Ramapo College Of New Jersey
School of Theoretical and Applied Science

Senior Project
CMPS 450
Spring 2020

EZ Cash App

Submitted By:

Nitesh Parajuli

Submitted To:

Professor Dr. Victor J. Miller

Paper Abstract:

This purpose of this paper is to describe the usage and design of the application – EZ Cash App. It describes the development process, architecture, coding structure, and challenges encountered. It also gives a brief overview of what a user experiences when using the application. The project is implemented using Android Studio – the official integrated development environment for Google's Android operating system. Java is the programming language used for the development of the application, and Firebase (a platform for creating mobile and web applications developed by Google) and Node.js are used for the backend. In addition, the project uses Dwolla and Plaid APIs to facilitate ACH payments, verify users and their bank account information within the application.

Introduction:

EZ Cash App: An android application that connects a group of people with common interests and backgrounds and facilitates communication and payment services between them.

Project Overview

The goal of the project was to create a simple and effective android application that allowed for ACH payments within a group of students who share their resources with each other. To amplify the user's experience, the communication functionality was later added to the goal of the application.

Eventually, the development of the project will be switched to a production environment with added security, and the application will be uploaded to the Google Play Store for a wider audience reach. Following is a brief description of the contents of this paper to follow:

Section 1: Highlights the goal of the project, the present use cases and possible use cases with slight modifications, and the installation guide.

Section 2: Describes UI flow of the application from the user's perspective. It will contain the working of the application as seen through users using it for the first time.

Section 3: Discusses the documentation and design of the code.

Section 4: Contains a description of all the Java classes used in the project.

Section 5: Includes the challenges faced while working on this project and things that would've been done differently had there been more time and resources. It also includes the things I learned and throughout the process of developing this application.

Section 6: Includes a summary of the overall project and details future steps.

Section 1: How did it All Start?

This section describes the initial objective of the project. Over time, the idea of implementing these objectives changed slightly to better fit the coding architecture and structure. The project is at a point where slight modifications will allow the project to be accessible to varied circumstances. It's adaptable to a lot of difference situations with slight modifications.

Project Inspiration:

The initial inspiration of the project was from an interview that I had for an internship. When I was asked on the projects that I've built, I talked about a simple application that I had developed in Summer 2019 – Split. Split is an android application which helps a group of college student to keep track of their payables and receivables. The purpose of the application is very simple. It does basic algebraic calculation of the various amounts that you owe to someone and vice-versa and determines the final payable/receivable amount. When I discussed about this application in my interview, the interviewer recommended me to integrate payment functionality within the application to make it even better. I took the advice and decided to build a new application which has a slightly different purpose and use it for my senior project.

There are a lot of apps that allow you to send payments and offer chat functionality – Facebook Messenger being one of the most popular, reliable, and dominant. There are, however, not many applications which simply provide both of these functionalities. Venmo is another example which is considered to be a successful payment service, but the users do not have the option to enjoy both functionalities at the same time and within the same platform. There are various groups of people who might need this service, such as roommates, classmates, people who play fantasy sports, etc., and having these services will ease the process for them.

Problem Statement:

> Lack of applications intended for connecting people and offering payment and chat functionalities without having to use two different applications.

Advantages Offered:

1) Easy and quick ACH payment service.
2) Efficient chat service without having to deal with a complex registration process.

3) An isolated platform which does not share any information to other applications or uninvolved parties.
4) Friendly and simple user experience for audiences of all age groups.

Further Usage:

The app can be further extended with simple modifications to fit the needs of group payments. A usage could be making the exact same payment to a large audience. Instead of making each payment multiple times, it'd be much more efficient to do it in one shot. This would be highly beneficial for a start-up company to make a mass payment to their employees.

Due to time constraints, the only features available in the current app are – login, registration, sending friend requests and adding friends, messaging service, and one-to-one payment service. Based on the profile set by a user, other users will be able to send the request and find what they are up to. Features like declining a request and unfriending is also available. Further steps would require making group chat, a dashboard which displays other friends thoughts and status, and including voice/video call APIs. There is a lot of room for growth in this app.

Project Features:

This project has the following functionality:

1. User login and registration.
2. Set up user profiles including display images and a status.
3. Access to a community of users with their basic profiles registered with the application and ability to send a friend request if interested in networking.
4. Ability to send text messages and images to a friend.
5. Ability to change profile pictures and status if needed.
6. Verify a bank account simply with their online banking login credentials. This will come in handy if a user lost or forgot their debit cards, as debit card information is hard to memorize as compared to online banking login details.
7. Ability to use almost any bank within the U.S.
8. Payments will only be made if both sender and receiver have a verified bank. This will reduce the chances of the payment being stuck in between.

Installation Guide:

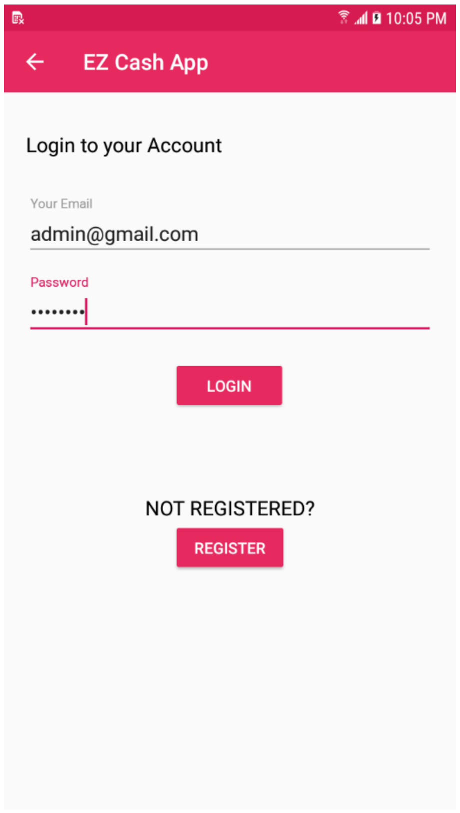
EZ Cash App is currently not available to download from the google play store. Here’s the step by step guide to install EZ Cash App in an Android Phone.

1) Download and install the latest version of Android Studio on a device.
2) Contact me at nparajul@ramapo.edu.
3) Either download the project from the GitHub (link will be provided) or extract the submission zip file into a directory.
4) Open Android Studio and click the option to open an existing project. Select the project downloaded folder.
5) Follow android studio’s instructions and build the project. Run the project by connecting an android device and get ready to explore everything that the application has to offer.

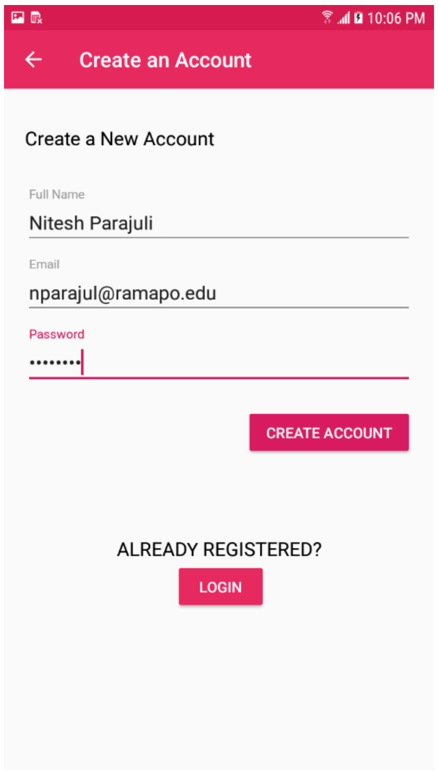
Section 2: Application Flow

With Firebase taking care of the login, registration, database, and image storage, and a Node.js server taking care of the ACH Payment functionalities, the focus of the application has been to implement event driven programming.

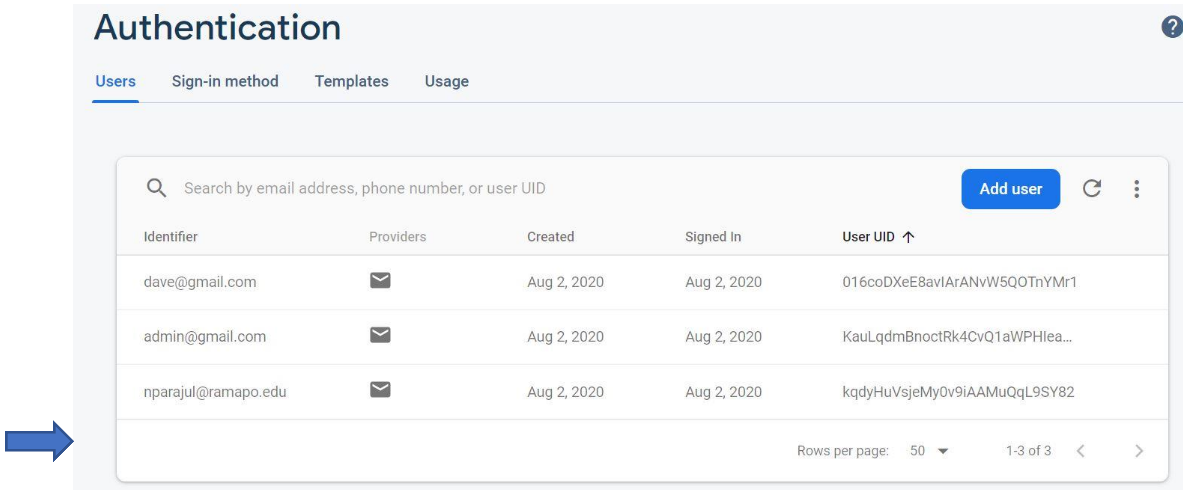
After successfully installing the application, the Login Page is the first thing a user will see when the application is opened. The login page allows user to enter their email and password to log in to the application.



Additionally, the page also contains a register button which redirects users - who haven't created an account before – to the registration page. Once the user creates an account, their information will be stored in the Firebase database under the collection named "users". The userId that will be obtained while using firebase's library to sign up with email and password will also be stored under the collection.

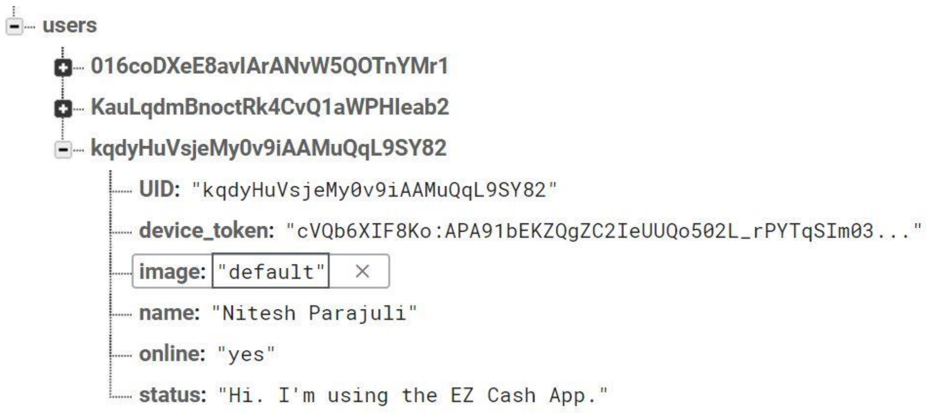


The authentication and real time firebase database will look as the follows when a user with the email ‘admin@gmail.com’ signs up.



Identifier	Providers	Created	Signed In	User UID
dave@gmail.com	✉	Aug 2, 2020	Aug 2, 2020	016coDXeE8avlArANvW5QOTnYMr1
admin@gmail.com	✉	Aug 2, 2020	Aug 2, 2020	KauLqdmBnoctRk4CvQ1aWPHlea…
nparajul@ramapo.edu	✉	Aug 2, 2020	Aug 2, 2020	kqdyHuVsjeMy0v9iAAMuQqL9SY82

Fig. Firebase Authentication

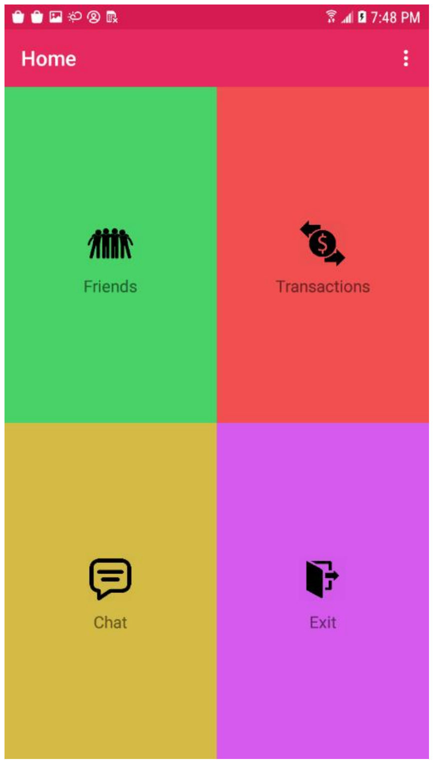


```
users
  016coDXeE8avlArANvW5QOTnYMr1
  KauLqdmBnoctRk4CvQ1aWPHleab2
  kqdyHuVsjeMy0v9iAAMuQqL9SY82
    UID: "kqdyHuVsjeMy0v9iAAMuQqL9SY82"
    device_token: "cVQb6XIF8Ko:APA91bEKZQgZC2IeUUQo502L_rPYTqSIm03..."
    image: "default"
    name: "Nitesh Parajuli"
    online: "yes"
    status: "Hi. I'm using the EZ Cash App."
```

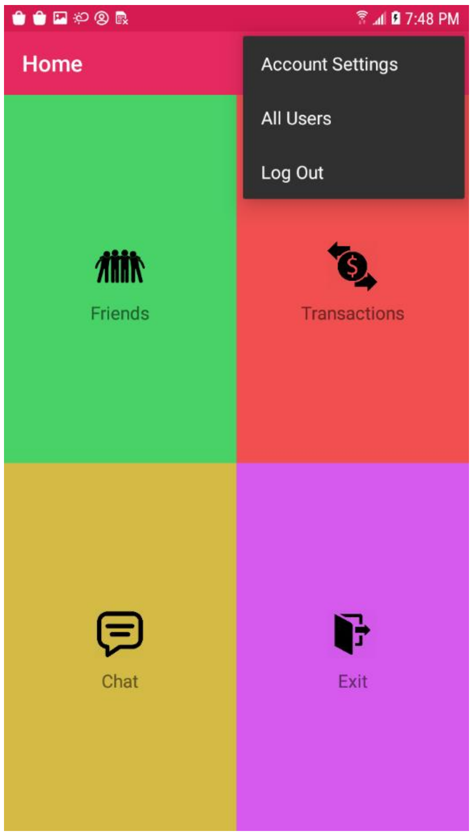
Fig. Firebase real-time database

As seen in the screen capture above, once Firebase libraries are used for signing up, Firebase provides a unique Id for the user. This ID is taken and saved in the Firebase real time database under the users collection along with the username, device token, online presence, image (set to default initially) and status (also set to a default status upon successful registration). After the user profiles are set up, the image and status values will be updated.

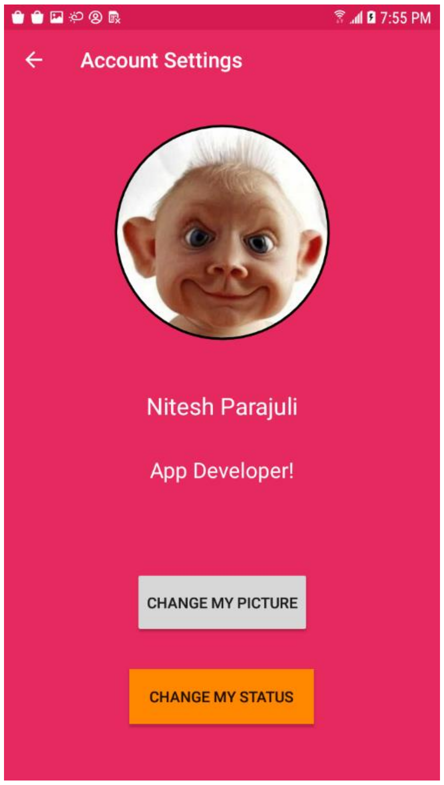
Once users login or register successfully, they will be redirected to the homepage of the application. The homepage has 4 primary clickable layouts that directs the user to different pages.



Also, it contains a toolbar menu containing 3 menu options: Account Settings, All Users, and Logout.



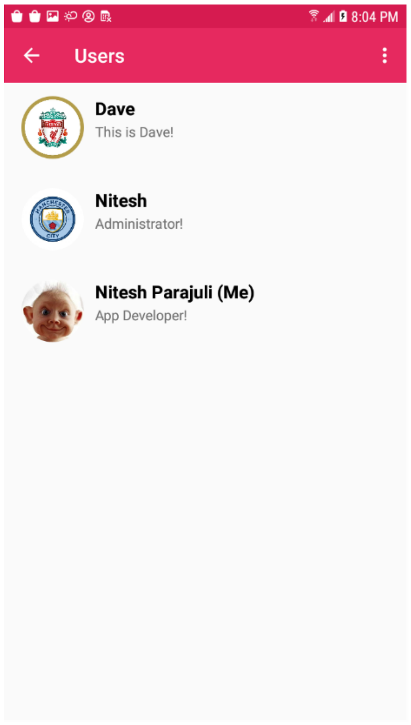
When the user clicks on Account Settings, a new page containing their profile picture and status, both of which have a default value initially, will open. Additionally, two buttons named “Change My Picture” and “Change my Status” will be located on the bottom half of the page. Clicking on the first will open the phone’s gallery and allow users to choose a picture, crop it, and select it as their profile picture. Clicking on the later will open a page containing a text field where the user can type their status and upon clicking the button their status will change. Either of these changes will update the respective value in the firebase database. After completing both of these steps, the accounts settings page will look like below.



Nitesh Parajuli

App Developer!

When the user clicks on All Users, a new page will appear with a list of all users who have registered with the application. This uses a RecyclerView to populate the page with information of Users as saved on their profiles, as shown in the picture below. When an item in the

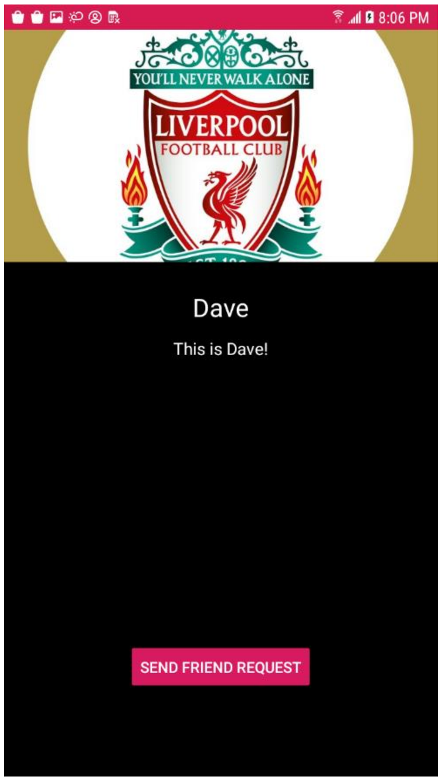


Dave
This is Dave!

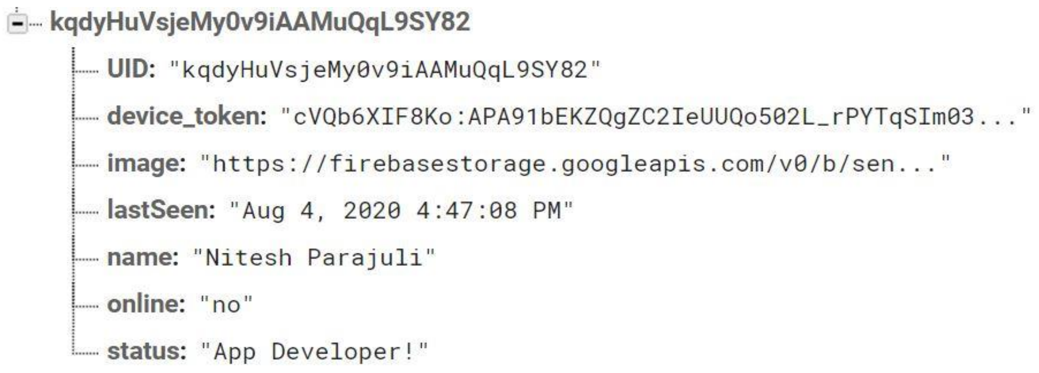
Nitesh
Administrator!

Nitesh Parajuli (Me)
App Developer!

RecyclerView is clicked, the user's profile page will load, and other users will be able to view their profile picture, status, and send a friend request. If a friend request is sent to the user, the button will change to "cancel the request" and will delete the friend request if clicked. If a user has already sent a friend request to the current user, the profile page will have two buttons - one for accepting the request and the other for declining the request. Accepting the request will create a new collection named friends and add the user ids of the user and store the date of when the request was accepted.

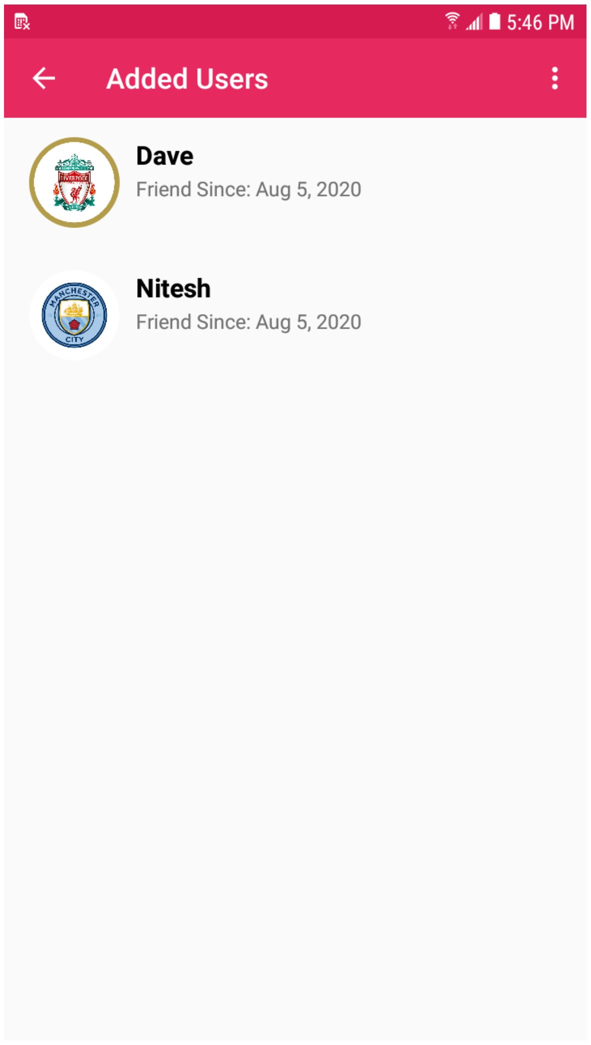


When the user clicks on Logout, the online status of the user will be changed to indicate offline presence and the timestamp will be set to the lastSeen value in the database. The user will be directed to the login page.



```
kqdyHuVsjeMy0v9iAAMuQqL9SY82
    UID: "kqdyHuVsjeMy0v9iAAMuQqL9SY82"
    device_token: "cVQb6XIF8Ko:APA91bEKZQgZC2IeUUQo502L_rPYTqSIm03..."
    image: "https://firebasestorage.googleapis.com/v0/b/sen..."
    lastSeen: "Aug 4, 2020 4:47:08 PM"
    name: "Nitesh Parajuli"
    online: "no"
    status: "App Developer!"
```

When the user clicks on Friends in the main activity, a list of users who are friends will be displayed. Like the all users page, a RecyclerView is used to populate the friends page with their name, image, and the date when the request was accepted. When a friend is clicked, an Alert dialog builder is shown with two options: View Profile and Send a Message. On clicking view profile, users will be directed to their friend's profile. The profile image and status will be shown along with an unfriend button. Clicking the unfriend button will delete their friendship and the record from the database.

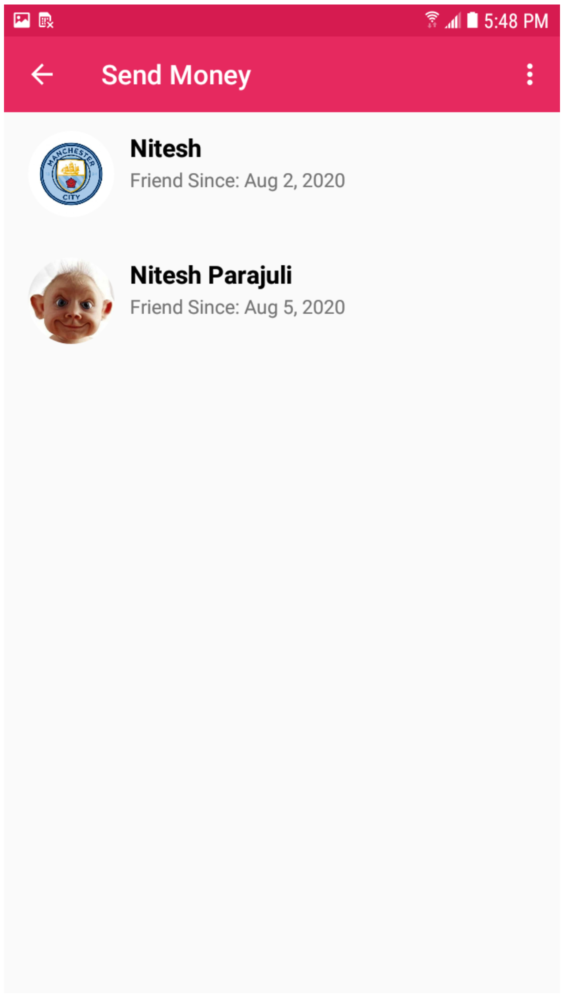


Dave
Friend Since: Aug 5, 2020

Nitesh
Friend Since: Aug 5, 2020

When send a message is clicked, the user will be directed to the messages page and is able to send a message to the friend. The details of this page will be discussed later in the chat section.

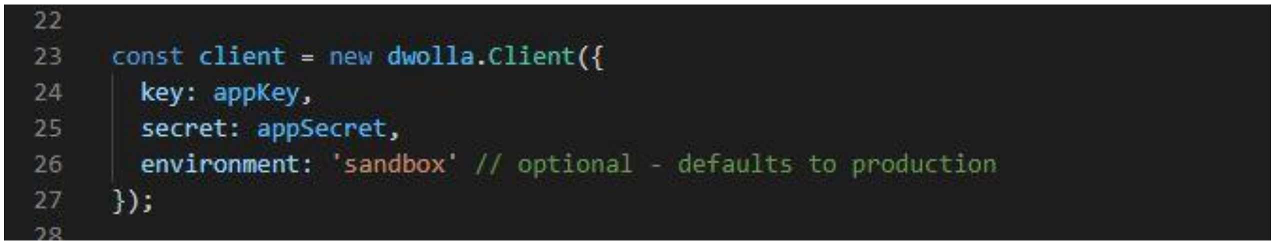
When the user clicks on Transactions in the main activity, a list of users who are friends will be displayed. The information shown is exactly the same as the Friends page. However, when a friend is clicked, the Alert dialog builder is shown with slightly different options: View Profile and Send Money. On clicking view profile, users will be directed to their friend's profile. The profile image and status will be shown along with an unfriend button. Clicking the unfriend button will delete their friendship and the record from the database.



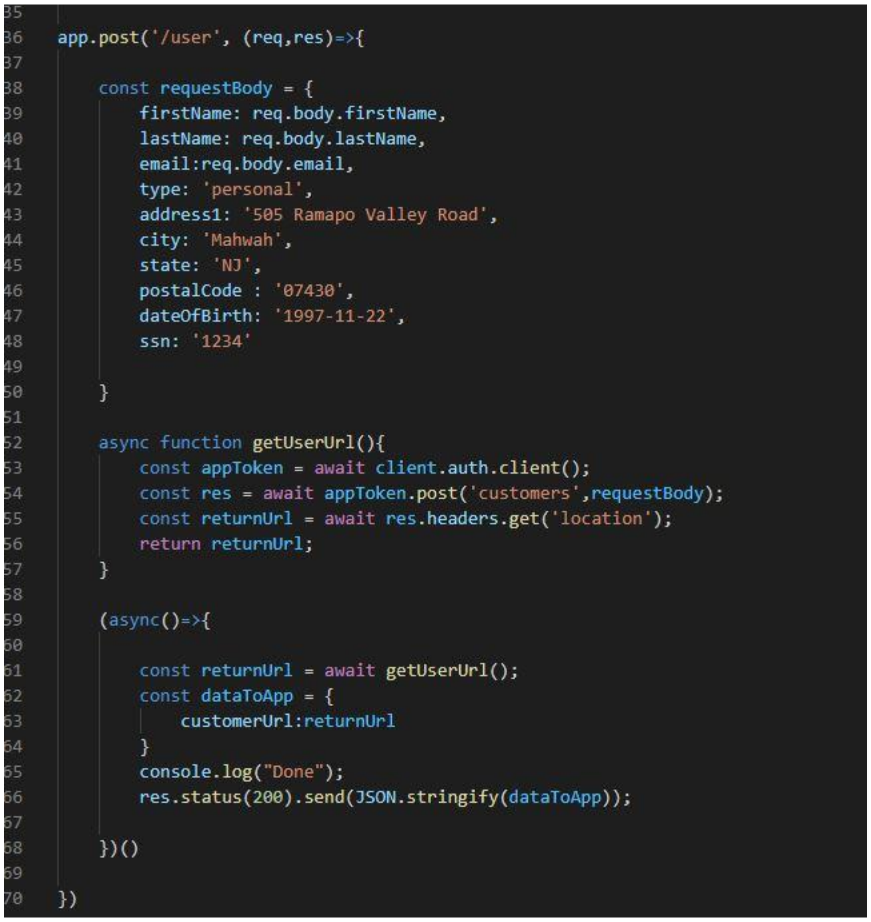
Nitesh
Friend Since: Aug 2, 2020

Nitesh Parajuli
Friend Since: Aug 5, 2020

When send money is clicked, a toast message will appear on the screen if the receiver hasn't yet verified his bank. Bank verification should be done through the menu options in the transactions page. The menu item contains Account settings (acts the same way as it does when it's clicked in the homepage) and Verify my bank options. When the user clicks on verify my bank, a different registration page will open. This registration page is responsible to add the user as a verified customer on Dwolla's end. When users fill out their information and click the button, a POST request will be made to the backend nodejs server.

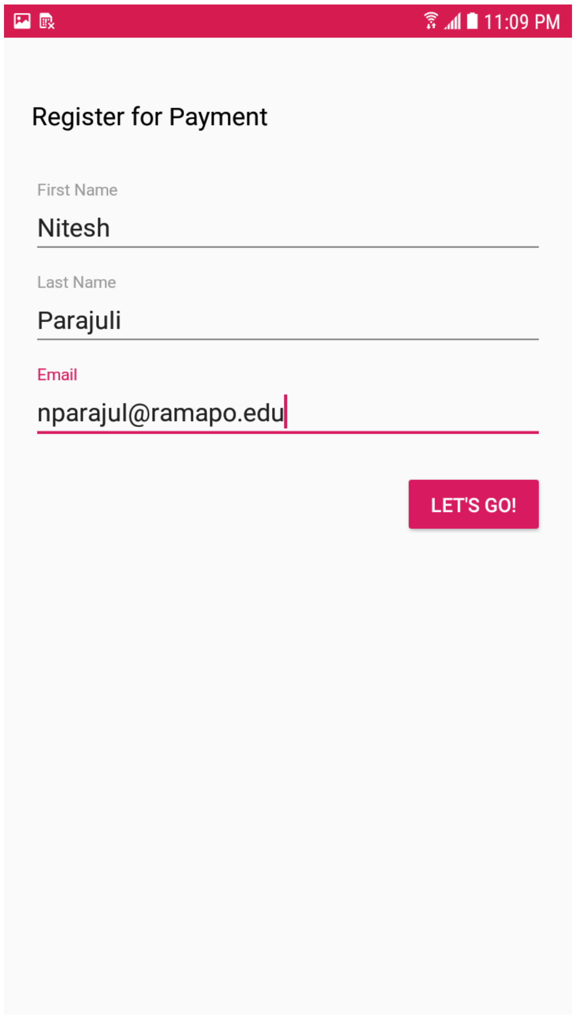


```
const client = new dwolla.Client({
  key: appKey,
  secret: appSecret,
  environment: 'sandbox' // optional - defaults to production
});
```



```
app.post('/user', (req,res)=>{

    const requestBody = {
        firstName: req.body.firstName,
        lastName: req.body.lastName,
        email:req.body.email,
        type: 'personal',
        address1: '505 Ramapo Valley Road',
        city: 'Mahwah',
        state: 'NJ',
        postalCode : '07430',
        dateOfBirth: '1997-11-22',
        ssn: '1234'

    }

    async function getUserUrl(){
        const appToken = await client.auth.client();
        const res = await appToken.post('customers',requestBody);
        const returnUrl = await res.headers.get('location');
        return returnUrl;
    }

    (async()=>{

        const returnUrl = await getUserUrl();
        const dataToApp = {
            customerUrl:returnUrl
        }
        console.log("Done");
        res.status(200).send(JSON.stringify(dataToApp));

    })()

})
```

The request is handled as per Dwolla's API documentation and a customerUrl is returned as a response and stored in the payment_details collection for each user. For this project, users are only asked for their first and last name, and their email. When the application will go in production in the future, information like address, last 4-digits of social security number, phone number, etc. will also be required.



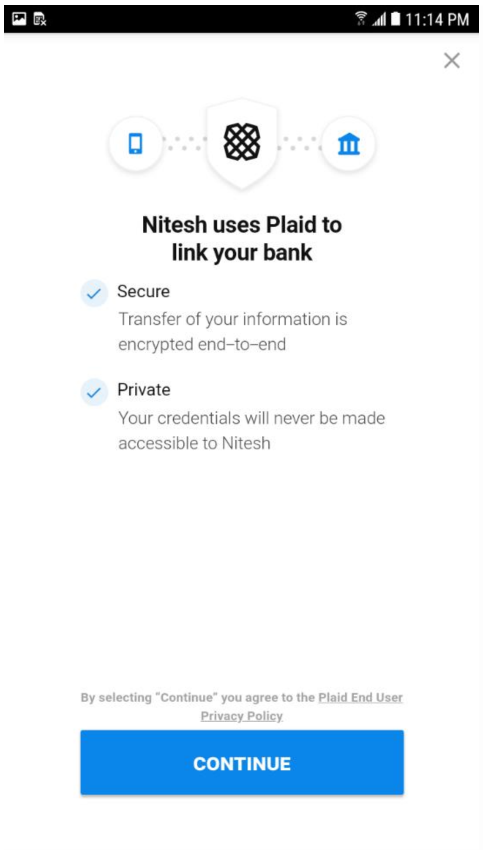
Register for Payment

First Name
Nitesh

Last Name
Parajuli

Email
nparajul@ramapo.edu

After the user is added as a verified user with Dwolla, a new screen will appear which facilitates the user to verify their bank information. This functionality is done with the help of Plaid, which has an integration with Dwolla. The verification is very simple. Users will be displayed a list of common banks and a search bar if the bank isn't listed. Plaid supports nearly all banks within the U.S. so users shouldn't face any inconvenience during this process.



11:14 PM

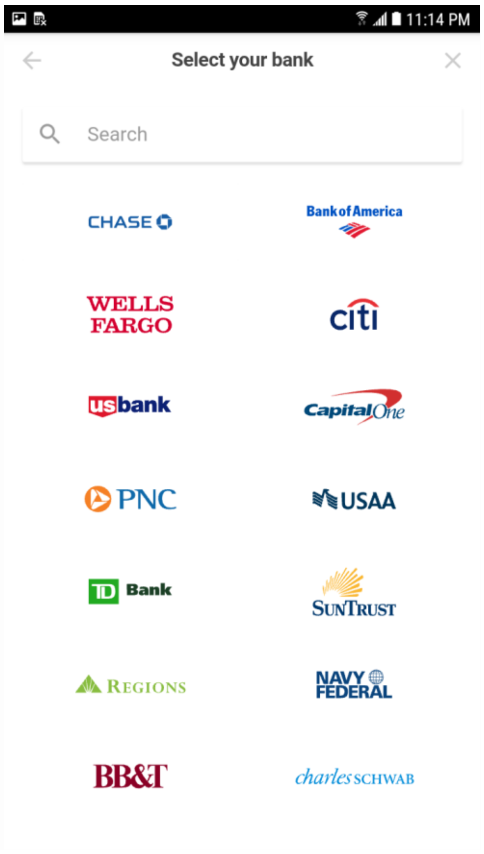
Nitesh uses Plaid to link your bank

Secure
Transfer of your information is encrypted end–to–end

Private
Your credentials will never be made accessible to Nitesh

By selecting "Continue" you agree to the Plaid End User Privacy Policy

CONTINUE

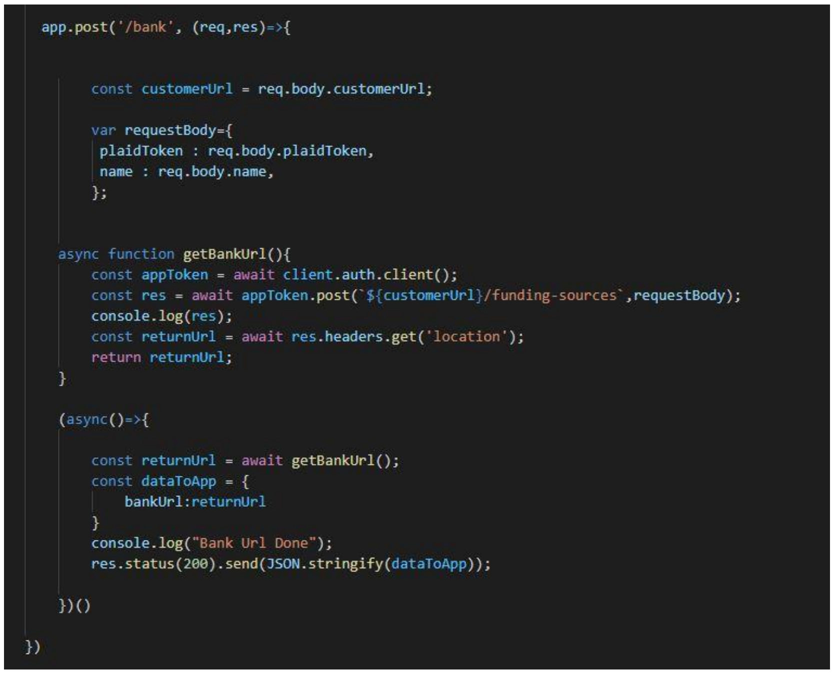


11:14 PM

Select your bank

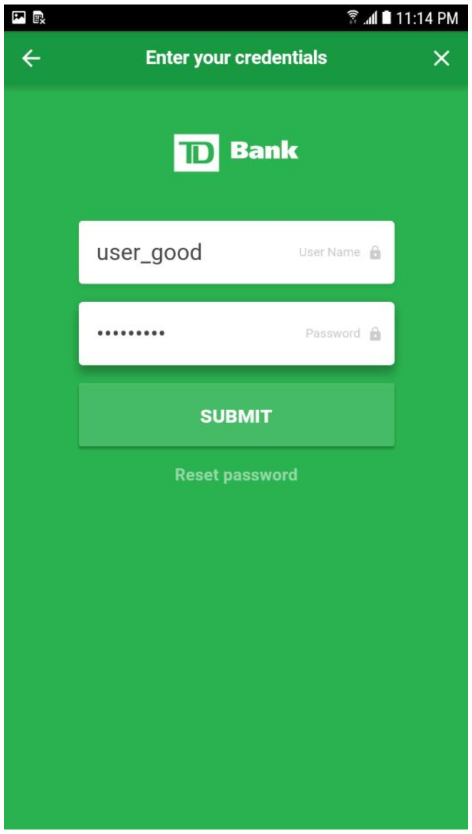
Search

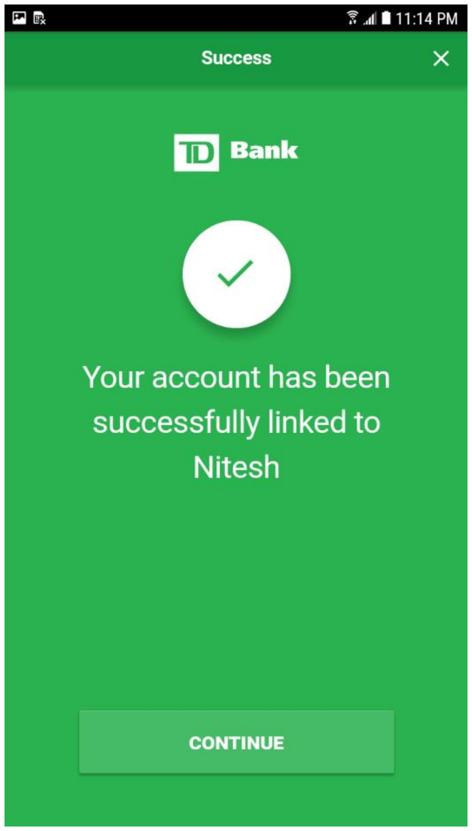
After they choose their bank and fill out their login credentials, Plaid will verify the bank and return a response body which contains a unique processor token. Then, another POST request will be made to the backend nodejs server where the processor token is passed with the request body.



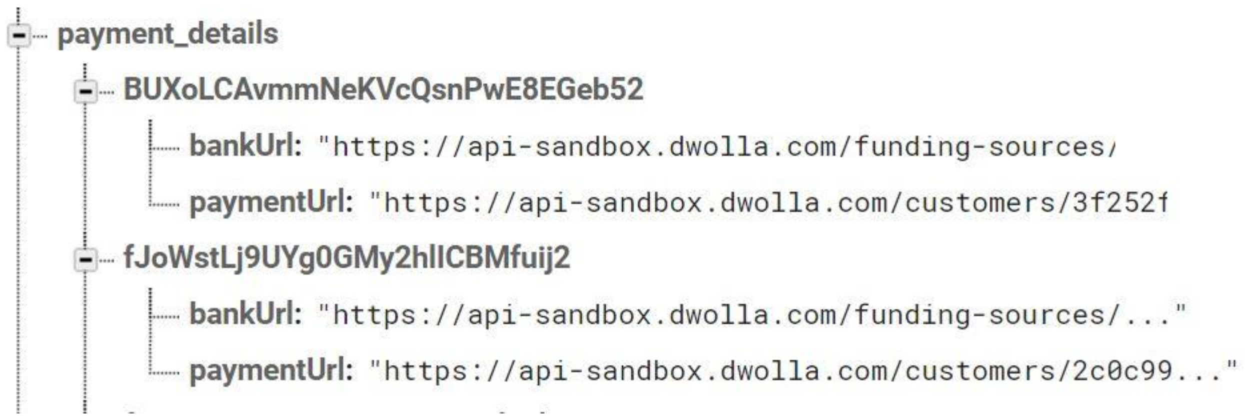
```
  app.post('/bank', (req,res)=>{


        const customerUrl = req.body.customerUrl;

        var requestBody={
         plaidToken : req.body.plaidToken,
         name : req.body.name,
        };


    async function getBankUrl(){
        const appToken = await client.auth.client();
        const res = await appToken.post(`${customerUrl}/funding-sources`,requestBody);
        console.log(res);
        const returnUrl = await res.headers.get('location');
        return returnUrl;
    }

    (async()=>{

        const returnUrl = await getBankUrl();
        const dataToApp = {
            bankUrl:returnUrl
        }
        console.log("Bank Url Done");
        res.status(200).send(JSON.stringify(dataToApp));

    })()

})
```

The request is handled as per Dwolla's API documentation and a unique bankUrl is returned as a response and stored in the payment_details collection for each user. For this project, the application is working in the sandbox(testing) environment due of restrictions such as the cost involved with production environment subscription, and Dwolla's requirement to give production access to verified businesses only. As this application is currently developed for my senior project only, I am not able to fulfill these requirements. One of the future goals with the project is definitely pushing the application to a production environment.



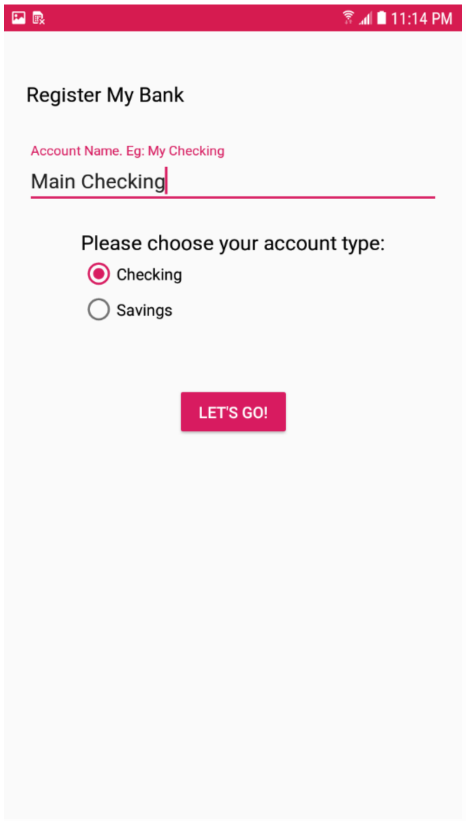


Upon user account creation and bank verification, the tokens will be added to the payment_details collection in the firebase database. The bankUrl will store the verified funding source link and paymentUrl will store the user's account link.



```
payment_details
    BUXoLCAvmmNeKVcQsnPwE8EGeb52
        bankUrl: "https://api-sandbox.dwolla.com/funding-sources/
        paymentUrl: "https://api-sandbox.dwolla.com/customers/3f252f
    fJoWstLj9UYg0GMy2hlICBMfuij2
        bankUrl: "https://api-sandbox.dwolla.com/funding-sources/..."
        paymentUrl: "https://api-sandbox.dwolla.com/customers/2c0c99..."
```

After the bank registration process is completed and the database is updated, a user should be able to send money to their friends provided that the receiver has also completed this process. If the receiver hasn't done it yet, a toast message - indicating so - will be displayed when send money option is clicked in the alert builder. Else, the user will be directed to the transfer money page and it will contain a field for the users to input the amount in USD and upon pressing the send button, the transaction will start. A POST request is made to the backend server with the amount, sender's bankUrl and receiver's bankUrl. The response will contain the status of the transfer which will always be pending initially as ACH transfers take a day to process. For our sandbox environment, the Dwolla dashboard provides a process bank transfers functionality and all transfers will be processed. The amount will be reflected in the receiver's funding source on the next day.



11:14 PM

Register My Bank

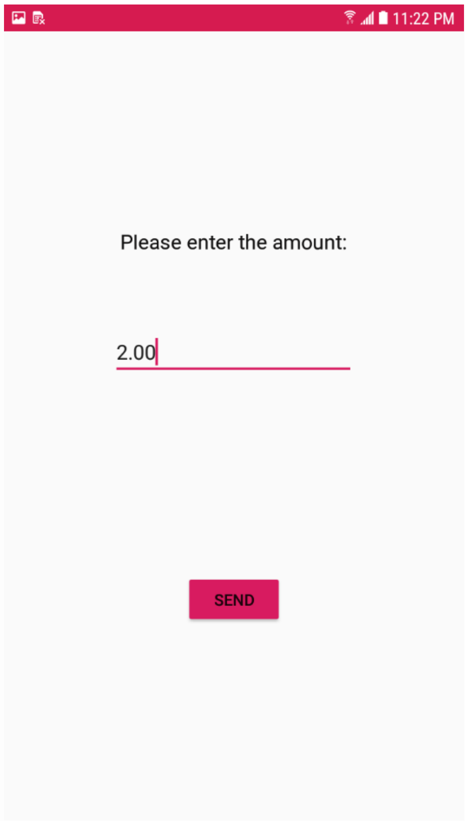
Account Name. Eg: My Checking

Main Checking

Please choose your account type:

[✓ Checking]

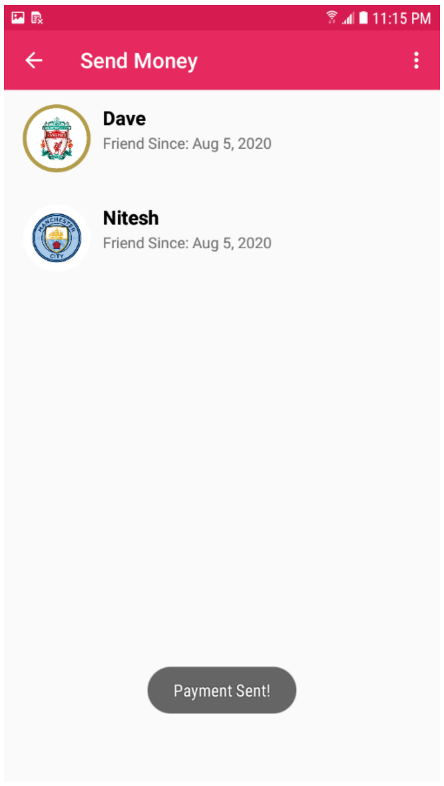
Savings



11:22 PM

Please enter the amount:

2.00



11:15 PM

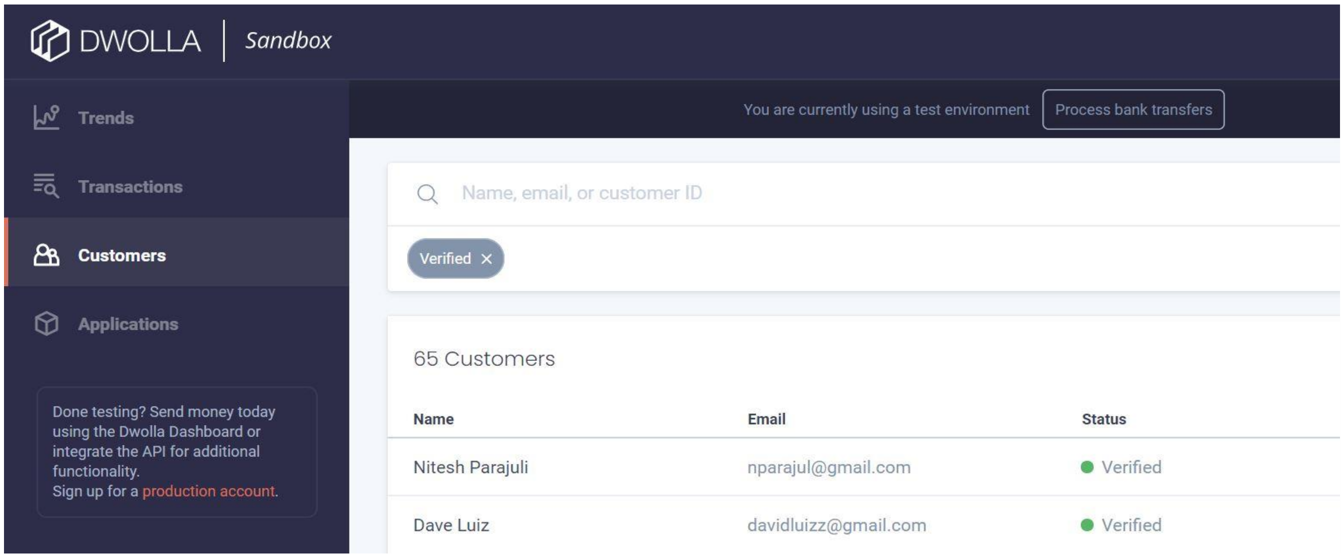
Send Money

Dave
Friend Since: Aug 5, 2020

Nitesh
Friend Since: Aug 5, 2020

Payment Sent!

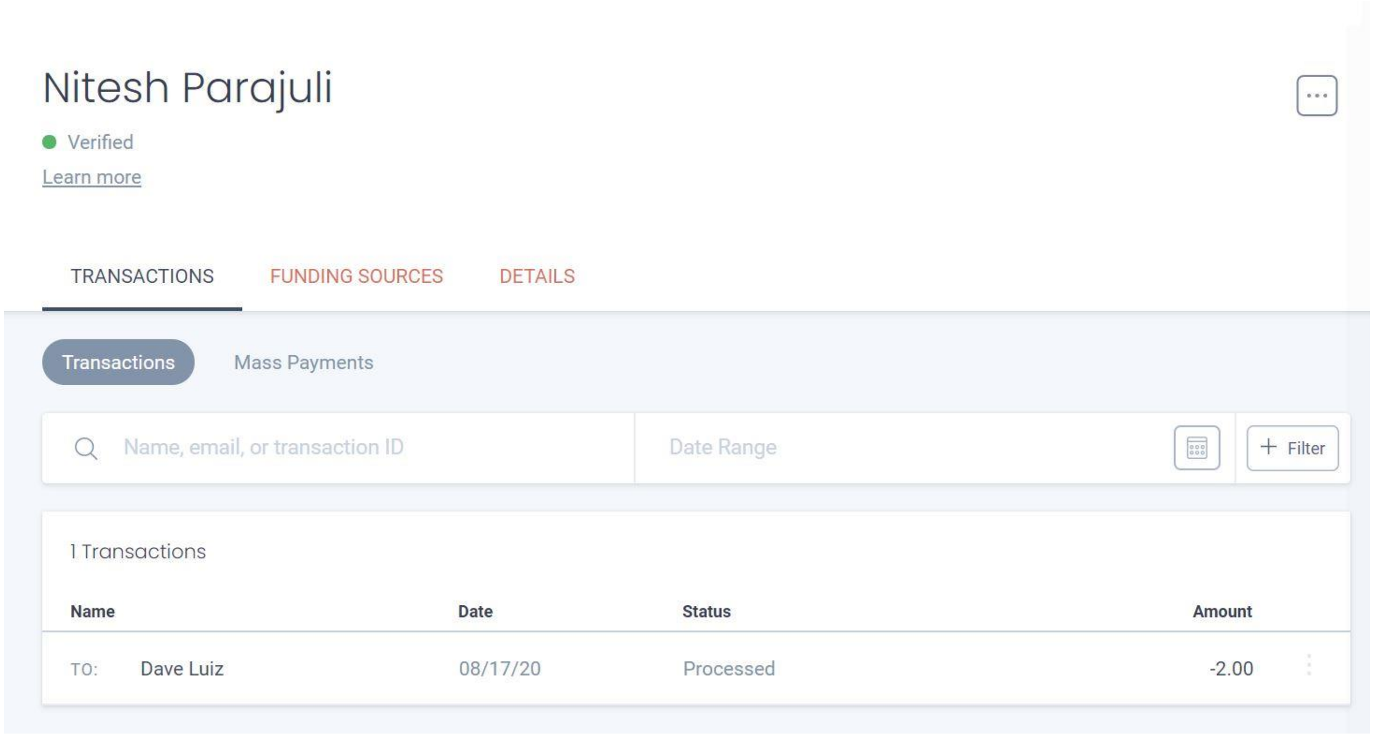
The transaction will be reflected in Dwolla's dashboard with the status of the payment. As discussed earlier, ACH payments will take a business day to process.



Done testing? Send money today using the Dwolla Dashboard or integrate the API for additional functionality.
Sign up for a production account.

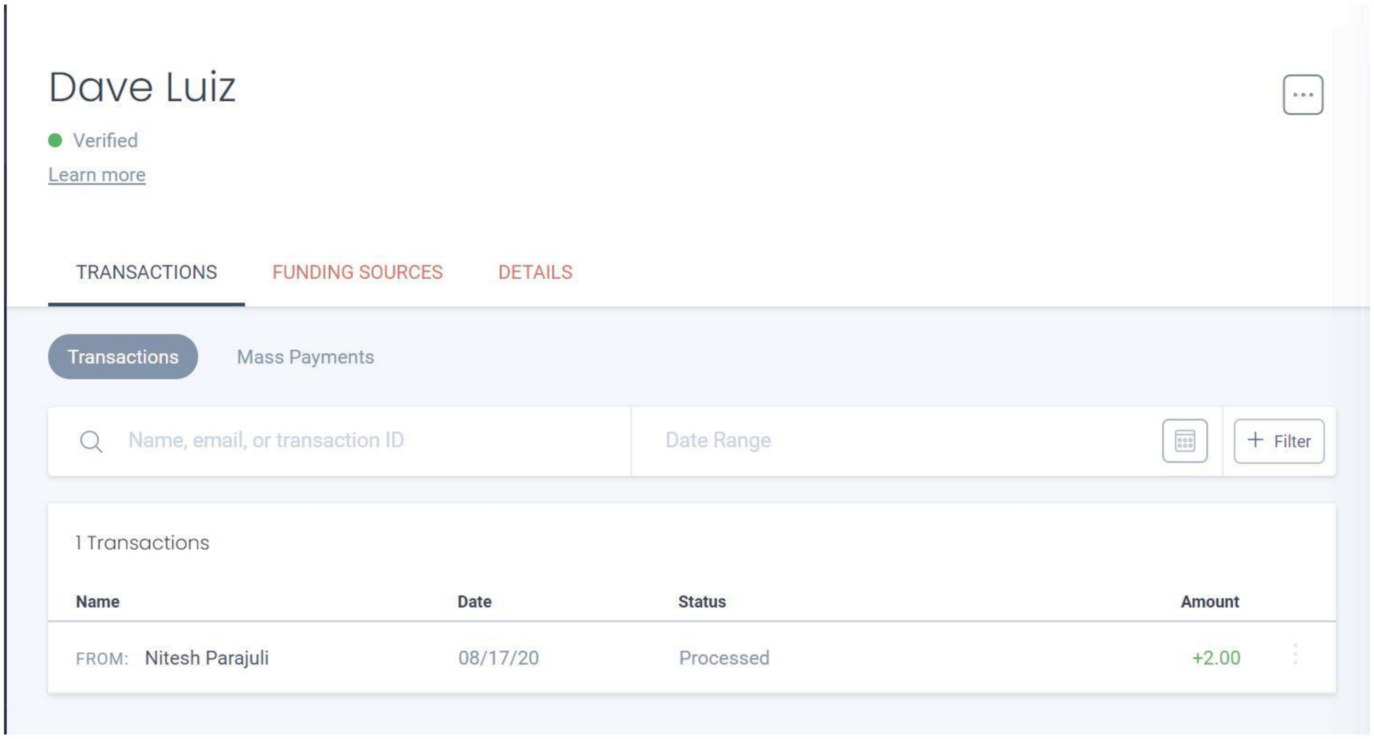
65 Customers

Name	Email	Status
Nitesh Parajuli	nparajul@gmail.com	● Verified
Dave Luiz	davidluizz@gmail.com	● Verified



1 Transactions

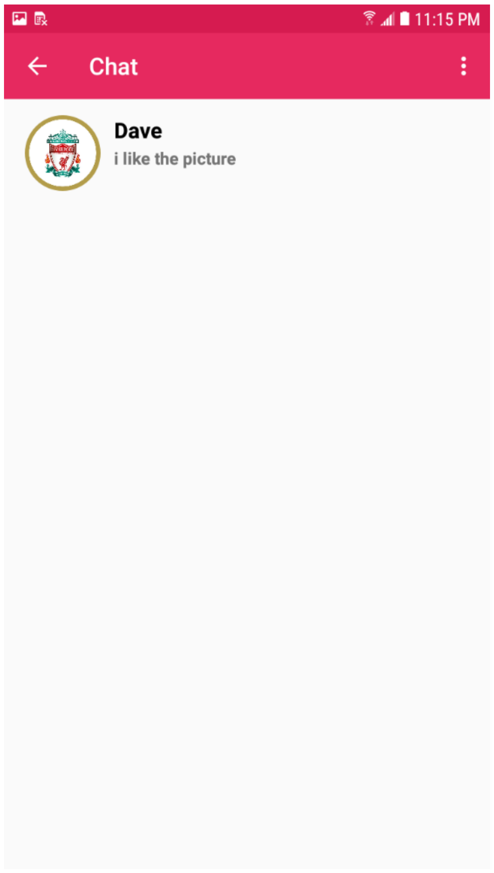
Name	Date	Status	Amount
TO: Dave Luiz	08/17/20	Processed	-2.00



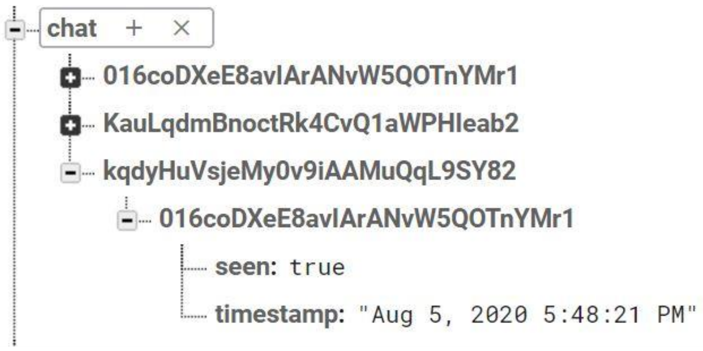
Name	Date	Status	Amount
FROM: Nitesh Parajuli	08/17/20	Processed	+2.00

All payment related services are handled server-side through the nodeJS server.

When the user clicks on Chat in the main activity, a list of friends who have sent a message to the current user or received a message from the current users will be displayed. Like the all users, transactions, and friends page, a RecyclerView is used to populate the chat page with users' name and last message sent/received. If the current user has received a message but hasn't opened the message yet, the item in the recycler view will be in bold.



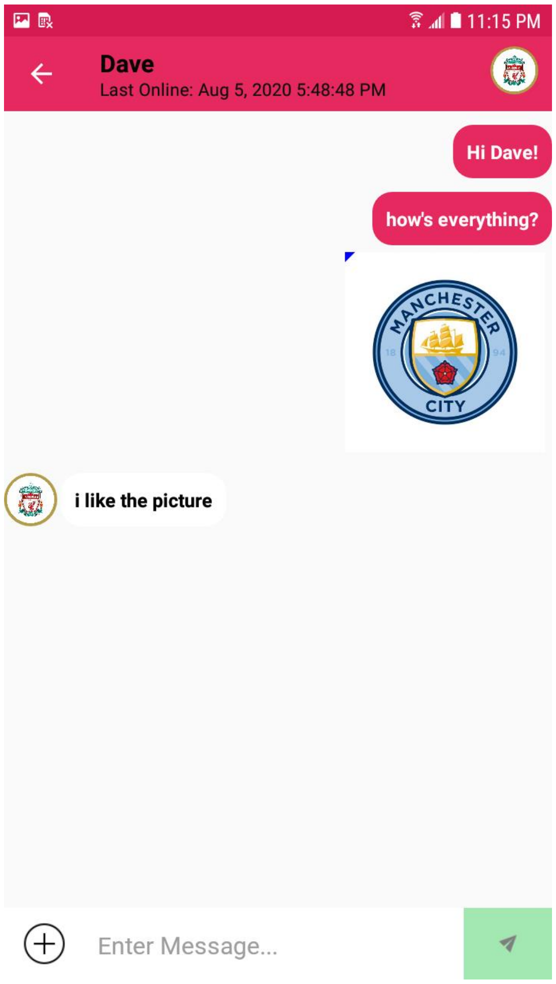
The information loaded on chat page is stored in the chat collection in firebase database. When a user sends their first message to a friend from the friends page, the record will be stored in chat collection. It will contain a Boolean seen value and the timestamp. Also, a new record will also be added to the messages collection and all text and image messages sent/ received will be added to the messages collection.



```
chat
  016coDXeE8avlArANvW5QOTnYMr1
  KauLqdmBnoctRk4CvQ1aWPHleab2
  kqdyHuVsjeMy0v9iAAMuQqL9SY82
    016coDXeE8avlArANvW5QOTnYMr1
      seen: true
      timestamp: "Aug 5, 2020 5:48:21 PM"
```

When a friend is clicked in chat activity, the messages screen will load. It will contain all the messages sent and received by the current user with their friend. The page will also display the online presence if the user is online and the lastSeen timestamp if the user is offline. The messages sent by the current users will be aligned on the right whereas the messages received

will be on the left. The page offers a scroll functionality to allow users to access previous messages sent or received.
On the bottom of the screen, the user will see a text field, and two clickable image views. The one on the left, representing add media item, will allow user to choose an image from gallery and send it to their friend whereas the one on the right is the send message button.



After a message is sent, the messages collection in the database will be updated and will append the new message to the collection.

Section 3: Code Design

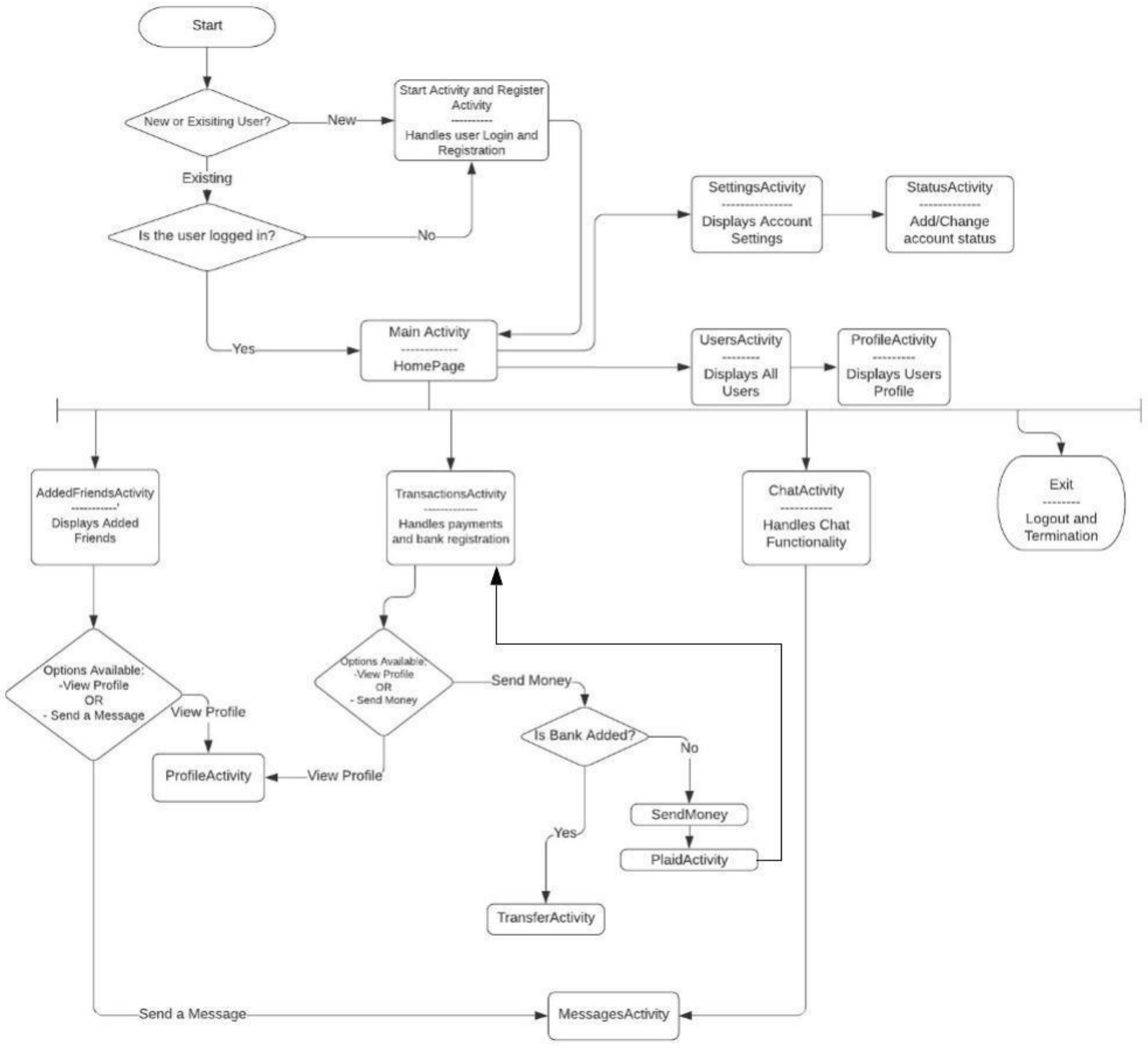
EZ Cash App is written in Java. The primary reason for choosing Java as the programming language is for its compatibility with Android Studio and since it is an event driven language. Firebase libraries are used for interaction with Firebase database and for all data related interaction with the back end. Since the application is based on events as determined by the user, and the data isn't stored in the app, rather is reflected based on minor changes in the real-time Firebase database, the code hasn't been divided into the Model View Controller design. Rather, the entire application follows as event driven design, and within the activity-based classes, data is taken from and stored in the real-time database based on user events.

To display the list of all users, friends, and chat users as shown in different screenshots in section 2, the application uses RecyclerView. RecyclerView displays a vertically scrollable collection of views, where each view is positioned immediately below the previous view in the list. It is modern and flexible approach to displaying lists and is enough to meet the needs of this project.

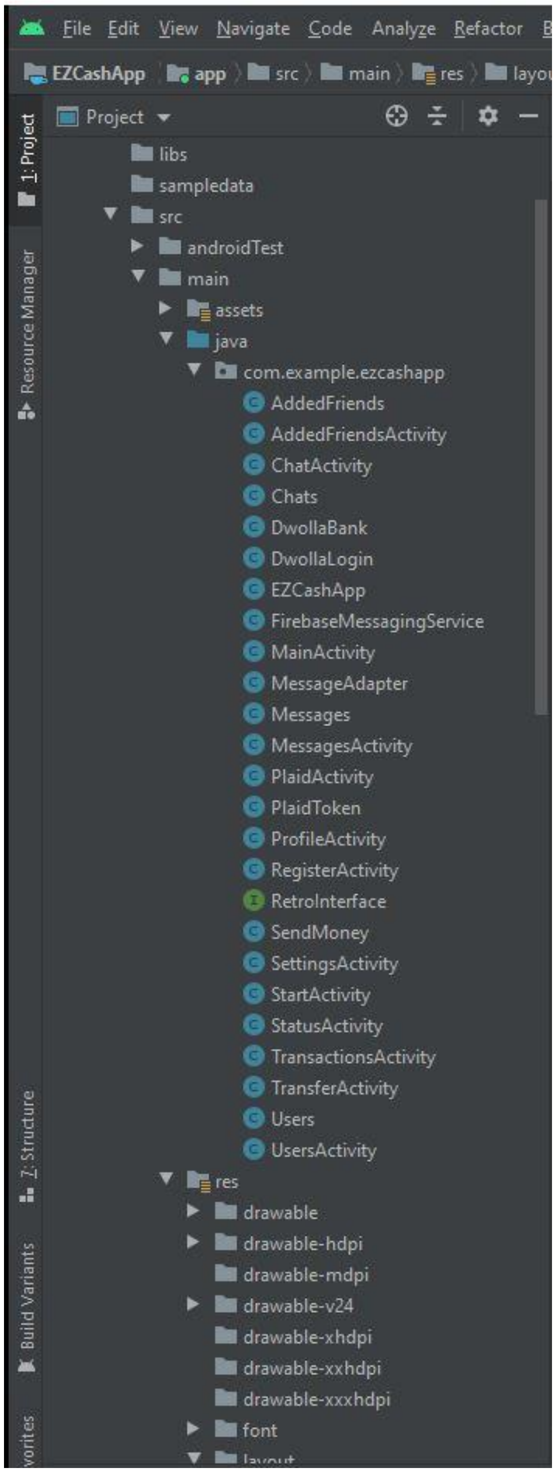
RecyclerView is similar to a listview or adapterview, but it requests views on demand from a FirebaseRecyclerAdapter in cases where new views are needed to be displayed on demand when the users scroll up or down. In order to display items in the list, setAdapter(FirebaseRecyclerAdapter object) is called to associate an adapter with the list.

To enable communication with the server, Retrofit - an awesome type-safe HTTP client for Android and Java built by awesome folks at Square – library is used to turn HTTP API into a Java interface. Retrofit makes it easy to consume JSON or XML data which is parsed into Plain Old Java Objects (POJOs). The nodeJS server plays a vital role to carry out the payment functionality within the application. The RetroInterface class connects the application and the backend server by describing the route of the post request based on the function that's called inside the application. The server uses Express, a minimal and flexible node.js framework, to handle all requests and response. In order to ensure asynchronous programming, the backend server uses callbacks, promises, and async/await. This will help perform long network requests without blocking the main thread.

The event driven flow and organization of the activity classes is best described with the following flowchart:



Here’s a screenshot of organization of the major self-written files and classes in the code:


File Edit View Navigate Code Analyze Refactor
EZCashApp app src main res
Project
1: Project
Resource Manager
Z: Structure
Build Variants

```
libs
sampledata
src
  androidTest
  main
    assets
    java
      com.example.ezcashapp
        AddedFriends
        AddedFriendsActivity
        ChatActivity
        Chats
        DwollaBank
        DwollaLogin
        EZCashApp
        FirebaseMessagingService
        MainActivity
        MessageAdapter
        Messages
        MessagesActivity
        PlaidActivity
        PlaidToken
        ProfileActivity
        RegisterActivity
        RetroInterface
        SendMoney
        SettingsActivity
        StartActivity
        StatusActivity
        TransactionsActivity
        TransferActivity
        Users
        UsersActivity
    res
      drawable
      drawable-hdpi
      drawable-mdpi
      drawable-v24
      drawable-xhdpi
      drawable-xxhdpi
      drawable-xxxhdpi
      font
```

Section 4: Class Description

Below is a brief description of all the major classes used in the project:

AddedFriends:

This is a Data Model Class for AddedFriendsActivity. It retrieves users' data from the friends Collection stored in the Firebase Database.

AddedFriendsActivity:

The controller class for the Friends page of the application This class populates the friends page screen after the user clicks Friends on the home screen. It displays a list of users who have added the current user as a friend and vice-versa. All friends are retrieved from the firebase collection – friends. Users will have the ability to send a message to their friends and view their profile.

ChatActivity:

The controller for the Chats page of the application. This class populates the chats page screen after the user clicks Chats on the home screen. It displays a list of friends who have communicated with the current user, and users have the ability to communicate with their friends. All messages are stored in firebase database.

Chats:

This is a Data Model Class for ChatActivity. It retrieves users' data from the chat Collection stored in the Firebase Database.

DwollaBank:

This is a Data Model Class for Dwolla's bank verification API request/response. This class helps to send/retrieve data to/from the backend Nodejs server when a POST request is made to verify a user's bank account.

DwollaLogin:

This is a Data Model Class for Dwolla's customer Registration API Request/Response. This class sends/retrieves data to/from the backend Nodejs server when a POST request is made to register a user with Dwolla.

EZCashApp:

This is a Base class for maintaining global application state. It describes specialized tasks that need to run before the first activity.

FirebaseMessagingService:

This is a Messaging Service class for friend requests. This class uses FirebaseMessagingService to send a notification to a user's device after they've received a friend request.

MainActivity:

The controller for the home page of the application. This class populates the home screen after successful login. It redirects the user to other pages of the application. It displays all functionalities offered by the app and redirects users to respective pages if they click on any functionalities.

MessageAdapter:

This is the Adapter class for chat messages. It provides an interface whose implementations provides data and control to the display the messages sent/received by a user.

Messages:

This is a Data Model Class for MessagesActivity. It retrieves users’ data from the messages Collection stored in the Firebase Database.

MessagesActivity:

The controller for the Messages page of the application. This class populates the messages screen after the user clicks a user in chat page. It displays a list of messages exchanged between two users. Users will also have the ability to send a new text or photo message.

PlaidActivity:

The controller for the bank verification page in the application. This class allows user to select their bank and login with their credentials to verify their bank. The screen displayed to the user is made possible by plaid's android sdk integration as described here : https://github.com/plaid/plaid-link-android

PlaidToken:

This is a Data Model Class for Plaid’s bank verification API request/response. This class sends/retrieves data to/from the backend Nodejs server when a POST request is made to generate a link token.

ProfileActivity:

The controller for the Profile page of the application. This class populates the profile page screen after the user clicks "View Profile" on the Added Friends/All Users page. It displays the user's profile image, name, and status. In addition, it also allows sending a friend request, declining a friend request, accepting a friend request, canceling a friend request, and unfriend functionality.

RegisterActivity:

The controller for the Registration page of the application. This class populates the registration page screen after the user clicks "Register" on the login page. It displays text fields for name, email, and password, and registers the user for the application. Upon successful registration, the user will be added to the users collection in firebase database and will be granted login authentication.

RetroInterface:

This class handles executes all POST requests made to the backend server from the application and communicates request/response with the backend nodeJS server.

SendMoney:

The controller for the Dwolla’s User Registration page of the application. This class populates the Register For Payment page screen after the user clicks "Send Money" on the SendMoney page, if User hasn't yet signed up with Dwolla. It allows user to create a registration with Dwolla which is the first step required to sending/receiving money.

SettingsActivity:

The controller for the Account Settings page of the application. This class populates the accounts setting page after the user clicks "Account Settings" from the toolbar menu. It displays the user's profile image, name, and status. In addition, it also allows user to set/change their profile image and status.

StartActivity:

The controller for the Login page of the application. This class populates the login page. The login page is the first page visible to the user after installing the application or starting the application after the user logs out of the application. It displays text fields for email and password and takes the user to MainActivity after successful login.

StatusActivity:

The controller for the status page of the application. This class facilitates user to add/change their status. The page is displayed when user clicks on the change status button in the Account Settings page.

TransactionsActivity:

The controller for the Transactions page of the application. This class populates the Transactions page screen after the user clicks Transactions in the home page. It displays a list of users who are eligible for receiving money (friends with the current user).

TransferActivity:

The controller for the TransferMoney page of the application. This class populates the Transfer Money page screen after the user clicks "Send Money" option from Transactions page. It displays a numeric field for users to enter the amount of money they wish to send to a friend. This page will only be visible if a user has successfully registered for payment with Dwolla and verified their bank with Plaid.

Users:

This is a Data Model Class for UsersActivity. It retrieves users' data from the users Collection stored in the Firebase Database.

UsersActivity:

The controller class for the All Uses page of the application. This class populates the All Users page screen after the user clicks All Users on the menu option. It displays a list of users who are registered with the application. The users will have the ability to send a friend request to all users.

Section 5: Challenges faced and future revisions

Working single-handedly on a project from scratch is not an easy task and a lot of challenges during development was expected. The magnitude of challenges increased with the progress made during development. During my initial days of development, the major challenges were solely based on my lack of experience with Android Studio. Errors such as dependencies version mismatch, faulty naming conventions, NullPointerReference, etc. took significant amount of time to get resolved but as time progressed, these errors were resolved within a few minutes. This was certainly a good learning experience as it helped me understand the working of Android Studio and the code design that's expected. There were some challenges with the Firebase integration during the mid-phase of development, but this didn't last for too long as there were many online resources available and they were well-explained and documented.

The majority of complex challenges were faced to integrate a P2P (peer-to-peer) payment functionality. I was hoping Google's API will easily help me to this. However, I later figured out that this is not yet possible. Then I switched my focus to other services like PayPal, Venmo, BrainTree, Stripe, and such but all of them had one out of the two flaws. The first flaw was they only offered a checkout functionality, which was not very helpful because my goal was not to send money to a centralized place like an e-commerce sites or apps. Instead I was more focused on sending money directly from one bank to another bank. The second flaw was due to my limitation on the goal of this project. Many services required my application to be registered as a business, fill out official taxpayer documents, and pay for the service that I was trying to implement. The payment cost was as high as $12,000/year and this was simply unrealistic.

But Dwolla came to rescue with a great resource for developers – a free sandbox environment. Their goal with this environment was to help developers do their development completely free without having to register for a production environment. This was exactly what I was looking for and they suggested me to take the Dwolla+Plaid Integration route to facilitate a peer-to-peer money transfer.

Integrating the payment service was a whole new project in itself. The first challenge was to learn how to make HTTP requests in android studio and how to do a client-side development. I only had this experience with web-development, and I decided to somehow use that to the best of my ability. I started off by writing the server-side code as it was not the major challenge. Then the communication between backend server and my application was challenging. I decided to use WebView in android studio but Plaid Android sdk which used WebView integration was deprecated. So, my only option was to use the LinkConfiguration method as described in Plaid's documentation. After a whole week of reading the documentation, I was finally able to connect all the dots and implement the functionality in the application.

Moving forward my goal will be to strengthen the security features and push the project to a development environment. The application will then be available on Google Play Store for users to download.

Section 6: Summary and Conclusion

The complexity of the project turned out to be significantly higher than initially planned. Each component of the project involved a lot of trial, error and research.

The most complex part of the project was figuring out the right architecture for this scale. I jumped from one implementation to another to accommodate growing complexity of the problems and features. After weeks of going through tutorials, articles, company forums, I landed on a well-crafted architecture which worked really well for this project. From this project, I also learned by experience that not all problems have one right solutions. There can be numerous ways to solve a problem. The first step is to solve it the most primitive way we can and start optimizing from there.

The software is at a stage I had initially intended. However, I hadn't anticipated the scale of the project in the beginning. As of now, the software has roughly 8000 lines of code (6000 lines of java code and 2000 lines of xml code). I'm glad I was able to do so much of it. This was definitely a huge learning experience. During my research and desperate search for solutions, I came to understand many industry terms, practices and common software architecture patterns that I would not have found on my own. I learned about reactive programming, observables, pagination, BLoC architecture, and ViewModels just by reading through other companies' forums, news articles and reddit posts. These kinds of solutions don't come in huge colorful books; You have to observe what other people are doing and think how their solutions applies to your problem at hand. Of course, not all solutions fit the problem, but they give you important ideas on how to come up with your own solutions.

There are some improvements I would like to make to the app after the submission of this project. I plan on migrating the whole code base to a different platform which can make the application run on iOS devices. Java is a great programming language, but java applications don't run on iOS devices. I want to launch this application for general use at some point in the future. For that, I want to make it available to both iPhone and android users. Out of all other solutions React looks like a good platform to learn. The overall architecture of the application will still stay the same even in its new code base. After the release, I feel confident that users will find this app useful for improving productivity and collaborating with their friends and families.