

Technical Specification

B.Sc in Computer Applications

Third Year Project

James Nolan & Kealan Thompson



Communal Costs developed for



ANDROID

0. Table of Contents

0. Table of Contents	2
1. Introduction	3
1.1 Overview	3
1.2 Glossary	3, 4
2. System Architecture	4,5,6,7
2.1 The end user	4
2.2 The Android user interface	5
2.3 Firebase	5, 6, 7
3. High - Level - Design	7, 8, 9, 10, 11, 12, 13, 14
Context Diagram	7
Data Flow Diagram	8, 9
Logical Data Flow	10
Class Diagram	10, 11
Firebase Functions	12
Firebase Authentication	12
Firebase Cloud Messaging	13
Firebase Test Lab	13
Firebase Real Time Database	14
4. Problems & resolutions	14, 15, 16
Location & scheduling	14
Merge conflicts in Android Studio	14
Gitlab asking for password continuously	14, 15
Learning curve with Android	15
Implementing list views	15
Implementing settings	15
Changing activities to services	15
Firebase beta functionality	15, 16
5. Installation Guide	16

1. Introduction

1.1 Overview

Communal Costs is a cost sharing application, that offers users the ability to set up an account to monitor the expenses of a household. Allowing users to keep track of communal spending and the contributions that each member owes. It has been developed for devices operating on android, from [API 19](#) upwards. Which takes advantage of android's notification system and real time data syncing to make sure all [collective](#) members have the same financial data at all times.

This is a real world problem that everyone encounters at one time or another. Communal Costs takes the confusion and possible conflict out of financial management. Making keeping track of household funds as easy as tracking expenses.

1.2 Glossary

API	Application programming interface – a tool used to develop software
collective	A group with a common purpose
admin	A user with all privileges
contributor	A user with all privileges but the ability to edit the collective structure
Ordinary member	A user that can only view the collective
UI	User interface – the graphical representation of the programme
JAVA	A programming language
backend	Part of a piece of software that handles data processing and manipulation
App Engine	Google's app engine platform, a tool to integrate google's app services and cloud support services
transaction	A financial transaction
Node.js	A programming language

npm	A command line tool, for access to the firebase backend
apk	Application installation file
ListView	A way of displaying objects in a dynamically scrolling list on an android device

2. System Architecture

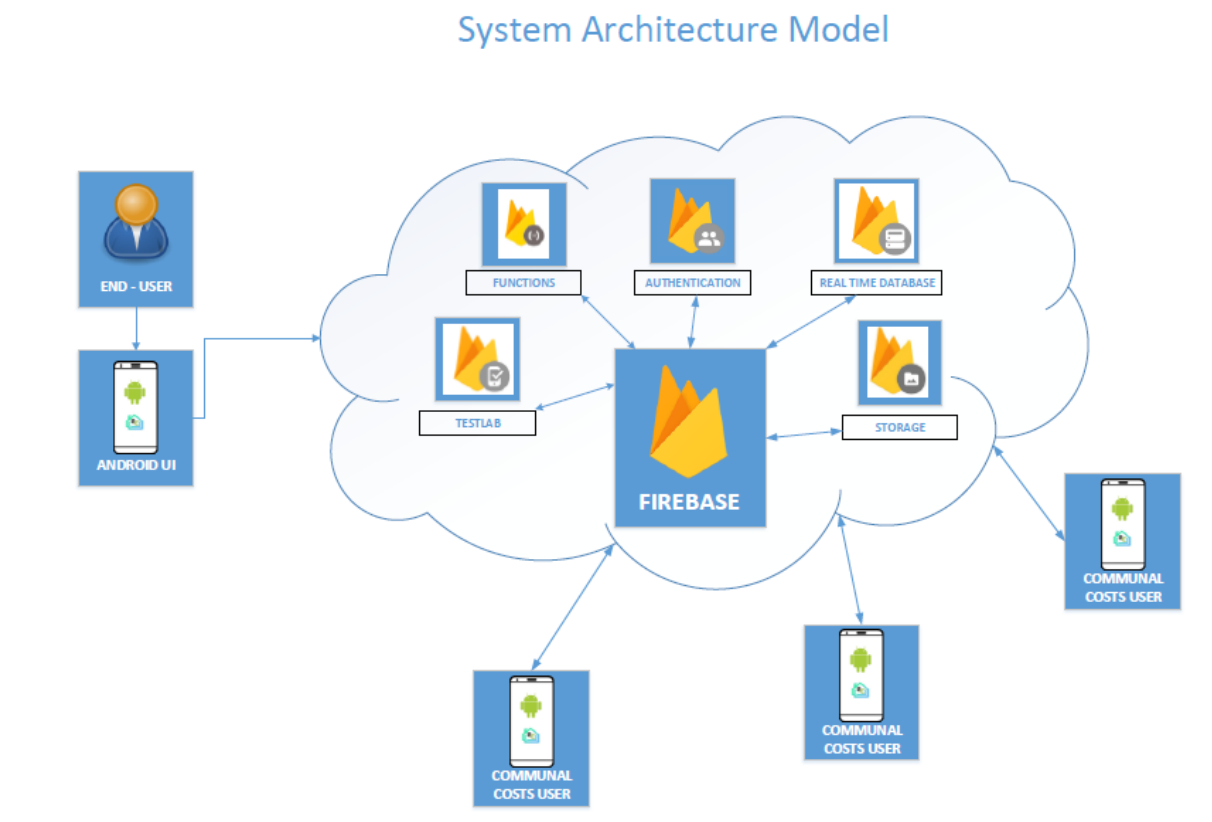


Fig2.1 System Architecture Model

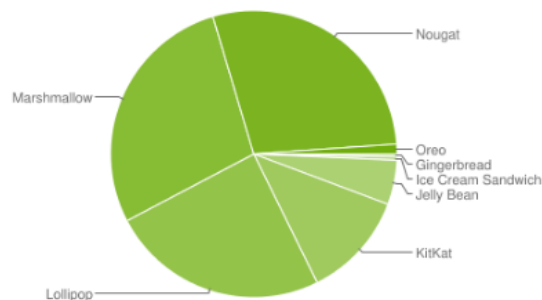
2.1: The End User

The end user who may be an [admin](#), [contributor](#) or [ordinary](#) member of a collective, inputs data into the user-interface. They receive notifications anytime a new transaction is added to a collective they are member of.

2.2 The Android User Interface

The user interface is easy to navigate and aesthetically pleasing. This is what the end-user interacts with to input various expenditure and gain information about their collective. The UI is programmed in [JAVA](#), with the minimum [API](#) level set at 19. We made this choice as this seems to be the best chunk of the market share presently.

Version	Codename	API	Distribution
2.3.3 - 2.3.7	Gingerbread	10	0.3%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	0.4%
4.1.x	Jelly Bean	16	1.7%
4.2.x		17	2.6%
4.3		18	0.7%
4.4	KitKat	19	12.0%
5.0	Lollipop	21	5.4%
5.1		22	19.2%
6.0	Marshmallow	23	28.1%
7.0	Nougat	24	22.3%
7.1		25	6.2%
8.0	Oreo	26	0.8%
8.1		27	0.3%



Data collected during a 7-day period ending on February 5, 2018.

Any versions with less than 0.1% distribution are not shown.

Src: <https://developer.android.com/about/dashboards/index.html>

2.3 Firebase

Firebase is a [backend](#) system provided by google, which is particularly suited to mobile applications. One of the really handy things about using Firebase is that it works offline, if users make changes on the app whilst they are offline, once they regain a connection, their changes are immediately pushed to Firebase.

Aspects of Firebase are built on Google Cloud (i.e. the App Engine), which we originally wanted to use in our project, however we decided against using the [App Engine](#) as Firebase provided a sufficient amount of features. Firebase is a core part of the Communal Costs application, and below we outline the various features of Firebase that are used in this project.

2.3.1 Real Time Database

The Communal Costs application relies heavily on the Real Time Database. The app stores user information, [transaction](#) information, [collective](#) information and test information on the real time database. Updates to the database are extremely efficient. Retrieving information from Firebase is heavily reliant on the use of objects locally and if these objects don't have the correct getters and setters, it can be tricky to debug.

2.3.2 Authentication

Another core feature is the Firebase Auth. This provides a framework which allowed us to determine which user was logged in and some of their account details. We used the Firebase Authentication userID as the primary key of the user in the real time database. This made it handy for us to access user information.

2.3.3 Functions

Our backend functions are written with [Node.js](#) using the firebase shell with [npm](#) on the command line. Without the backend functions, our application would not work. They rewrite the database when a user is added or when a user joins a collective. They also keep count of the amount of users on our app and handle notifications.

```
=== Deploying to 'communal-5d872'...  
  
i  deploying functions  
Running command: npm --prefix functions run lint  
  
> functions@1.0.0 lint C:\Users\C5228122\Documents\CASE3\DCU\3YP\Project Code\code\firFun\functions  
> eslint .
```

Fig 2.2 npm command line

```
+ functions: Finished running predeploy script.  
i functions: ensuring necessary APIs are enabled...  
+ functions: all necessary APIs are enabled  
i functions: preparing functions directory for uploading...  
i functions: packaged functions (40.94 KB) for uploading  
+ functions: functions folder uploaded successfully  
i functions: updating function colNotifications...  
i functions: updating function addCollectiveIDtoMemberAccountsUpgrade...  
i functions: updating function rmCollectiveIDtoMemberAccountsUpgrade...  
i functions: updating function countUsers...
```

Fig 2.3 functions being deployed to cloud

2.3.4 Test Lab

The firebase test lab is a really cool feature that allows you to test your [APK](#) across multiple devices with varying API levels. This is a great indication for how well your app performs across multiple devices.

2.3.5 Storage

The Firebase Storage feature is similar to the real time database. However, we did not really get to harness this feature to a great extent. The firebase cloud storage allows you to store videos and images and reference them in the real time database.

3. High-Level Design

The Context Diagram

The context diagram below (Fig 3.1) shows a brief top level view of how the end user interacts with the system. The end user can modify, view and create collectives & transactions, the application returns an overview of the collectives/transactions relevant to the user and also supplies notifications. The system updates the collective and user trees of the database when the user interacts with them. When new information is created in one section of our database, the necessary replications are made to the other side of the database to ensure data is uniform.

CONTEXT DIAGRAM

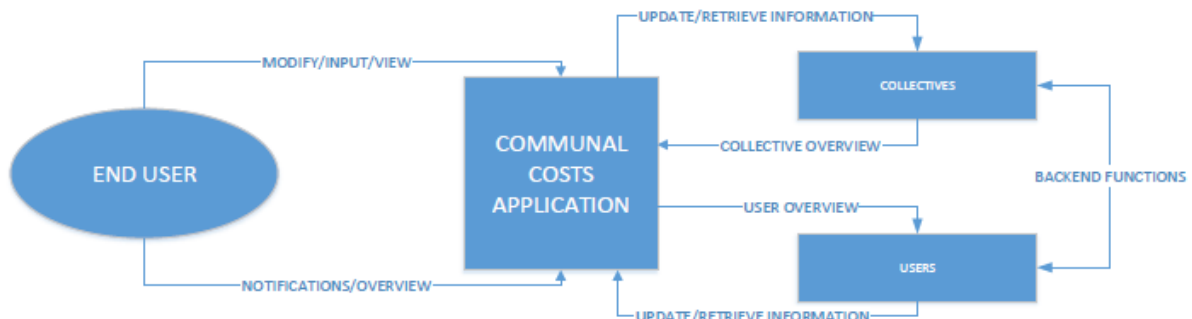


Fig 3.1 The Context Diagram

The data flow diagram (Fig 3.2) shows how entities interact with process which are interacting with digital stores in the Communal Costs Application.

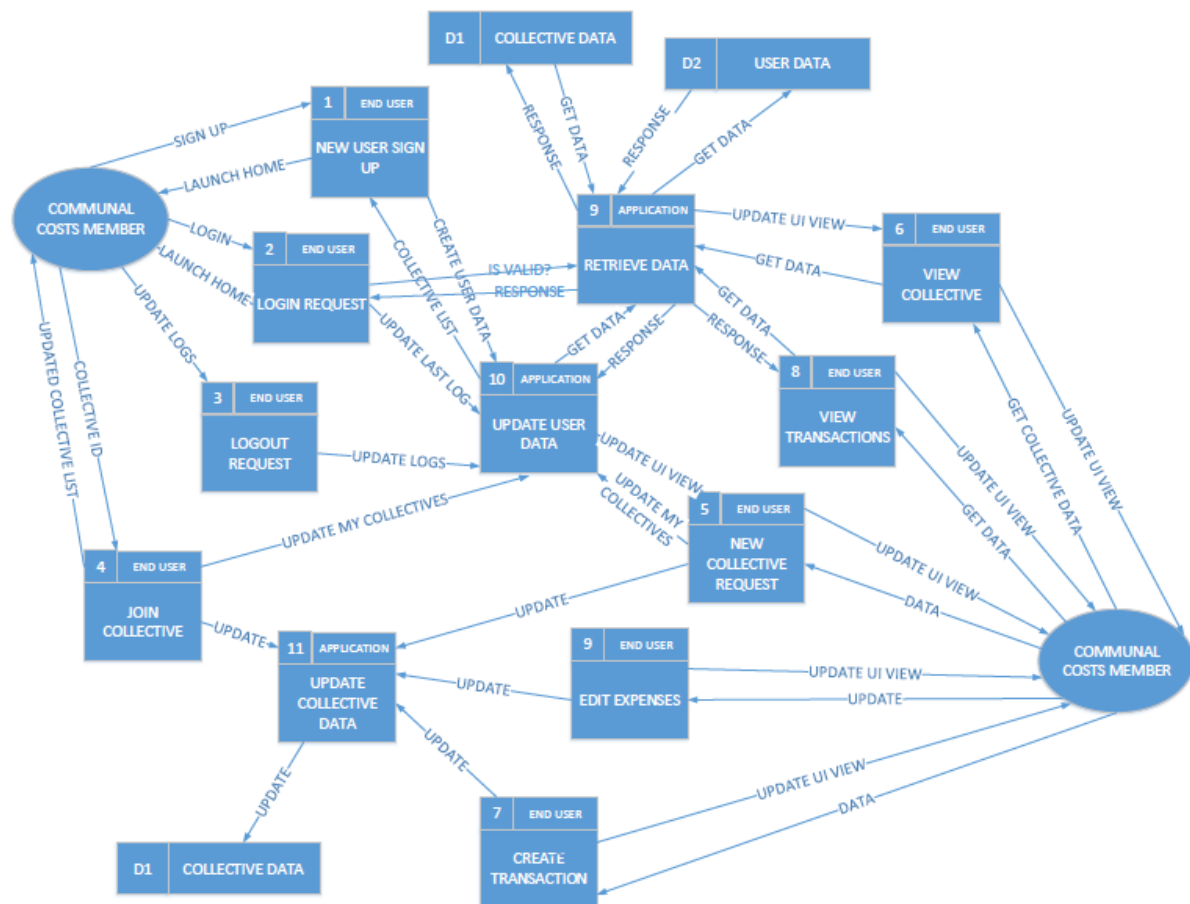


Fig 3.2 Data Flow Diagram

<u>DFD Key</u>		
<i>Type</i>	<i>Name</i>	<i>Description</i>
Process(1)	New User Sign Up	User entity has signed up to the system. Update user data
Process(2)	Login Request	User entity has logged into the system. Update user data
Process(3)	Logout Request	User entity has logged out of

		the application. Update user data.
Process(4)	Join Collective	User entity has joined collective x. Update user and collective data.
Process(7)	Create Transaction	User entity has created transaction x in collective y. Update collective data.
Process(9)	Edit Expenses	User entity has edited transaction x in collective y. Update collective data.
Process(5)	New Collective Request	User entity has created a new collective. Update collective and user data,
Process(8)	View Transactions	User entity has opened a Transaction x in collective y.
Process(6)	View Collective	User entity has opened collective x from their list of collectives
Process(9)	Retrieve Data	A process requires data to complete.
Process(10)	Update User Data	A process is requesting to update the user data.
Process(11)	Update Collective Data	A process is requesting to update the collective data.
Entity	Communal Costs User	The end user who is interacting with our system
Data Store	Collective Data	All collective information, transaction, members, creator, and collective details are stored here.
Data Store	User Data	All user data is stored here, their list of collectives, their email, display name and further details.

Logical Data Flow

The below logical data flow outlines the basic processes that occur in the system.

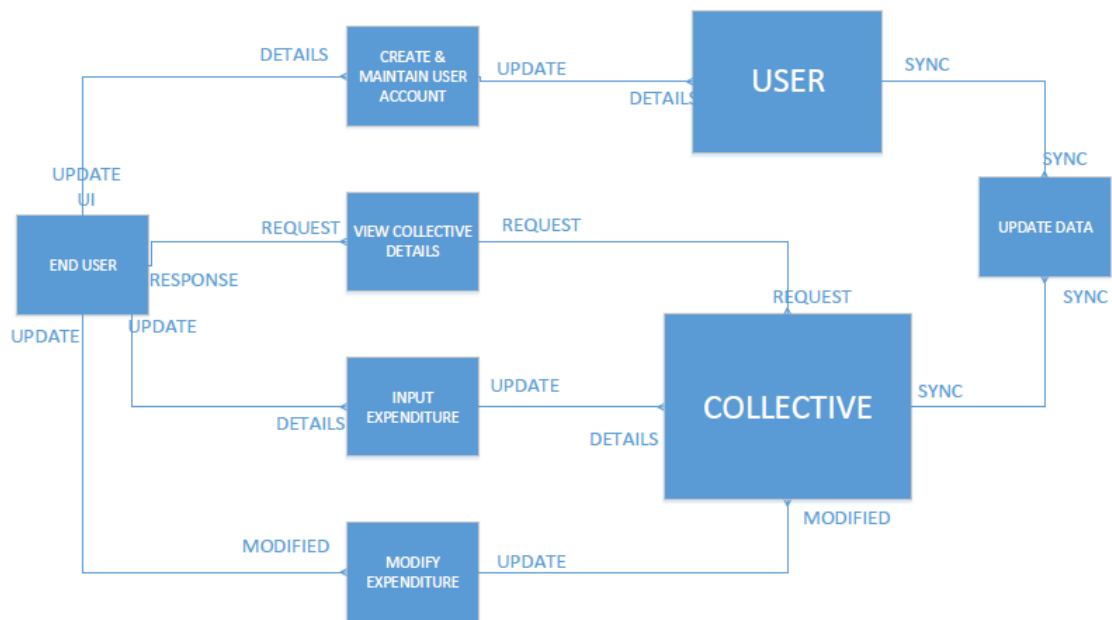


Fig 3.3 The Logical Data Flow

Class Diagram

Below is our class diagram. The class diagram shows all our classes (activities + objects + services) and the methods and attributes they contain. The application contains 14 classes in total.

Firestore Functions

This is the console view of the Firestore functions. There is a function to add collective ID's to the user tree when an email address is added to the member section of any collective. The collective notifications works in conjunction with firestore cloud messaging to provide notifications when new transactions are added to a collective. The last method removes collectives from the user "myCollectives" arraylist when a collective is deleted.









Function	Event	Executions	Median run time
addCollectiveIDtoM...	 ref.create /collectives/{colName}/members/{i}	45 	567.32 ms
colNotifications	 ref.create /collectives/{colName}/transactions/{i}	80 	558.94 ms
countUsers	 ref.write /users	96 	550.80 ms
rmCollectiveIDtoMe...	 ref.delete /collectives/{colName}/members/{i}	48 	550.80 ms

Fig 3.5 Firestore Functions Console

Firestore Authentication

Below shows a list of all our current users. There are presently 20 users. In this window, we can delete, disable and reset user passwords.


















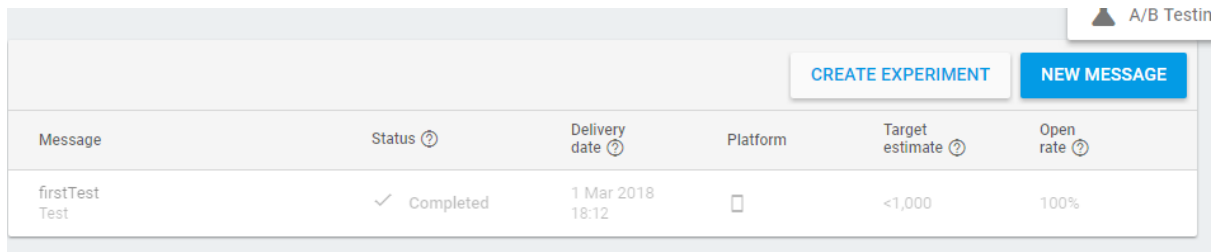
 Search by email address, phone number or user UID ADD USER  				
Identifier	Providers	Created	Signed In	User UID 
jamesyboyoo@yahoo.ie		1 Mar 2018	1 Mar 2018	0FVYGWRyEoRNDajAM7QbhdUV...
jamezyboyoo@yahoo.ie		9 Mar 2018	9 Mar 2018	GH1xcdbkaJh1kQRaxBe999Bm4f...
mail@valid.com		8 Mar 2018	8 Mar 2018	IHUeliqSxfYxH5YjqNRNbt90Or22
kealanthompson@hotmail.c...		3 Mar 2018	3 Mar 2018	Ol8QVpk1V6T1ETM5TYf6PK61rxz2
test@hotmail.com		9 Mar 2018	9 Mar 2018	PWMW4r41vagfV5aBNjx8jqvtVd2
hcgjtycr@jcdh.com		26 Feb 2018	26 Feb 2018	SgN6PqlXFMSDoMmYxsODt3KWz...
james.nolan38@mail.dcu.ie		21 Feb 2018	9 Mar 2018	TkFRWX97lgOQqb3GZSYA3MKGjY...
shffk@sgh.com		8 Mar 2018	8 Mar 2018	VB7vXOSty3gnAGYOBsAGV5dDKY...
shriek@mail.com		9 Mar 2018	9 Mar 2018	Vh0QUhNNVIESGi4ClIx9oFtkpg1y1
lufthansa@airways.com		2 Mar 2018	2 Mar 2018	dA1aU42oiBT0fufuKC6iVSR18U2
fir@fir.com		8 Mar 2018	8 Mar 2018	jlmjsVKBxCVML0BSAMtLUOXuY2g2
test@mail.com		8 Mar 2018	8 Mar 2018	mMBEbL8MIGc4AKH7WZCmobrv...
test@mail.ie		9 Mar 2018	9 Mar 2018	oG8arhilM5bHZyuq7zZ52HdPTIq2

Fig 3.6 Firebase Authentication Console

Firebase Cloud Messaging

By accessing the specific token generated by a device, we can message individual users.



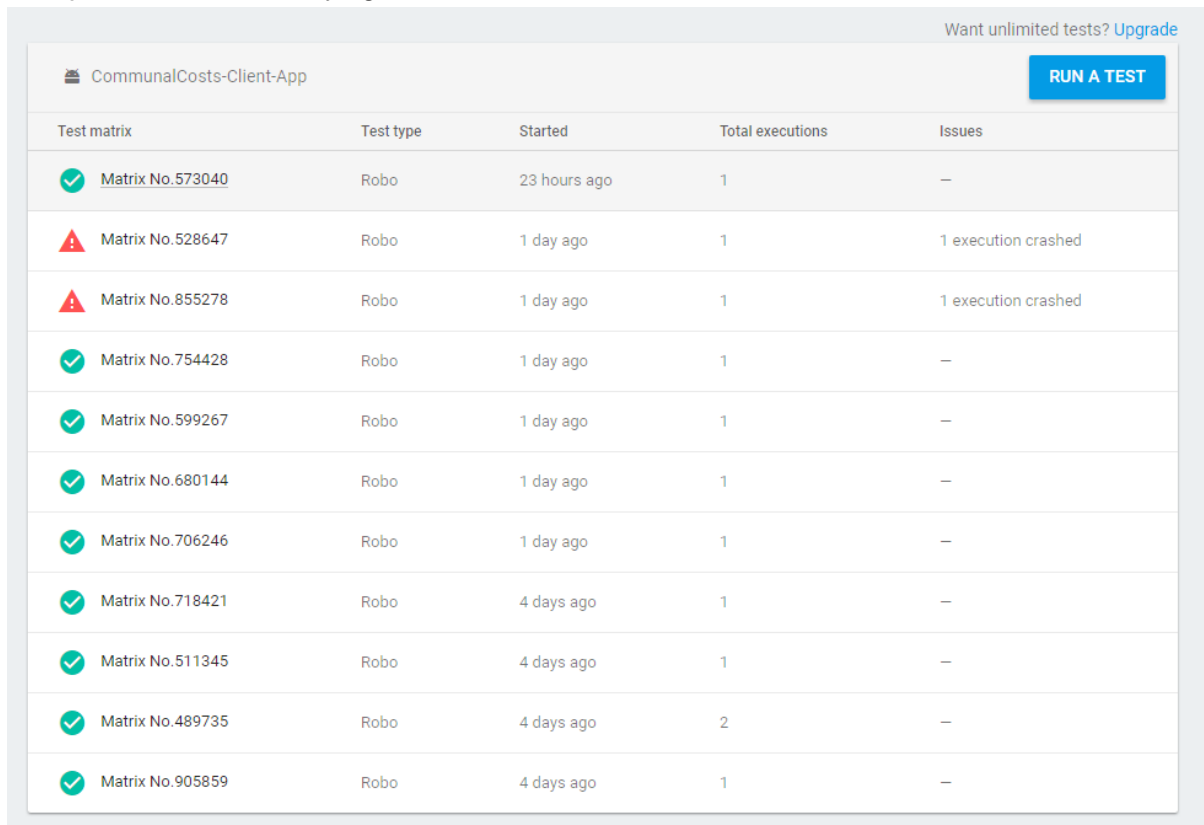
The screenshot shows the Firebase Cloud Messaging console interface. At the top right, there is a user profile icon and the text 'A/B Testin'. Below this, there are two buttons: 'CREATE EXPERIMENT' and 'NEW MESSAGE'. The main part of the console is a table with the following columns: Message, Status, Delivery date, Platform, Target estimate, and Open rate. A single row is visible in the table with the following data: Message: firstTest Test, Status: ✓ Completed, Delivery date: 1 Mar 2018 18:12, Platform: (empty), Target estimate: <1,000, Open rate: 100%.

Message	Status ?	Delivery date ?	Platform	Target estimate ?	Open rate ?
firstTest Test	✓ Completed	1 Mar 2018 18:12		<1,000	100%

Fig 3.7 Firebase Cloud Messaging Console

Firebase Test Lab

A really cool feature of Firebase, the test lab allows you to test your application across multiple devices with varying API levels. You can run a robo test or an instrumentation test.



The screenshot shows the Firebase Test Lab console interface. At the top right, there is a link 'Want unlimited tests? Upgrade'. Below this, there is a header for the application 'CommunalCosts-Client-App' and a 'RUN A TEST' button. The main part of the console is a table with the following columns: Test matrix, Test type, Started, Total executions, and Issues. The table contains 11 rows of test results, each with a status icon (green checkmark or red triangle), a matrix number, a test type (Robo), a start time, a total number of executions, and a list of issues.

Test matrix	Test type	Started	Total executions	Issues
✓ Matrix No.573040	Robo	23 hours ago	1	—
⚠ Matrix No.528647	Robo	1 day ago	1	1 execution crashed
⚠ Matrix No.855278	Robo	1 day ago	1	1 execution crashed
✓ Matrix No.754428	Robo	1 day ago	1	—
✓ Matrix No.599267	Robo	1 day ago	1	—
✓ Matrix No.680144	Robo	1 day ago	1	—
✓ Matrix No.706246	Robo	1 day ago	1	—
✓ Matrix No.718421	Robo	4 days ago	1	—
✓ Matrix No.511345	Robo	4 days ago	1	—
✓ Matrix No.489735	Robo	4 days ago	2	—
✓ Matrix No.905859	Robo	4 days ago	1	—

Fig 3.8 Firebase Test Lab Console

Firestore Real-Time Database

Below are screenshots of our real-time database. It contains our user information, collective information, the amount of users and a tree for unit testing.

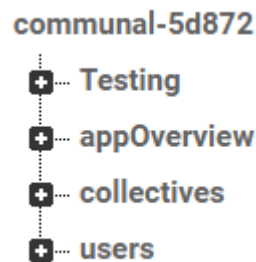


Fig 3.9 The real time database but collapsed

4. Problems and Resolutions

Location and scheduling

As one half of the team for this project was in Dublin and the other half was in Linz, Austria until February. There was a heavy reliance on remote work systems and instant messaging services to ensure that the project stayed on track. Although due to the difference in locations, there were also differences in our schedules, being in different time zones meant that we had to accommodate the hour difference into any meeting plans or collaborative work. While also keeping each other informed of our differing schedules, with both of us studying in different universities. While this was a challenge it meant that for the most part there was always someone working on the project and we had little conflict due to both of us working on the same thing at the same time.

Merge Conflicts in Android Studio

At the start, this was a real pain as most of the work was taking place in the same activities. However, we got the hang of resolving merge conflicts after an hour or two researching Stack Overflow and google documentation.

Gitlab asking for password continuously

This was a frequent issue, even when both of us were in different countries. Gitlab would sometimes ask for the password to “gitlab.computing.dcu.ie” and would not allow you to push or pull from the repo. Removing SSH keys and resetting git global parameters did not

resolve the issue. This was a nuisance as it delayed collaboration in the later stages of our project.

Learning curve with Android Development

With both of us having some experience coding on projects of the same scale we underestimated the learning curve associated with becoming proficient android developers. Each day we had to accommodate a significant portion of our development time to learning the necessary background work to develop efficient systems within our application.

Implementing list views

This was a challenging iteration of development, changing from displaying a basic home screen to displaying all information from the database in a [ListView](#) with the ability to click on and edit entries.

Implementing Settings

This was quite different to any of the other activities we implemented, relying on different base classes and a fragment structure. This proved difficult to conceptualise at first.

Changing activities to services

Due to the size of the application and all the processes in the main thread it was necessary to move some elements to services instead

Firebase beta functionality

Some functionality for firebase, especially with regard to firebase functions and the test lab, is still being developed by Google. Developing the firebase functions took quite some time as neither of us had experience with Node.js and one bug we encountered in the development was resolved by google 2 days before it occurred for us!

For instance, on March 9th, [npm](#) told us that an update was available (Fig 4.2) when we were updating the cloud functions. We were encountering an issue as the functions wouldn't deploy on Firebase (time out error when pushing) and due to the timeout the function got stuck in deployment mode on the backend. The solution according to stack overflow was to navigate to the google cloud page and check if there was a warning sign on any method (shown in fig 4.1). Clearing the functions from the google cloud functions platform did not work as it did not replicate across to the Firebase functions platform. We managed to wipe the firebase functions section and this resolved the issue. Heart attack averted!

Filter functions		Columns				
<input type="checkbox"/> Name ^	Region	Trigger	Memory allocated	Executed function	Last deployed	
<input type="checkbox"/> ⚠ addCollectiveDtoMemberAccountsUpgrade	us-central1	Firebase Database: create	256 MB	addCollectiveDtoMemberAccountsUpgrade	09/03/2018, 13:46	⋮
<input type="checkbox"/> ⚠ colNotifications	us-central1	Firebase Database: create	256 MB	colNotifications	09/03/2018, 13:46	⋮
<input type="checkbox"/> ⚠ countUsers	us-central1	Firebase Database: write	256 MB	countUsers	09/03/2018, 13:46	⋮
<input type="checkbox"/> ⚠ rmCollectiveDtoMemberAccountsUpgrade	us-central1	Firebase Database: delete	256 MB	rmCollectiveDtoMemberAccountsUpgrade	09/03/2018, 13:46	⋮

Fig4.1 Firebase Function Error

```

Select npm
[.....] \ extract:npm: verb lock using C:\Users\C5228122\AppData\Roaming\npm-cache\_locks\staging-d8cc7fdd3
16f1f70.lock for C:\Users\C5228122\Documents\CASE3\DCU\3YP\Project Code\code[.....] \ extract:npm: verb lock
k using C:\Users\C5228122\AppData\Roaming\npm-cache\_locks\staging-d8cc7fdd316f1f70.lock for C:\Users\C5228122\Documents
[.....] \ extract:npm: verb lock using C:\Users\C5228122\AppData\Roaming\npm-cache\_locks\staging-d8cc7fdd

```

Fig4.2 Update for npm released

5. Installation Guide

We have uploaded our APK to our project Google Drive. Our app is free to use with no adverts. Follow the below steps to install. Please note, we recommend you install our app through your mobile device. Compatible with Android API 19+ only.

1. Open the following url in the browser on your mobile (Click [here](#))
2. Tap on “Download”.
3. In the system tray, select the file you have downloaded, or access the file through the “downloads” section of your browser.
4. If you have not already, please navigate to your settings window and turn on “allow installation of apps from unknown sources”
5. You should now be able to successfully install the Communal Costs Client Application!
6. Sign up with your email and password today!