

CALIFORNIA STATE UNIVERSITY, NORTHRIDGE

Web Application with Mobile Client for the  
Short-term Restaurant Job Marketplace

A graduate project submitted in the partial fulfillment of the requirements  
for the degree of Master of Science in Computer Science

By

Sri Krishna Chaitanya Kotapati

May 2017

The graduate project of Sri Krishna Chaitanya Kotapati is approved:

---

Prof. Li Liu

---

Date

---

Prof. Robert Mcilhenny

---

Date

---

Prof. Richard Covington, Chair

---

Date

## Acknowledgements

I would like to acknowledge and thank my thesis committee chair Dr. Richard Covington for his continued guidance throughout the semester, his experience and knowledge that he shared with me over the period along with his patient temperament that helped me in reaching the level of sophistication my thesis has.

I would also like to thank Dr. Robert Mcilhenny and Dr. Li Liu for their support and agreeing to be a part of my thesis committee. A special thanks goes to my graduate coordinator Dr. Ani Nahapetian who was there to resolve any questions concerning to the thesis process and kept me up to date with the deadlines and gave information related to any helpful workshops.

## Table of Contents

Signature Page .....	ii
Acknowledgements .....	iii
List of Figures .....	vi
List of Tables .....	vii
Abstract .....	viii
Chapter 1: Introduction .....	1
Chapter 2: Background/Trade Study .....	3
2.1 Comparison to similar apps .....	7
2.2 Requirements Capture .....	7
Chapter 3: System Design .....	9
3.1 Overview of Features .....	9
3.2 Selection of Framework and Technologies .....	11
3.2.1 Architectural Pattern .....	12
3.2.2 System Backend .....	12
3.2.3 System Frontend (UI) .....	13
Chapter 4: Implementation .....	14
4.1 Development approach .....	14
4.1.1 Waterfall vs Agile .....	14
4.1.2 Continuous Development and Integration .....	15
4.1.3 Deployment .....	16
4.2 Technologies and Framework .....	16
4.2.1 Framework .....	16

4.2.2 Languages and Tools .....	18
4.3 User interface .....	25
4.3.1 Login Screen.....	25
4.3.2 Forgot Password Screen .....	26
4.3.3 Sign up Screens .....	27
4.3.4 Manager Screens.....	29
4.3.5 Worker Screens.....	33
4.3.6 iOS vs Android .....	35
4.3.7 Notifications .....	36
4.4 Source Code .....	38
4.4.1 Encryption .....	38
4.4.2 Notification Handling .....	38
4.4.3 Maps .....	40
4.4.4 Camera/Display Picture .....	42
4.4.5 REST API Calls .....	43
4.4.6 MongoDB .....	45
4.5 Unit Testing.....	47
4.6 User Testing .....	48
Chapter 5: Conclusion.....	49
5.1 Summary .....	49
5.2 Development Experience .....	49
5.3 Challenges and Future Scope .....	49
References .....	52

## List of Figures

Figure 1: Application Flow .....	10
Figure 2: System Architecture .....	11
Figure 3: MVC Framework for the System .....	17
Figure 4: MongoDB Structure .....	21
Figure 5: Login Screen.....	25
Figure 6: Forgot Password and Change Password Screen.....	26
Figure 7: Sign-up Screen .....	28
Figure 8: Manager Screen (Tab 1) .....	29
Figure 9: Manager Screen (Tab 2) .....	30
Figure 10: Manager Screen (Tab 3.1) .....	31
Figure 11: Manager Screen (Tab 3.2) .....	31
Figure 12: Manager Screen (Tab 4) .....	32
Figure 13: Manager Screen (Tab 5) .....	32
Figure 14: Worker Screen (Tab 1) .....	33
Figure 15: Worker Screen (Tab 2) .....	33
Figure 16: Worker Screen (Tab 3) .....	34
Figure 17: Worker Screen (Tab 4) .....	35
Figure 18: iOS vs Android.....	36
Figure 19: Notifications when app is closed.....	37
Figure 20: Notifications as popups, when the app is open.....	37
Figure 21: Postman App (Used for testing REST API Calls).....	47

## List of Tables

Table 1: Volume of Customers in a Week (Average).....	4
---	---

## Abstract

### Help Wanted Application for Mobile

By

Sri Krishna Chaitanya Kotapati

Master of Science in Computer Science

In this project, I analyze the mobile technologies and develop a mobile client which act as a bridge between the restaurant managers or food chains and its job seekers and provide an ecosystem to collaborate, specifically for the domain of temporary short-term informal employment. I use Cordova/Ionic Framework for building the mobile client, Node.js/Express.js for the backend and MongoDB for the database. The server will be hosted on AWS (Amazon Web Services) instance. Some of the main features of the mobile client in this project are, sending out real-time notifications to jobseekers in the neighborhood of the job posting, managers will be able to accept or reject via Tinder like interface i.e., swipe right to accept or swipe left to reject the job applicants. Both managers and job seekers will be able to view the other on a map for easier access.



## **Chapter 1: Introduction**

In today's world, restaurants and food chains hire a limited number of workers (like waiters, dishwashers, bartenders etc.); managers of these chains rely completely on the worker's goodwill to show up daily for work. What if and when a worker misses work for any reason (for example: Sick or Car breakdown), what choices do the managers have to fill this gap. Some of them may be:

- Ask other waiters to call their friends and family
- Run through their Rolodex of contacts and call them one-by-one
- Or pay over-time to the workers who showed up

All of the above options are time consuming or work overload that impacts their business adversely, the reputation of the business is at stake.

The problem here is that there are people looking for a temporary job, may be part time or may be full time but are not able to find the suitable jobs. The app "Uber" is a proof of such problem. One of the main reasons for its success is that the people have the option to work as an Uber driver whenever they want. This does not require any particular skill set or any qualifications other than the knowledge to drive. This idea of people willing to work in their free time can also be applied to the restaurant business. One does not require any qualifications to work in a restaurant and if any skills are required, they are very easy to learn (like in a couple of hours). But there is no system or "app" in place to address this problem by bringing the people looking for jobs and the managers of food chains onto a same platform.

In this day and age where technology is revolutionizing/disrupting the normal industry processes, the above analysis shows the need for a system which can make the interaction between employers and job seekers much easier.

The objective of this project is to develop a “Help Wanted” mobile client, by using the available technologies to solve this gap and/or need in the food-chain industry. The client will be more in the Job Search Category of apps rather than a Recommendation category as this will not recommend people to the managers instead it’ll only connect everyone who applies to a job. This mobile client is for the managers to post any available jobs that they require to be filled and for people looking for hourly/daily jobs in the restaurant business. Connecting these two sides, providing them with instant notifications about the change in status of their posting will be its main features.

## **Chapter 2: Background/Trade Study**

### **Case Study 1: Ironwood Tavern**

During the early stages of this project, I had the opportunity to talk with Mr. Sam Avadhi an experienced manager in the restaurant business. Mr. Avadhi was part owner and the General Manager for a restaurant named Ironwood Tavern based in Leesburg, Virginia. for 3 years before it was acquired. In our discussion, I had asked him about the staffing model, (i.e. staff onboarding, retention, etc), customer demographics, customer volume, etc. that he directly managed and other things about the business.

From our discussion, below are the steps involved in hiring a staff member:

1. Posting the vacancy using a “For Hire” or "Help Wanted" sign outside the restaurant and also in the “Craigslist” with his cell phone as the primary contact number.
2. Interested candidates would contact him then they would set up a meeting/ interview.
3. Interview process would review their
  - a. Job application for work authorization and other legal aspects
  - b. Previous work experience (if any)
  - c. The reason for leaving previous work
  - d. Their overall lifestyle, their demeanor (approachable, pleasant to talk etc.)
4. Qualified candidate would then go through their standard Human Resources and legal process. This process involves a background check for criminal records,

Employment Eligibility (using form I-9) and collect their W-4 (for Federal Tax deductions).

5. The new hire will then go through a series of training and tests. For example, a job like bartender or waiter would be trained for 3 days and then there would be a “Menu Test” to make sure they know the items in the menu. While a concierge job would be trained in greeting and scheduling customers.

Ironwood Tavern had a seating capacity for 120 guests/customers. The customers’ volume varies depending on the time of the day, the day of the week and events on that day like game or valentine day etc. On an average the number of customers on each day of the week during lunch and dinner were as follows:

DAY OF THE WEEK	LUNCH	DINNER
MONDAY	40	60
TUESDAY	40	60
WEDNESDAY	40	60
THURSDAY	60	100
FRIDAY	80	200
SATURDAY	200	250
SUNDAY	80	100

*Table 1 Volume of Customers in a Week (Average)*

As a General Manager Mr. Avadhi job was to maintain necessary staffing levels for better customer service. For example: Each waiter/server can support up to 6 tables with best service, any more tables would affect the service levels. On a regular Saturday, the

restaurant had up to 60-70 concurrent guests for lunch, So Mr. Avadhi planned to support 15-18 tables which means he needed 3 waiters to support Saturday Lunch. Mr. Avadhi would look at the roster of scheduled staff members and would decide to post a job (if required). He used similar formulas to staff other positions like dishwashers, cooks, bartenders etc.

When asked about the case where one or more staff members don't show up. He said its generally the staff's responsibility to replace their shift with someone who is not working on that day. So generally, staff tend to exchange their shifts with each other if needed. But in case the one who is not able to show up has no one to cover their shifts, it becomes a burden on manager to find someone or he himself must do that work. He said there were people who wanted to work for one day or two days (temporarily) whom he would call if needed. Other option was to call a Temporary Help Agency which would send out people to the restaurants as required. These kind of workers as they were a temporary replacement would skip all the recruitment process and were directly put to work with very little or no training.

#### Case Study 2: Arbor Grill

My personal experience of working at "Arbor Grill" at CSUN, I was called for an interview after I applied for the job, the interview was general questions like have you worked before, do you have experience of handling cash etc. I had never worked before implying I had no experience at all. Still, I was hired and went through the other legal formalities of filling some forms. Training was only for a day, I was shown how things are done, where the equipment is and in no time, I was working as a normal employee

would. The process of hiring was the same as described by Mr. Avadhi. Also, the problem of staff not showing up and that being a burden on the manager did exist at the place I worked too. I was called in multiple times during the week even when I had no shift, to replace someone else's shift because the other employee called in sick. The concept of temporary employee would not work in the case of Arbor Grill as it comes under a bigger corporation, "The University Corporation" (TUC) and to take in a temporary employee they would have to go through the complete hiring process which can be time taking. But if say there is a need for employee at Arbor Grill and no one else working at Arbor Grill can show up, the manager can call someone else from a different store say "Freudian Sip" (a café at CSUN) because the employees at Freudian Sip are the employees of TUC.

This is not the case in restaurant business. Generally, the restaurants are owned and run by the same management. Hence they have the flexibility to hire and fire whenever required as there are no contracts signed in this business. The hiring-firing process keeps on going around the year. This is where an app would be very helpful to both the manager and the jobseeker. An app that can connect the jobseeker and the manager whenever there is a requirement of a job. As this is for a temporary job or a short-term job, it is up to the manager to either go through the whole hiring process or just directly get the jobseeker into work. Apps or websites like "ZipRecruiter" or "Craigslist" do this but they only allow users to post or search for jobs. They do not inform the users if and when a job is available. The managers have no option to view the job applicants to accept or reject, they have to call for an interview and then decide.

The app proposed in this project will address this problem, will notify the users if a job is posted or if someone applied to the job and give a choice to the manager to accept or reject the applicant.

## **2.1 Comparison to similar apps**

A website like craigslist <sup>[1]</sup> is one example. It is a very simple site where anyone can post anything, like buy/sell something or post advertisements for services or looking for someone who can help them, etc. It is a website for everything. As a jobseeker one has to search for a specific job to find if it is available or not. This must be done every time one is looking for a job or something else. Same goes on with the manager side or the one posting a job. After posting, one has to wait for someone to see it and reply to it then meet him/her in person to decide to go ahead or not. This is a complicated process.

Apps like “Job search by ZipRecruiter <sup>[2]</sup>” takes in the jobseekers’ details like their resume, cover letters, phone numbers, email addresses etc. and then bombards them with unrelated advertisements. The jobseeker is sent a notification only if the app finds that the details match the requirement.

## **2.2 Requirements Capture**

When Mr Avadhi was told about the idea of the app he was very excited about it and said it would help them very much, and would make their work easier. When asked about any specific requirements he wanted the app to do, he wanted to view profile photos and other details of the jobseeker before he could hire them. He also wanted to control the distance the notification is sent out to. He wanted something like a

rating/review system so it can help them out in deciding to hire or not. He was even ready to pay a fee, which would definitely guarantee them a help. All this was from the side of the recruiter or the one looking for help.

From the perspective of the jobseeker, I met with the people who were already working in those joints and asked them if I make an app to help them out with their job searching what features would they like to see. Most of them wanted to be notified whenever a job is available, instead of them searching through all the related and unrelated ones. They also wanted the traditional search and apply option to be there. These were the main features required and only then I started with the app and taking these as its “User Stories”.

The ultimate goal was to make job searching easier and more convenient at least for the Blue-Collar Jobs or for the Food chain managers and its job seekers.



## **Chapter 3: System Design**

The main goal of the proposed approach is to design a mobile client that provides an easier and faster way for the manager to post jobs which then instantly get blasted to all the registered jobseekers within 'x' (chosen by the manager) miles using job seeker's pre-defined location. Upon receipt of the real-time notification, the job seeker can quickly review the job details and apply for the job using the app.

The Manager then gets notified as jobseekers are applying for the job in real-time. He/she can quickly review the profile of the Jobseeker and can accept or decline the job applicants.

### **3.1 Overview of Features**

The app shall have two versions, one for the manager and the other for job seekers. The 'manager' version shall collect the details about the restaurant like its name and address, along with manager's profile. The 'jobseeker' version shall collect their profile details like name, age and a photo.

A manager looking to hire, posts the job of a Waiter providing job details like number of hours, pay, etc. and publishes it. Upon publishing, the app broadcasts a notification to all the job seekers who are within the radius of reach (given by the manager). The job seeker upon receipt of the notification clicks on it to open the app, review the job posting and apply if he/she are interested.

The manager shall now receive notifications/alerts when an application is received. He/she shall be provided with a card view of all the jobseekers, who then can review the

applicant's profile in a Tinder like view (i.e. to accept / reject). Once a candidate is chosen/rejected the app shall send notification to the job seeker about the decision.

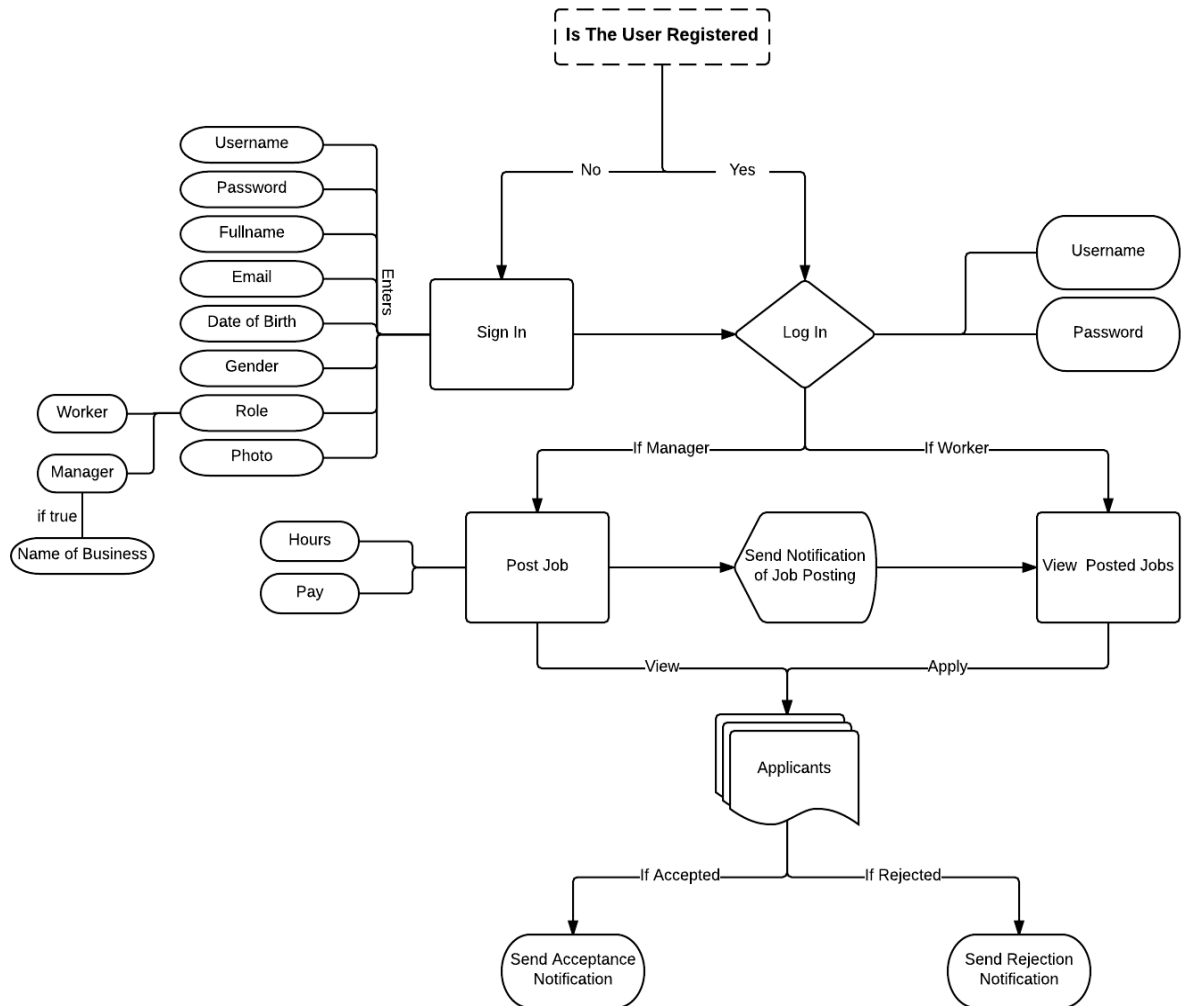
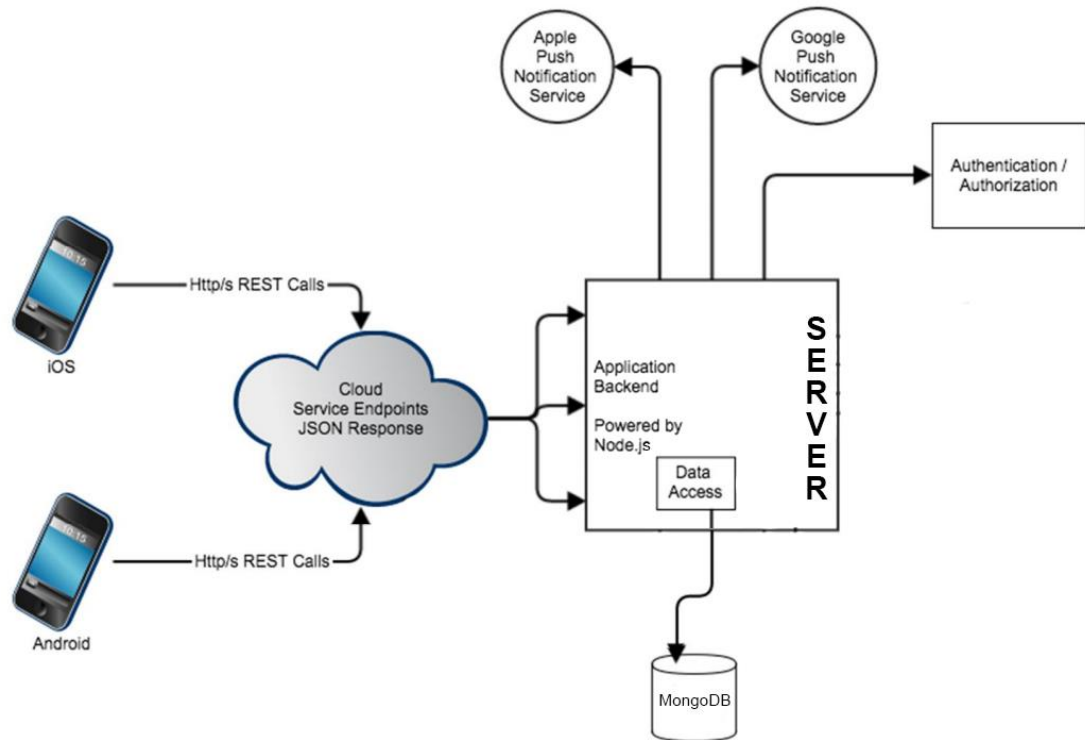


Figure 1 Application Flow

### 3.2 Selection of Architecture, Framework and Technologies

The architecture for most mobile clients is more or less the same. The below figure shows how a common mobile client's system architecture is designed.



*Figure 2 System Architecture*

First there is the actual mobile client (Android or iOS) which communicates to the server (using HTTP/ REST Api call in this case). The server manages what kind of data to receive and what type of data to send back to the client. The server is either connected to or has the database in it. The database is where all data coming from the client and data that is required by the client is stored. The server also has features like authentication (which client is authorized for the data and which is not), also is connected to the push notification services provided by the different mobile operating systems. This way server will be able to send data to the clients without the client asking for it.

### 3.2.1 Architectural Pattern

It is important to adopt a modular approach for the foundation of the system which has such a wide range of characteristics and services. Taking in a modular approach contributes to maintaining the dependency between each and every element to its minimum which can help while adding, changing or removing the system features with slight or no consequence on the remaining features. Moreover, it is also important to have modularity not just among different system components, features and/or services, but also among those individual components, features and/or services. A single feature consists of visual components through which the end user interaction, connection with the database, performing CRUD (Create, Read, Update, Delete) operations and backend processing on the data from the database and on the input from the user. It is important to maintain a clear distinction between these components, which help in a better maintainability of the system as a small change would not consume a lot of development time.

### 3.2.2 System Backend

Once the framework to be used for developing the system is set in place the next step is to select a backend language for the system that will not only handle the business logic of the system but will also communicate with the database. It is better to opt for a stable release of a secure, robust and open source language which has been in use for a while. Apart from these qualities having a large and active community for the language is also a factor to consider so that the roadblocks faced during the development process can be tackled down without wasting a lot of time in figuring out the problem.

### 3.2.3 System Frontend (UI)

The frontend of the system plays a major role in setting the look and feel of the system and the selection of the frontend technologies is very critical in getting that "first impression" from the users. Since this system is a mobile based app, it must be accessible from any device with an active internet connection. Thus, it is best to opt for an existing framework which can adopt to the device of the end-user with minimal effort and a framework which has an active community should be preferred over others so that one can take advantage from the other co-users of the framework.

## **Chapter 4: Implementation**

### **4.1 Development approach**

#### **4.1.1 Waterfall vs Agile**

Waterfall is a linear approach to software development. In this methodology, the events are done in the following sequence:

- Gather and document requirements
- Design
- Code and unit test
- Perform system testing
- Perform user acceptance testing (UAT)
- Fix any issues
- Deliver the finished product

In a Waterfall development project, each of these represents a distinct stage of software development, and each stage is finished before the next one can begin. This approach has a major drawback, if any issue rises in mid-way of the development cycle, the complete process must be restarted. Hence there is a possibility of not finishing the project by the deadline.

On the other hand, Agile is an iterative approach to development. This approach emphasizes the rapid delivery of an application in complete functional components. Rather than creating tasks and schedules, all time is “time-boxed” into phases called “sprints.” Each sprint has a defined duration (usually in weeks) with a running list of

deliverables, planned at the start of the sprint. Deliverables are prioritized by the customer's needs. If all planned work for the sprint cannot be completed, work is reprioritized and the information is used for future sprint planning.

When the project was started just the high level understanding of the business need was known, and later time was spent on high-level needs or requirements during the requirements phase of each sprint. The Agile development approach was followed to build the product, using this incremental development approach, this helped to fine tune as the product was built and tested.

The development plan that was created was broken down into sprints and each sprint was addressing a piece of the functionality. Only after each sprint was developed and tested, it was moved to the next phase where it was integrated with earlier sprints thereby following continuous development methodologies. This Agile approach helped me in:

1. Planning the available time by not spending majority of time upfront defining the detailed requirements
2. Minimizing code rewrites and finalizing the requirements at the same time
3. Helped in testing small independent modules as compared to testing the entire product at one time

#### 4.1.2 Continuous Development and Integration

Dividing the project execution into smaller sprints helped to focus on small functionality while the basic building blocks of the entire project were built.

This process of building small blocks or micro-apps and integrating them with others and deploying them provided quicker turnaround time to resolve bugs as there was only the need to focus on the new code rather than earlier tested code.

#### 4.1.3 Deployment

The system is deployed on a cloud infrastructure because of its scalable, robust, secure and customizable environment. The EC2 (Elastic Compute Cloud) offered by AWS (Amazon Web Services) <sup>[3]</sup> running an instance of Microsoft Windows Server 2012 is used for the deployment of the backend and DB. To maintain the consistency between the development environment and the deployment environment, Windows Server was used instead of Linux. The port 80 was opened on the instance for the app to communicate with the Express framework.

## 4.2 Technologies and Framework

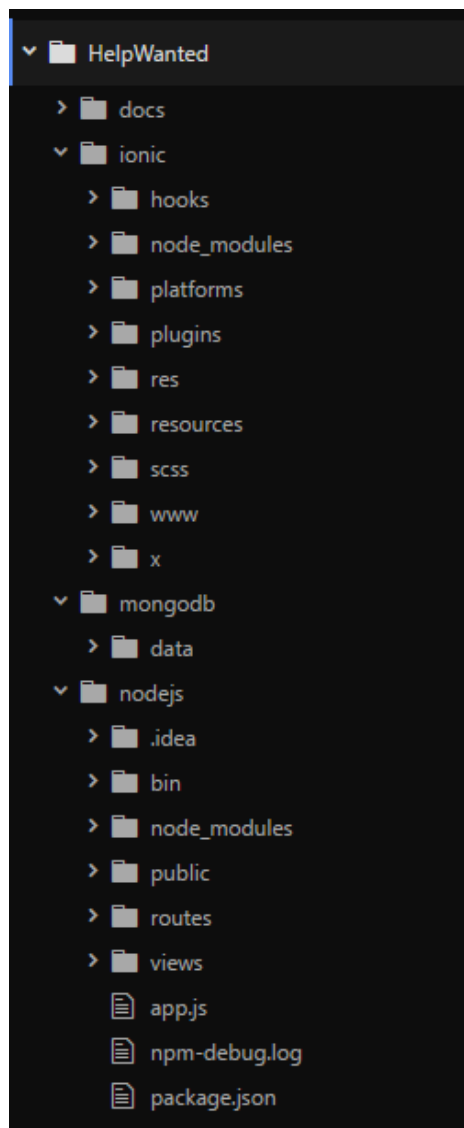
#### 4.2.1 Framework

The realization of the system architecture as explained in the previous chapter is done following the popular MVC (Model-View-Controller) architectural pattern. The main reason to follow the MVC framework is the simplicity with which it can be easily adopted to any existing or a new system. Moreover, the benefits reaped from this framework are far greater than the effort required to implement this framework. The most basic MVC structure is where the system is divided into three partitions, the View (V) which contains the visual aspects of the system along with the system interface, the Model (M) which contains the business logic of the system and the connection with the



database, the Controller (C) which serves as a connection between the View and the Model. However, it is up to the developer to add sub partitions or add new partitions.

Thus, the modularity offered in the MVC framework helps in keeping each and every component and features of the system separate while making the addition of new components in the future easy to integrate with the existing system. This concept of keeping things separates is known as Micro Services. Below is the screenshot of the partition for the MVC structure,



*Figure 3 MVC Framework for the System*

As shown in the above screenshot, the "ionic" is the View (V), "nodejs" contains the Controller (C), "mongodb" contains the Model (M) of the system.

#### 4.2.2 Languages and Tools

A secure, scalable, and well-supported development framework was needed. Java, Microsoft.Net and Node.js were the choices in hand.

Java has a rock-solid foundation, IDE support, threading capabilities, debugging features and vast array of libraries. But Java has its downsides, primarily related to speed of development and the burden of legacy overhead, making it a questionable choice when it comes to modern, streamlined mobile client app development.

Microsoft .Net also is a great language, but adopting .Net would require picking a specific .Net language like C#, and that would mean spending more time learning the language as supposed to writing the actual logic.

Where Node.js on the other hand is easy to understand, as it allows the developers to write JavaScript on both the client side and the server side. This means similar patterns or similar libraries on both back-end and front-end development could be used. Unlike Java or .Net, Node.js runs based on callbacks. This approach helps the application to not pause or sleep if any intensive task is given, but it assigns a callback to it and proceeds to the next line hence it stays available for other requests. Whenever the intensive task gets completed, the callback will be called to process any necessary statements. Node.js is a JavaScript runtime built on Chrome's V8 JavaScript engine. Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient <sup>[4]</sup>. Node.js

package ecosystem, NPM (Node Package Manager), is the largest ecosystem of open source libraries in the world. Hence Node.js was a good fit for the needs of my application which required real-time fast replies with no delays. And, I was already familiar with JavaScript so there was no need for me to start from the scratch.

Hence the backend of the developed system is based on Node.js (v7.5.0). Node.js is just a platform for building server-side applications using JavaScript. Hence it cannot be used as it is for the purpose of this project. Express.js is such a framework based on Node.js which can be used for building web applications using the principles and approaches of Node.js. It helps to organize the web application into an MVC architecture on the server side. This framework is open source and very widely used, hence has a large user base/active community which helped me a lot during development.

After the server side development was decided, what kind of database to use needed to be decided. The two choices were SQL and No-SQL databases.

SQL databases are primarily called as Relational Databases (RDBMS) <sup>[5]</sup>. These are table based databases which means that these represent data in form of tables which consists of rows and columns. These require a predefined schema and are vertically scalable meaning these can be scaled by increasing the computation power of the server. SQL databases use SQL (Structured Query Language) for defining and manipulating the data. These are a good fit for a complex query intensive environment. SQL database examples: MySQL, Oracle, Sqlite, Postgres and MS-SQL.

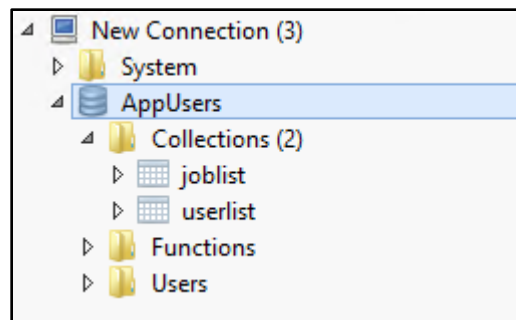
No-SQL databases on the other hand are primarily called as non-relational or distributed databases. These are document based databases which means that the data is

represented as a collection of key-value pairs. These do not have a standard schema or it has a dynamic schema. The data is unstructured in these types of databases. These are horizontally scalable meaning these can be scaled by increasing the number of database servers. The data is stored similar to how a JSON object stores its data. NoSQL database examples: MongoDB, BigTable, Redis, RavenDb, Cassandra, Hbase, Neo4j and CouchDb.

In the beginning of this project there was no idea of what data of the user will be stored, or what types of data would be stored. Hence there was no structure or schema for the database. This was the main reason the No-SQL databases were chosen, so that fields can be added whenever required without disturbing the already existing data. But as the project went on, it was found that the decision of using a No-SQL database had its own advantages. One major advantage being the ability to store images, as this was a major part of the app. In a MySQL database to add an image to the database only reasonable solution that could be found was to save the image on to the local storage and save its path to the database which did not seem to be an efficient way to do it, because every time an image was required the path had to be searched in the database, go to that path, search for the image and then return it to the origin of request. But in a No-SQL database like MongoDB, an image can be stored easily into the database by saving the binary of images (0s and 1s). This is fast and efficient way to send and receive images to and from the database.

The data that was required for the mobile client had to be transferred quick to reduce load times. If a SQL Database had been used, multiple tables would have to be made, one for the users and their details, one for managers containing the jobs that they

had posted, one for workers containing the jobs they had applied and all these to work seamlessly two or more tables would have required to be combined using SQL Joins clause, which would result in a very complex structure and would also have been complicated to write a query for. In the No-SQL database used there was no such problem and the queries were also very easy to write. The same will be shown in section 4.4.6 and requires only two collections for meeting my requirements (See Figure 4). The No-SQL Database used is, MongoDB as it was the most popular database for web applications and is a perfect fit for Node.js applications as MongoDB uses the same structure as a JSON Object which works seamlessly with the JavaScript used in Node.js.



*Figure 4 MongoDB Structure*

Once the backend of the mobile client was planned out for, the next step was to plan the front end of the system, i.e. the Mobile app. As this project is meant to be available to everyone, making an app that was compatible with all kinds of Mobile OS (iOS and Android for the scope of this project) had to be considered. There were two options available, one was to build native apps or to build a hybrid app.

A native app is a smartphone application developed specifically for a mobile operating system (using Objective-C or Swift for iOS and Java for Android). Since the

app is developed within its ecosystem following the guidelines of the OS, the interaction/look and feel is consistent with most other native apps. Thus, the end user is more likely to use and navigate the app faster. Native applications also have the advantage of being able to access the built-in capabilities of the user's device (e.g., GPS, camera etc).

Hybrid applications are, at core, websites packaged into a native wrapper. They can be made to look and feel like a native app, but ultimately outside of the basic frame of the application they are a website. Basically, a hybrid app is a web app built using HTML5, CSS and JavaScript, wrapped in a native container which loads most of the information on the page as the user navigates through the application. Also, various hardware capabilities of user's device can be accessed using various plugins.

But a major advantage of Hybrid apps over Native which outweighs all other disadvantages is, it needs to be coded only once whereas in Native apps one has to code separately for different Mobile operating systems. This means the app would have to be coded for android using Java once and again for iOS using Objective-C or Swift. This would take up a lot of time and effort. But in a Hybrid app, there would only be a need to code once and the same code can be used for an Android app and an iOS app with no extra effort necessary. This would mean more time could be spent on the actual app than on fixing bugs separately in individual native apps.

Thus it was decided, Hybrid App would be a better option over a Native App. Next step was to decide what framework of Hybrid Apps to use. There are many options available in the field of Hybrid Mobile App Development, some of them being Ionic,

Mobile Angular UI, Intel XDK, Sencha Touch etc. Every one of them is an excellent framework, but it was decided to go with the Ionic Framework, because it has been in the market for a long time and has a very strong and active community which would help out if there would be hiccup somewhere. It has a very rich plugin repository and offers some other small features like pull-to-refresh, infinite scroll, sliding boxes etc. that other frameworks cannot offer.

Ionic Framework is a complete open-source SDK (Software Development Kit) for hybrid mobile app development. It is built on top of AngularJS and Apache Cordova.

AngularJS is a JavaScript based open source front-end web application framework maintained by Google. This is used generally for declaring dynamic views in web-applications and extend the functionalities of vanilla HTML.

Apache Cordova is a mobile application development framework that enables a developer to build applications for mobile devices using CSS3, HTML5 and JavaScript instead of relying on platform-specific APIs like those in Android or iOS. The core of Apache Cordova applications use CSS3 and HTML5 for their rendering and JavaScript for their logic. HTML5 provides access to underlying hardware such as the camera and GPS. Apache Cordova can be extended using native plugins, allowing developers to add more functionalities that can be called from JavaScript, making the communication between native layer and HTML5 easier.

The Ionic Framework, as mentioned before, is built on top of Apache Cordova. So, it basically has all the features of Cordova with some UI components added to it which help develop rich and interactive apps. It uses Cordova to deploy the app natively

by using the native SDKs. All the Cordova plugins like, the plugin to control the camera of the user's device or the plugin to use the GPS of user's device can all be used within Ionic very easily.

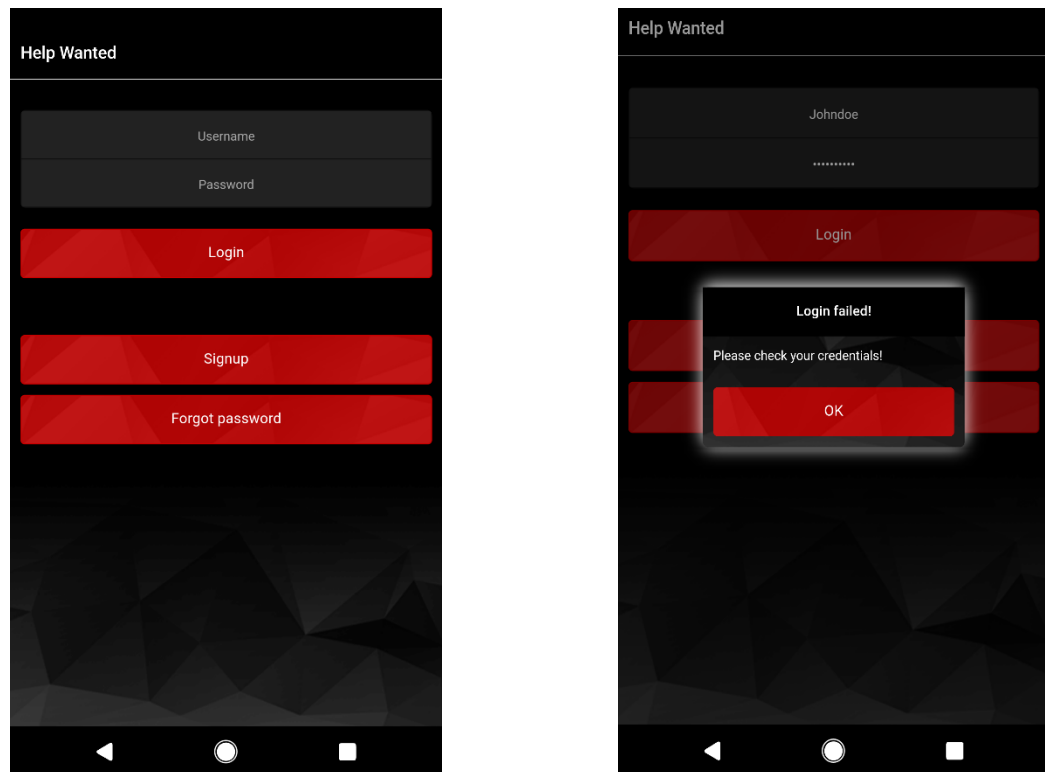
After the backend and the front end of the mobile client were decided, there was a need to make them both talk, transfer data to and from. As both the backend and front end use JavaScript, it seemed right to use a service that did not need the JavaScript data to be converted into some other form before sending it and then converting it back to JavaScript after it is received. So, for this there was only one good option i.e., RESTful Web Services. Representational State Transfer (REST) is an architectural style, and an approach to communications that is used in the development of web service. Almost every RESTful service uses HTTP (Hypertext Transfer Protocol) as its underlying protocol. Thus, it uses the HTTP protocol's methods like GET (used to request data from a specified resource) and POST (used to submit data to be processed to a specified resource) to enable communication between clients and servers. Data can be sent in form of JSON objects using these REST calls to and from the server to the mobile client.



## 4.3 User interface

### 4.3.1 Login Screen

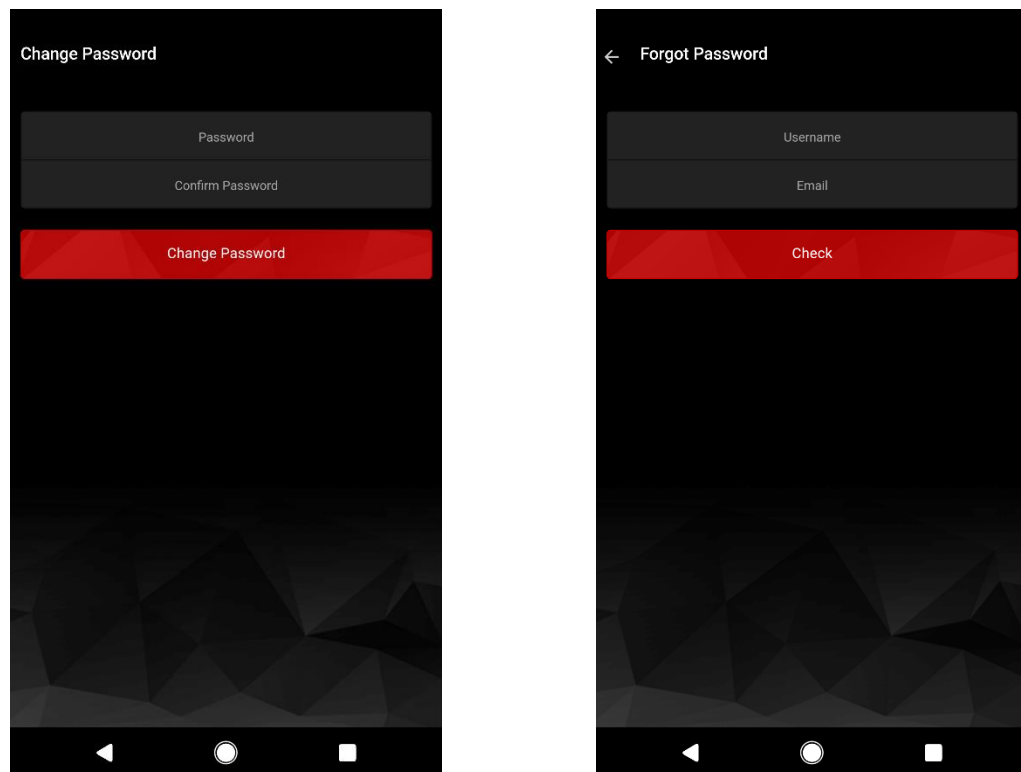
The Login Screen has the main elements like a textbox for username, password field for the password, Login button to login the user. Other than these there are two more buttons, one a Sign-Up button for new users to sign in and a Forgot Password button for the users who forgot their passwords and want to change it. After the user enters the username and password, it is checked with the DB, if the pair matches the user is logged in, otherwise they are shown a popup saying the details entered are incorrect.



*Figure 5 Login Screen*

#### 4.3.2 Forgot Password Screen

The Forgot Password Screen takes in the Username and Email of the user and checks with the DB, if the pair matches, the user is displayed a Change Password screen to update their password. If the pair doesn't match, they are displayed a popup saying the username and email don't match.



*Figure 6 Forgot Password and Change Password Screen*

#### 4.3.3 Sign up Screens

The Sign-up part has 3 different steps or screens. The first screen takes in the user's details like the username, password, Full Name, email address, date of birth, gender, Role (Manager or Worker). If the role selected is “Manager”, it asks for the Name of the Business. The second screen asks for the location of the user, by default it uses the GPS Location of the users mobile and puts pin on the maps provided by Google. The pin is draggable, so the user can decide the location to be stored by the system. The third screen asks the user to upload a picture. Here the user has a choice of either to click a picture using the Mobile’s built-in camera or upload a picture already available on their mobile. This picture will be displayed wherever required. The following set of screenshots (Figure 7) shows the Signup process (the three steps).

← Sign up

Username

Password

Confirm Password

Full Name

Email

Date of Birth

Gender ▾

Role ▾

Next

← Sign up

JohnDoe

.....

.....

John Doe

johndoe@abc.com

Date of Birth 03/20/1986

Gender Male ▾

Role Worker ▾

Next

← Your Location

Move the Marker to your address

Map Satellite

Arbor Court

California State University, Northridge...

Jacaranda Walk

Delmar T Oviatt Library

Sequoia Hall

Matador Walk

Matador Wi


Google

Map data ©2017 Google Terms of Use

Next

← Login

Open Gallery



Signup

Figure 7 Sign-up Screens

The login, sign up, and forgot password screens are common to all kinds of users. The difference comes in when a user logs in. If the user is a manager or a worker depending on their profile are shown different screens.

#### 4.3.4 Manager Screens

If the user who logged in is a manager, there are 5 different tabs shown. The first tab displays a map showing the location of nearby workers using markers and also the location of the manager using a different kind of marker. Tapping on the markers of workers reveals their full name (See Figure 8).

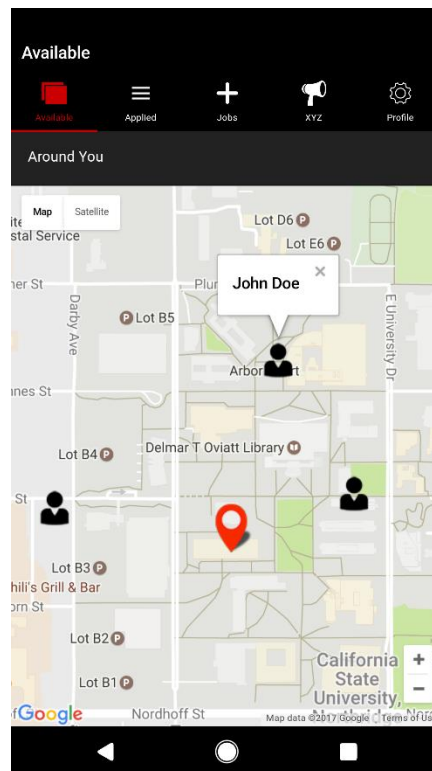
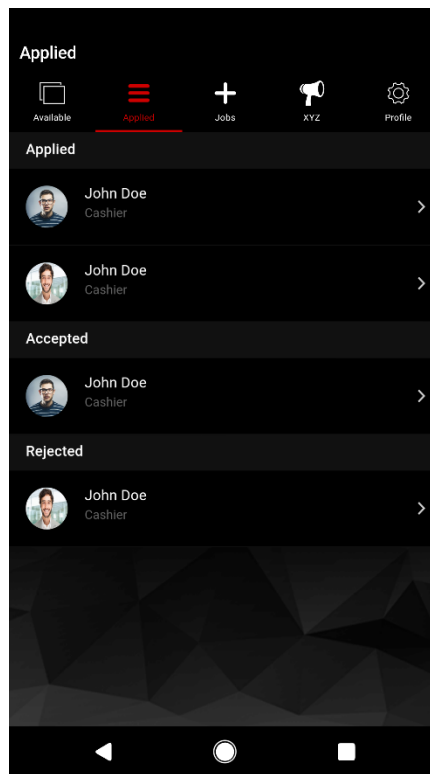


Figure 8 Manager Screen (Tab 1)

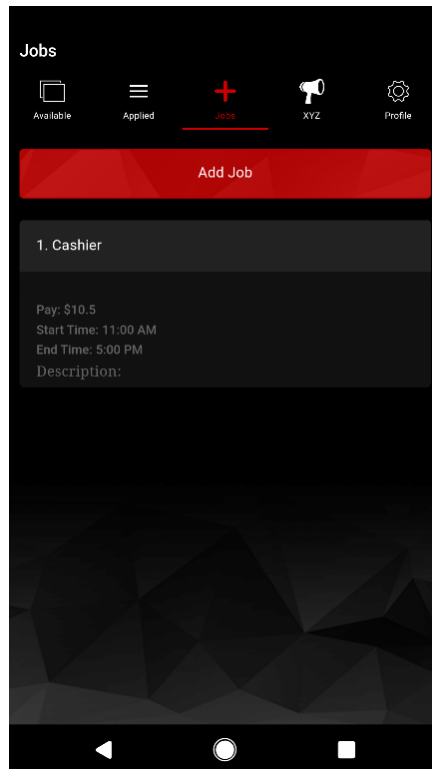
The second tab displays all the profiles of workers who have applied to any of the newly posted jobs, anyone the manager had accepted or rejected for any of the previously posted jobs. These are shown in 3 different categories i.e. Applied, accepted and rejected. Their picture along with their name and the job they had applied for is displayed. Tapping it opens up a screen with the same details along with the pay per hour field. The manager can also refresh the screen to see if anyone new has applied by scroll-to-refresh (See Figure 9).



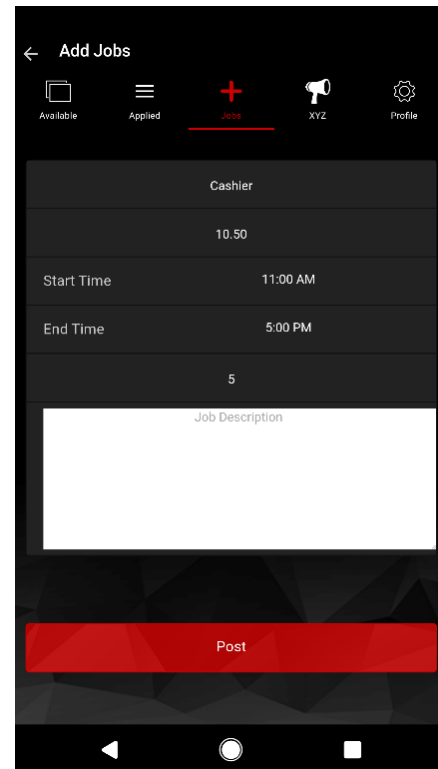
*Figure 9 Manager Screen (Tab 2)*

The third tab is where the manager can either view the previously posted jobs or add a new job using the 'Add Job' button (See Figure 10). When this button is pressed, it navigates to the 'Add New Job' screen where the manager can enter details like the Job Title, pay per Hour, the Start time and end time of the job, Description about the job and

the Beacon Radius. The Beacon Radius is the number of miles within which the notification of Job Posting will be sent to. Say the Radius is set to 20 miles, the notification will be sent to all the workers within 20 miles of the Business (See Figure 11).



*Figure 10 Manager Screen (Tab 3.1)*



*Figure 11 Manager Screen (Tab 3.2)*

The fourth tab is where, if anyone new has applied for a job posted, their picture with their name and the job title applied for is shown in a neat stack of cards form. The manager can swipe the card right to approve them or to the left to reject them (See Figure 12). This immediately sends them a notification letting them know about the decision and also the same is updated in the second tab screen.

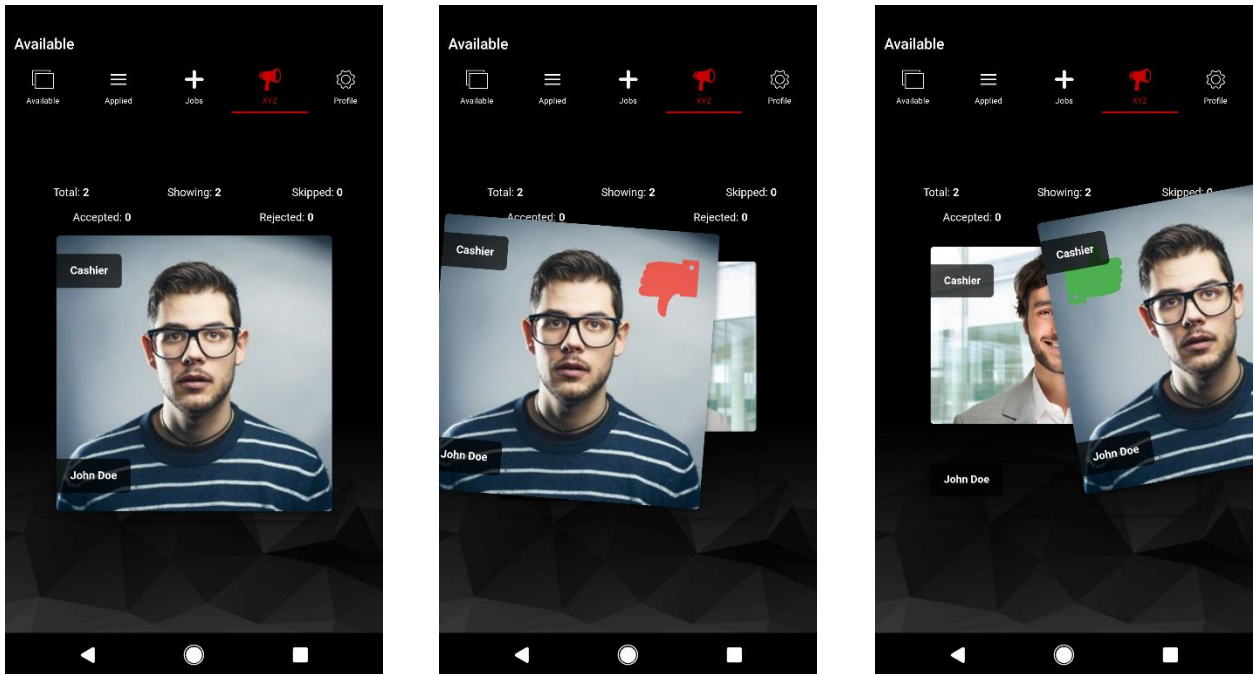


Figure 12 Manager Screen (Tab 4)

The fifth tab is basically the profile screen displaying the picture and other details of the user along with the Logout button and Change Password button (See Figure 13).

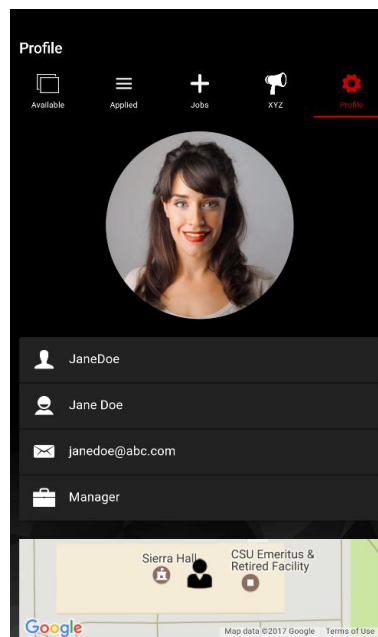


Figure 13 Manager Screen (Tab 5)



#### 4.3.5 Worker Screens

If the user who logged in is a worker/Jobseeker, there are only 4 tabs displayed. The first tab (Figure 14) displays a map showing the location of nearby jobs using markers and also the location of the user using a different kind of marker. Tapping on the markers of jobs reveals the job title and the amount of pay.

The second tab (Figure 15) is pretty much the same as on the manager screen only difference is it displays the jobs the user had applied and the ones they were accepted into or rejected from.

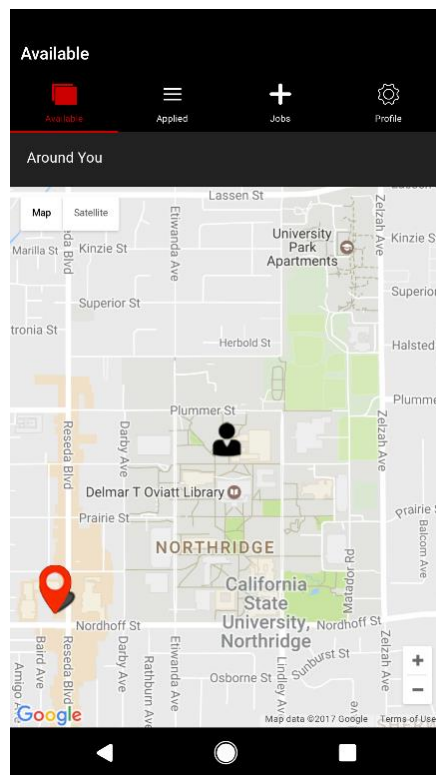


Figure 14 Worker Screen (Tab 1)

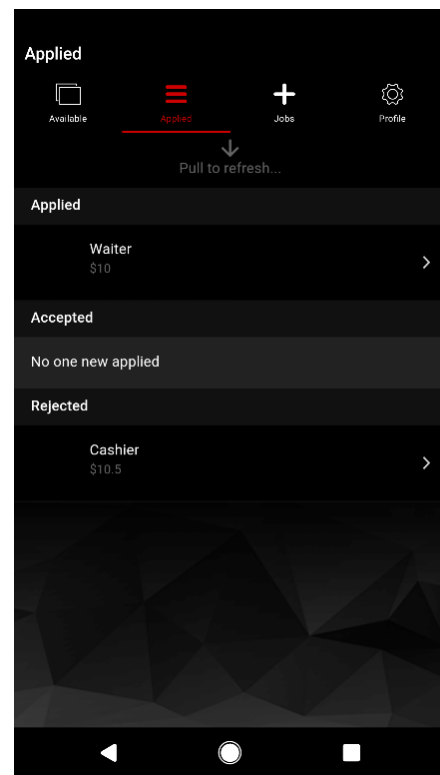


Figure 15 Worker Screen (Tab 2)

The third tab displays all the job postings around him within a certain radius of miles from his location. The user can change that radius and view all the jobs within that radius. When he finds the job suitable, tapping on it opens up a different screen with more details about the job and has the option to apply for that job. Tapping on the ‘Apply to this Job’ button sends out a notification to the manager who posted that job.

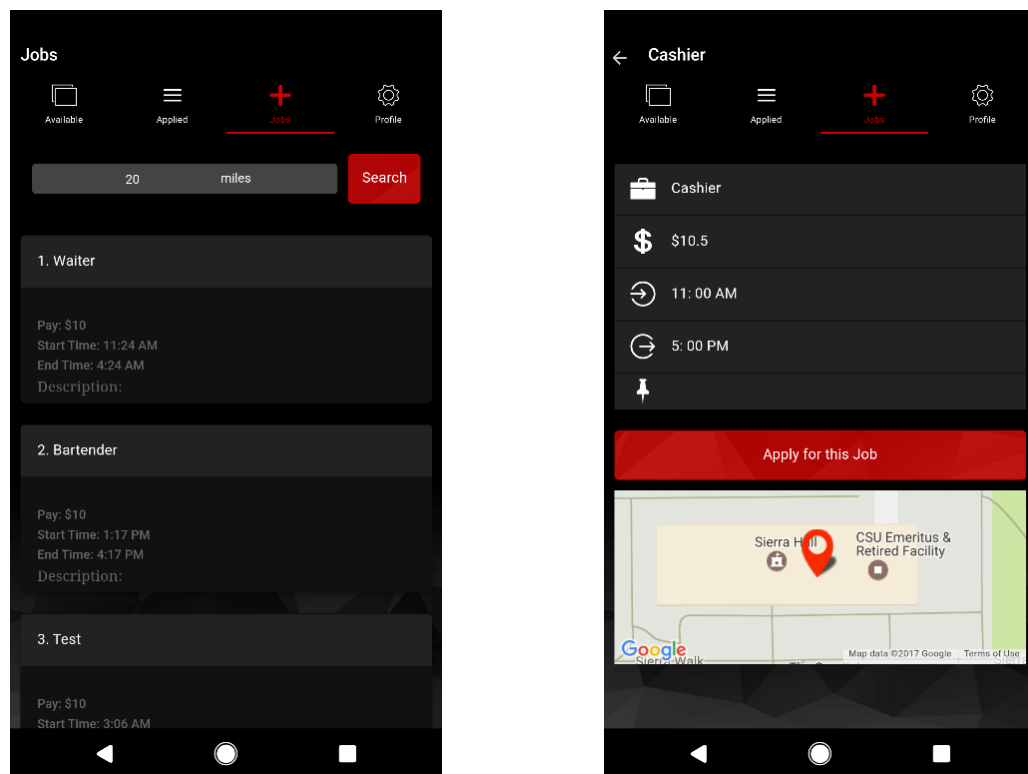
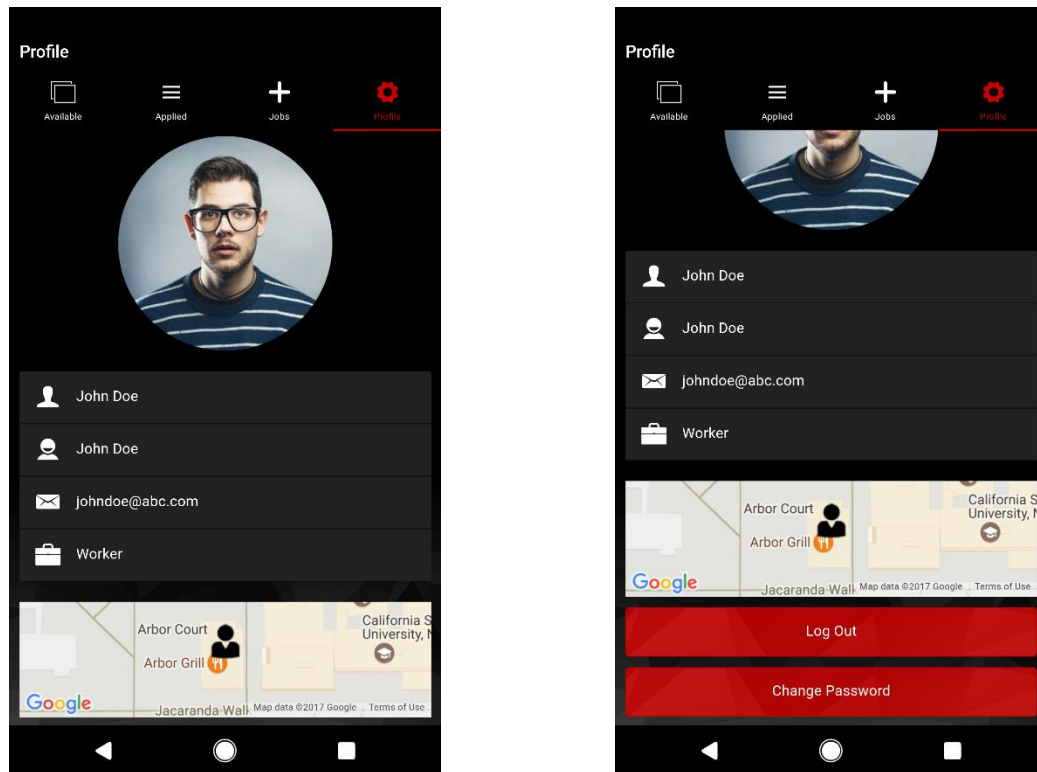


Figure 16 Worker Screen (Tab 3)

The fourth screen is like the profile page, displaying the details of the user along with a logout button and a change password button.

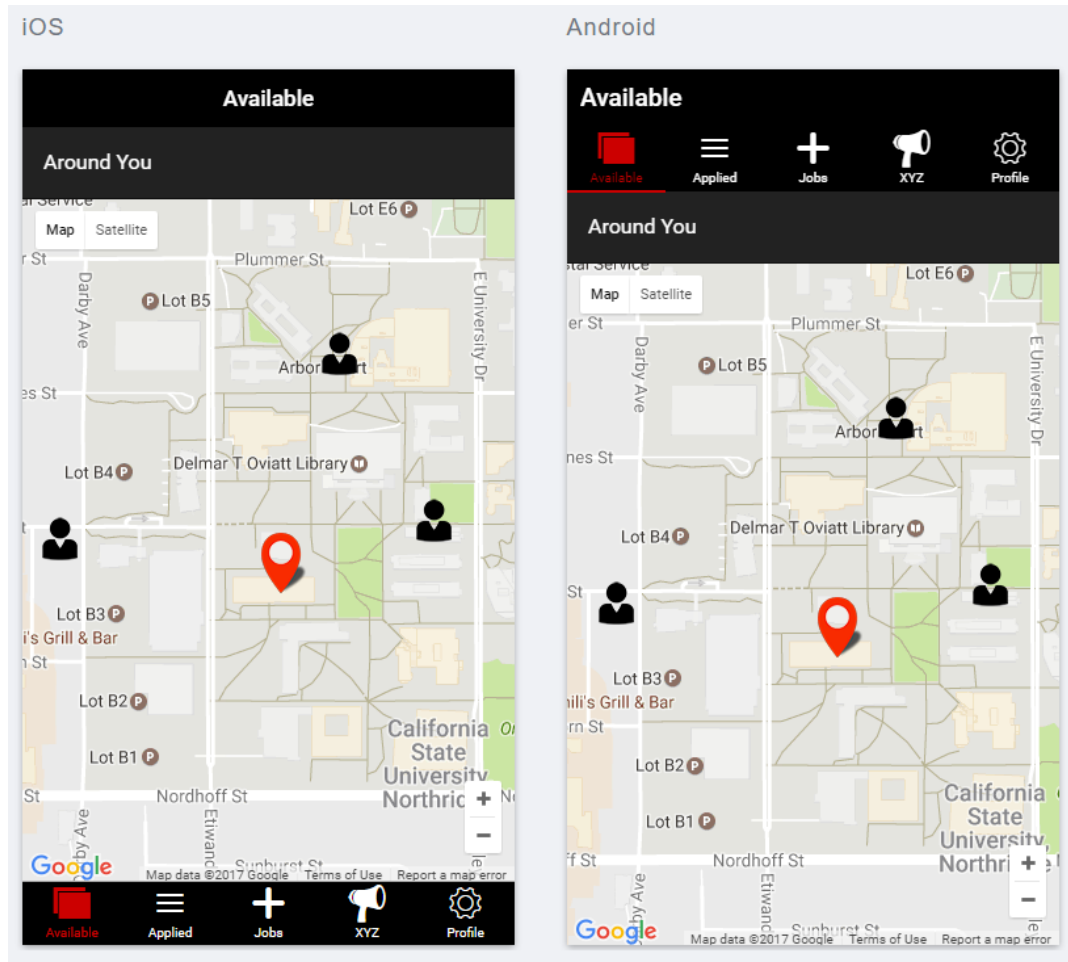


*Figure 17 Worker Screen (Tab 4)*

#### 4.3.6 iOS vs Android

The advantage of using a Hybrid Mobile Framework is, one need not code multiple times for different platforms. The Ionic framework does all the work of deploying the same code onto separate platforms. The only difference in the interface between the iOS and Android is the way in which the tabs are shown.

In the iOS, the tabs are displayed on the bottom of the screen, whereas in Android they are shown on the top of the screen. This is the only difference, other than this every other screen is the same, every other functionality is the same.

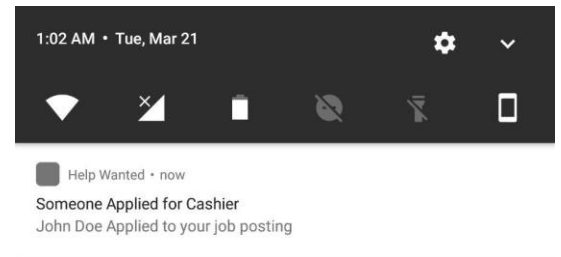


*Figure 18 iOS(Left) vs Android (Right))*

#### 4.3.7 Notifications

When a notification is received, there are two ways of handling it. One is when the app isn't open or is in the background and the other one is when the app is open i.e. In the foreground.

If the app is open and a notification is received, it will show up as a popup displaying the information. If the app is closed or in the background, the notification will show up as any other notification would i.e. in the notification drawer (See Figure 19).



*Figure 19 Notifications when app is closed*



*Figure 20 Notifications as popups when the app is open*

## 4.4 Source Code

### 4.4.1 Encryption

Encryption of passwords is done by using a npm module, CryptoJS and its AES algorithm for a more secure password.

```
$rootScope.base64Key =  
CryptoJS.enc.Hex.parse('0123456789abcdef0123456789abcdef')  
$rootScope.iv =  
CryptoJS.enc.Hex.parse('abcdef9876543210abcdef9876543210');  
  
var encrypted = CryptoJS.AES.encrypt(  
    $scope.data.password,  
    $rootScope.base64Key,  
    { iv: $rootScope.iv }  
);  
var password = encrypted.ciphertext.toString(CryptoJS.enc.Base64);
```

If a user enters the password as “qwerty1234@!”, after encryption and hashing the password that is sent to the server will look like “y+EPTIdb/5Bpv8bFufiI5w==”.

### 4.4.2 Notification Handling

For handling the notifications, Google’s Firebase Cloud Messaging (FCM) was used. Firebase Cloud Messaging (FCM) is a cross-platform messaging solution by Google that lets one reliably deliver messages at no cost. Using FCM, one can send notification messages to notify a client app. Basically FCM acts as a middleman between the app server and the mobile client. The server sends a message to FCM, then the FCM delivers that message to the targeted users (which are attached to the message sent to FCM from the server).

To use this feature a cordova plugin called cordova-plugin-fcm<sup>[6]</sup> was used in the mobile client.

```
FCMPlugin.getToken(  
    function (token) {  
        $rootScope.token = token;  
    },  
    function (err) {  
        console.log('error retrieving token: ' + err);  
    }  
);  
  
FCMPlugin.onNotification(function(data) {  
    $ionicPopup.alert({  
        title: data.title,  
        template: data.payload  
    });  
});
```

Every time a user logs in, the FCM server sends a token that is unique to the user's mobile that is sent to the database and is saved for future use.

If a notification is received by the user when the app is in the background it is automatically handled by the user's mobile OS but if the app is open and a notification is received, the app must be told how to handle that notification. As seen in the code above, whenever a notification is received and the app is open, the notification is shown in form of a popup window (Section 4.3.7).

On the server side, a different module called as “fcm-node” is used.

```
var serverKey = 'AAAAvYfixGU:APA91bErvr5btqby7eW1EE3nTduPhj-  
1Tg39Eu0AA3uJlROenYjE5mce8c2zXXUjpId7xD2-BvqpVBNG7E-oMfW3Hw_aMR-  
MrMlWfRt8odJ1hoX2eypqYYv7RZRVv_M8H4oWEHUAVyUS';  
var fcm = new FCM(serverKey);  
  
var message = {  
    to: docs3[0].token ,
```

```

notification: {
  title: 'Someone Applied for ' + docs[0].title,
  body: fullname+ ' Applied to your job posting' ,
  sound: 'default'
},
data: {
  title: 'Someone Applied for ' + docs[0].title,
  pay: fullname+ ' Applied to your job posting'
}
};
fcm.send(message, function(err, response){
  if (err) {
    console.log("Notification sent except for few");
  } else {
    console.log("Successfully sent with response: ", response);
  }
})

```

Here the “serverKey” is unique to the app, and any message sent using that server key, the notification will be sent. In the “message” variable, “to” field must contain the tokens of the users one want the notification to go to. The above code if executed will send a notification to the manager informing them someone applied to the job they posted.

#### 4.4.3 Maps

For showing the locations on map, Google Maps <sup>[7]</sup> were used. The following lines of code must be added in the html file wherever the map is required to be shown.

```

<script src="http://maps.google.com/maps/api/js?key=AIzaSyA86p53TudF-
YmCk7gWJ1oByhEU0j049kg"></script>

```

Here, “key=AIzaSyA86p53TudF-YmCk7gWJ1oByhEU0j049kg” is the Google Maps Api Key that is unique to this app and is provided by Google. Without this key, Google servers wouldn’t reply any calls made to it.



```
<div id="maploc" data-tap-disabled="true" style="width:100%;height:80%;"></div>
```

In the java script files, the logic which shows the location and other markers, the following is the code:

```
var latLng = new google.maps.LatLng(position.coords.latitude,
position.coords.longitude);
    gps = latLng;

    var mapOptions = {
        center: latLng,
        zoom: 18,
        mapTypeId: google.maps.MapTypeId.ROADMAP,
        disableDefaultUI: true,
        zoomControl: true,
        mapTypeControl: true
    };

    $scope.maploc = new google.maps.Map( document.getElementById
("maploc"), mapOptions);

    var marker = new google.maps.Marker({
        position: latLng,
        map: $scope.maploc,
        draggable:true,
        animation: google.maps.Animation.BOUNCE
    });
    marker.addListener('click', handleEvent);
    marker.addListener('drag', handleEvent);
    marker.addListener('dragend', handleEvent);
}
function handleEvent(event) {
    gps = event.latLng;
}

if(navigator.geolocation){
    navigator.geolocation.getCurrentPosition(onMapSuccess,
onMapError, {enableHighAccuracy: true });
}
```

The variable “latLng” holds the coordinates of the current location of the user (using the GPS in the user’s smartphone).

The variable “mapOptions” has all the variables required by Google Maps to display, like the coordinates of the center point, the zoom value and other map controls.

The variable “marker” is used to display a pointer like icon on the map for the user which can be dragged around for a more precise location which the user can choose.

The above code is used to display a map with a marker pointing at the user’s location (See Figure 7).

#### 4.4.4 Camera/Display Picture

In the sign up screens, to upload a picture of the user using the camera on their smartphone, the plugin “cordovaCamera” was used. The following line of html code was added where ever the picture needs to be displayed.

```
<div class="ion-profile-picture no-picture" ng-click = "takePicture()" style="background-image: url('{{user.picture}}');"> </div>
```

Here “ion-profile-picture no-picture” is the class which displays the picture in a circular window.

The code used to open the camera app and take a picture is:

```
$scope.takePicture = function (options) {  
  
    var options = {  
        quality: 80,  
        destinationType: Camera.DestinationType.DATA_URL,  
        sourceType: Camera.PictureSourceType.CAMERA,  
        allowEdit: true,  
        encodingType: Camera.EncodingType.JPEG,  
        targetWidth: 1024 ,  
        targetHeight: 1024 ,  
        popoverOptions: CameraPopoverOptions,  
        cameraDirection: 1,  
    }
```

```

        saveToPhotoAlbum: false
    };

    $cordovaCamera.getPicture(options).then(function(imageData) {
        $scope.user.picture = "data:image/jpeg;base64," +
imageData;
    }, function(err) {
        // error
    });
}

```

The properties for the smartphone to take the required picture are given in the variable “options”, like the quality of the picture, encoding format, height and width, to use the front camera (cameraDirection: 1). After the picture is taken, it is converted to base64 format (text format) which makes it easier to send using REST API calls and also saving into the MongoDB.

#### 4.4.5 REST API Calls

A REST API defines a set of functions which developers can perform requests and receive responses via HTTP protocol such as GET and POST. Basically, these calls allow the data to be transferred in the form of JSON (JavaScript Object Notation).

```
$rootScope.ipaddress = "http://52.38.153.189:80";
```

This is the IP address of the server that was setup for this app on the AWS EC2 instance running Windows Server 2012. This is where the ExpressJs (Server) and MongoDB (Database) are running. The ExpressJs keeps listening on port 80 for any calls and handles accordingly.

```

router.post('/login', function(req, res) {
    var db = req.db;

    var userName = req.body.username;
    var passWord = req.body.password;

```

```

var token = req.body.token;
var collection = db.get('userlist');
collection.find({"username": userName, "password":
passWord}, {}, function(e, docs) {

    if(docs.length == 1 )
        res.send("Success");

    else
        res.send("Failed");

});
});

```

In the above function, “req.body” contains these variables. The username and password are checked with the ones in the MongoDB, if there is a match then a “Success” message is sent and if no match is found a “Failed” message is sent. “res.send()” sends back a response to the source of request after the request has been handled.

To run the above function a HTTP Post call must be made from the mobile client with the required fields.

```

$http.post($rootScope.ipaddress+'/login',
{"username":$scope.data.username,"password":password}).then(function(re
sp){

    if(resp.data=="Success"){
        Data.setUser(resp.data);
        var alertPopup = $ionicPopup.alert({
            title: 'Login Successfull!',
            template: 'Welcome '+resp.data.fullname
        });
        $state.go('tab.dash');
    }
    else{
        var alertPopup = $ionicPopup.alert({
            title: 'Login failed!',
            template: 'Please check your credentials!'
        });
    }
})

```

Here the call is made to “http://52.38.153.189:80/login” with the body containing username and password. If it receives a “Success” message from the server, it shows a popup saying “Login Successful” “Welcome” with the full name of the user, and the function “state.go()” displays the home screen of the user. If it receives a “Failed” message from the server, it shows a popup saying “Login Failed” “Please check your credentials”.

#### 4.4.6 MongoDB

In MongoDB, there are many functions that can be performed on the data in the database. To achieve this a module called “monk” was used in the Node.js. Only the functions “find”, “update”, “insert” were needed for the app. Few examples of how they are used are as follows,

```
var collection = db.get('userlist');
collection.update({"username":
userName}, {$set: {"token": token}}, function(e, docs) {
    res.send(
        e === null) ? { msg: 'Success' } : { msg: e }
    );
});
```

Here initially the collection (similar to a table in SQL Database) is set to “userlist” which contains all the usernames and passwords of all the users. Then “collection.update” is used to update a particular field. Either you can add a new field or update the value of an existing field. The database entries are stored as a JSON Object and hence is very customizable. In the example above, “collection.update()” adds a field named “token” to the entry where the username matches the variable “userName”. If the update is successful, “res.send()” sends a “Success” message or else it sends the error message.

```
collection.insert({"username": userName,
  "role": role,
  "password": passWord,
  "email": email,
  "fullname": fullname,
  "DOB": dob,
  "gender": gender,
  "location": location}, function(e, docs) {
  res.send(
    (e === null) ? { msg: 'Success' } : { msg: err }
  );
});
```

Here “collection.insert()” is used to add a new entry to the collection in the database. In the example above, a new user is being added to the collection with all his/her details like username, password, role, email, fullname, date of birth, gender, and location. Similar to the previous example, if the insertion is successful it sends a “Success” message otherwise it sends the error message back.

```
collection.find({"username": userName, "password":
passWord}, {}, function(e, docs) {
  if(docs.length == 1 )
    res.send(docs);
  else
    res.send("Failed");
});
```

Here “collection.find()” is used to find/search for a particular entry. In the example above, an entry where the username is the value of variable “userName” and the password is the value of variable “passWord” is searched. If found, the whole entry is saved in the variable “docs” and is sent back and if it does not exist, a “Failed” message is sent.

These three functions are used throughout the ExpressJS to handle data in different types of REST API calls like “forgotpassword”, “changepassword”, “signup”.

## 4.5 Unit testing

Unit Testing is a level of software testing where individual units/ components of a software are tested. The purpose is to validate that each unit of the software performs as designed <sup>[8]</sup>. My unit testing was based on the White Box Testing methodology, i.e. I had access to view the code, hence my test cases were built around breaking the code and boundary conditions.

I had to test individual units like in the app if there is no internet connection, is the app giving me a popup saying network disconnected. If the entered username is wrong, is the app notifying the user, if the connection to server is a bad one, is the user notified to try again or not. Basically, the GUI was tested extensively.

For testing the REST API calls, an app known as “Postman” was used to test if the server is replying to the received calls in the right manner and format or not. If the MongoDB is saving the data in a proper collection or not. All these were thoroughly tested.

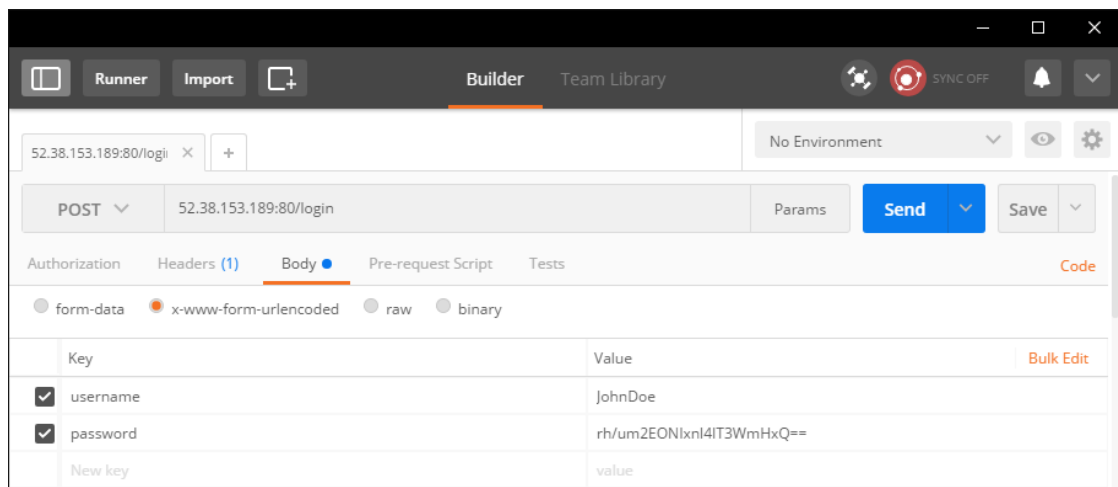


Figure 21 Postman App (Used for testing REST API Calls)

## 4.6 User Testing

For user testing the app, I went to the same people whom I had met for requirement gathering. I had only limited time and access to them, so couldn't meet them more often, hence could not take a survey like feedback. I had to show them as much as possible and ask if they wanted any more features or anything needed to be changed, just to be sure I was on the right path and if this app would be helpful to them.

Few improvements that were suggested were, initially during the signup process, for the location of the user I was not using the GPS to determine their location, instead it was asking for the user to drag the marker to their desired location. They suggested to use the GPS on their smartphone to determine the initial location and then they could drag the marker to more precise location. By the next sprint I had put in that feature and as I had the GPS coordinates I also was updating their location whenever they log in. But they did not like that feature as, if they logged in to the app just to check for jobs when in a market or some other place other than their home, app would show their location as the market which would lead to confusion when sending out notifications. So, I had to keep their location fixed at whatever they chose to put it during their signup process.

One other such feature that was added was in the display picture screen. I had put in the option for only taking a picture from the camera, my idea was that this would force the users to take only their picture which would avoid uploading of any other random image from the internet. But people wanted the option to upload from their local phone storage.



## **Chapter 5: Conclusion**

### **5.1 Summary**

In this project, a mobile client was successfully developed which acts as a bridge between the restaurant/hotel managers or food chains and its job seekers and providing an ecosystem for them to collaborate.

### **5.2 Development Experience**

This project also helped me learn new technologies that I was not familiar with in the beginning. This experience will help me out in my future endeavors as a developer. I learned how to build a mobile client that can be exported to any mobile platform without the need of native OS coding. I also learned to host a server in the cloud, create and manage a database, and make the app communicate with the server. Amazon Web Services and their tutorials helped me a lot. Following the Agile framework also helped me be on track and finish the project without any issues of staying behind the schedule. The use of micro services helped me in testing and coding units separately without being dependent on each other in turn reducing errors.

### **5.3 Challenges and Future Scope**

I did not face any major challenges other than the fact that I had to learn each and every technology used for this project. I wouldn't call it a challenge though; it was more of a learning experience.

Amazon Web Server offers free service for one year. My first subscription had expired, but that did not automatically kill the server, so I started getting charged for running it. So, to avoid that I had to move the whole of Node and Mongo DB code onto another instance which changed the IP address and I had to change the IP address in the app too.

Even though I used a Hybrid Mobile Framework, to generate an app that is executable in the Apple ecosystem of devices, it needed a Mac Os device which was not available hence the app was tested only for Android devices and built accordingly. Also for the notifications to work on the iOS devices I needed to have an Apple Developer account which was not free and again was not feasible. So, the scope of the app in this project is limited to Android devices only. But if required in the future it can be expanded into other mobile ecosystems.

For future developments in the app, instead of signing in using the traditional method, we can add authentication using Google, Facebook, Twitter etc. making it easier to login for the users. The project did not include a traditional instance plan. There was no attempt to address concerns like profitability and return on investment. There are many ways to achieve this, first is in-app ads, second is like a premium account, say the manager takes a premium subscription which guarantees a help that has already been given a good rating and checked by us. A premium subscription of the job seekers can be like they'll be the first to be notified if a job has been posted or provide him with transport to a location which is far from the radius, as we already have their location and their destination. This will not only give a guarantee to the job seeker that he will get a job but also guarantee the food chain the help they want without they worrying about the distance.

Other major future scope is that the app for now is made only for restaurants and food chains, but this can also be expanded to other Blue collar jobs or one day jobs or any other short term contract jobs. By adding another screen where the user can select what kind of job are they looking for or what kind of help they are looking for. Say someone needs a plumber to fix a problem at their home, when a user chooses the “plumber” option, the app will send notifications to only the ones who have selected “Type of work looking for” as “Plumber” while signing in, instead sending the notification to everyone that's around.

## References

1. “Craigslist”, Retrieved on October 2<sup>nd</sup>, 2016 from  
<https://losangeles.craigslist.org/>
2. “Zip Recruiter”, Retrieved on October 3<sup>rd</sup>, 2016 from  
<https://www.ziprecruiter.com/>
3. “Amazon Web Services EC2”, Retrieved on November 13<sup>th</sup>, 2016 from  
<https://aws.amazon.com/ec2/>
4. “NodeJS Tutorials”, Retrieved on October 26<sup>th</sup>, 2016 from  
[https://www.tutorialspoint.com/nodejs/nodejs\\_introduction.htm](https://www.tutorialspoint.com/nodejs/nodejs_introduction.htm)
5. “SQL vs NoSQL Database Differences Explained with few Example DB” By Luke P.,  
Retrieved on October 7<sup>th</sup>, 2016 from  
[http://www.thegeekstuff.com/2014/01/sql-vs-nosql-db/?utm\\_source=tuicool](http://www.thegeekstuff.com/2014/01/sql-vs-nosql-db/?utm_source=tuicool)
6. “Fire Cloud Messaging Plugin”, Retrieved on October 28<sup>th</sup>, 2016 from  
<https://github.com/fechanique/cordova-plugin-fcm>
7. “Google Maps Javascript API Documentation”,  
Retrieved on October 10<sup>th</sup>, 2016 from  
<https://developers.google.com/maps/documentation/javascript/tutorial>
8. “Unit Testing”, Retrieved on February 12<sup>th</sup>, 2017 from  
<http://softwaretestingfundamentals.com/unit-testing/>