March 18th, 2016

S3

Hire

Technical Design Documents

SENG 321
Fall 2015

Prepared by
Andrei Taylor • Ben Hawker • Jake Cooper • Jonah Boretsky

SoftStart

Contents

List of Figures iv

List of Tables v

Glossary vi

Executive Summary vii

1 Introduction 1

2 Functional Specification 1
- 2.1 Objectives . . . 2
- 2.2 Market Position . . . 2
- 2.3 Technical Requirements . . . 3

3 User Interface 3
- 3.1 Authentication . . . 5
- 3.2 Job Board . . . 7
- 3.3 Jobs . . . 8
- 3.4 Profile . . . 11

4 Use Cases 12
- 4.1 Environments . . . 12
  - 4.1.1 For Hirers . . . 12
  - 4.1.2 For Hirees . . . 12
- 4.2 Scenarios . . . 13
  - 4.2.1 Hiring from Home . . . 13
  - 4.2.2 Getting Hired on the Go . . . 14

5 Data Management 15
- 5.1 Implementation . . . 15
  - 5.1.1 Pre-Posted . . . 15
  - 5.1.2 Posted . . . 16
  - 5.1.3 Offer . . . 16
  - 5.1.4 In-process . . . 16
  - 5.1.5 Finished . . . 22
- 5.2 Stack . . . 22
  - 5.2.1 Front End . . . 26
  - 5.2.2 Back End . . . 26
  - 5.2.3 Database . . . 27

5.2.4 ECommerce 27

6 Class Structure of Data Transfer Objects **27**
6.1 Person Object 28
6.2 Review Object 28
6.3 Job Object 30
6.4 Offer Object 30

7 System Interactions **31**
7.1 Responding to Job Posting 31
7.1.1 Client 31
7.1.2 Server and Database 31
7.2 Notification of Job Request 32
7.2.1 Server and Database 32
7.3 Posting a Job 33
7.3.1 Client 33
7.3.2 Server and Database 33
7.4 Search for a Job 34
7.4.1 Client 34
7.4.2 Server and database 34
7.5 Delete/Edit Job 35
7.5.1 Client 35
7.5.2 Make An Offer 35
7.5.3 Client 36
7.5.4 Server and Database 36
7.6 Accept/Decline Offer 37
7.6.1 Client 37
7.6.2 Server and Database 37
7.7 Complete a Job 38
7.7.1 Client 38
7.7.2 Server and Database 38
7.8 Cancel A Job 39
7.8.1 Client 39
7.8.2 Server and Database 39
7.9 Review A User 40
7.9.1 Client 40
7.9.2 Server and Database 40

8 Testing **41**
8.1 Unit Tests 41

8.1.1 Registration . . . 42
8.1.2 Login . . . 43
8.1.3 Profile . . . 44
8.1.4 Hire Board . . . 45
8.1.5 Hire Map . . . 46
8.1.6 Job Interactions . . . 47
8.1.7 Feedback System . . . 48
8.1.8 HireChat . . . 49
8.1.9 Logging out . . . 50
8.2 Integration Tests . . . 51
8.2.1 Review . . . 51
8.2.2 Offer . . . 51
8.2.3 Person . . . 52
8.2.4 Job . . . 52
8.3 Functional Testing . . . 53

9 Reporting Criteria **54**
9.1 Gerrit Code Review . . . 54
9.2 Continuous Integration . . . 56
9.3 Test Driven Development . . . 57

10 Management Plan **57**
10.1 Gerrit . . . 57
10.2 SCRUM . . . 58
10.3 Scheduling and Employees . . . 58
10.3.1 Project Schedule . . . 59

11 Summary **60**

12 References **61**

List of Figures

1 Authentication Process 5
2 Sign Up Process 6
3 Job Listings 7
4 Jobs the User is Interacting With 8
5 An Active Job and Job Chat 9
6 An Job being Completed and Reviewed 10
7 The Users Profile 11
8 Responding to a posting 14
9 Application Flow 15
10 Saving a Posting Implementation 17
11 Searching Jobs 18
12 Edit or Delete a Job 19
13 Make an Offer 20
14 Accept or Decline an offer 21
15 Complete a job 23
16 Cancel a job 24
17 review a job 25
18 Hire Architecture 26
19 Class Diagram of Data 29
20 Gerrit's Interface 54
21 Gerrit's Code Review Process 55
22 Inline Comments In Gerrit 56
23 History in Gerrit 56
24 TDD Workflow [1] 58

List of Tables

1 Developers Roles and Assignments 59
2 Expected Completion Dates 59

Glossary

DTO An object that carries data between processes. 27, 30, 60

iOS The mobile operating system created by Apple for the iPhone and iPad product lines. 22

API Application Programming Interface. 27

client A system consuming files, such as webpages, over the internet. 22

database An organized collection of data that can be easily altered and accessed. 22

Digital Ocean A reliable cloud based server provider, `https://www.digitalocean.com/`. 22

DTO Data Transfer Object. vi, 27, 30, 60

front-end The portion of the application that the user interacts directly with, e.g., a website or mobile app's interface. 22

horizontal scaling A strategy for allowing a platform to handle more traffic by adding more servers. 22, 26

iOS iPhone Operating System. vi, 22

PayPal The largest on-line payment system provider, `https://www.paypal.com`. 27

relational A digital database whose organization is based on the relational model of data. 27

scalable A scalable system is one where performance does not suffer as the user-base grows. 27

server A computer serving files, such as webpages, over the internet. vi, 22

strongly typed language A programming language is more likely to generate an error if the argument passed to a function does not closely match the expected type. 27

Executive Summary

SoftStart was given the task of developing and implementing an application that allows users and contractors to find each other locally. Ability provided us with requirements surrounding this task, and *SoftStart* moved forwards in creating an RFP that would accurately convey the requirements for the project.

Here at *SoftStart* we strive to make the best software in the world. We're looking forward to bringing the Hire platform into reality. This report details the Technical Design that we have for the Hire application.

The report begins with a look at the user interface on the client. The interface is designed to be used with iPhones. We look at how users authenticate with the system. How the job board will look and the interface for individual Jobs. We conclude this section with a look at the Hiree/Hirer profile page.

The next section is our Use Cases. In this section *SoftStart* details our key uses of the application for both the Hirers and Hirees. We then examine some of the most common scenarios for Hire users and how those can be accomplished quickly and easily within the application.

Data Management is next and goes over how data flows within the system using sequence diagrams. This is a high level overview so it is not overly technical. Rather, it is a good starting place for a new developer to get up to speed with the system. It outlines the tasks in the use cases section as well as how the system reacts to a user initiating a offer, deleting a job and more.

The report then moves into the structure of the data outlined above. This shows how our data is represented in memory and and how it is stored in the database. In this section there are some class diagrams to help get a visual feel for the way our system is laid out. This section is a bit technical so it may be difficult to understand unless you are experienced in computer programming.

The System Interactions section will show pseudo code of the Use Cases as well as show low level logic of the systems architecture. This is a real mock up of how the system is built and we are basing the data flow on these pseudo code elements. We go through each and every one of the use cases including deleting jobs, accepting and declining offers and more. This section is highly technical and we would suggest having a strong background in computer science if you are intending to understand this portion of the document.

The testing section outlines our plans for integration testing. We will be using a suite of both integration and unit tests to build a robust platform. The section outlines the tests for registration, login, bookings, the hire map and the job interactions. It also outlines our plans for who is responsible for testing each section of the system. This is when the report gets back to being intended for the general public and this section is written to be understood by anyone who is using the application.

The final section of this report is the reporting criteria which goes over our plans for Continuous Integration through Gerrit and a discussion on how we will accomplish

this. It also gives a brief overview on how Gerrit works and why it is a invaluable tool in the *SoftStart* ecosystem.

This report will provide a clear picture of our plans for the Hire platform going forward. *SoftStart* is excited to make Hire a reality.

1 Introduction

This report details *SoftStart's* formal high level overview of the Hire Application. The document will outline our plans for the interface, uses, data management, architectural design, external interaction model, testing plan and reporting method for Hire. Throughout the document we will discuss in great detail the features, interactions, functionality and value delivered to customers.

The document begins by evaluating our market position and determining our technical requirements. Section 3 will cover the user interface that will be presented to users. This section contains detailed wire-frame mockups which will be converted into applications when Hire is built. These wire-frames include all of the intended features of hire, such as the photo-acknowledge confirmation and the Hire Map. Section 4 will outline some of the use cases that will be commonly performed in the application. These use cases will allow readers to glimpse into how users are intended to use Hire, and what kind of value can be drawn from the application as a user. The use cases include scenarios for both Hirers and Hirees, as well as detailed systems diagrams to illustrate the functionality at a technical level. Section 5 will give insight into the hardware we are building for, and the expected performance of the application. This section also covers the implementation, stack, and additional technical functionality not covered in previous sections. Section 7 covers the system interaction information, and contains pseudocode for specific tasks in the implementation. Section 8 contains detailed examples of the unit, functional, and integration testing structure. Finally, sections 9 and 10 contain information about how quality of code is assured and management of the software developers is applied to assure the utmost efficiency and quality.

As an end-to-end solution, Hire is excited to deliver an application of only the highest quality.

2 Functional Specification

This section will describe in detail Hire's intended capabilities, appearance, and interactions with users. Section 2.1 will cover the intended functionality and what services the application will provide to its users. Section 2.2 will outline how we intend to differentiate ourselves from the competing applications in the hiring field. Lastly, section 2.3 will give insight into the hardware we are building for and the expected performance of the application.

2.1 Objectives

Currently there is need for a mobile job search platform that is easy to use and approachable for users of all ages. There is evidence that such an application would provide many benefits to users, as competing products lack many essential features.

Hire aims to establish a strong community, in which users are able to feel comfortable looking for work, or hiring others. To accomplish this, we will provide many features that allow users to get to know one another before the Hire, as well as a rating/review system after the work is completed. Over time, we hope that reviews and ratings will build up and help drive the application and it's community.

Another goal of ours is to help users find the best possible options for their needs. This means that we will make suggestions based on location of users and relevance for their given skills. We want users to have the ability to quickly find work that fits their specific skills, and give employers the best fit people for their job.

We believe that everyone has skills and knowledge that they may offer, and we want to provide a platform that anyone can use. We feel that people of all ages should be able to benefit from the application, and it should be as easy to use as possible. Hire will focus on providing users with a seamless and intuitive interface that can be easily navigated by those with little to no technical experience.

2.2 Market Position

Our research has shown there are many competing applications in this market including AirTasker [2], Task Rabbit [3], and ThumbTack [4]. Because of this pre-existing competition, we are putting a large amount of effort into finding the areas our application will be competitive against the current solutions. Our application will focus on the two areas that we feel need the most improvement; monetization and task screening.

The first area is monetization, many of the currently competing applications take 10% of each sale. We feel 10% is much too large of a chunk of the transaction and will lower this just 5% of the transaction. This will help us to break into the market quickly as the people working for other task apps will see immediate increases in their profits when they are hired using our application

We will also focus on making the payment process as simple and straight forward as possible. There will be no cash involved during the Hire process and all money that is exchanged will be handled securely through our application.

The second area is the lack of proper screening of job requests. The mis-classification of job requests is often stated as the largest downfall of task based hiring applications [5].

Our team will focus on classifying incoming requests into the appropriate category so our clients and our users will know exactly what is requested before a Hire is made.

2.3 Technical Requirements

The technical requirements for Hire are as follows:

- The application(s) must uphold 99% up time for any 24-hour period
- The applications(s) must support all current mobile and platforms.
- The application(s) should utilize languages and technologies that are currently, flexible, and extensible.
- The application(s) should be closed source and of standard compliance.
- The application(s) should support over 100,000 users concurrently.
- The application(s) should support, but not require, two factor authentication.
- The application(s) should facilitate simple information changes, including password and personal information.
- The application(s) should support account recovery.
- The application(s) data should be secured via industry standard cryptographic means.
- The application(s) should support loss of Internet connection and reconnect without damaging the user's experience.
- The application(s) should contain a logging system which keeps track of users interactions.

3 User Interface

At *SoftStart* we think that the most important part of an interface is not what you put in, its what you keep out. Our interfaces are designed to be minimal and clean while allowing the user to reach their intended action in minimal time.

The following interface prototypes (mockups) showcase the core interface layouts for Hire. Additionally the following features are included in the mockups.

Hire Board A board that contains all of the jobs Hirer's have posted around you.

Hire Map A map which contains way point representations of the jobs in the Hirer Board. Clicking on a way point reveals the job and it's payout. Way points are color coded using a legend ranging from manual labour to desk work.

Hirer/Hiree Feedback System Through the rating system, Hirers and Hirees can rate each other and give feedback for future job seekers to look over when making their decision.

Social Media Authentication Authentication with Facebook, Google, and Twitter APIs.

HireChat An integrated chat service that allows Hirers and Hirees to converse seamlessly over the Internet to determine if adequate service can be provided.

Escrow Payment System When a Hirer and a Hiree have agreed on a task, the Hirer will escrow a payment to the application, which will hold funds until the task has been completed by the Hirer.

Photo-Acknowledge Confirmation When Hirees are finished their task, they take a picture of their completed activity. The Hirers will then confirm the task and payment will leave escrow.

3.1 Authentication

A user's credentials will be cached on the devices however on fist launch they will have to log in (figure 1b) or sign up (figure 2) which will be done from the splash screen (figure 1a).

(a) Splash Screen (b) Log In Form

Figure 1: Authentication Process

The Sign Up screen will offer to connect to a social media account (figure 2a) before having the user fill out a sign up form as shown in figure 2b.

(a) Social Media Sign Up (b) Sign Up Form

Figure 2: Sign Up Process

3.2 Job Board

Jobs will be listed for Hirees in the job Board tab (figure 3) where the jobs can be viewed as either a list (figure 3a) or on a map (figure 3b). Viewing a job will allow a Hiree to make an offer for how much they would be willing to complete the job for (not shown).

(a) Job Board Tab

(b) Jobs displayed on the Map

Figure 3: Job Listings

3.3 Jobs

Jobs that the user is currently interacting with, either as a Hirer or Hiree will be listed in the Jobs tab shown in figure 4a. From the Jobs tab the user can also create new jobs as shown in figure 4b.

(a) Jobs Tab (b) New Job Form

Figure 4: Jobs the User is Interacting With

Looking at an accepted job the user can see current information about the job including the agreed price, current status, and location as shown in figure 5a. From here the user can get directions to the job which will open in their selected maps application or chat with the corresponding Hirer or Hiree as shown in figure 5b. Once their done the job, they can also mark it as complete which is show in figure 6a.

(a) An Active Screen (b) Chat for a Job

Figure 5: An Active Job and Job Chat

When a user is ready to mark a job as complete they do so from the job screen. This is shown for a Hiree in figure 6a. After the job has been marked complete payment is transferred and the user can leave a review as shown in figure 6b.

(a) A Hiree Completing a Job (b) A Job Being Reviewed

Figure 6: An Job being Completed and Reviewed

3.4 Profile

The user can view and edit their Hire profile on the Profile tab as shown in figure 7a. From here they can

(a) The Profile Tab

(b) Editing a Users Profile

Figure 7: The Users Profile

4 Use Cases

As we hope to make the application beneficial and approachable for a wide range of individuals, the target age for our application are people from ages 18 - 75. Individuals seeking work are expected to be in the ages of 18-60, while those looking to hire others are expected to be in the range of 25-75. Users may be either male or female.

4.1 Environments

Hire will focus on two main environments,

At Home Either around the house or at work, wherever the user is settled.

On the Go Travelling, commuting or just out and about, wherever the user is going.

In both these environments, Hire will be easy to use and readily available for Hirers and Hirees.

4.1.1 For Hirers

When a Hirer is either at home or at work, Hire will provide accessible and contextual information. The suggestions for both jobs and Hirees will tailor themselves to the location of the user and their context. For example, at work around lunch time a suggestion might be lunch delivery with the option to drop it off with reception. Hire will remember previous contexts like where you like your coffee from.

When a Hirer is out on the go, perhaps traveling in a foreign city or commuting, Hire will provide contextual suggestions based on your location. This could be suggesting food delivery in the area or someone to grab you coffee as you drive between meetings.

4.1.2 For Hirees

When a Hiree is at home, Hire will provide situational notifications of jobs in their area. The information made available to the Hiree will be tailored based on schedule, previous experience, and interests. The Hiree will have control over these settings so they can be notified when a job appropriate for them is available.

When on the go, a Hiree will have the option to be made aware of local jobs that relate to their skill sets. Hirees could also be made aware of possible tasks when during a trip

or a commute at their discretion. For example a Hiree could be notified of a food delivery on their way home from work.

4.2 Scenarios

The following section details two example interactions between Hirer or Hirees and the Hire platform. The two scenarios are,

Hiring from home Sarah would like some help unclogging her shower drain which is full of hair.

Getting hired on the go Andy gets notified about a job to pick up a Hirer's dry cleaning on his way home from work.

4.2.1 Hiring from Home

Sarah's roommate has complained about the shower not draining and now Sarah is tasked with cleaning out the drain. Sarah tries cleaning it with a drain snare but quickly realizes she doesn't have the stomach for it.

Sarah decides to contact someone for help. She grabs the Hire app from her devices app store. Being a first time user Sarah is prompted to sign up upon opening the app. She chooses to create an account using her Facebook account and the app pulls her details from Facebook. She is then prompted to add a credit card. The interface provides either the option to fill in credit card details or to read the details off her credit card using the camera. After choosing the latter Sarah is brought to the home screen where the on boarding wizard offers to help her create her first job.

Sarah fills out the fields for job description, location (inferred by the app), offer amount, notes, and time frame. Sarah's job is called "Clean out my shower drain", she gives it a time frame of a couple hours and an offer amount of $20. In the notes Sarah mentions that she already has a drain snare.

About twenty minutes after posting, Sarah receives a notification that Joe, a Hiree, has offered to do her job but for $30. After reviewing Joe's hiring history and reviews, Sarah agrees and a hold is placed for the $30 on her credit card. Joe sends a message through Hire to Sarah asking if she has any Draino before he comes over. Sarah responds that she does and sends Joe her address.

Joe shows up a half hour later and gets started cleaning out the drain while Sarah watches TV in the other room. When Joe's done, he marks the job done on his phone and logs a picture with the app. Sarah is notified and also marks it as done which transfers

Figure 8: Responding to a posting

the funds to Joe's PayPal. Joe leaves and a little while later Sarah is prompted to fill out feedback for Joe, she gives him five stars for his prompt service and cheerful attitude.

4.2.2 Getting Hired on the Go

Andy has about an hour left at his work when he gets a notification from the Hire app. Someone nearby would like their dry cleaning picked up and delivered and it's on Andy's route home from work. The job is offering $15 and, after opening the notification, Andy accepts. He sends a message to Dan, the Hirer, saying he will drop the dry cleaning off in about an hour when he's off work. Dan responds and sends Andy the confirmation number and address to pick up the dry cleaning.

Andy gets off work and drives to the dry cleaners. He picks up the clothes and heads to Dan's place. He arrives and gives Dan the dry cleaning. On the way back to his car, Andy marks the job as done. Several minutes later he's notified that Dan did as well, and the funds are deposited into Andy's PayPal account. Andy is offered the chance to offer feedback for Dan and gives him five stars for being prompt and responsive.

A couple weeks later, once Andy has been hired by a few different Hirers, he gets a summary of his feedback in an email. He sees that he received all good feedback.

5 Data Management

Referring to figure 9, we can see a high level overflow of the application flow for Hire.

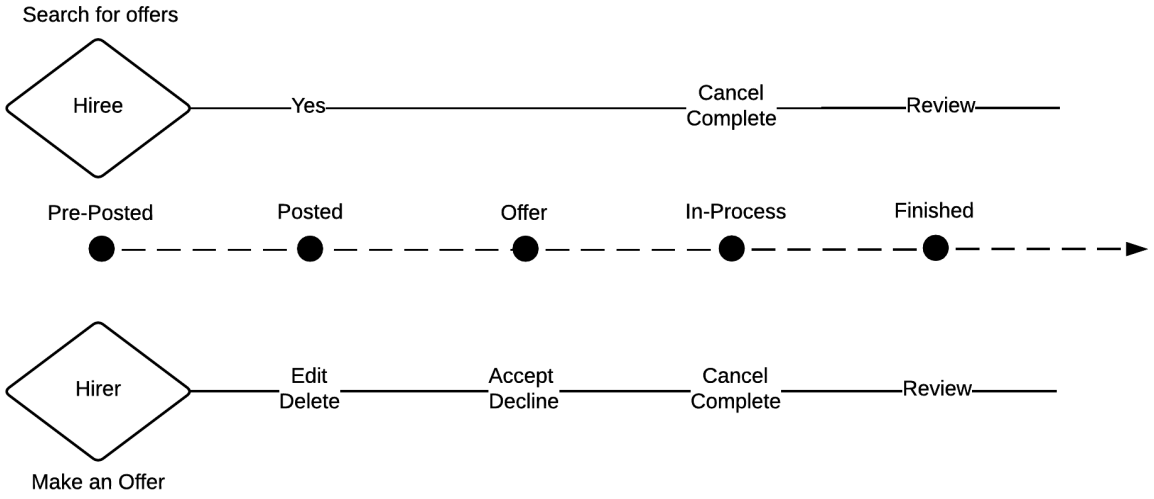

Figure 9: Application Flow

We can break out application flow into 5 distinct sections:

1. Pre-posting: The offer has yet to be posted.
2. Posted: The stage at which the job has been posted, but no offers have yet been made.
3. Offer: The stage at which the offer has been made, but the Hirer has not yet accepted or declined the offer.
4. In-Process: The job is currently in progress by the Hiree.
5. Finished: The job has been finished by the Hirer.

5.1 Implementation

Below are the implementations for our 5 distinct sections represented as UML diagrams.

5.1.1 Pre-Posted

For Hirers When a user saves a posting they will send a request to the server for validation of the job. A valid request will be sent to our database and saved. Upon a

successful save the application will send back confirmation to the client of a successful posting as shown in figure 10.

For Hirees Hirees may simply search for jobs using the job board or job map. This process is represented in figure 11.

5.1.2 Posted

The posted section indicates that a job has been posted, but an offer has not yet been accepted by the Hiree.

For Hirers Hirers may edit or delete their jobs at this point. If additional information becomes relevant for the posting, Hirers can edit the posting to reflect this new information. If the server is no longer required (an outside party completed the contract, or simply the Hirer no longer wishes the service to be completed), the Hirer can delete a job if they desire. This interaction can be viewed in 12.

For Hirees Hirees may make offers to Hirers during the posted phase. The Hiree will fill out all the relevant information about the job, including their offer, and will send it to the Hirer. This process is outlined in figure 13. If the Hirer is interested in the offer, the job will move to the Offer phase.

5.1.3 Offer

The offer section serves only to satisfy the Hirer. During the offer section, Hirers are free to observe the offers of Hirees as they come in. Once they select an offer that works for them, the job will move onto the in-process phase.

For Hirers At this point Hirers may accept or decline an offer. Accepting the offer closes the job posting, declining the offer deletes the offer using the offer ID. This interaction can be seen in the UML diagram in figure 14.

5.1.4 In-process

The in-process stage signifies that the task is currently being completed by a Hiree. During this stage users have the ability to mark the option as completed, or cancel the

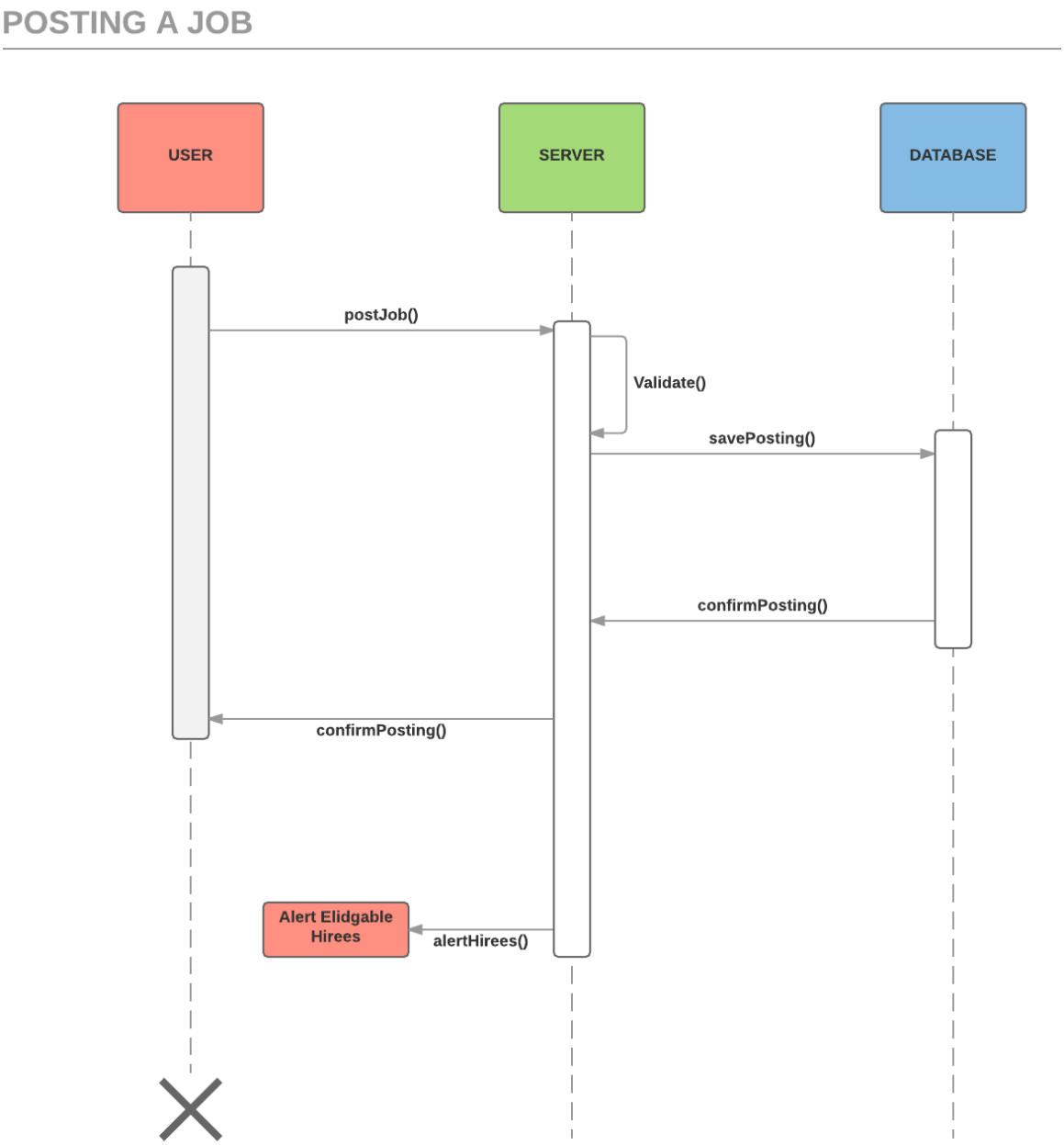

Figure 10: Saving a Posting Implementation

SEARCH FOR A JOB

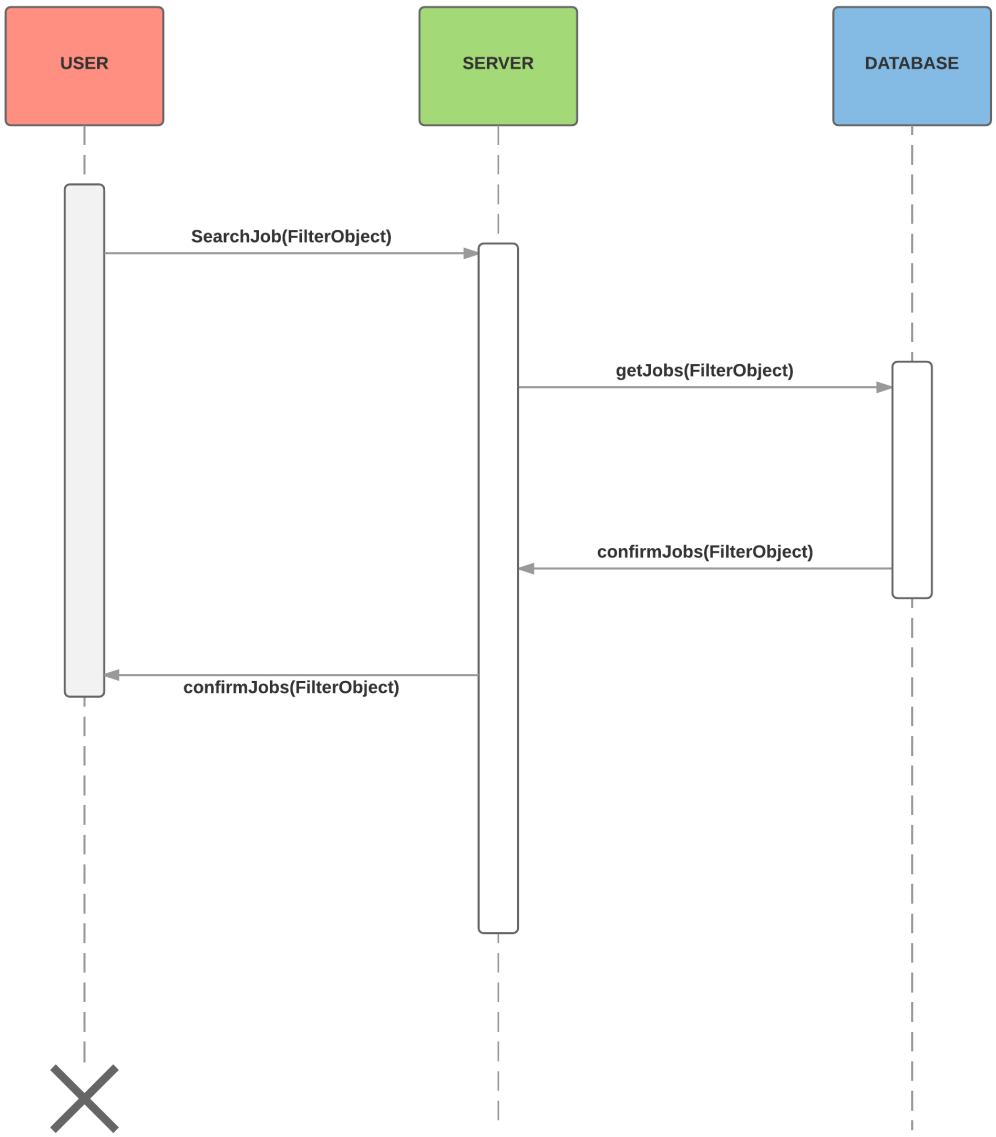

Figure 11: Searching Jobs

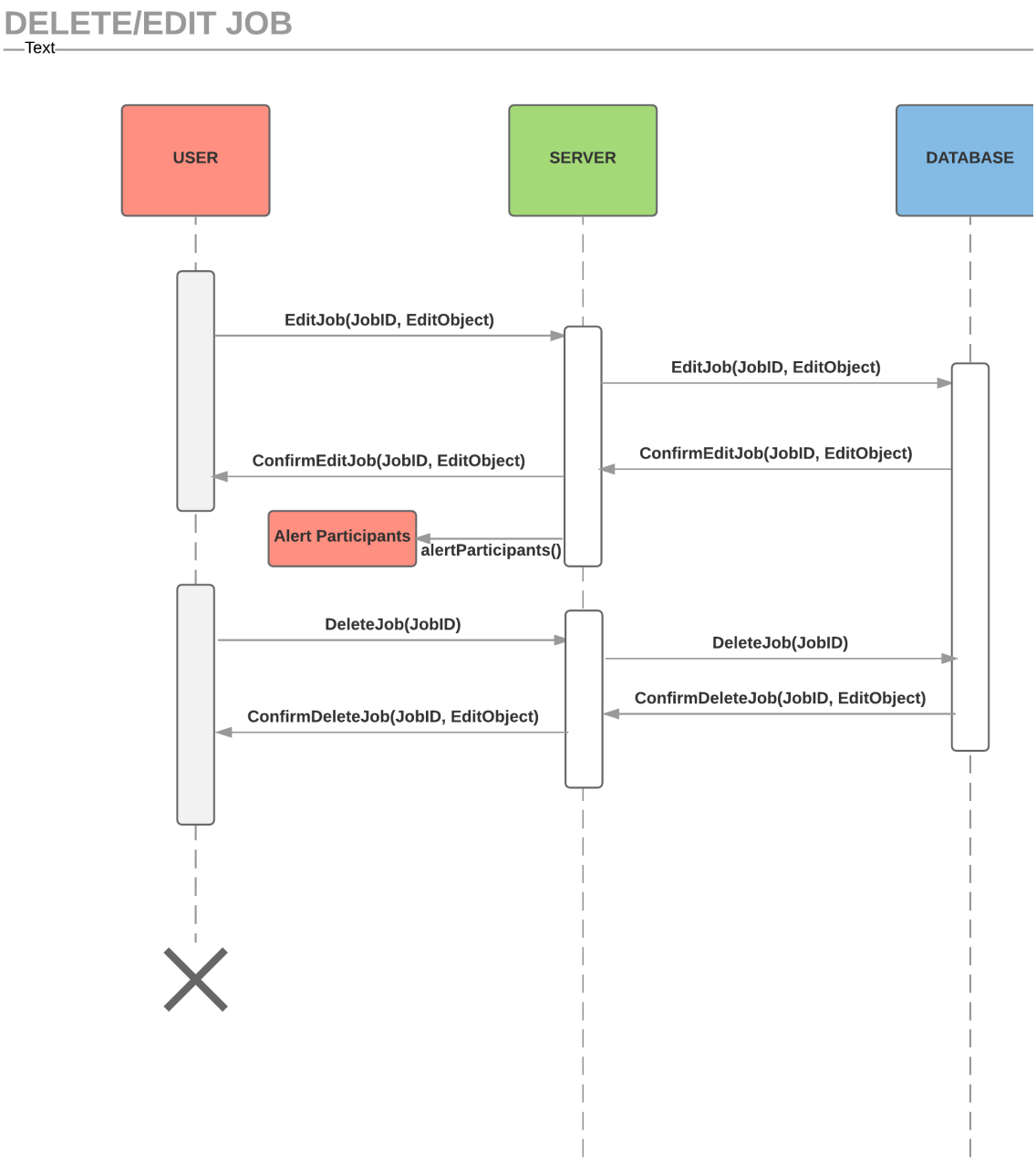

Figure 12: Edit or Delete a Job

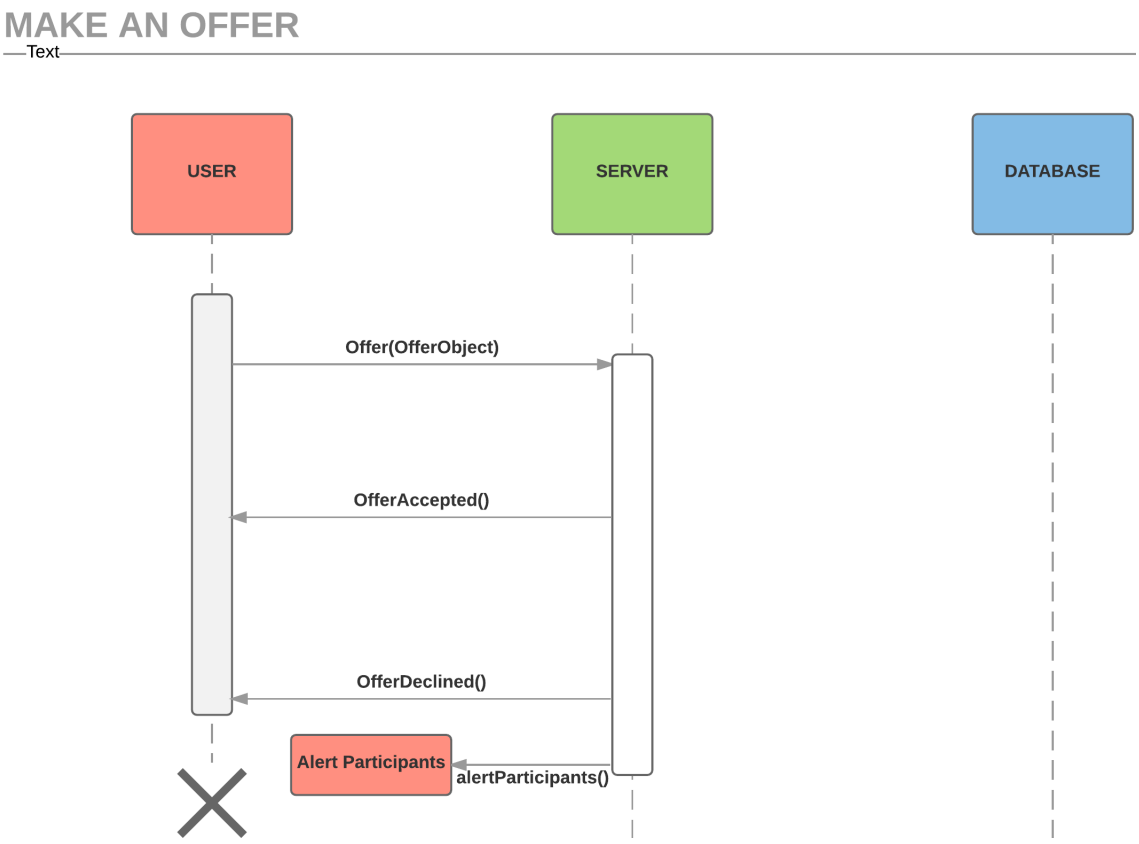

Figure 13: Make an Offer

ACCEPT/DECLINE AN OFFER

Text

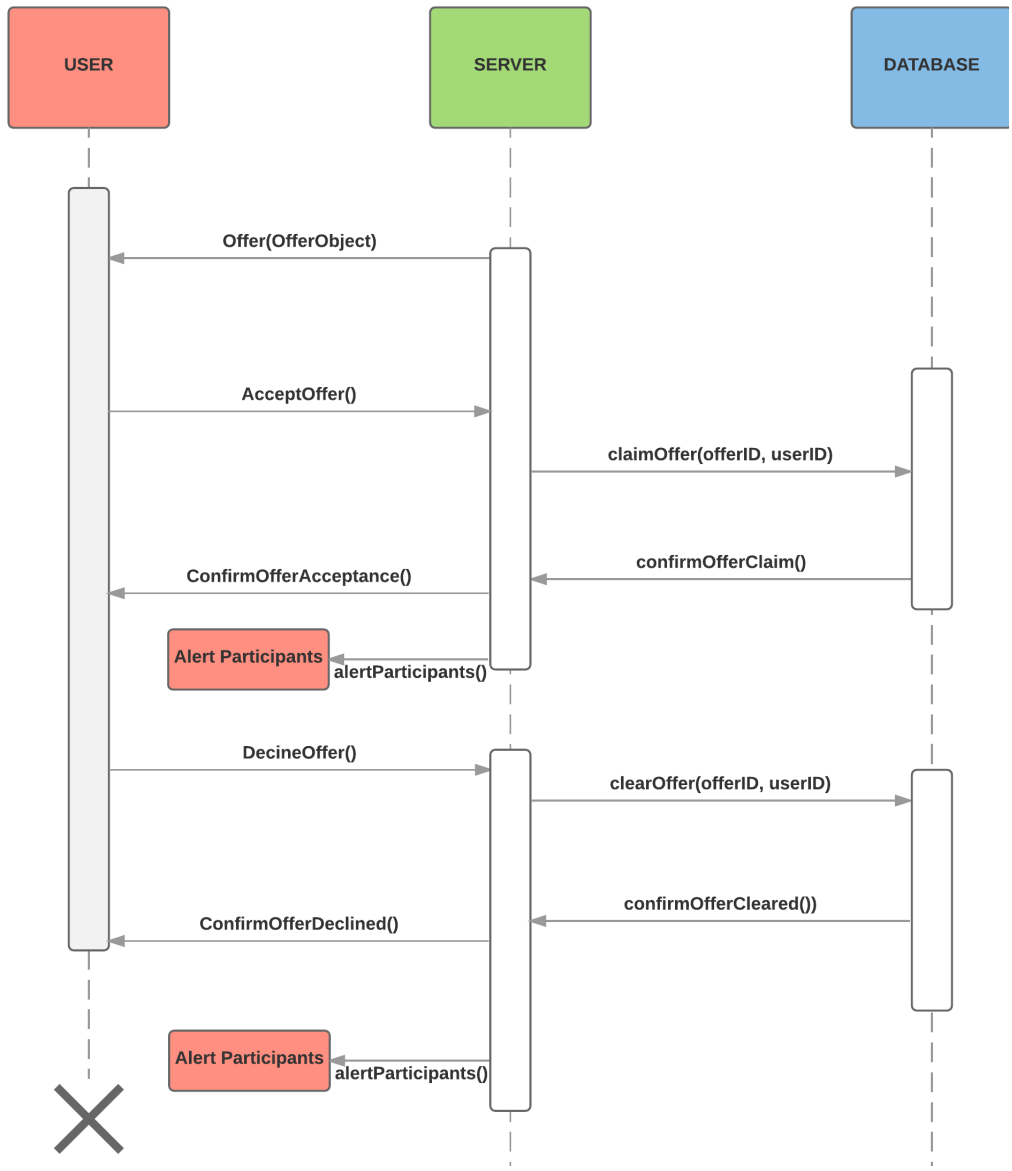

Figure 14: Accept or Decline an offer

job.

For Hirers Once the Hirer has seen the task to be complete, he will mark it as such in the Hire application. The Hirer also has the option to cancel anytime during the task for any reason. Upon cancellation, the Hirer must provide a reason for the cancellation.

Hirees Like Hirers, Hirees also have the option to cancel a job if they are not satisfied with their work environment. Upon completion, Hirees will initiate a complete event. The complete event will prompt them to take a picture of their work to verify that the task has been finished. Once both the Hirer and the Hiree have completed the task, the job will move to the Finished phase.

5.1.5 Finished

This finished section exists to wrap all loose ends of the project up. During the finished section, Hirers and Hirees will be able to rate each other. These reviews and interactions will exist as part of their user records for the remainder of the time they are on Hire.

For Hirers and Hirees Once the task is finished, the posting is transferred to a finished stage. At the finished stage, the job is considered completed by all parties involved and the review section opens up to both Hirers and Hirees. Hirers have the opportunity to post information about the quality of work completed by the Hiree, and Hirees have the ability to post about interactions with the Hirer. At the end of the review process, both Hirers and Hirees are asked to ranks each other based on their experience throughout the entire process. Once ranked, other users will be able to see this information about the Hirer/Hiree when making future decisions about who to hire.

5.2 Stack

This section will outline our technology stack and will showcase the data flow throughout the application. Our technology stack will be organized into a client server model. Connections will be initiated through a front-end (iPhone Operating System (iOS)) with all data being saved into a single database.

This server will be securely hosted by Digital Ocean and will be implemented with horizontal scaling as a top priority. This means that instead of moving our server to a more powerful machine as we get more traffic, we will simply add more servers and split the load between them.

COMPLETE A JOB

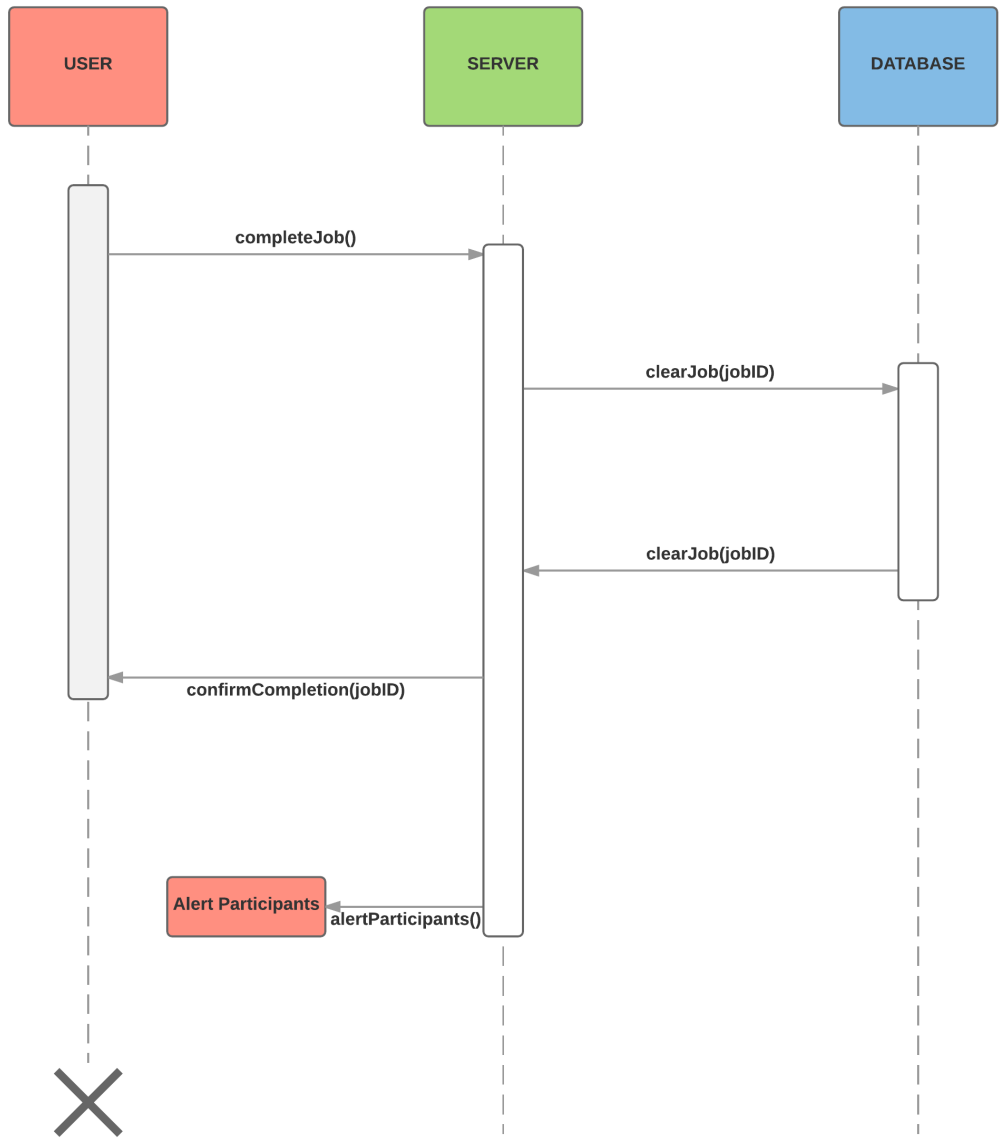

Figure 15: Complete a job

CANCEL A JOB

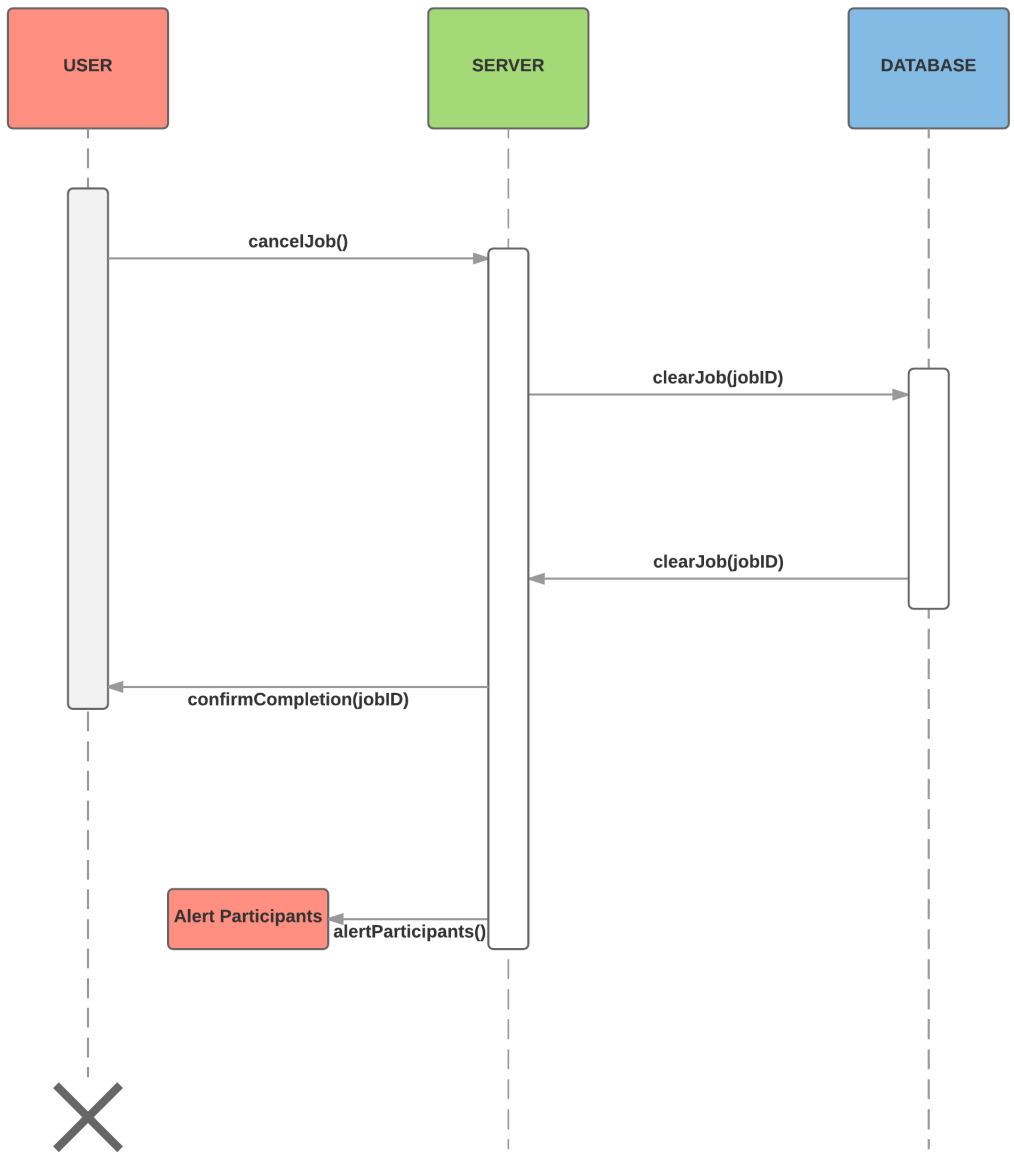

Figure 16: Cancel a job

REVIEW A USER

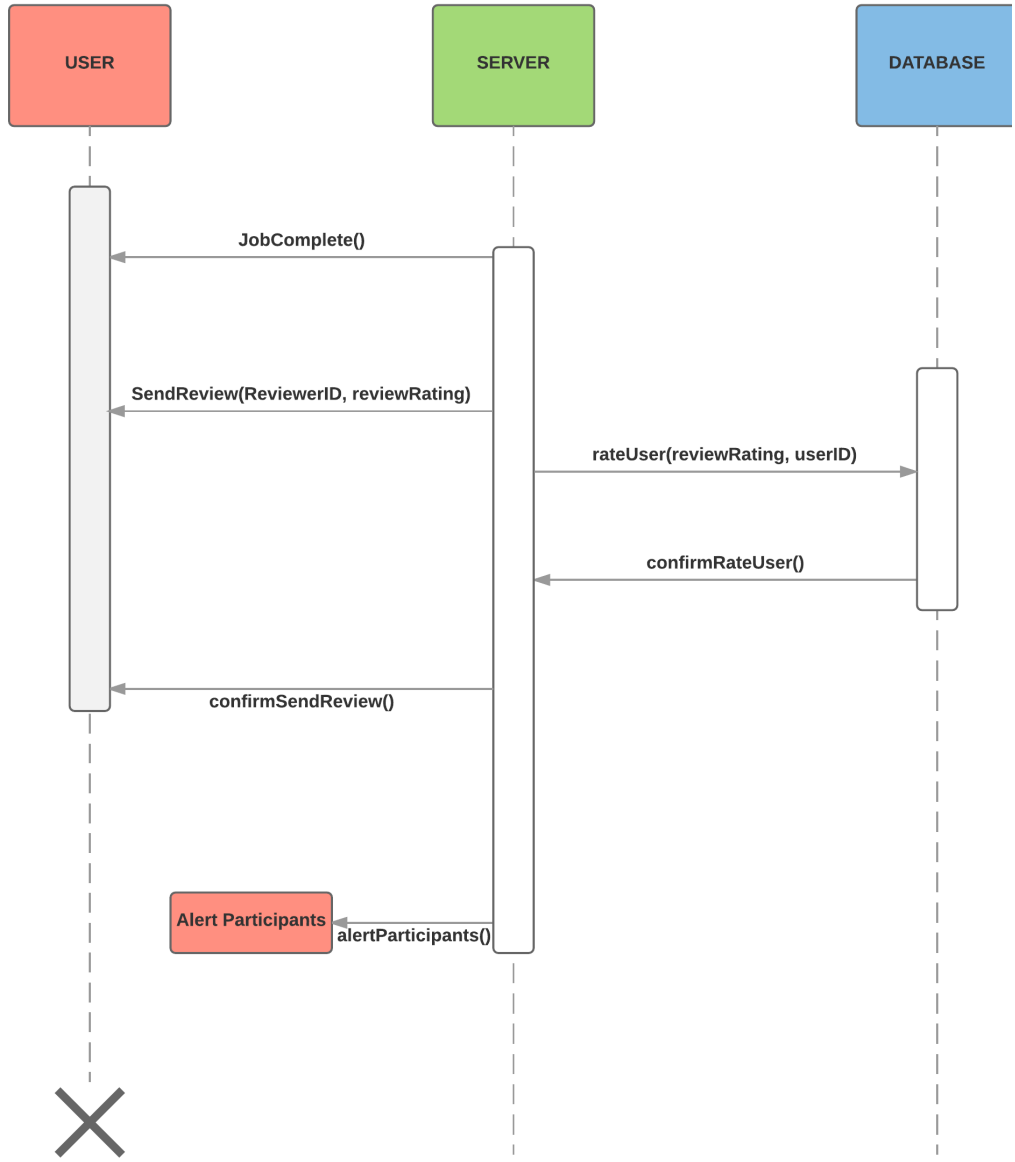

Figure 17: review a job

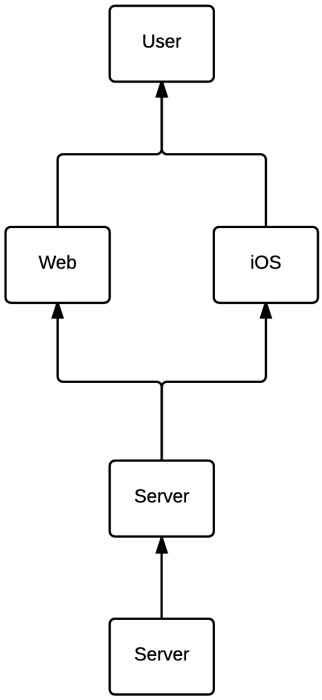

Figure 18: Hire Architecture

5.2.1 Front End

iOS Application

Mobile implementation will be done via iOS. Developing on iOS allows users to download the app from the App Store and allows the user to quickly access Hire from anywhere. This advantage makes the user far more inclined to keep using the application once they download it, as it is readily available for use. Furthermore, building an iOS applications allows Hire to push notifications directly to the users phone, notifying them instantly when users are trying to connect with them, jobs have been completed, or new jobs are posted in their area. iOS currently possess the largest market share, and the largest revenue stream for application. We feel that Hire will be a good fit into the iOS app ecosystem to due its service application structure.

5.2.2 Back End

Scala

Scala is a programming language that focuses on horizontal scaling. This language was

chosen to make the application fast, efficient and easy to scale by adding more machines. Scala is a strongly typed language which makes code organization more effortless and code more clean. Furthermore, the semi-functional nature of Scala makes the application back-end very flexible to future issues and hangups. Since Scala is built on the JVM, Hire will have access to the millions of Java libraries available to users around the world.

5.2.3 Database

CockroachDB Cockroach is a relational database which is scalable and extremely flexible in the number of servers required to run it. If one server goes down, cockroach will automatically distribute the load to another. Relational databases offer a speed that cannot be matched by non-relational databases, at a cost of more upfront work when it comes to extracting and adding information to the system.

5.2.4 ECommerce

Monetary transactions will be completed using the PayPal Application Programming Interface (API) [6]. This will ensure that monetary transactions are fast, secure and simple. This will also offset the amount of time we spend setting up commerce.

6 Class Structure of Data Transfer Objects

A hierarchical class structure is defined in figure 19. This is a outline of the data structure and not a class diagram for the overall system. These are the objects that will be passed between the client, the server and the database and will be used to outline the behaviour of the system.

Each of these objects will have a unique identifier which can easily be queried in the database. If a Data Transfer Object (DTO) states that it holds another DTO then the actual information that is being held is this unique identifier which can then be queried in the corresponding database.

The DTOs are intended to be thought of as both the representation in the database as well as the representation in memory as the code is running. Therefore, any functions or members in a DTO are accessible while the object is stored in memory and any DTO that is stored in the database only has access to the members listed within the class.

This used of DTOs allows for a consistent usage of these classes throughout the program even if we decide to switch servers, or switch databases. This will allow our team to be

agile during development and to make the codebase as versatile and simple as possible for future developers.

6.1 Person Object

The person object is an abstract class which is instantiated using any of the available subclasses. This is currently limited to Google, Twitter, Facebook and Hire.

Each of the subclasses is implemented with the ability to login, get a profile picture, get the persons name, and also to get all of their reviews. This class is abstract because the methods to do each of these tasks will vary for each of the different login systems, so instantiating the methods should be up to the developer and should be isolated for each of the logins while allowing for a standard interface regardless of whether the user is logged in with Facebook, Google, Twitter or any other login system we decide to use in the future.

Another, choice that we made was to let an administrator be a user the system with additional behaviour. This allows the administrator to view a super set of the features of the application. This will be accomplished with a check if a user is a Administrator (the isAdmin function).

When a Person is found to be an administrator they are able to run the Remove posting and ban user functions as described in figure 19. The administrator can be a Hirer or Hiree and can perform all functions that these users can perform.

Note that the connection between this class and the Jobs class is dual, meaning that a job is associated with two People and a Person is associated with a job. This is also true of the Review object, as a Person is associated with a Review and a Review is associated with multiple People. The connection between a Person and a Offer Object is one-to-many meaning a Offer has a single person associated with it but a person can have many offers.

6.2 Review Object

The Review object will hold onto the information associated with a Job. A person can have multiple reviews and each review is associated with one Job. This allows a user to have multiple reviews from the same person should they have recurring jobs with the same user. This also reduces coupling with the Jobs Class as a Job is not associated with any reviews, the review is looked up by the Job or by a Person.

By decoupling these reviews we not only save space in our Database but we also allow for the system to run queries for both jobs and for people in parallel instead of in

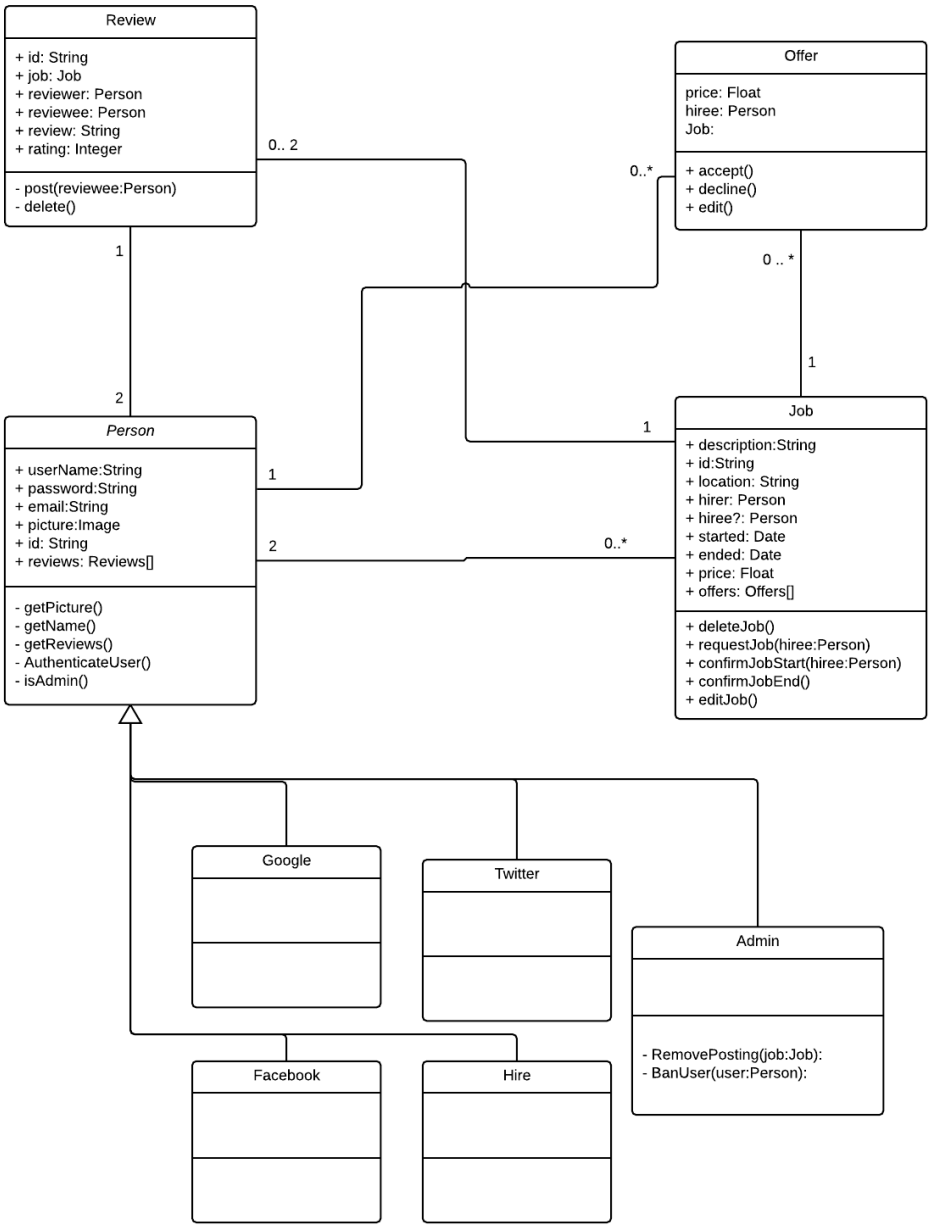

Figure 19: Class Diagram of Data

succession.

The functions of the review object will handle the creation and deletion of the object in the database and nothing more. For now editing will not be possible as this would increase the complexity of the Review object in our system. It would also require logic to check the current state of the review and to update it based on that. We will only save a single review from the Hirer and a single review from the Hiree and block any additional requests to save a Review for a job. If a edit is necessary, the current system will require them to delete the review and upload a new review.

6.3 Job Object

Jobs hold information about the the work that is being provided, the location the work is being performed at and and link to the Hirer and Hiree. The price agreed upon is also saved in the Job order and is stored with the job in the database.

They also are responsible for their own creation and deletion and for handling all requests for job confirmation, starting and ending. The information for when a job starts or ends is contained exclusively in this object to make meta data easy to collect about peak time for Jobs and duration. No date information is stored in any other class for this reason.

The Job DTO contains information about the Person class but has no reference to the review class. This was intentional to reduce coupling within the system. Although there is no reference to these in the class, each job can have a maximum of two reviews, one from the Hirer and one from the Hiree.

Jobs must contain a Hirer but do not have a Hiree until an offer is accepted. This makes the Hiree a optional field in the Class overview.

6.4 Offer Object

The Offer object is used to describe offers on a job. Offers can be accepted or declined and are initiated by the Hiree. We made the decision not to save the Hirer in the offer as this is redundant information (the job is associated with the Hirer).

There is no limit on the number of jobs that can be associated with an offer.

7 System Interactions

The Hire application is split into three main sections: The client, the server and the database. These different levels of the system interact with one another to define the behaviour of the application. This section will outline the data-flow used in the client, server and database and how how they will interact to perform tasks.

This section employs the use of pseudocode to describe the interaction between the systems. A

7.1 Responding to Job Posting

Users can request to do a job via the job page, which can be accessed through the job board. When the user requests a job, the server must process the request by first checking that the job is available, and saving the request in the database. The client sends the request to the server to be processed and waits for a response. The response it receives determines what the user will see upon requesting the job.

7.1.1 Client

```
requestJob():
    req = requestJob()
    if(isOk(req)):
        showClientConfirmation()
    else:
        showClientError()
```

7.1.2 Server and Database

The server handles whether or not the users request is accepted. It first queries the database to determine whether or not the job is currently available. If it is not available, it returns an error. If it is available, then it saves the request in the database, and returns a confirmation to the client.

```
JobAvailable(Job, Hiree):
    if(db.find(Job).isAvailable):
        SaveRequest(Job, Hiree)
    else:
        confirmRequest(err)

saveRequest(Job, Hiree):
    res = db.find(Job).set(requested, true)
```

```
    if(res):
        confirmRequest(confirmed)
    else:
        confirmRequest(err)

confirmRequest(err, confirmed):
    if(confirmed):
        hireRequest(Job, Hiree)
        confirmRequest(confirmed)
    else:
        sendError()
```

7.2 Notification of Job Request

Now that the request has been sent, the Hiree no longer has any involvement in the data transfer. They have received a confirmation of the request, and now must wait for the request to be accepted. The server must now notify the original poster of the job, the Hirer, giving them the information of the user who has requested the job.

7.2.1 Server and Database

```
hireRequest(Job, Hiree):
    sendNotification(Job, Hiree)

saveAccept(Job, Hiree):
    saved = db.find(Job).set(Hiree, Hiree)
    if(saved):
        acceptSent(Job);
        jobAccepted(Job)
    else:
        throw err()

acceptSent(Job):
    sendAcceptedNotification(Job, Hirer)

jobAccepted(Job):
    sendAcceptedNotification(Job, Hiree)
```

Jobs will be listed for Hirees in the job Board tab (figure 3), where the jobs can be viewed as either a list or on a map. Viewing a job will allow a Hiree to make an offer for how much they would be willing to complete the job for.

7.3 Posting a Job

Users may post jobs to the application by clicking the "New" button, which is located on the job board, or on the "jobs" page. The user can insert a title for the job, a description of the job, how much it pays, and specify the location and time when it is to be completed. Then they can select "Create Job", which initiates the following interactions.

7.3.1 Client

The client is very simple it does some basic validation of the job in the device, then sends the postJob request to the server. If the server validates the job then the user sees a confirmation. If there is any error the user will see a descriptive error about what went wrong. This quick validation is duplicated on the server with more tests to make sure the job that is being saved is valid.

```
postJob(Job):
    if(Job.isValid()):
        confirmed = sendJobToServer(Job)
        if(confirmed):
            showConfirmation()
        else:
            showClientError()
    else:
        showClientError()

onAlertForHiree(Job):
    showCLientNewJobs(Job)
```

7.3.2 Server and Database

The server and database validate the posting has all required fields and has correct structure. If this is not the case the server replies that the data is incorrect and the client handles this accordingly.

The Server also will send out a notification to all eligible Hirees stating that a new posting is available. Inside of this method will be another call to the database but this is not shown in the following pseudocode.

```
validate(Job):
    if(Job.isValid()):
        confimed = db.savePosting(Job)
        if(confirmed):
            sendConfirmation()
```

```
            alertHirees()
        else:
            sendClientError()
    else:
        sendClientError()
```

7.4 Search for a Job

Users can search for jobs in the Job Board page. From this page, the user can select the search button, and input their query. This action will filter out jobs that are unrelated to their search query, and list those that are relevant in the Job Board.

7.4.1 Client

The client exhibits a simple if else data flow. If the search is successful the jobs are displayed, if the search is unsuccessful the client is shown a descriptive error.

```
searchJob(FilterObject):
    search = sendSearchToServer(FilterObject)
    if(search != null):
        showJobsToClient()
    else:
        showClientError()
```

7.4.2 Server and database

The database and server communicate and filter the query into a chunk that is returned to the client. There is almost no logic in this section as the server will expect the client to handle the errors in this task. The query is returned to the client. The length of the query will be configured by the client.

In the event that a query has no results it is up to the client to show an appropriate error.

```
getJobs(FilterObject):
    search = db.query(FilterObject)
    showJobsToClient()
```

7.5 Delete/Edit Job

Hirers may edit or delete their jobs if additional information becomes relevant for the posting or if the service is no longer required. This interaction can be viewed in figure 12.

7.5.1 Client

Clients editing jobs will send the job to the server and expect a confirmation or error. Should the client be a eligible Hiree, they will receive a configurable notification window showing them the updated job offer.

If a client deletes a job, they should expect a confirmation or error.

```
editJob(Job):
    confirmed = sendUpdatedJobToServer(Job)
    if(confirmed):
        showConfirmationToClient()
    else:
        showClientError()

onAlertEdited(Job):
    showClientEdits(Job)

deleteJob(Job):
    confirmed = sendUpdatedJobToServer(Job)
    if(confirmed):
        showConfirmationToClient()
    else:
        showClientError()
```

7.5.2 Make An Offer

The offer section serves only to satisfy the Hirer. During the offer section, Hirers are free to observe the offers of Hirees as they come in. Once they select an offer that works for them, the job will move onto the in-process phase as seen in figure 13. The figure does not show the database interaction because this is intended to model the client reaction to the offer being accepted.

When a offer is accepted all Hirees that put in offers for the job are notified that the job has been taken.

7.5.3 Client

The client makes an offer on a posting then later a event is fired that their offer was either accepted or declined. All potential Hirees that were in the running for the job are notified that the job was taken. There are two notifications here, offerDeclined is to send the user a notification personally that their offer was declined and alertParticipants is used to update the potential Hirees that the job was filled by another applicant.

```
makeOffer(Job):
    confirmed = sendOfferToServer(Offer)
    if(confirmed):
        showConfirmationToClient()
    else:
        showClientError()

onAlertAccepted(Offer):
    offerAccepted(Offer)

onAlertDeclined(Offer):
    offerDeclined(Offer)

onAlertParticipants(Offer):
    updateLocalOffer(Offer)
```

7.5.4 Server and Database

The server here is simplified as this is a event received from external factors. When the server receives notification that the Hirer has chosen a Hiree it begins sending messages to all associated people with that Job. To see the full interaction proceed to the next section.

```
onOfferAccepted(Job):
    confirmed = db.find(Job)
        .confirmOfferAccepted()
    if(confirmed):
        sendSuccess()
    else:
        sendError()
        return
    sendAcceptance(Person)
    alertParticipants(db.findOffers(Job))

onOfferDeclined(Offer):
    confirmed = db.find(Job)
        .confirmOfferDenied()
    if(confirmed):
```

```
        sendSuccess()
    else:
        sendError()
        return
    sendDecline(Person)
```

7.6 Accept/Decline Offer

Hirers may accept or decline an offer. Accepting the offer closes the job posting and declining the offer deletes the offer using the offer ID. This interaction can be seen in the UML diagram in figure 14.

7.6.1 Client

```
onAlertOffer(Offer):
    if(Person.pressedAccept):
        acceptOffer(Offer)

acceptOffer(Offer):
    confirmed = sendAcceptanceToServer()
    if(confirmed):
        showClientSuccess()
    else:
        showClientError()
```

7.6.2 Server and Database

The server here is simplified as this is a event received from external factors. When the server receives notification that the Hirer has chosen a Hiree it begins sending messages to all associated people with that Job. To see the full interaction, proceed to the next section.

```
onOfferAccepted(Job):
    confirmed = db.find(Job)
        .confirmOfferAccepted()
    if(confirmed):
        sendSuccess()
    else:
        sendError()
        return
    sendAcceptance(Person)
    alertParticipants(db.findOffers(Job))
```

```
onOfferDeclined(Offer):
    confirmed = db.find(Job)
        .confirmOfferDenied()
    if(confirmed):
        sendSuccess()
    else:
        sendError()
        return
    sendDecline(Person)
```

7.7 Complete a Job

Once the Hirer has seen the task to be complete, he will mark it as such is the Hire application. The Hirer also has the option to cancel anytime during the task for any reason. Upon cancellation, the Hirer must provide a reason for the cancellation as seen in figure 15. The client will send event to the server deming the job complete the server and database will clear the job and update the clients to post a review.

7.7.1 Client

The client in this case initiates the confirmation that the job has been done, this results in the server clearing the job and marking it as passed.

```
completeJob(Job):
    confirmed = sendCompletionToServer(Job)
    if(confirmed):
        showClientSuccess()
        askClientForReview()
    else:
        showClientError()
```

7.7.2 Server and Database

The Server responds to the event from the client by clearing the offer from the database. The client is expected to handle any erroneous results. Completion is broadcast to all those involved with the job this is currently only the Hiree, but in the future there may be more users with a stake in the job and it should be coded in a way that makes it easy to add in these additional notifications.

```
onJobCompleted(Job):
    confirmed = db.find(Job)
        .confirmJobCompleted()
```

```
    if(confirmed):
        sendSuccess()
    else:
        sendError()
        return
    alertParticipants(db.findOffers(Job))
```

7.8 Cancel A Job

Like Hirers, Hirees also have the option to cancel a job if they are not satisfied with their work environment. Upon completion, Hirees will initiate a complete event. The complete event will prompt them to take a picture of their work to verify that the task has been finished. Once both the Hirer and the Hiree have completed the task, the job will move to the Finished phase as seen in figure 16.

7.8.1 Client

The client (either a Hirer or Hiree) in this case initiates that the job is cancelled, this results in the server clearing the job and marking it as cancelled.

```
cancelJob(Job):
    confirmed = sendCompletionToServer(Job)
    if(confirmed):
        showClientSuccess()
    else:
        showClientError()
```

7.8.2 Server and Database

The server responds to the event from the client by clearing the offer from the database. The client is expected to handle any erroneous results. Cancellation is broadcast to all those involved with the job (this is currently only the Hiree but in the future there may be more users with a stake in the job and it should be coded in a way that makes it easy to add in these additional notifications).

```
onJobCompleted(Job):
    confirmed = db.find(Job).confirmJobCompleted()
    if(confirmed):
        sendSuccess()
    else:
        sendError()
        return
```

```
alertParticipants(db.findPartici(Job))
```

7.9 Review A User

Once the task is finished, the posting is transferred to a finished stage. It is at this stage the job is considered completed by all parties involved and the reviewing section opens up to both Hirers and Hirees. Hirers have the opportunity to post information about the quality of work completed by the Hiree, and Hirees have the ability to post about interactions with the Hirer. At the end of the review process, both Hirers and Hirees are asked to ranks each other based on their experience throughout the entire process. Once ranked, other users will be able to see this information about the Hirer/Hiree when making future decisions about who to hire. This behaviour is seen in figure 13

7.9.1 Client

The client (either a Hirer or Hiree) in this case has received the review event and has written the review locally on their phone.

```
completeJob(Job):
    confirmed = sendCompletionToServer(Job)
    if(confirmed):
        showClientSuccess()
        askClientForReview(Job)
    else:
        showClientError()

askClientForReview(Job):
    confirmed = sendreviewToServer(Job)
    if(confirmed):
        showClientSuccess()
    else:
        showClientError()
```

7.9.2 Server and Database

The takes the review posts it in the appropriate places then alerts participants (this is currently only the Hiree but in the future there may be more users with a stake in the job and it should be coded in a way that makes it easy to add in these additional notifications).

```
onJobCompleted(Job):
    confirmed = db.find(Job)
```

```
    .confirmJobCompleted()
if(confirmed):
    sendSuccess()
else:
    sendError()
    return
alertParticipants(db.findParticipants(Job))
```

8 Testing

Any well defined software system must be rigorously tested in order to be declared complete. Hire is no exception to this rule. The following section is a detailed description of test procedures and validation methods for the Hire application. Tests are performed in order to ensure that the program produces the expected outputs in all use cases.

Unit tests are designed to test the smallest unit of functionality. Unit tests will be included to test our smallest testable units, and run frequently to ensure that no minor changes affect our application negatively. We will be using a source control system, along with continuous integration to maintain our code-base. Changes will be pushed to the source control system frequently, and unit tests will be run at that time. Unit tests will be extensive to ensure high code coverage.

Integration tests will also be used in our test environment. These integration tests will be run when the code is pushed to a release branch, to verify that our modules and classes interact in the expected way. Integration tests will not be run as frequently as the unit tests, as they take longer to complete.

Finally, Functional tests will be used to assess the entire system's functionality.

Testing will be carried out by the members of the Hire team. Andrei will be responsible for integration testing. Jake will be in charge of writing extensive unit tests, and implementing new unit tests, as features evolve and new functionality is added. Ben and Jonah will be responsible for managing the system tests and automation test suite. Hire will also use the SCRUM methodology to ensure that the team remains well informed on all areas of the system.

8.1 Unit Tests

8.1.1 Registration

Module Overview The registration system provides a means for the user to create an account and access all of the jobs on the Hire application. The registration can be done through a Hire account, or using one of our supported API logins.

Inputs Hire Registration Form: Username, Password, Email, Name Supported API Registration: API Registration Key

Positive Test Cases

Action The user creates an account using the Hire Registration form.

Result A success message is displayed and the user is prompted to confirm their account via their email.

Action The user creates an account using a supported API registration system.

Result A success message is displayed and the user is prompted to confirm their account via their email.

Action The user confirms their account via email.

Result A success message is displayed and the user is moved to the Hire Home page.

Negative Test Cases

Action The user tries to register with an email that is already in use.

Result This results in an error message saying they the email they are trying to register already belongs to an account. Additionally, the user will be asked if they would like to reset the password for their email.

Action The user tries to register with an email that does not contain a routing address (e.g @gmail.com)

Result The request will not be sent to the server. The user will receive a message stating that their email is invalid.

Action The user fails to register through one of the supported API registration systems.

Result This results in an error message saying that the external API registration has failed. The user will be asked if they want to make a Hire account only if a connection can be made to Hire's authorization servers.

8.1.2 Login

Module Overview The login system provides a means for the user to access their account and all of the jobs available on Hire.

Inputs Username and Password

Positive Test Cases

Action The user logs in using their correct Hire account credentials.

Result A success message is displayed and the user is moved onto the Hire home page.

Action The user logs in using their API system login.

Result A success message is displayed and the user is moved onto the Hire home page.

Negative Test Cases

Action The user tries to login with the incorrect credentials once.

Result This results in an error message saying they have failed their login.

Action The user tries to login with the incorrect credentials three times or more.

Result The user is prompted to reset their password.

Action The user enters nothing in the username and/or password box.

Result The request is not processed, and an error message stating they must fill out all fields is displayed.

Action The user tries to enter non-ASCII characters into the text-box.

Result The request is not processed, and an error message stating they must use ASCII characters is shown.

8.1.3 Profile

Module Overview The profile system allows the user to access all of the information pertaining to their profile.

Inputs Username, Password, Email, Name, API Access.

Positive Test Cases

Action The user updates their profile by adding another API access point to their area.

Result The user can now sign in with that API login system. On the backend, the API login key is now bound to that user.

Action The user updates their email, phone number, etc on the profile page.

Result These changes are saved to the database and the user can see them the next time they navigate to the page.

Action The tries to add a valid credit card to their account.

Result The credit card is validated using Luhn 10 and a request is made to verify that the credit card can accept charges.

Negative Test Cases

Action The user tries to modify their email to an invalid email.

Result An error is displayed saying that the email is invalid and the user must change the email to a valid email.

Action The user's credit card expires.

Result If this is the only user's credit card, they will be prompted to add a new credit card or they will be unable to post jobs through Hire. If the user has additional credit cards, nothing will occur.

8.1.4 Hire Board

Module Overview The Hire board allows users to view jobs in a list. It also allows them to sort through this list and view jobs that they might be interested in.

Inputs Search queries.

Positive Test Cases

Action The user searches for a job using the text search with no invalid characters

Result The displayed jobs match the search query, either via the title, or some content of the ad.

Action The user taps on an item in the jobs list.

Result The posting should expand to give the user a better idea of what they job entails. The expanded section should show price, contact information, a contact name, and the description of the task. Pictures are optional.

Negative Test Cases

Action The user searches with nothing in the search bar.

Result A popup appears stating that the user has not filled out the search bar and must do so if they wish to refine their job search.

Action The user searches for a job with a query that returns no results.

Result The jobs list should state that the query returned 0 results.

Action The user performs a search for a job but has no access to the Internet.

Result The jobs list should state that a connection to the Hire servers was not made and that the user should retry when they have a good connection.

8.1.5 Hire Map

Module Overview The Hire map allows users to see jobs in their area placed on a map. From here, users can select a job by clicking on one of the waypoints which are color coded based on the type of job.

Inputs N/A

Positive Test Cases

Action The user clicks on waypoint.

Result Information about the job is displayed in a similar manner to when users tap on an item in the jobs list section. The formatting may be different, but the content should still be the same. That is, contact information, description, etc should all be present.

Action The user performs a pinch, drag, or any other feature that is present when using something like Google maps.

Result The map should move, zoom, and do any action just as the user would expect. There should be no surprises when it comes to interaction with the map.

Negative Test Cases

Action The user taps somewhere on the map that does not contain a waypoint.

Result Nothing should happen.

8.1.6 Job Interactions

Module Overview Job Booking denotes the section of interactions that occur when Hirers and Hirees are ready to exchange services. Money transfer, confirmation, and a variety of other actions are covered within this section.

Inputs N/A

Positive Test Cases

Action The Hiree sends the Hirer a request and the Hirer accepts the request.

Result The Hiree should receive a notification that the request has been accepted. The job should now show up under their "Current Jobs" list. Additionally, the Hirer should be charged and the payment sent to the Hire Escrow Service.

Negative Test Cases

Action The Hiree sends the Hirer a request and the Hirer declines the request.

Result The Hiree should receive a notification that the request has been declined. The job should now show up under their "Rejected Jobs" list. The Hirer should NOT be charged for this interaction.

8.1.7 Feedback System

Module Overview The feedback system allows Hirers and Hirees to rate each other out of five and optionally provided a detailed comment to justify their rating.

Inputs Rating out of 5 and optional text justifying this rating.

Positive Test Cases

Action The Hirer leaves a rating for a Hiree.

Result This rating should be visible on their profile. For the Hirer, it should be under "My Sent Reviews" and for the Hiree it should be under "My Received Reviews". The review should indicate whether the person was the Hirer or the Hiree.

Action The Hiree leaves a rating for a Hirer.

Result This rating should be visible on their profile. For the Hirer, it should be under "My Received Reviews" and for the Hiree it should be under "My Sent Reviews". The review should indicate whether the person was the Hirer or the Hiree.

Negative Test Cases

Action A Hirer/Hiree tries to add a rating less than 0 or greater than 5.

Result Hire should prompt the user that this is not allowed and that the user must select an integer value in the range of 0 and 5.

Action A Hirer/Hiree tries to add a rating of 3.5.

Result Hire should prompt the user that this is not allowed and that the user must select an integer value in the range of 0 and 5.

8.1.8 HireChat

Module Overview The HireChat allows Hirers and Hirees to communicate on Hire quickly and efficiently.

Inputs Text which is converted to chat messages. This text may also contain unicode emojis.

Positive Test Cases

Action User A sends a valid message to user B.

Result User B should receive the message, as well as a time stamp indicating when user A sent the message.

Action User A receives a message, but does not respond to User B.

Result User B should see a read receipt, indicating that user A has seen the message.

Negative Test Cases

Action A user tries to send an empty message.

Result The user should be unable to send an empty message, as the send key will not be enabled.

8.1.9 Logging out

Module Overview Logging out allows the user to remove their account from the device.

Inputs A button press indicating the user would like to logout.

Positive Test Cases

Action A user presses the logout button.

Result They are asked if they are sure they would like to logout.

Action The user confirms the logout message.

Result The user is logged out and moved back to the login screen.

Negative Test Cases

Action The user presses the logout button, but decides they are not ready to logout. They then press "No" on the logout confirmation.

Result The user is not logged out.

8.2 Integration Tests

The following integration tests are split into their respective classes. Each method is tested using the following procedures. Integration tests are used to verify interaction between different modules.

8.2.1 Review

Post() is used to post a review. It involves both the Hirer reviewing, the Hiree being reviewed, and the review itself. Upon posting a new review, the review should be visible on the Hiree's page who is being reviewed.

input: A new review, and two users
output: review is created and visible on Person's page

Delete() is used to remove an existing post. The user who originally wrote the post is able to delete the post. This involves a Person, and a review.

input: Existing user review, original poster of review
output: Review is removed from system and not visible on Person's page

8.2.2 Offer

Accept() and **Decline()** are available to a Hirer that originally posted a job. This option becomes available after a request for a job has been sent. The original poster then has the option of either accepting the job request, or declining the request.

accept():
input: Available Job
output: Job is now active

decline():
input: Available Job
output: Job remains available

edit() is used to edit offers. Offers can also be edited by either Person. It is a form of agreement, that determines when the job starts and ends, they pay for that job. It

involves the Hiree, the Hirer, and job offer.

inputs: Available Job, a Hirer and a Hiree
output: Job is updated

8.2.3 Person

AuthenticateUser() is used to give a Person access to the application.

AuthenticateUser():
The following Integration Tests apply only to admins.

RemovePosting() is used by an Admin to remove a job posting that may not comply with terms of use for the Hire application.

Input: Job
Output: Job is removed

BanUser() is another method available to Admins. This method is used to ban a user from the application.

Input: User
output: User is banned from the application

8.2.4 Job

deleteJob() is used to delete a Job from the Job Board. It involves the original poster of a Job, and the Job itself

inputs: Job, Hirer
output: Job is removed from application

RequestJob() is used by a a Person who is interested in a job. They request a job, which triggers a notification for the original poster.

inputs: Job, Hirer
output: Job request is saved, and Hirer is notified

ConfirmJobStart() and **ConfirmJobEnd()** are used to confirm the job start and end times. They are determined via the Job Offer.

confirmJobStart()
inputs: Offer, Job
output: Job start time is set

confirmJobEnd()
inputs:Offer, Job
output: job end time is set

EditJob() allows the job to be edited by the original poster of the job.

inputs: Job, Hirer

output: Job is updated

8.3 Functional Testing

Functional Testing will be done in preparation for releases to assess the overall functionality of the application. It is a type of testing where we assess the outer layer of the application, and how a user may interact with it.

Functional testing is useful in regression testing and can be executed via automated tests, as well as manual testing. Automated tests will be implemented using the Selenium iOS framework. Selenium is a tool that is able to access the interface and simulate user interaction.

Automated tests are useful because we can ensure that the tests remain consistent between releases. Therefore we can pinpoint any changes in functionality that may have been introduced to the program during the current iteration, and fix them or edit the tests before release. The tests must cover an adequate range of use cases to ensure high coverage.

While automated testing is very useful, it is limited to specific use cases, and can often miss edge cases. Manual testing will also be done to ensure that we do not miss potential bugs.

Functional tests will be executed by the Quality Engineering team at Hire. The team will consist of full-time coop students. They will be responsible for running automated testing, as well as manual testing.

9 Reporting Criteria

Our reporting and code evaluation will all be run thorough Gerrit. Below is our plan for code review and Continuous integration using Gerrit

9.1 Gerrit Code Review

Gerrit is a tool that helps track changes to software and makes it easy to see changes to the code. This allows developers to easily track and manage their colleaguesâĂŹ work and to suggest edits before the code is pushed to customers.

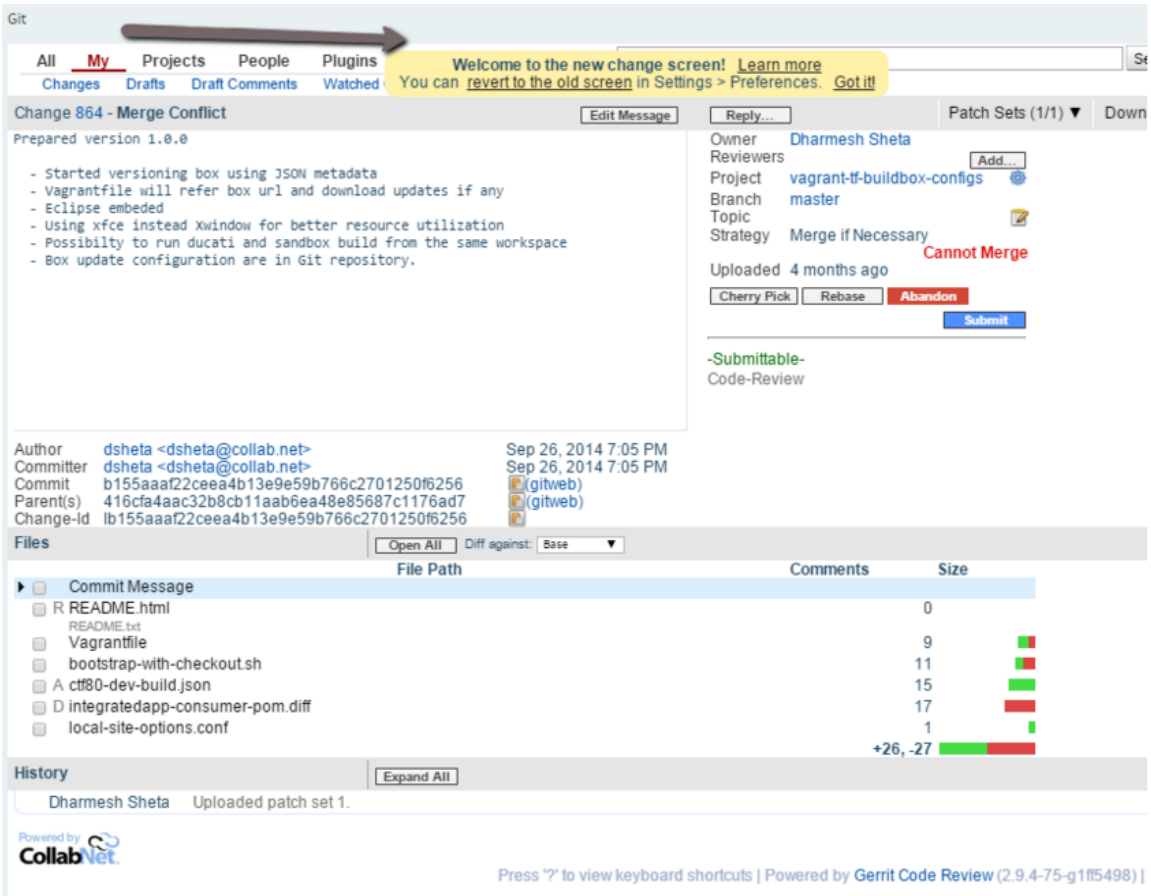

Figure 20: Gerrit's Interface

A code review is an examination of the proposed changes to a piece of software by one's peer.

Code Review often encompases the term Continuous Integration which is the process of building, running and pushing code to production. Typically the code review process is complemented by running the tests before someone reviews the code. This is because

the act of reading someone elseâĂŹs code is mentally exhausting, and time-consuming. A developer does not want to review code that does not pass the tests that the developer has written, or code that that does not compile. Gerrit helps with this process.

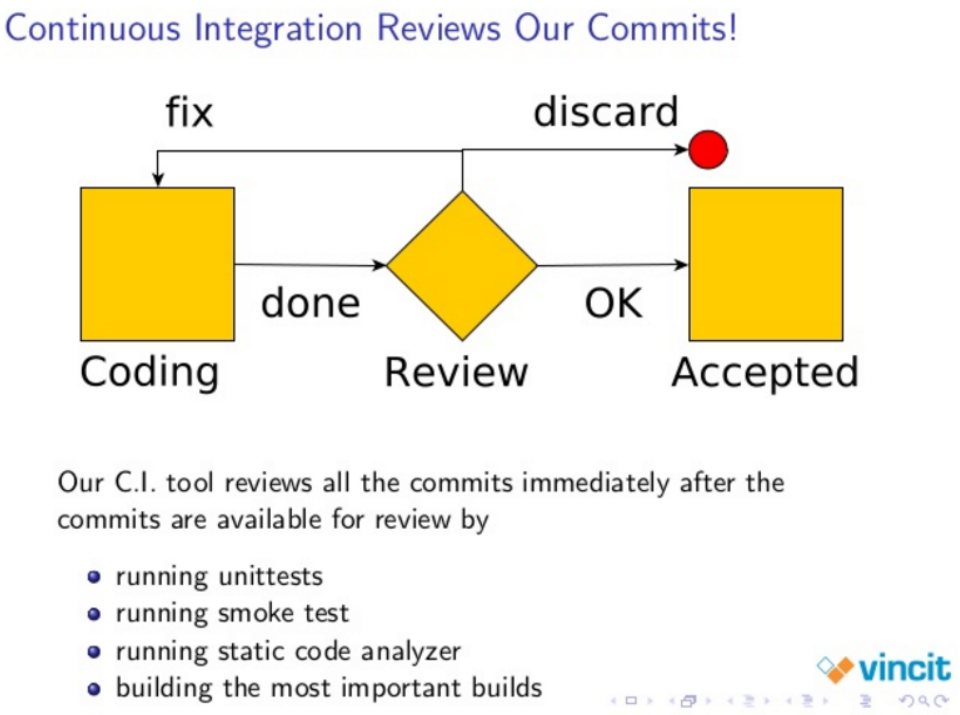

Figure 21: Gerrit's Code Review Process

Figure 21 shows the typical process. A developer will write code, and when they are finished they will ask for a review. The developer will send the code to gerrit, then gerrit initiates the Continuous Integration process before another developer is asked to read the code.

Continuous Integration Tools, such as teamcity, (See Section 1.3) are automated programs that build and run the code that is up for change. This helps to confirm that the developer did not make a human error and accidentally break something in the codebase.

Gerrit also allows you to open a dialogue on the code in question and allows a central place for the discussion. Figure 22 shows an inline comment on a piece of code this is the typical method of communication within Gerrit.

This comment is now a permanent part of the history of the project and when future developers look at this piece of code they are able to see the conversation between the developers and see their reasoning for allowing this bit of code in the repository.

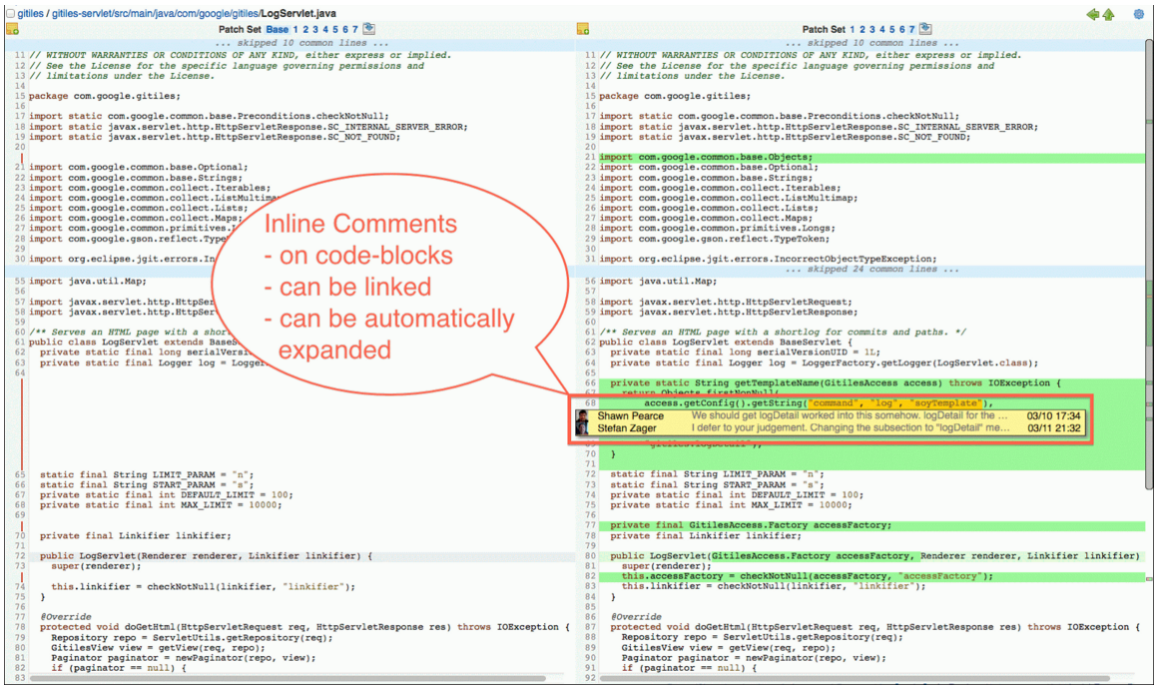

Figure 22: Inline Comments In Gerrit

9.2 Continuous Integration

Continuous Integration (CI) is the process of merging code automatically into a central repository several times a day. When changes are sent to the server, the CI tool will build, compile and run the code, run the tests then pass the information back to the team in a easy-to-read format that is displayed in Gerrit. If the tests do not pass or the code does not build or your tests do not have enough code coverage then the CI tool will warn you that this code should not be merged into the central repository.

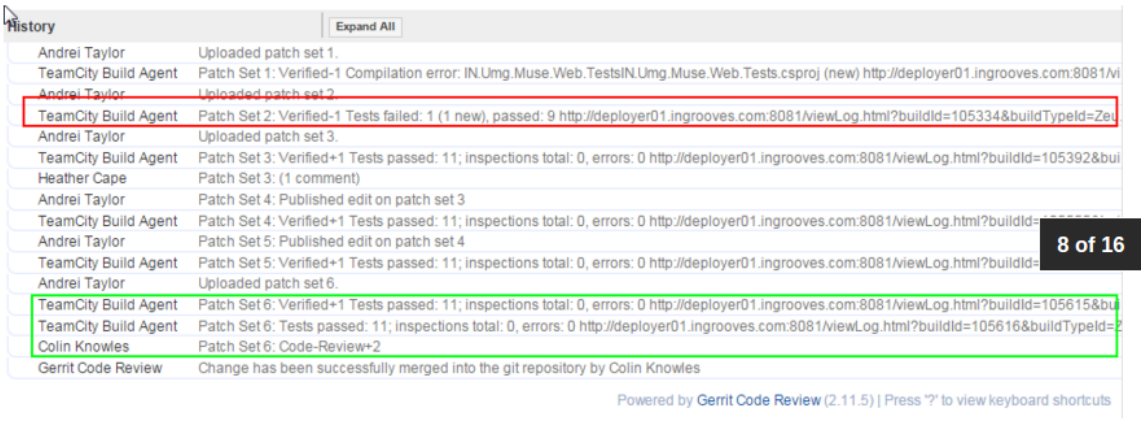

Figure 23: History in Gerrit

Figure 23 shows a typical history in a Gerrit Project. A user (in this case Andrei) will upload a Patch Set, which is a series of changes to the code. By uploading these to the CI

tool, Andrei is requesting that they be merged into the central repository. The CI tool, before any other humans interact with the code, will build the code and run the tests.

This first step signals either a failure, denoted in Gerrit by a -1 code review (in red), or a +1 (in green) which signals that all the tests have passed and that the code is ready to be reviewed. Once this signal is sent, a reviewer has access to the code and can read over the change for errors and coding style. They can make comments and suggest changes and also stop the code from being merged into the main codebase.

This CI step saves a lot of time as reviewers do not spend their time reviewing code that does not work and they can rely on the tools to perform this step for them.

9.3 Test Driven Development

Our developers will efficiently create this system using test driven development and the best available tools. Test driven development is an agile, test-first approach to development, where tests are written in advance, and the code is written later.

The general workflow of TDD is, red, green, refactor, as shown in figure 24 below. A new test is written before implementation of new functionality. This test should fail initially. The code is then updated and refactored until the new test passes. Then the code should be refactored and cleaned up further, and all the tests should still pass. The idea of test driven development is to be constantly improving the system, while maintaining high test coverage. TDD has the added bonus of keeping the code streamlined and clean.

10 Management Plan

This plan is the formal document for SoftStart's implementation of testing the application. This represents many hours of planning and evaluating our options to find the optimal solution to the problem. We believe that our plan for testing outlines a industry leading approach to testing and evaluating software.

The following section discusses how the team will be able to implement the proposed system in the given time frame, and some of the key techniques and tools that will be used by the team.

10.1 Gerrit

Our Gerrit instance will allow us to track developer contributions to the project and also to run our integration and unit tests quickly. The History feature in Gerrit will allow

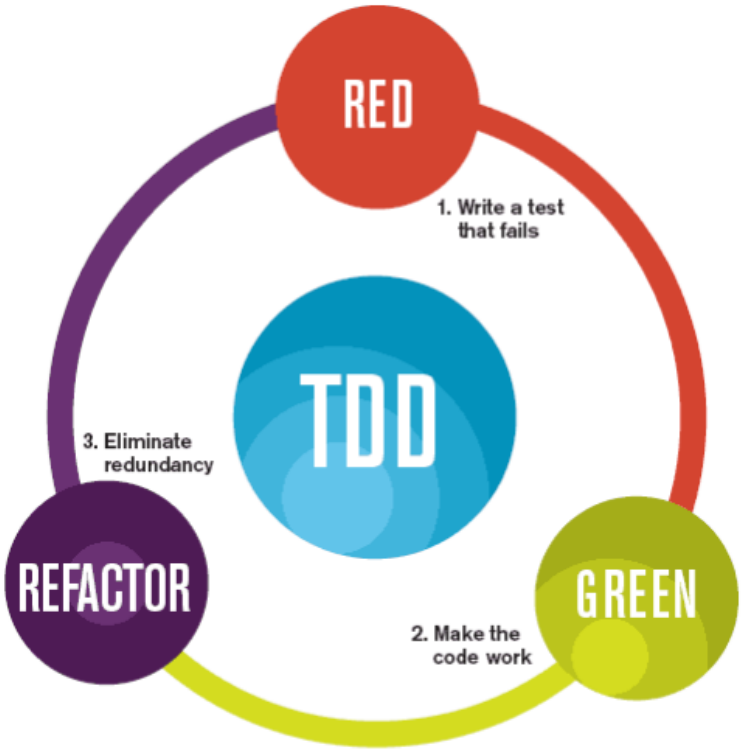

Figure 24: TDD Workflow [1]

developers to update the repository and revert the inevitable bugs that are introduced into the system. This will also be useful in determining the amount of work that each task required, and how much time the developers have put into their tasks. We will use these analytics to aid us throughout the course of development.

10.2 SCRUM

Scrum is an agile software development methodology. It will involve daily team meetings, which will be used to discuss features, functionality, problems that have come up and also assess non compliance of team members.

10.3 Scheduling and Employees

The Hire team hopes to complete development of the application by the end of April. Given the short time frame, we have assigned roles and tasks to each member of the team. Table 1 describes the employee's roles and tasks that they will be responsible for during development.

Name	Role	Tasks
Andrei Taylor	Back-End Developer	Back-End development, Databases, Server Code, Integration Tests
Ben Hawker	Designer	Front end UI, functional test suite and automation
Jake Cooper	Front-End Developer	Front-End Development, interaction with server, unit tests
Jonah Boretsky	Marketing	Management, documentation, functional test suite and automation
Co-op Students	Functional Testing	Automated and manual testing

Table 1: Developers Roles and Assignments

10.3.1 Project Schedule

The project's estimated completion date is April 4th. Table 2 displays expected completion dates for various milestones in the development life cycle.

Task	Expected Completion
Conceptual Design Document	March 4
Presentation to Customer	March 8
Technical Design Document	March 18
User Manual	March 22
Project Demonstration	March 29
Release	April 4

Table 2: Expected Completion Dates

11 Summary

Hire allows users to post jobs and find workers in their area. Through the jobs board, Hirees can make offers to Hirers about their task. After agreeing on a price, the Hirees will complete this task. Upon photo validation, funds will be released to the Hiree.

Hire utilizes an iOS application for the front end and a Scala service for the backend. Cockroach DB is used as the database to make sure the system is stable and fault-tolerant.

This document outlines our plans for dataflow within the system and shows the DTOs and logic of our implementation.

We will be implementing a plethora of integration and unit tests which are run by Gerrits Code Review System to make a reliable and scaleable application.

SoftStart is excited to make Hire a reality.

12 References

[1] “Tdd diagram,” Website, http://blog.simpleit.us/wp-content/uploads/2013/06/tdd.gif.

[2] “Airtasker,” Website, https://www.airtasker.com/.

[3] “Task rabbit,” Website, https://www.taskrabbit.com.

[4] “Thumbtack,” Website, https://www.thumbtack.com/.

[5] “Thumbtack was a great idea and it sucks,” Website, http://basecamppro.com/2014/05/book-publishing/thumbtack-was-a-great-idea-and-it-sucks/.

[6] “Paypal rest api,” Website, https://developer.paypal.com/docs/api/.