Gyg Inc.

Isuru Nanayakkara
Shawn Mitchell
Anthony Timberlake
Jonathan Luetze
Drew Vivian

Table of Contents

Change History

Version	Summary	Author	Date
1.0	Added intro/Scope	Shawn Mitchell	02/15/2018
1.1	Added Comparisons with Leading Alternatives Table	Anthony Timberlake	02/15/2018
1.2	Added Features	Jonathan Luetze	02/15/2018
1.3	Added Functional / Non Functional tables.	Isuru Nanayakkara	02/15/2018
1.4	Added Comparisons with Leading Alternatives Description paragraphs	Anthony Timberlake	02/22/2018
1.5	Added Non-Functional specific items	Isuru Nanayakkara	02/27/2018
1.6	Added title page/table of contents	Shawn Mitchell	02/27/2018
1.7	Included informational graphics	Isuru Nanayakkara	03/01/2018
1.8	Made minor changes to Comparisons with Leading Alternatives table and descriptions	Anthony Timberlake	03/01/2018
1.9	Added extra features for Gyg features page	Jonathan Luetze	03/01/2018
1.10	Added use case, class, and activity diagrams	Andrew Vivian	03/01/2018
1.11	Simplified/updated Comparisons with Leading Alternatives features descriptions	Anthony Timberlake	3/29/2018
2.0	Added Sequence Diagrams	Jonathan Luetze	3/29/2018
2.1	Added description to sequence diagram	Shawn Mitchell	3/29/2018
2.2	Added description of Login Sequence Diagram	Anthony Timberlake	3/29/2018
2.3	Added description to JobList	Isuru Nanayakkara	3/29/2018

2.4	Added description to Job	Jonathan Luetze	3/29/2018
2.5	Added login sequence diagram	Andrew Vivian	03/30/2018

1. Introduction

a. Purpose

This app will create an easy market for people looking for side jobs and those with side jobs. It will handle the transactions and also help create a safe way for people to find jobs or workers for their jobs. The purpose of this document is to outline the requirements specifications for the Gyg project. It is to be used as a guide to help with the development of Gyg. The document outlines the features and technology that will be used to build the app. It also contains outlines of UML diagrams of use cases and the functional and nonfunctional requirements for the app.

b. Scope

The scope of this project is to create a user friendly front end and back end that handles all transactions. The back end will hold a database that stores the jobs and user information.

c. Definitions

Gyg - Side job

2. Features

0. Login Page
0.1 Create Profile

1. **My Gygs (list of Gygs that the user does)**
    a. Completed/Pending Jobs View (sorted by date)
        i. Info button
            1. Time and date of transaction
            2. Payment details
            3. QR code to start/finish job
        ii. How much money the user has received/will receive
    b. View private comments for users that have worked for you before/that you have worked for before
2. **Find Gygs (user finds Gygs that they can do)**
    a. Filter button (will be displayed at the top of screen):
        i. User filter
        ii. Category filter (type of Gyg, e.g. moving lawn)
        iii. Area filter (if user wants to do something outside of their area)
        iv. Gyg post in the list will have an accept and info button as well as pay
3. **Post Gyg (user posts Gyg that they need another user to do)**
    a. Create a new Gyg
        i. Gyg Title
        ii. Gyg Category
        iii. Gyg Area
        iv. Gyg Description
        v. Gyg pay
            1. Might include a range that can be set (future feature)
        vi. Date and Time
            1. Ability to edit (would then send a notification to the other person)
    b. View posted Gygs
        i. Info button
            1. Displays everything entered in new Gyg
        ii. Date and time uploaded
    c. Completed Gygs
        i. Info button
            1. Displays everything entered in new Gyg
            2. Date and Time completed
            3. User served
            4. Personal notes on Gyg (Future feature)
        ii. Transaction button (links to transactions screen in profile, highlighting selected Gyg transaction)

4. **My Profile**
    a. Name
    b. Picture
    c. General working area
    d. Skillset
    e. Description
    f. Reviews (from other Gyg Users)
    g. View Past & Future transaction (different screen)

3. Functional / Non-functional Requirements

Front End

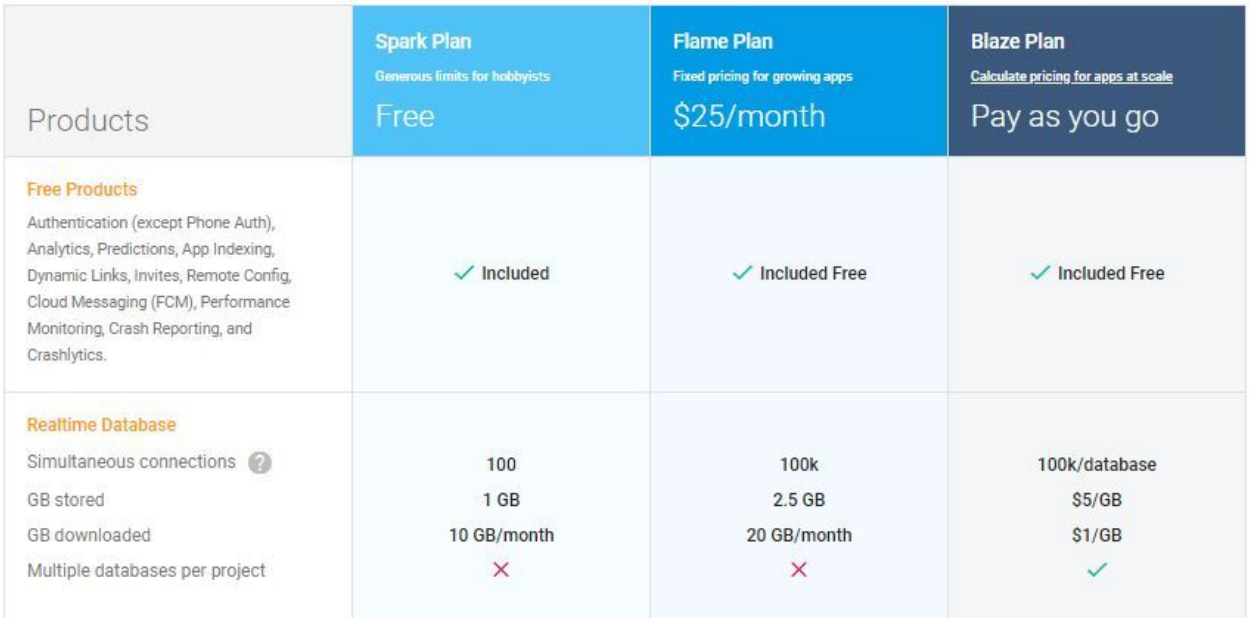
User Expectations	● User must be able to post a job they want done. ● User must be able to offer a job that they can do. ● User must be able to update or add new information about themselves.	Functional
User to User interaction	● Prior to accepting a job, the employee user must scan the QR code and the employer user must accept it. This verifies job acceptance. ● Upon job completion the employee user must scan again with the employer user to verify job completion. Once transaction is complete the payment process can move forward.	Functional
Notifications	● User can post a job and leave app. If a job is accepted a notification will be sent to the user and they can view/accept/decline the job.	Functional
Map	● Location services / map GUI for locating Gygs or posting Gygs of your own	Non - Functional

Back End

Title	Description	Special Notes
Firebase for data storage	Use firebase to store user data. This includes user name, skills, general location etc.	Functional
App engine written in Node.js	Use Google App Engine to process user requests. Will be written in Node.js. Most user interactions will make calls to the server. The server will process this information and relay it back to the app.	Functional
Payment handling. Possibly PayPal API	Once job transaction is complete. The financial service will process the funds transfer appropriate for the job.	Other options are Plaid and Stripe. Functional
Server must handle delays	If the users have bad connections then the server must properly synchronize the data.	Non-functional

Non Functional Specific Features:

1. **Availability:** The app will be available in the States first. In the future we will expand it to support it globally.
2. **Capacity:** The app will be able to support at least 100 active users at a time. This is due to using Firebase as a backend. Only 1GB of JSON data can be stored every month.



Products	Spark Plan Generous limits for hobbyists Free	Flame Plan Fixed pricing for growing apps $25/month	Blaze Plan Calculate pricing for apps at scale Pay as you go
Free Products Authentication (except Phone Auth), Analytics, Predictions, App Indexing, Dynamic Links, Invites, Remote Config, Cloud Messaging (FCM), Performance Monitoring, Crash Reporting, and Crashlytics.	✓ Included	✓ Included Free	✓ Included Free
Realtime Database			
Simultaneous connections	100	100k	100k/database
GB stored	1 GB	2.5 GB	$5/GB
GB downloaded	10 GB/month	20 GB/month	$1/GB
Multiple databases per project	✗	✗	✓

Figure 1: Screenshot of Firebase storage plans and limitations.

3. **Scalability:** The app back-end will be easily scalable thanks to the use of Google App Engine and the Node.js runtime environment.

4. **Pushing Updates and Servicing:** Updates to the backend can be done without having to update the Android portion of the app. Updates to the Android files will require pushing updates to Play Store.

5. **Data Handling:**
    - Firebase will be the database being used.
    - Data will be structured in JSON format.
    - Data will be retained once stored.
    - Data will only be accessed by authenticated users.
    - Firebase authentication features will be heavily enforced.
    - Data about users will be as minimal as possible.

6. **Performance:**
    - No heavy processing will be done on the front-end
    - Front end only sends small chunks of data to the back-end.
    - Back end does all API calls, calculations, Firebase data handling etc.
    - Performance will be based on the user's device and internet connection.

7. **Supportability:**
    - The Firebase Flame Plan is cost effective. $25 dollars is a reasonable price if expansion is required.
    - Google Cloud Services will require additional fees but they are pay as you go.
    - Pricing will rise relative to the number of users.
    - If enough money can be generated from the app it will compensate for this cost.
    - 5 million monthly users will cost $2038/month.

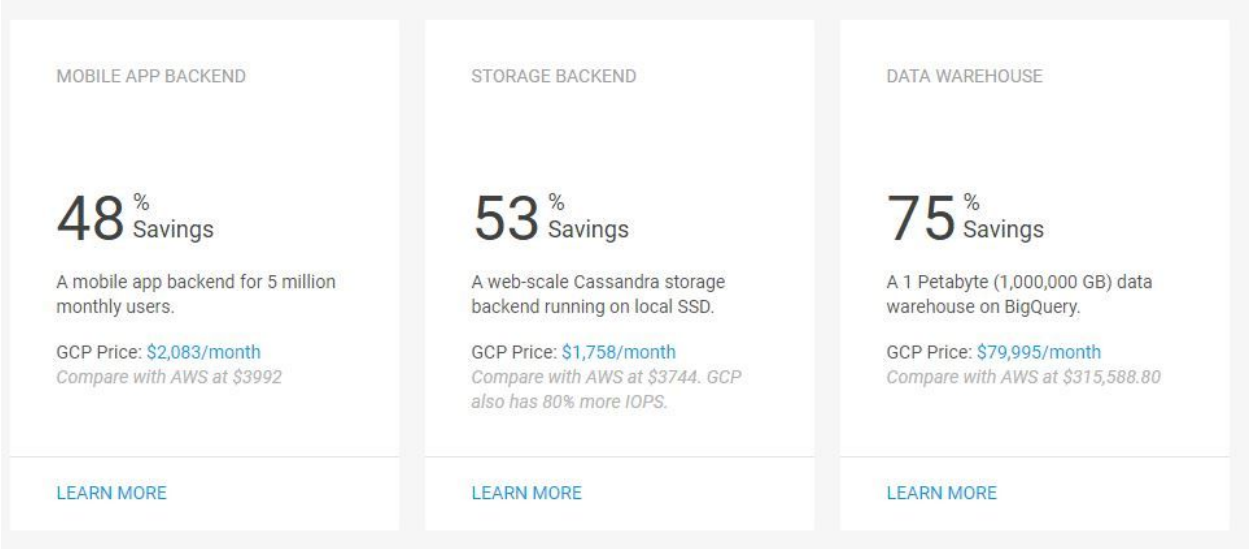


Figure 2: Google Cloud Backend Pricing

4. Comparisons with Leading Alternatives

Requirement	post/offer job	External Finance API Support	Update user profile	QR Code verification to start job	QR Code verification to end job	Push notification
Facebook	full	none	partial	none	none	partial
taskRabbit	partial	none	full	none	none	full
craigslist	full	none	partial	none	none	partial
FieldAgent	partial	none	none	none	none	none
Handy	partial	none	full	none	none	full

Facebook: Users can post or offer jobs they need done on Facebook buy/sell/trade pages, but the information about individual users is either limited to those posts or their individual profiles, which may be desired to be kept private. In terms of ensuring completeness of a job and external financial API support, Facebook has no means for either. Users can make posts seeking a person to satisfy a job and may get a notification for comments on said post, but they will be mixed with all other facebook notifications and users may have push notifications for these altogether disabled.

taskRabbit: TaskRabbit is geared towards specific tasks that the user can search for. There are profiles available for more information for users and users can leave the app and receive push notifications for updates to specific jobs available. There is no use of QR codes to ensure a job's completion.

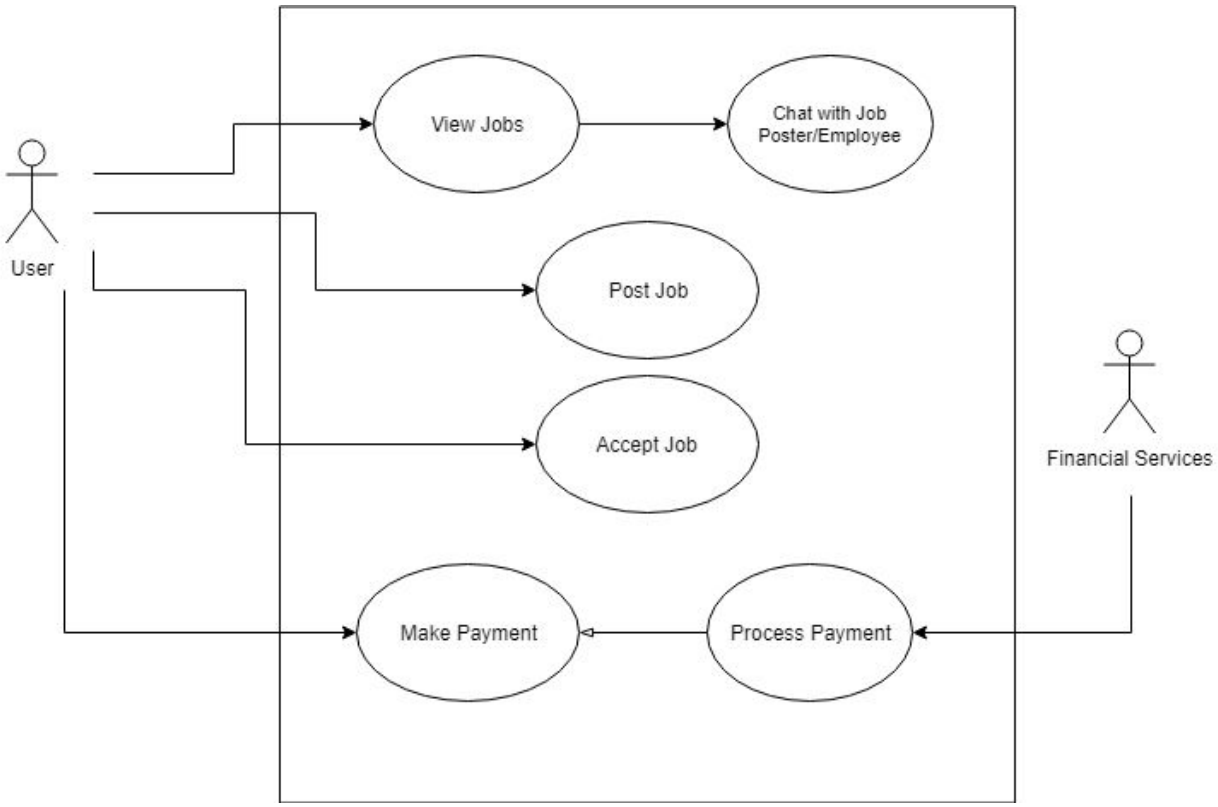
Handy: Handy is also aimed at specific tasks that the user can search for. This app also supports profiles and push notifications.There is no use of QR codes to ensure a job's completion in this app either.

Craigslist: Users have no form of profiles and can only keep information within posts made for a local area offering. Craigslist also has a bad reputation for being unsafe or untrustworthy. There is no QR code insurance for job completion or external financial API support, but upon response to a post, users receive email notifications.

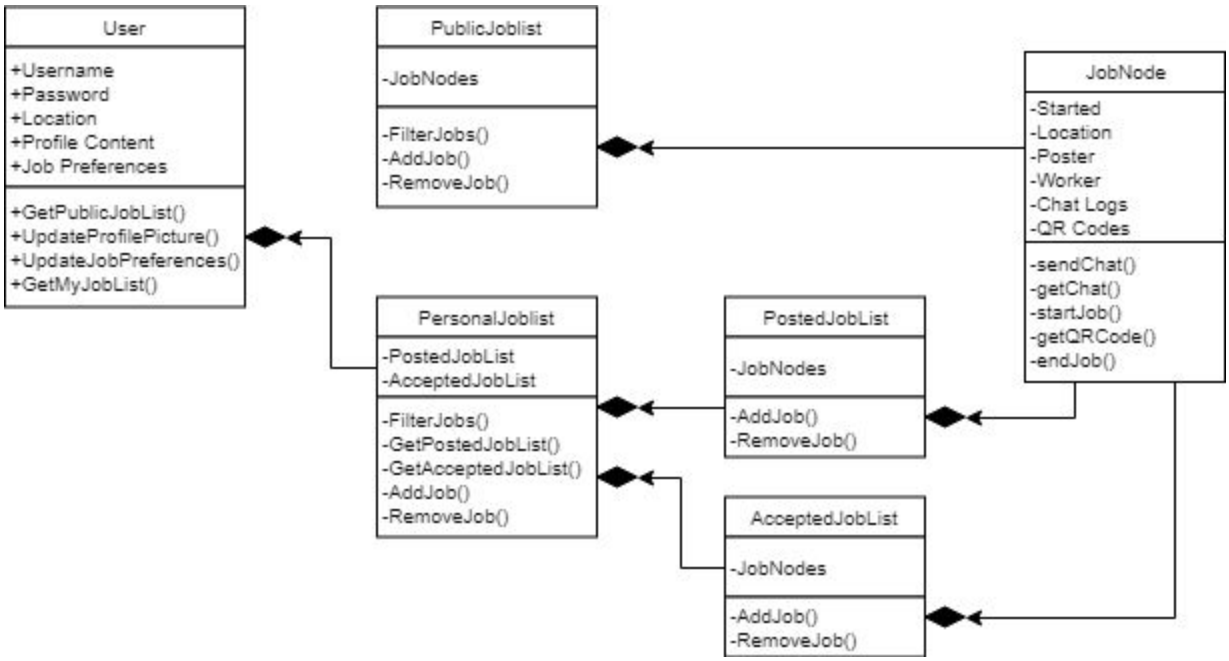
FieldAgent: Users complete local tasks for money, but cannot offer their own jobs they need done at the time. There are profiles per user, but only to keep up with the amount of money and tasks taken for each user. There is also no external financial API support.

5. Diagrams

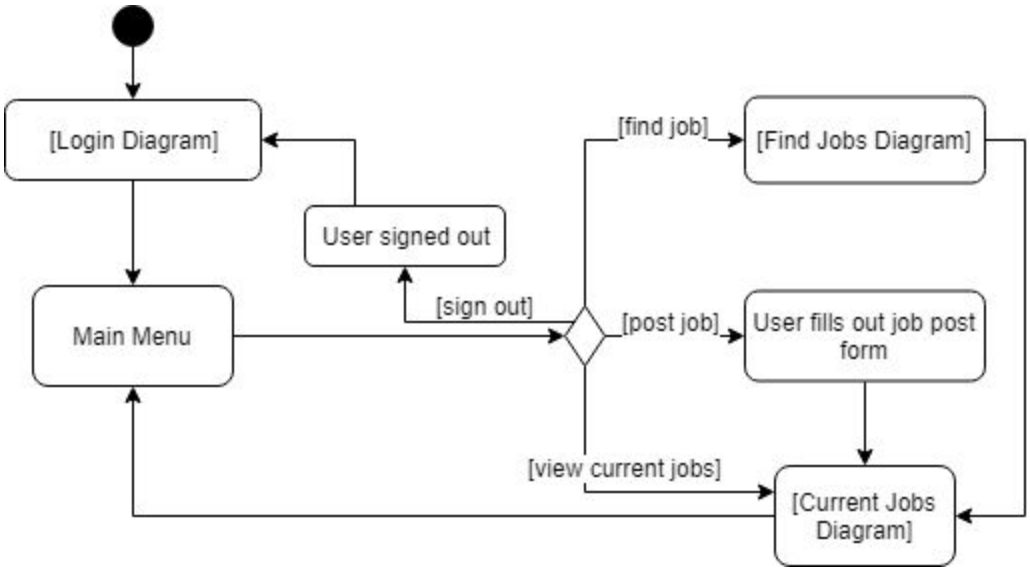
Use Case Diagram



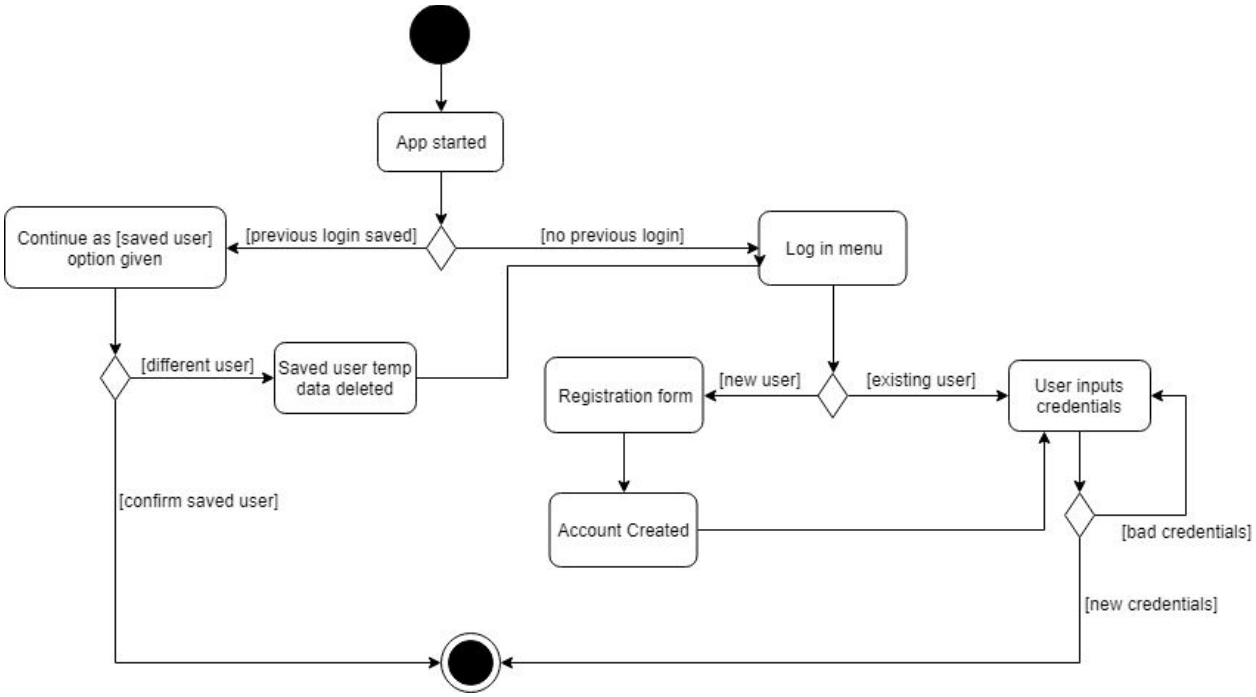
Class Diagram



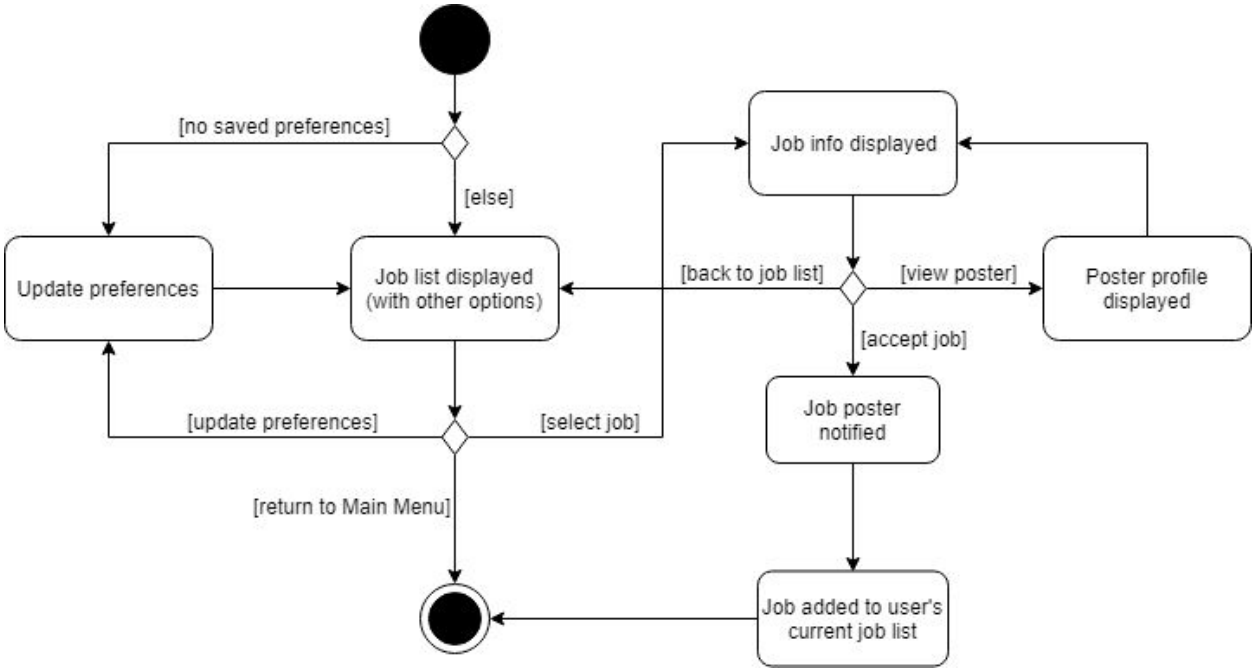
Full-app Activity Diagram



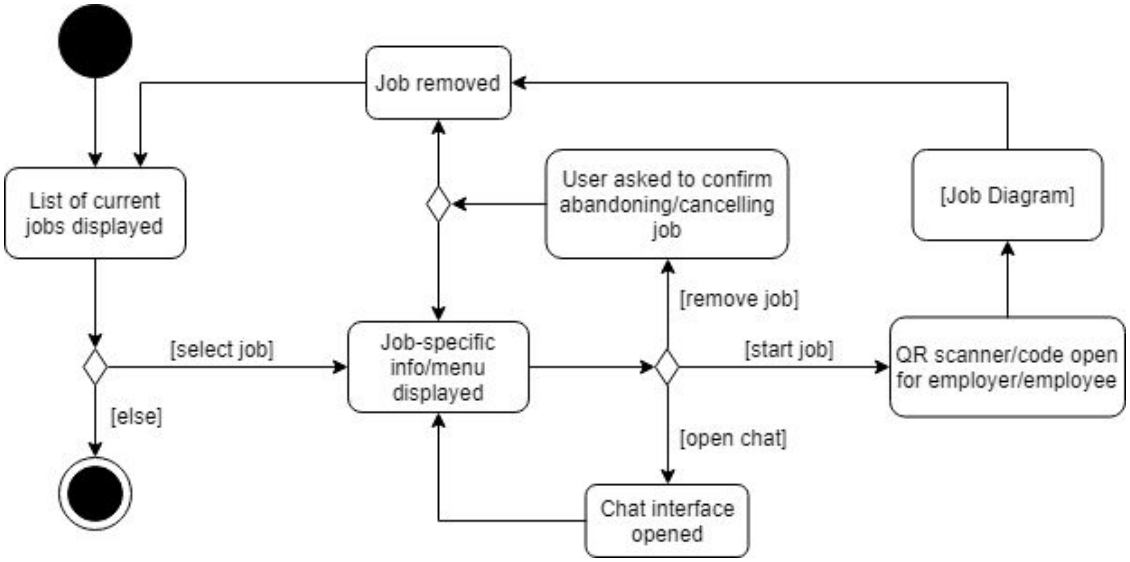
Login Diagram



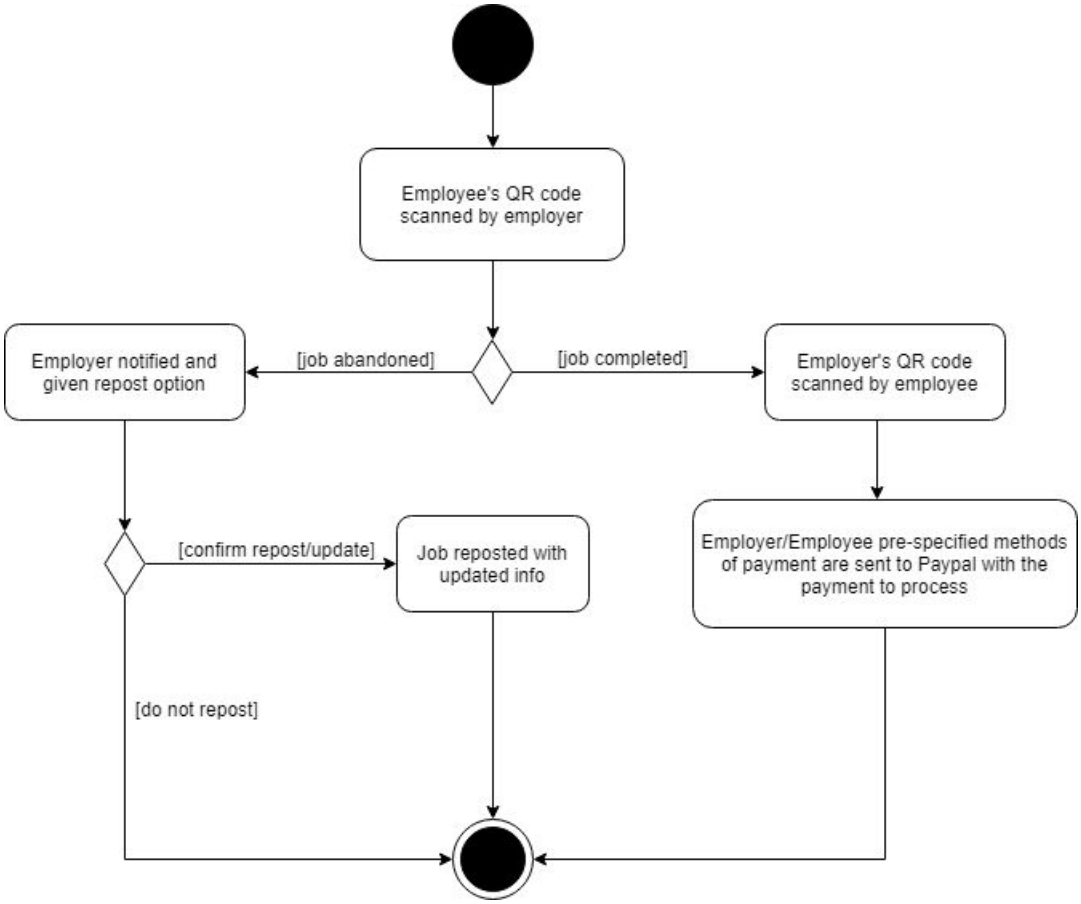
Find Jobs Diagram



Current Jobs Diagram



Job Diagram


Employee's QR code scanned by employer
Employer notified and given repost option
[job abandoned]
[job completed]
Employer's QR code scanned by employee
[confirm repost/update]
Job reposted with updated info
Employer/Employee pre-specified methods of payment are sent to Paypal with the payment to process
[do not repost]

Design Section

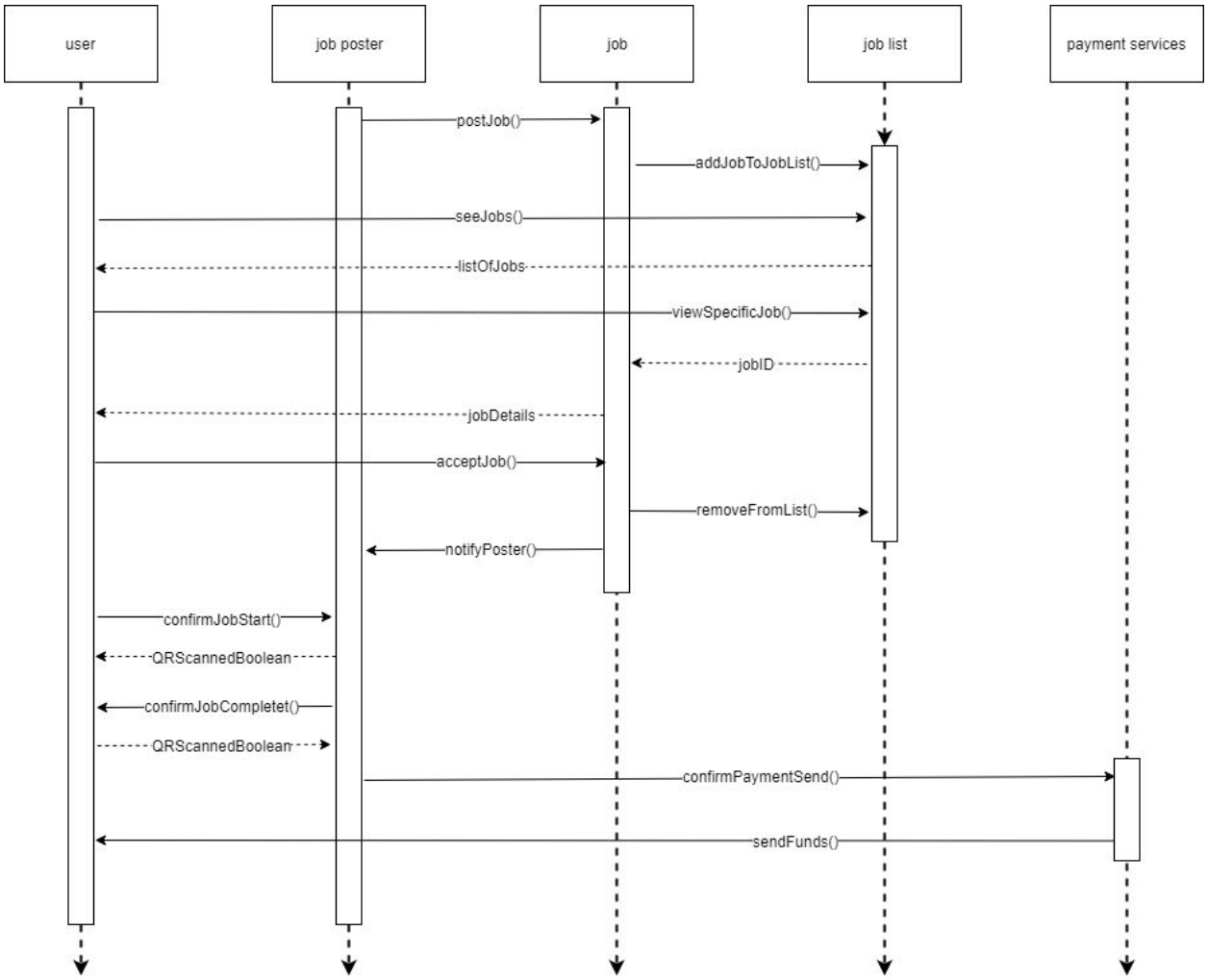


Figure 3: Overall Sequence Diagram

Job:

- Receives information from job poster to create a job object, and will then add the job object to the job list. After the user accepts a job, they will receive the details from the job object, the job object will be removed from the job list, and the job poster will be notified of the update.

Job List: The list of jobs made public in the network.

- A job can be added to the list. Once added all users should have access to the updated joblist. A user can view specific jobs available on the master list. A job can also be deleted from the list when required.

Figure 4: User and Job Poster Sequence Diagram

The user can see all jobs available and can also see jobs they are currently working on. From the job list they can accept a job and confirm its start by scanning the job posters QR code. Once both user and job poster have scanned each others QR code the funds will be sent the user.
The job poster can post jobs and will be notified when a user has accepted the job. They can also confirm the payment to send to user.

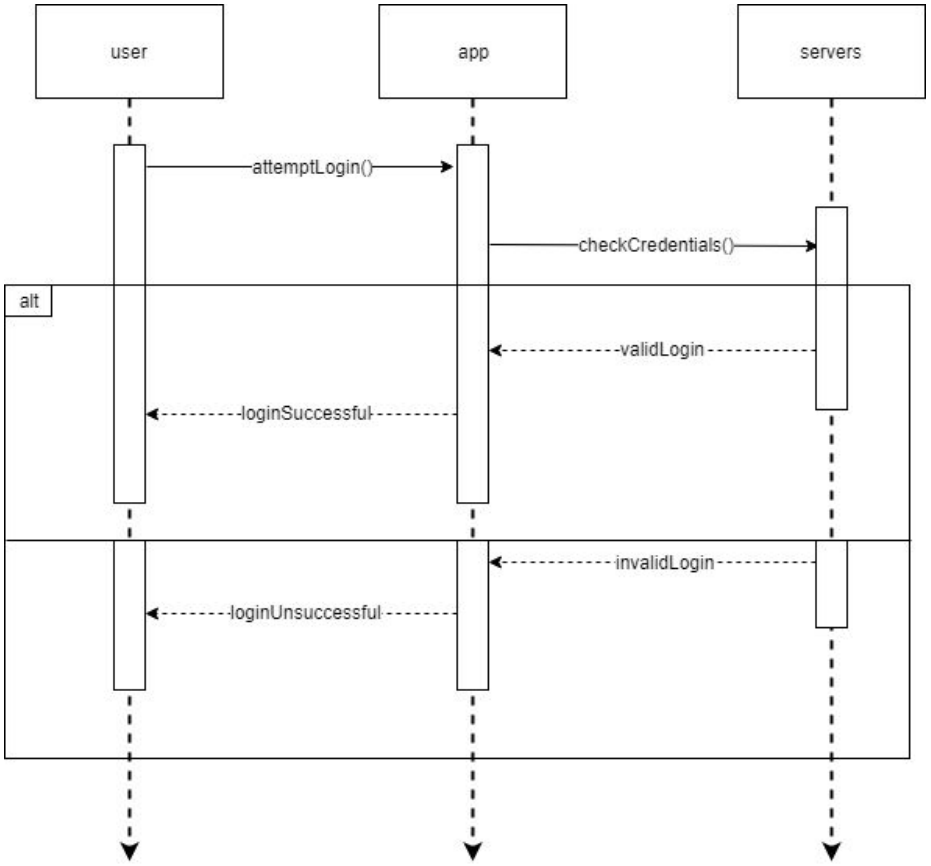


Figure 5: Login Sequence Diagram

The user will first attempt to login using his or her credentials. This will call attemptLogin(), which will be called to the app. The app will then call checkCredentials() to the server to see if the credentials the user entered can be validated in the database. Then, depending on whether the credentials are matched in the database or not, it will respond to the app with either validLogin or invalidLogin. This will then give the response on the app to the user of either loginSuccessful or loginUnsuccessful.