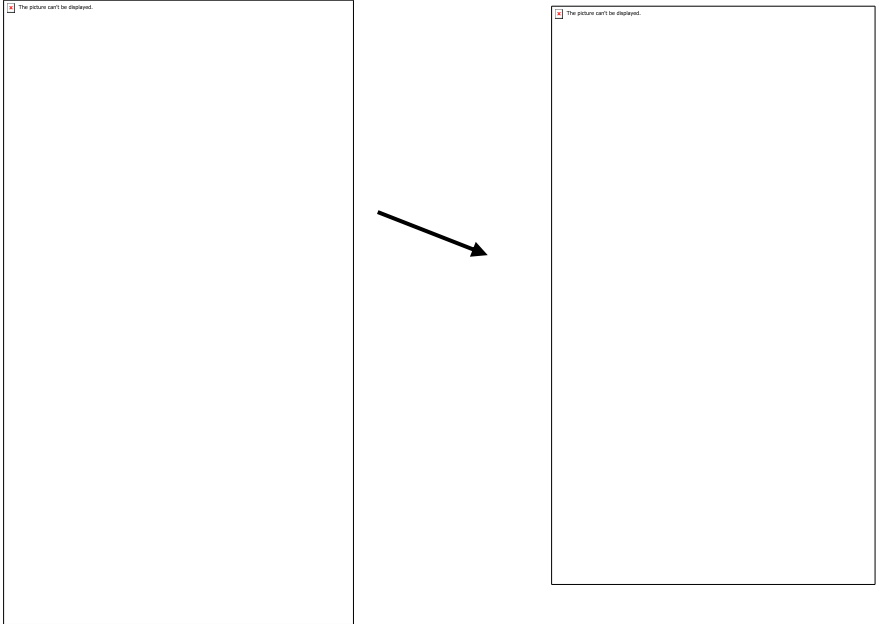
HistoryFragment

Test #	02
Test Description	Clicking any of the items opens the correct DetailsActivity

Steps to run test	3. Complete a transaction 4. Go to History Window 5. Click an item
Expected Result	The correct DetailsActivity opens based on the item clicked.
Actual Result	The correct DetailsActivity opens based on the item clicked.
Screenshot	

NewsFeedFragment

Test #	03
Test Description	Clicking any of the items opens the correct DetailsActivity

Steps to run test	1. Click one of the items
Expected Result	The correct DetailsActivity opens based on the item clicked.
Actual Result	The correct DetailsActivity opens based on the item clicked.
Screenshot	

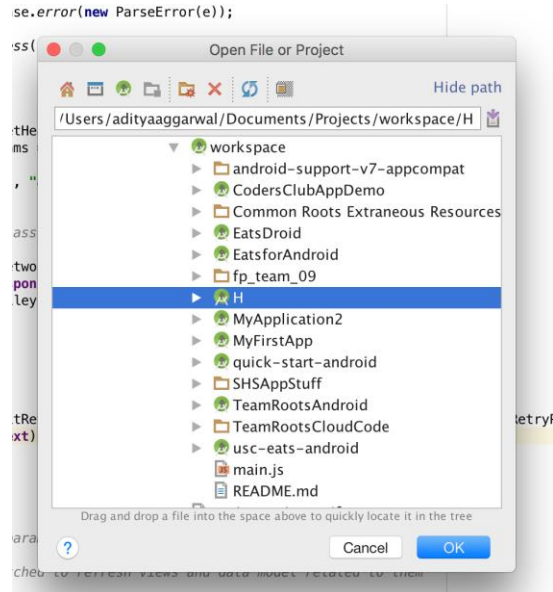
Deployment

We will be deploying this app to the play store so it will be easy for anyone to download.

A member of the team has an Android account listed used Adagg123 and knows how to get most apps up (like this one) in less than 24 hours.

However, you can also open up the app on Android Studio like this:

To deploy this application within Eclipse, import the project labeled 'H' into Android Studio. This should generate a project called H with java, and xml files. The java folder contains the java files that serve as the backend, while the xml files contain the front end design of our App. The drawable folder contains the images we planned to use for the app, to run the app, just press the green arrow button on top of the Android Studio application.



On the server side you will need to open the app in Eclipse NEON or later. You will also need to run an instance of tomcat. And before that you must compile the code with the command

```
mvn clean install package
```