B.Sc. in Computer Applications
Third Year Project 2017/2018

Functional Specification

Communal Costs

Kealan Thompson (12417338) and James Nolan (14316461)

Check out our online project blog: https://jameskealan3yproje.wixsite.com/jameskealan3yp
Email: jameskealan3yproject@gmail.com

0. Table of contents

Note, any words marked in blue are defined in the glossary.

1. Introduction

1.1 Overview

We aim to develop an **easy to use**, **efficient** application which makes life easier for those who manage communal costs. Our app will **keep track** of the **incomings and outgoings** for a **household**, a **club/society** or a **campaign/fundraiser**, while also providing **financial advice** on the data submitted. We will focus on **transforming users data** into more accessible forms, providing users with a **safe and secure way** of storing financial data, and offering very **convenient** ways of entering user **input**.

End-Users can **create an account** of their own, and subsequently **create** or **join** a "**collective**" of a certain **type**. Each varies according to key differences in both **structure** and user **privileges**. Collectives can be split into three **types** - a **household**, a **club/society** or a **campaign/fundraiser**, along with the possibility of **creating** your own collective, if the provided collectives do not meet your needs. These types vary according to the **levels** of **access** given to members, to access and alter the group's finances. Upon creation of a "collective" a **unique ID** is created to allow others to join.

Each collective has **three** different "member types", **administrators**, **contributors** and **ordinary members**. Administrators have **control** over the entire collective, contributors can only make entries and ordinary members can only view the expenses. All members can query any entry.

Our app will provide an **effective** means by which people can **collaborate** on shared income and expenditure.

What separates our app from what's already on the market is, that while there are other financial applications out there that offer the ability to send or receive payments, or that deal with a **very specific** purpose such as splitting bills there are very few exhaustive applications that offer a **wide range** of financial services. As a result of amalgamating a number of features and offering some new ones e.g. **pdf scraping** as an input method. application.

1.2 Business Context

The possible business applications arising from this application are **advertising** and **marketing** the app on the android app store. We could include in app advertising possibly as a discrete **banner** at the bottom of the app or a **popup ad** underneath some of the product's data **visualisation**, with the aim of not being too **obstructive** so as to discourage use. We could then possibly introduce a paid or subscription **service** to remove adds and introduce **greater functionality** if there is a demand.

We could also sell the app as a **paid** application on the android app store. Generating revenue as a once off payment, or if possible as a subscription based application.

1.3 Glossary

Communal	A tool for user in a community
Club/Society	A group which play a certain sport/a group with a similar interest
Collective	A general name for a group that is carrying out an activity together
Household	Specifically referencing shared accommodation
Campaign/Fundraiser	A short term collaborative project which needs funding
Administrator	A user with the highest privilege level. That can carry out any activity within the collective. Apart from those that require multiple administrators to prevent misuse.
Contributor	A user with secondary privileges. That can view and add entries.
Ordinary Member	A user that has the lowest privilege level and can only view and query entries
SQL Injection	A common database security exploit, allowing unauthorised access to the database
Sandbox - in terms of android security	a security mechanism for separating current running programs
IaaS	Infrastructure as a service - relevant to cloud computing
API	Application Programming nterface - a software building tool ie a library or data access tool

2. General Description

2.1 Product Functions

The aim of this product is to provide an app where users can:

Login - to their own personal account. Which can be a member of, to varying degrees of access, to a number of collectives.

Join a collective - existing collectives can be joined, pending the approval of one or more of the collective's administrators.

Create a collective - the user can also create their own collective if they desire. This can be tailored to their financial situation and goals.

A variety of income and expenditure data entry methods - we will include many different data entry methods, to ensure that it is convenient for users to enter data and keep their financial records up to date. We will allow users to submit a **bank statement** pdf, from which the application will then record the available financial data from it. This file **will not be stored** on the database, it will only be used to extrapolate information from. A **receipt scanner** is another input method, allowing users to specify how much was spent towards the collective and the scanner will then gather the rest of the financial data from the receipt, ignoring irrelevant information. While the image itself also acts as **proof** of financial expenditure as well as a **convenient** method for entering user's financial data.

Have confidence in the security and encryption of their data - we want users to be confident, that at every layer of the design process, the safety of their data has been given the utmost consideration and only the most secure design methods have been chosen. For example, we chose to develop with the google app engine as it handled some of the more abstract security concerns, allowing us to focus on more important features.

Alert the user, to modifications to the collective and expected modifications if absent - to implement an app alert system, that operates through a combination of push notifications and an in app notification centre. While alerting users to changes it will also remind the user of expected changes, eg expenses that haven't been logged.

Offer advice on taxation and other financial advice, relevant to the user's current financial position - we aim to find reliable resources to scrape current financial information, e.g. we are currently in contact with revenue to see if they have an api that we could use and alert users to events that affect their financial situation. This should also cover advice on tax brackets, expected tax due in comparison to paid so far, at the year's end and advice of any purchases for which tax can be claimed back or tax credits that are applicable.

Set financial goals - we will allow users and collectives to work towards financial goals, both concrete and abstract. For example, a concrete goal: to save x euro or an abstract goal: to save more. The abstract version would then report the percentage change in savings during the designated period. Rather than relevant to a goal position the focus is relative to the starting position.

Support forums and email support - we wish to ensure users that any and all queries will be handled as quickly and conveniently as possible.

2.2 User Characteristics and Objectives

The app **aims to target the student population** specifically, as we see this as a population with a need to budget. Societies, shared accommodation, and so on all provide ample opportunities for use of this app. We don't foresee any particular technical challenges with users in this area as students tend to be a fairly technically proficient demographic. The main concern would be that the **U.I. is user friendly** enough so as to be **intuitive** and not cause confusion. As a result, our focus is to have a clear and simple U.I. with only a few options at each stage and a further create your own option for users that would like to delve further into the options that the app can provide.

2.3 Operational Scenarios

Creating an account

Upon opening the application, you will be **prompted** to either create an account or login with an existing account. We hope to enable users to login with existing accounts, e.g. facebook, **google play account**, an email account, or to manually enter their data and create an account. We should only require a name, email address and a collective to join or create. Once an account has been set up, the user will receive a confirmation email with a link to verify their email address which once completed will full establish their user account.

Creating a collective
Upon a successful login, the user will be able to **view, join, create** or **edit** collectives. Since this is a new user we shall run through the process to create a collective. Tap on the create collective option and choose the collective type that most suits your needs or tap on the more/create your own collective type option. To create a collective that better suits your needs.

Inputting financial data
After purchasing goods for the group, a picture of the receipt can be taken as both **proof** of purchase and a **source** of financial **information**. With the option to specify which of the goods purchased were specifically for the **group** and which were **personal** items. Another way of entering a substantial amount of financial data at once is to allow the app to access a **pdf** of your **bank statement** on your phone and it will scrape the applicable financial information from the document. Then offering the user the opportunity to **accept** or **reject** to add individual transactions from the bank statement.

Viewing financial information
Users will be able to view both, **summary** and **individual** transaction data. Once an account reaches a point where enough information is available summary **data analysis** will be carried out. Providing users with a quick view of the whole group's finances. However up until that point we will still provide basic summary features that do not rely on a wealth of information e.g. transactions since a certain date, or on a certain date, average cost of expenses so far. Users will also be able to view individual transaction entries and query them.

2.4 Constraints

We wish to develop a sleek, seamless application. One which users do not notice any loading times or have to wait to update details. As such, we would hope to possibly **duplicate** critical data locally on user's devices so as to avoid a heavy reliance on an active data connection. Possibly also having the app update as much information as possible while at the introductory screen so that necessary information is loaded without explicitly stating it. Therefore alleviating wait times at points which they would generally be encountered e.g. when later viewing collective information.

Since the app deals with sensitive financial information, security will be of paramount concern at each and every point of the design. To ensure that at no level users are vulnerable to their financial data being accessed by those without the given access. So far we have factored that into the our choice of database and framework to design our database.

3. Functional Requirements

3.1 User login

Description
The system must be able to **store** and **recognise** all username, password combinations and external site logins i.e. through facebook. While being **secure** and not open to common security **exploits** such as a **sql injection**.

Criticality
This is **essential** to the functionality of the app, without it we would not be able to store user information remotely. Without a database login system and a remote database of financial information, we would have to store all data **locally** on the device. Which could then lead to **data loses** if a key user removes the app or there is some hardware error.

Technical issues
The use of external login sources will add extra **complexity**, especially if there are quite a few to implement. However from our research, it seems clear that most of this work will be handled by **API's** from the various login sources. Creating a **database** of **user emails** and passwords for those that choose that account setup method will be simple.

Dependencies with other requirements
All the users data and access to this data will be tied to their user account so this is of paramount importance.

3.2 Create Collective

Description
The system has a **number of collective types**, with the option for users to **create** their own collective if desired. Each collective varies in it's size, intended **purpose** and **privilege** structure among members.

Criticality
This is a **critical** function of the app. Without it, it's no longer a communal costs app. It is instead a financial advice app for individuals. Which limits its functionality and the scale on which it operates.

Technical Issues
This should be relatively straightforward, entering the constraints that the collective is built on such as user **privileges** and member **limits**, and adding a list of users that are members of the collective along with the collectives financial goals and services offered. While ensuring that all financial data inputted is linked to a certain collective.

Dependencies with other requirements
The user **database** is essential for forming collectives along with the set of privileges.

3.3 Enter Financial Data

Description
This is another pillar of our application without convenient ways to enter data it will deter users from entering financial information. If we left it to manual data entry, it would be unlikely that users would enter enough data for this app to offer detailed financial advice. Instead we focus on alternative methods such as **analysing** bank statement pdfs and **image processing** i.e. receipts. This should make the application a **convenient** alternative to manually keeping track of financial information.

Criticality
This is another mission **critical** function. Without it, it will be a major barrier to user uptake and as such the app won't be used.

Technical Issues
This will be one of if not the most technically demanding aspects of the project. Image processing in particular will be quite a **complex** task.

Dependencies with other requirements
The whole system depends on financial information to give advice on and transform. While measuring progress towards goals against.

3.4 Transform Financial Data

Description
It will be necessary to **transform** financial data to make it more digestible. Some of which include: introducing easy to read diagrams and charts measuring progress towards current financial goals, comparing income and expenditure to **expectations**.

Criticality
This is another feature **critical** to users, otherwise the app just offers the ability to make note of financial information.

Technical Issues
This should be straightforward enough, fetching the data from the database, comparing it to goal states and transforming it, in order to become more **user friendly.**

Dependencies with other requirements
All of the above requirements must be implemented, this is more of a finishing touch for when we have a working system in place as it draws from and works on all the data gathered up until that point. However it is still a critical feature as otherwise the app just offers the ability to make note of financial information.

3.5 Secure Communication

Description
Due to the sensitive nature of the data processed by the application secure encrypted communication and data storage is of the utmost importance.

Criticality
This feature is highly **critical**, as who would use a financial application that they could not be confident would safely and securely store their data.

Technical Issues
I can envision quite a few technical issues here as neither of us have any experience with encryption or security. However it will offer us a great learning opportunity.

Dependencies with other requirements
All other aspects of the app rely on the secure and safe storage and communication of data.

3.6 App Alerts

Description
Having a way of **reminding** users of their financial commitments and perhaps offering financial advice to help them towards their financial goals would be a very **useful** feature. For example, if there is a pattern of input over a certain period and one week there is no input, the app would alert the user to this fact, we plan to offer an explicit and implicit version. The implicit version operates on the basis of usual input patterns. While the explicit is implemented when the user sets a reminder for this specific event. This will be carried out through developing functions on the backend and the frontend.

Criticality
This is not an overly critical feature however it would be a **nice feature** to see implemented in a finished application. Without it the app would lack the desired feedback and user encouragement.

Technical Issues
I can’t foresee much difficulty in implementing an alert system from a technical perspective.

Dependencies with other requirements
This will depend on the rest of the above infrastructure being implemented as this is merely a way of communicating important information to the user.

3.7 Smart Advice

Description
To offer the user **advice** based on their current financial position. Such as whether they are able to claim tax back on their expenses, which tax bracket they are in and any other publically available information that would be advantageous to include.

Criticality

This feature is not mission critical however it would be a **good feature** to have making the app more useful and thus attracting more users.

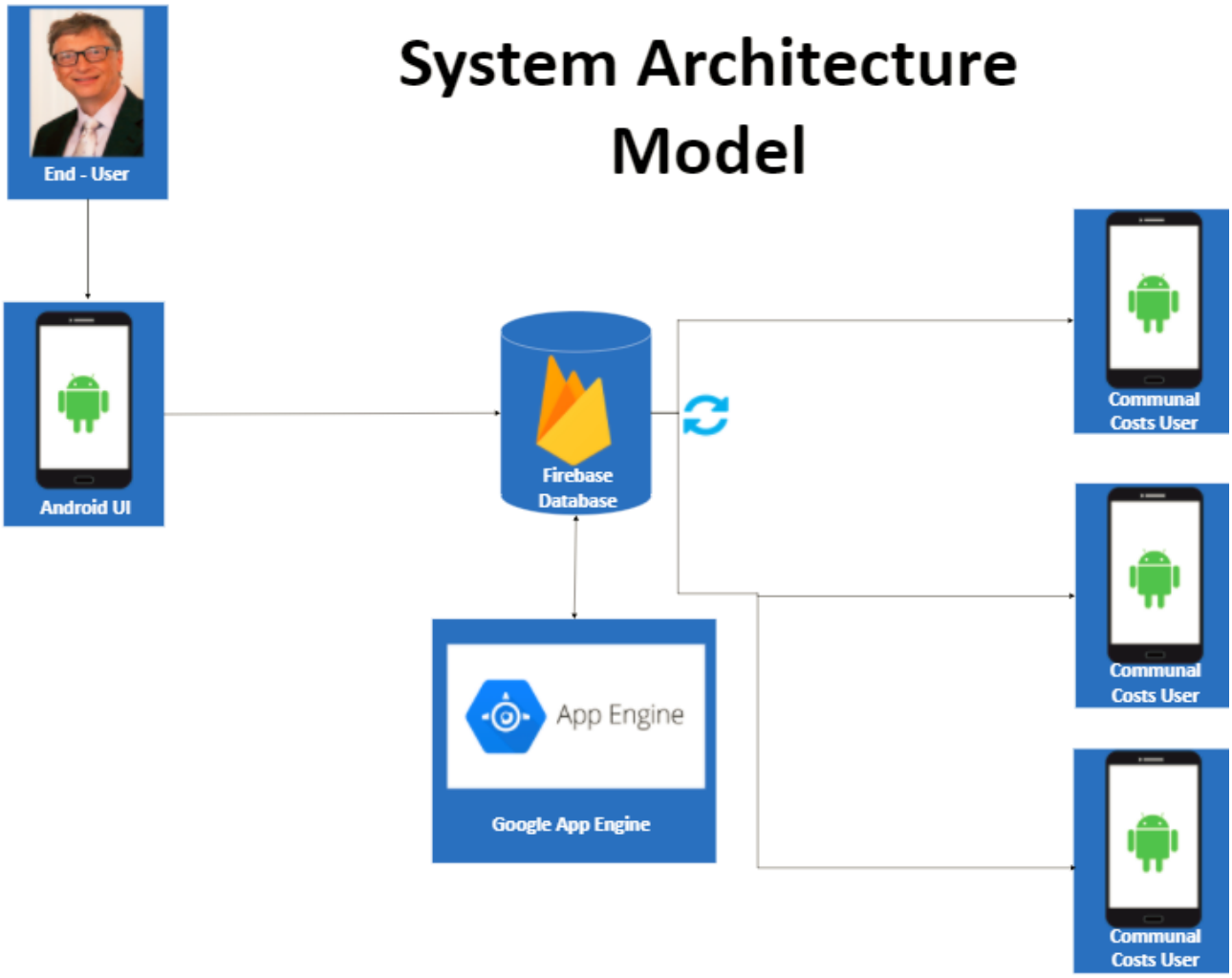
Technical Issues
The only foreseeable technical challenges here would be ensuring that all information and advice is **up to date**. Which would mean that we would have to carefully choose where we get out information from and how **regularly** to update it.

Dependencies with other requirements
This feature will depend on the rest of the above infrastructure being implemented as this is merely a way of communicating important information to the user based on their position.

4. System Architecture

Figure 4.1 shows an **overview** of the system architecture: the end user, the **android user-interface** where the end-user will be able to access and **edit collectives securely**. Our app will be developed with the **Google App Engine** so we will develop **python scripts** here to query the Database and process the information which will then be sent out to all users.

There are quite a few advantages of using **Firebase** and **Google App Engine**, especially when it comes to **google accounts**. The Google App Engine and Firebase also implement **sandboxing** which is quite good for us as it ensures the environment is secure which is of the utmost importance for us and our users.



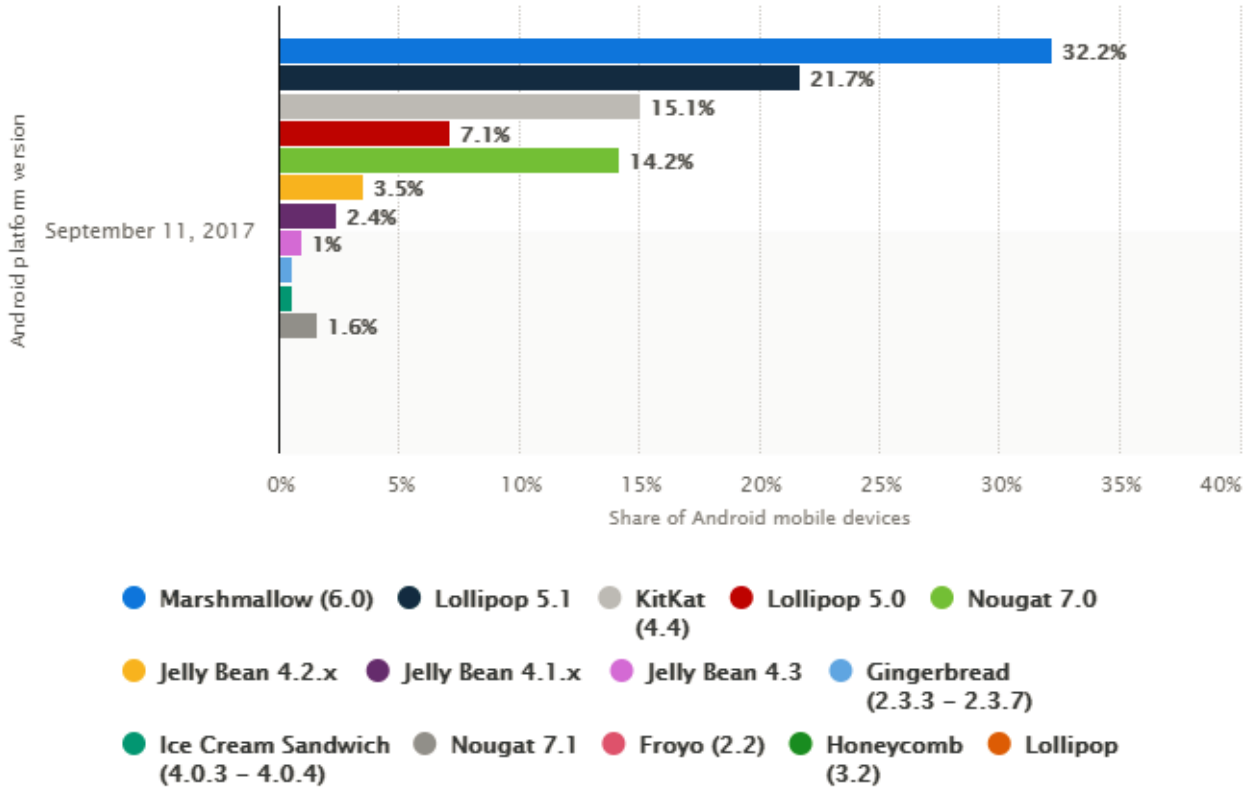
System Architecture Model - Figure 4.1

4.1.1 End User

The end user, who may be an **admin**, **contributor** or **ordinary member** of a collective inputs data into the user-interface through various means.

4.1.2 Android User-Interface

The **user interface**, must be easy to use and aesthetically pleasing. This is what the end-user **interacts** with to input various expenditure and gain information about their collective. The UI will be **programmed** in Java for android for minimum **API level 19** as the current market share is minimal for earlier API levels. (see figure 4.2 below)



current market share androidOS - Figure 4.2

4.1.3 Firebase Database

This is a database offered by **google** which is built into **Android Studio**. It will store our user and collective data. Firebase will also handle the **user authentication** and user synchronisation.

4.1.4 Google App Engine

The google app engine is a **cloud based hosting environment** for our backend. We plan to implement this with **Python**. The app will be **sandboxed** here for security reasons and will always run as it runs on cloud IaaS. It will perform operations on the data which it receives from Firebase which will then **sync** it across **all devices**.

5. High-Level Design

Below is the **context diagram** (figure 5.1) which shows the top level design of our system. The **administrator** can **modify, input and view** expenses. **Contributors** can **input and view** data and modify their own entries. **Ordinary members** can just **view** expenses. All **users receive notifications** from the app. The app then **stores** this information in a **collective** and there is also input from the **Revenue** with regards to information for the **smart advice**. This information is referred to as "**context information**".

5.1 Context Diagram

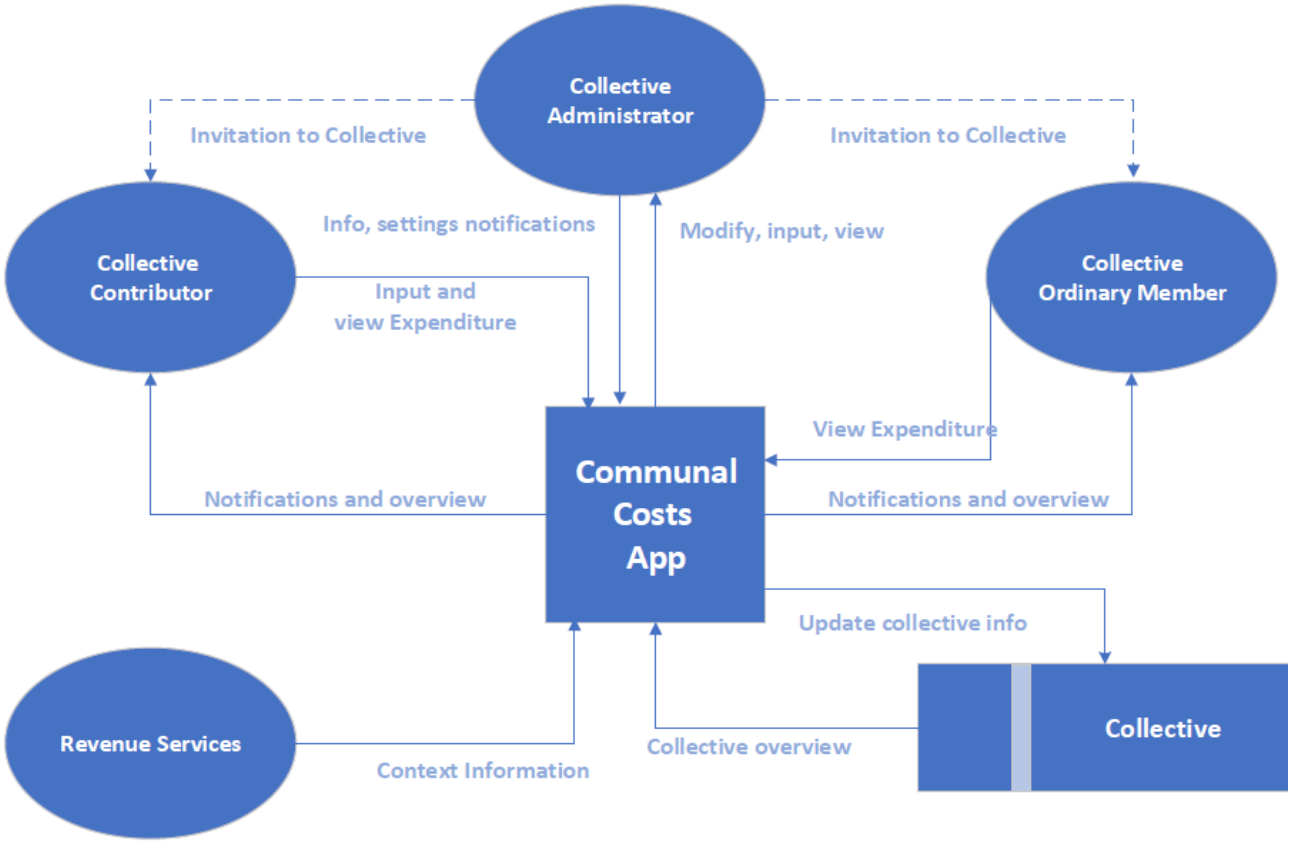


Figure - 5.1

5.2 Data Flow Diagram

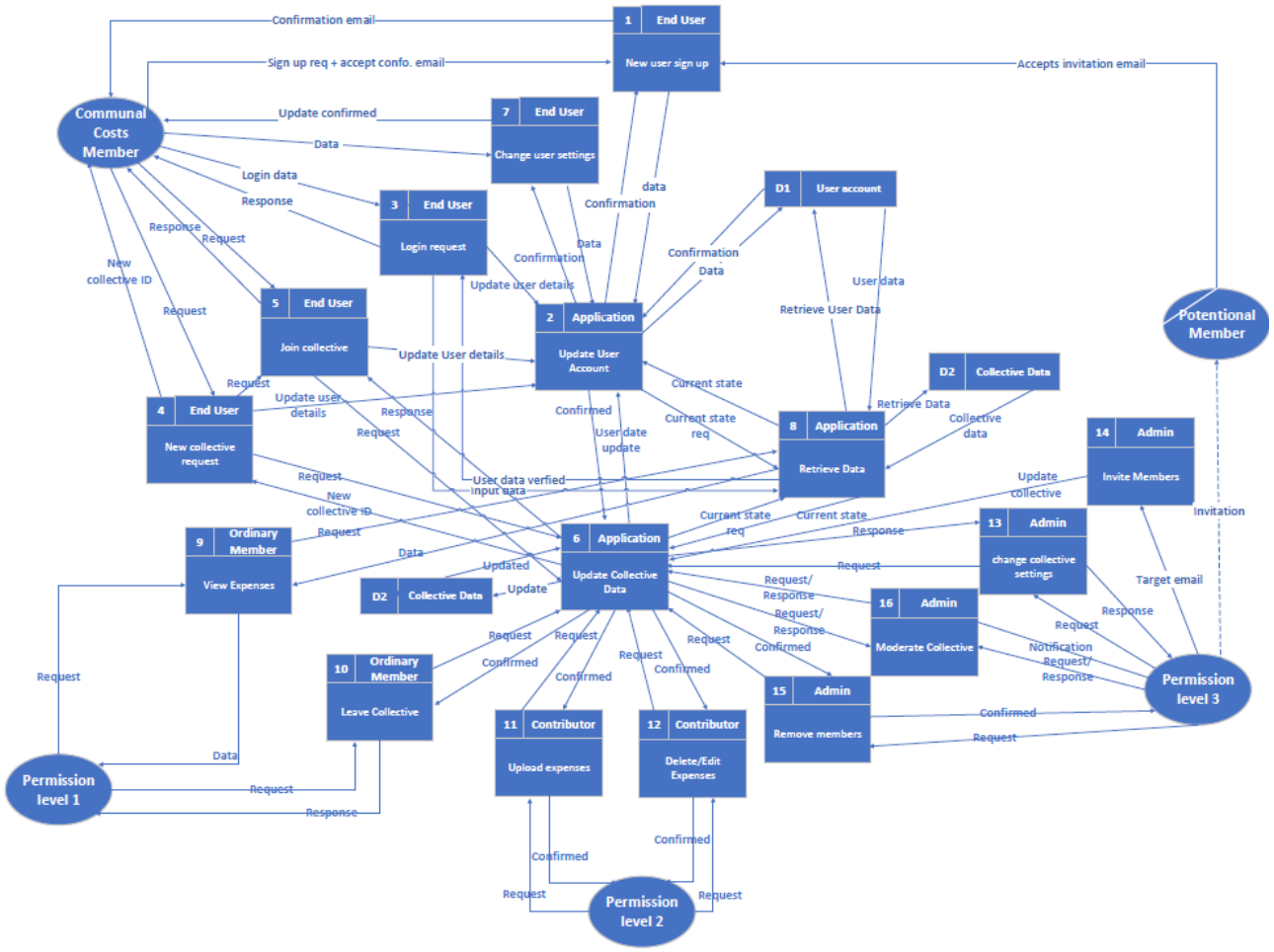


Figure 5.2 - View full DFD in PDF format :- https://goo.gl/zY3CTE

DFD Key		
Type	***Name***	***Description***
Entity	Communal Costs Member	Refers to a user of the Communal Costs app who has successfully registered for the app.
Entity	Permission Level 1	Refers to a user who is a registered member of our app with ordinary member permissions of a collective
Entity	Permission Level 2	Refers to a user who is a registered member of our app with contributor permissions of a collective
Entity	Permission Level 3	Refers to a user who is a registered member of our app with administrator permissions of a collective
External Entity	Potential Member	Refers to someone who has been invited to the system but not yet registered for an account.

Data store	User account (D1)	Refers to the section of the DB which contains user account information. This information includes when they last logged in, what collectives they are a member of, basic personal information.
Data store	Collective data (D2)	Refers to the section of the DB which contains information about all collectives. This section is further split into specific collectives and these sections contain the collective specific information such as, number of members, active and pending members, member permissions, content of a collective.
Process(2)	Update User account	This is one of the inner processes. Whenever any outer process calls this process, it will update the user account data store with the information is has received. It will also send back a confirmation of successful update back to the outer processes to be sent back to the entities that originally triggered the process.
Process(6)	Update Collective Data	This is one of the inner processes. Whenever any outer process calls this process, it will update the collective data, data store with the information it has received. It will then send back a confirmation to the original process which then sends it to the entities that triggered it.
Process(8)	Retrieve Data	This is the main inner process. This process can only be called by the inner processes and the Login request process. This retrieves information from each of the data stores and passes it back to the process that called it. In the case of the inner processes, it sends back the current state of user data or collective data so it can be edited.
Process(1)	New user sign up	Triggered when a user requests a new account, immediately an entry is made on the database and a confirmation email is sent to the sign-up email. Once confirmed, the account is activated.
Process(7)	Change user settings	A confirmation is provided and the User account data store is updated
Process(3)	Login Request	This is unique in the sense that is has direct access to one of the inner processes. It is able to confirm whether the credentials provided match any existing user. Information is also stored with regards to last successful/attempted login and failed login attempts
Process(5)	Join collective	The user needs to know the collective ID or have been invited to the collective by an administrator. Their user account is updated with any new

		collectives
Process(4)	New collective request	This process triggers the join collective process also and creates a new entry in the collective data store and adds it to the users data store.
Process(9)	View Expenses	Permission 1+ : Expenses are retrieved from the collective and displayed
Process(10)	Leave collective	Permission 1+: This is updated in user account and collective data.
Process(11)	Upload expenses	Permission 2+: can be done by various means, but this is updated in the collective account and also the user account in their list of current expenses.
Process(12)	Delete/Edit expenses	Permission 2+: user account and collective data store are updated.
Process(15)	Remove members	Permission 3+: user accounts and collective data are updated.
Process(16)	Moderate expenses	Permission 3+: to change any existing expenses or remove them - collective and user information updated
Process(13)	Change collective settings	Permission 3+: collective data is updated
Process(14)	Invite members	Permission 3+: collective data is updated and user account is updated.

5.3 Logical Data Flow

Below is the **logical data flow** (figure 5.3). This outlines the **basic logic** behind the **system** and how users (entities) **interact** with it.

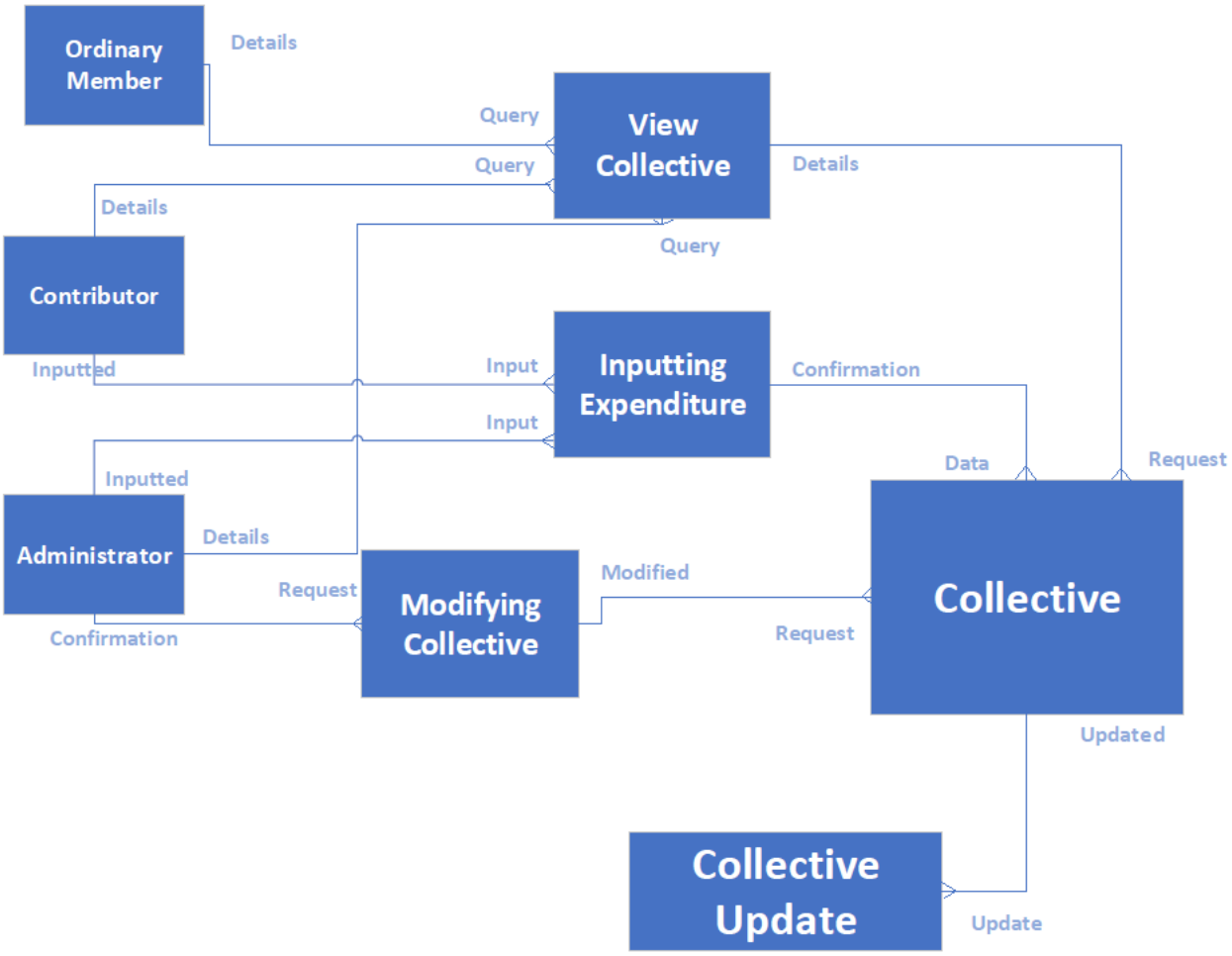


Figure 5.3

6. Preliminary Schedule

6.1 Tasklist

Our **tasklist** is split into 3 main sections, the *Preliminary Deliverables*, the *Development stage* and The *final stretch*. The development stage contains 4 main prototypes, Alpha, Beta, Gamma and the Final stage, along with testing stages for each.

At first, the focus will be on getting our backend up and running and ensuing **secure communication** between the elements of the backend and the backend-frontend connection.

This task list shows the tasks to be completed, who is responsible and the start / finish time along with the amount of days allocated. Christmas and study periods have been included. See **Fig 6.1** below.

	Task Name	Responsible	Duration	Start	Finish	Predecessors	Status
1	**Preliminary Deliverables**		26d	27/10/17	01/12/17		
2	Proposal Submission	team	1d	27/10/17	27/10/17		
3	Presentation Preparation	team	9d	30/10/17	09/11/17		
4	Presentation	team	1d	10/11/17	10/11/17		
5	Functional Spec. Preperation	team	9d	21/11/17	01/12/17		
6	Functional Spec. Submission	team	1d	01/12/17	01/12/17		
7	**Developement**		67d	02/12/17	05/03/18		
8	*Prototype Alpha*		32d	02/12/17	15/01/18		
9	Set up Google App Engine & Firebase	team	3d	02/12/17	05/12/17		
10	Sign-In + Sign Up	team	2d	06/12/17	07/12/17	9	
11	Account + Collective Structure	team	4d	08/12/17	13/12/17	10	
12	Implement collective functionality 1		3d	14/12/17	18/12/17	11	
13	**Christmas**		3d	24/12/17	26/12/17		
14	Implement collective functionality 2	team	7d	27/12/17	04/01/18	13	
15	Secure communication	team	7d	05/01/18	15/01/18	14	
16	Alpha testing	team	2d	16/01/18	17/01/18	15	
17	*Prototype Beta*		25d	09/01/18	12/02/18		
18	**Examination period**		15d	09/01/18	29/01/18		
19	Account permissions	kealan	3d	30/01/18	01/02/18	18	
20	App alerts	james	3d	02/02/18	06/02/18	19	
21	Collective goals	team	4d	07/02/18	12/02/18	20	
22	Beta testing	team	2d	13/02/18	14/02/18	21	
23	*Prototype Gamma*		9d	15/02/18	27/02/18		
24	Receipt reader	james	3d	15/02/18	19/02/18	22	
25	PDF reader	james	3d	20/02/18	22/02/18	24	

26	Support e-mail forum	kealan	3d	23/02/18	27/02/18	25	
27	Gamma testing	team	3d	28/02/18	02/03/18	26	
28	*Final Prototype*		4d	28/02/18	05/03/18		
29	Discussion forum	team	4d	28/02/18	05/03/18	26	
30	Forum Testing	team	1d	05/03/18	05/03/18	27	
31	**The final stretch**		5d	05/03/18	09/03/18		
32	Demo preparation	team	3d	05/03/18	07/03/18	27	
33	**Final Testing**	team	3d	06/03/18	08/03/18		
34	Submission to supervisor	team	1d	09/03/18	09/03/18		
35	Demonstration	team	1d	TBD			

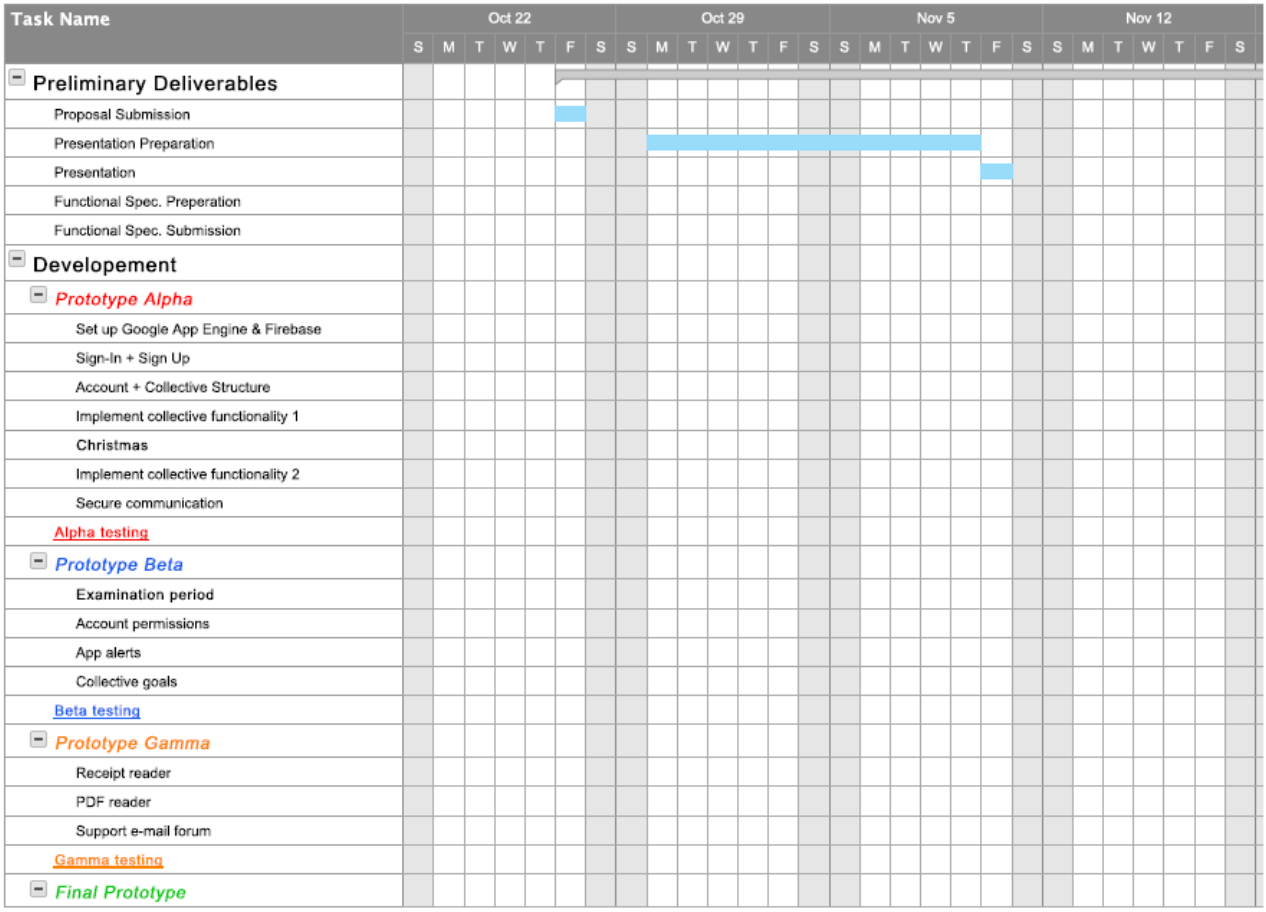
Fig 6.1 - Tasklist

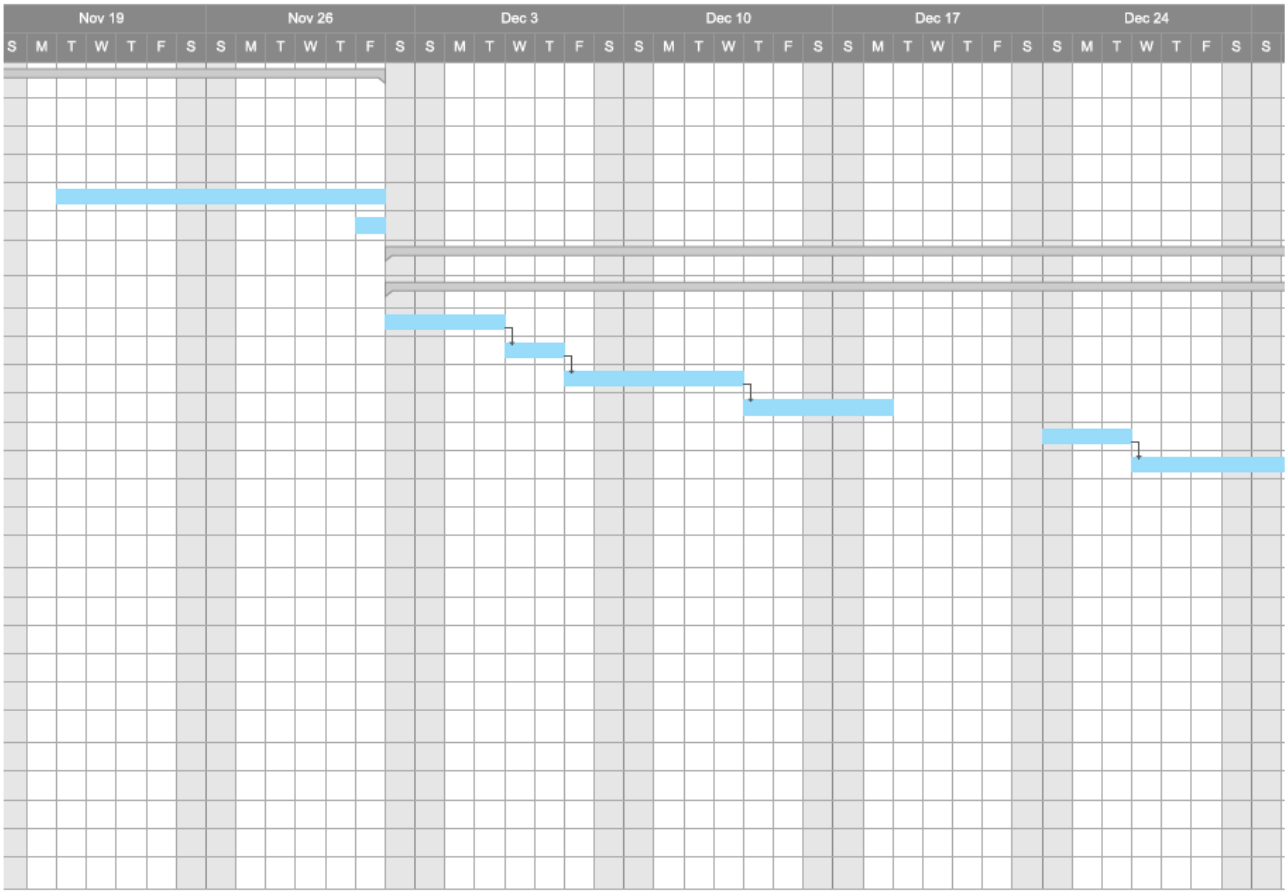
6.2 GANTT Diagram

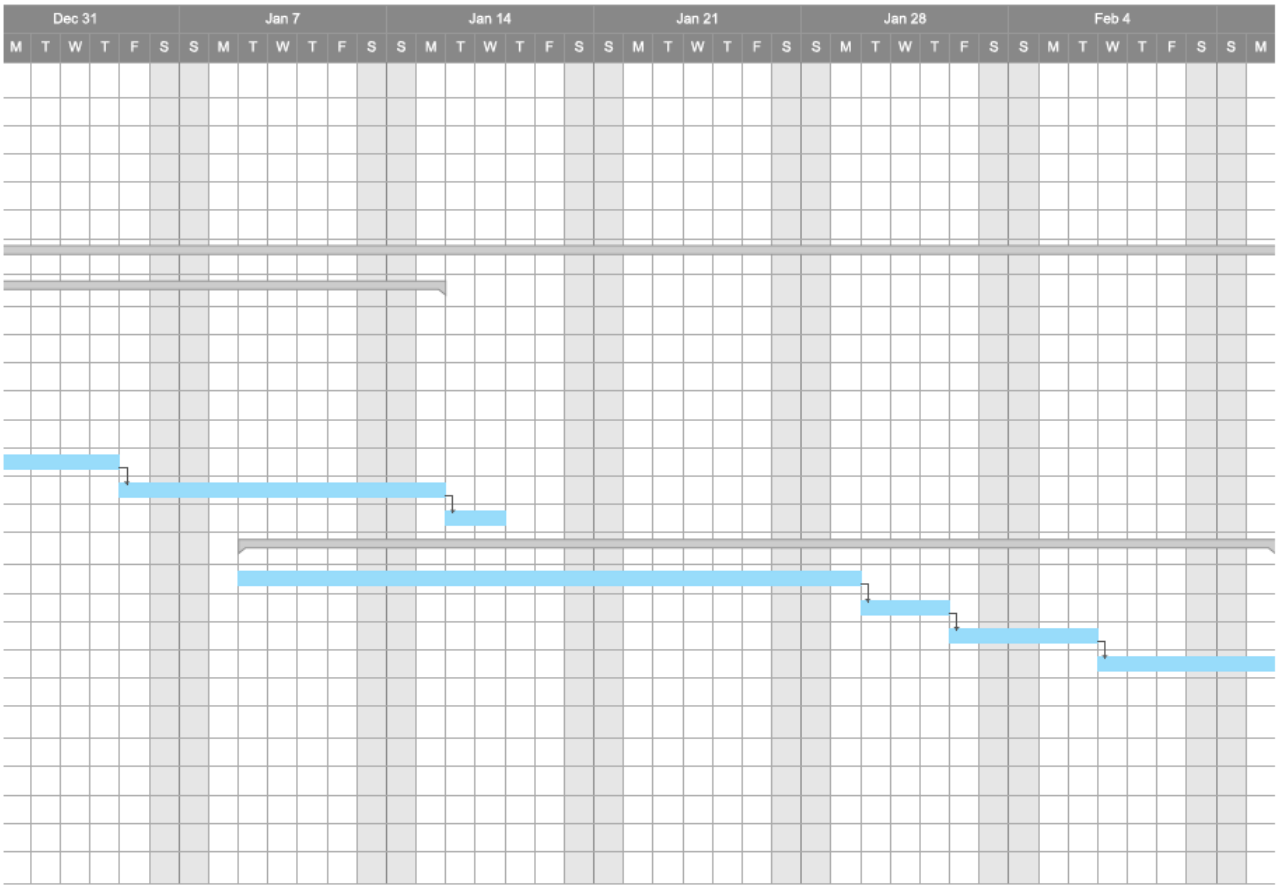
From this task list we derived a **GANTT Diagram** which shows the various stages of our project in a visual format outlining the various milestones and our expected finish date.

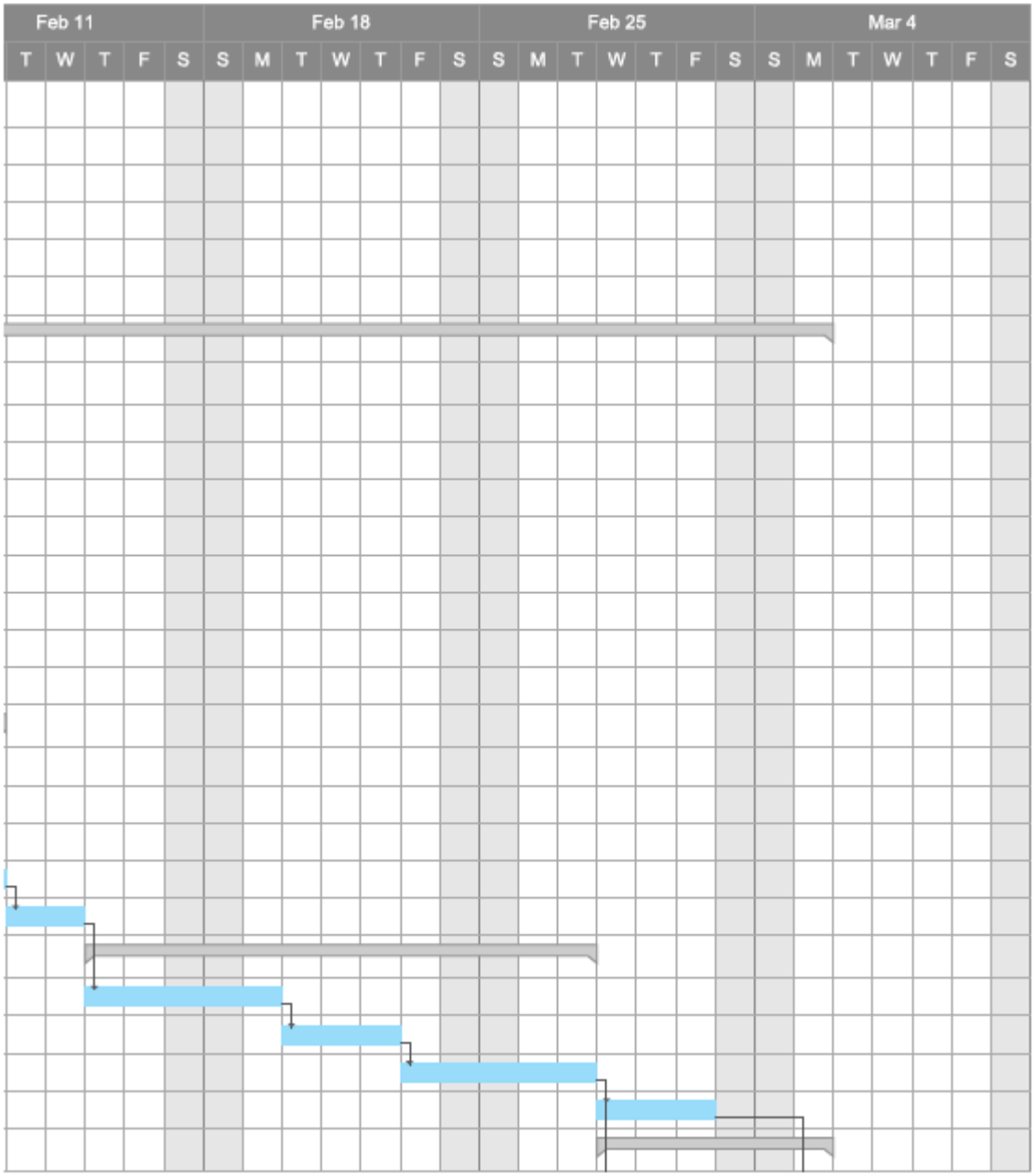
The **GANTT diagram** shows tasks line by line and shows the relation to it's predecessors through a black arrow. These tasks can only start once the predecessor has been completed. The gray bars represent the timeline of the three main sections. See **Fig 6.2** below.

view full GANTT pdf - https://goo.gl/4HQBEv




Nov 19
Nov 26
Dec 3
Dec 10
Dec 17
Dec 24
S M T W T F S S M T W T F S S M T W T F S S M T W T F S S M T W T F S S M T W T F S S


Dec 31
Jan 7
Jan 14
Jan 21
Jan 28
Feb 4
M T W T F S S M T W T F S S M T W T F S S M T W T F S S M T W T F S S M T W T F S S M



	Task Name	Oct 22							Oct 29							Nov 5							Nov 12						
		S	M	T	W	T	F	S	S	M	T	W	T	F	S	S	M	T	W	T	F	S	S	M	T	W	T	F	S
29	Discussion forum																												
30	Forum Testing																												
31	The final stretch																												
32	Demo preparation																												
33	Final Testing																												
34	Submission to supervisor																												
35	Demonstration																												

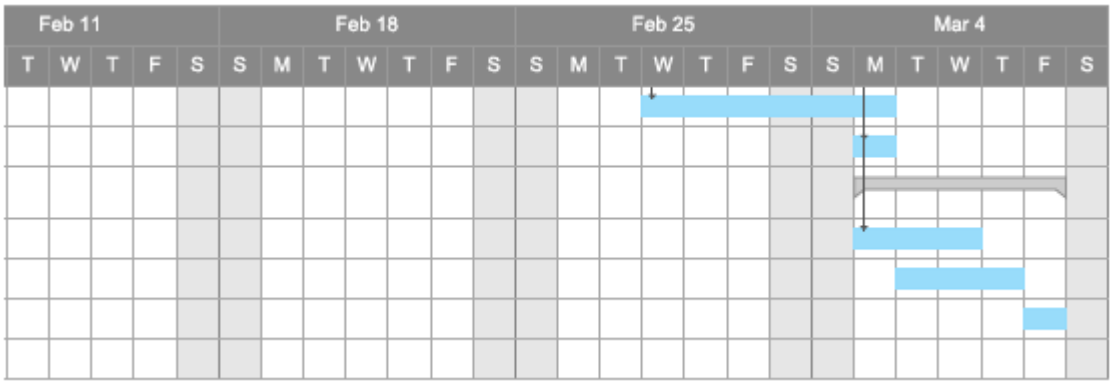


Fig 6.2.1 - GANTT Diagram (broken down)

7. Appendices

https://cloud.google.com/solutions/mobile/mobile-firebase-app-engine-flexible - Using Firebase and Google App Engine for an Android project. To assist with understanding system architecture section.
https://cloud.google.com/appengine/docs/standard/python/tools/using-local-server - Using local development server to simulate the engine environment before deploying to the Cloud.
http://searchcloudcomputing.techtarget.com/definition/Infrastructure-as-a-Service-IaaS - Infrastructure as a Service for Cloud Computing.
https://cloud.google.com/appengine/ - Google App Engine
https://firebase.google.com/ - Google Firebase
https://www.w3schools.com/sql/sql_injection.asp - SQL Injection
https://en.wikipedia.org/wiki/Sandbox_(computer_security) - Sandbox (Computer Security)