# Hyer

Location Based Job Listing and Job Search with Payments

## The Problem

A multitude of job posting apps exist, such as jobs.ca, Workopolis, craigslist, etc. Out of those many job posting apps, a small portion of them are for short-term jobs, or a simple task that one person wishes to pay another person for. And even furthur, out of those apps, none seem to be able to filter jobs relative to your current location, and list jobs that can be on the other edge of the city. This results in constant clicking into job postings, finding they are too far and going back. Additionally none of these apps seem have a sort of payment system incorporated, which also for many people who live using their card, or do not like carrying coins, can be a nuisance.

## Hyer

Our application, is the solution to those problems. Our app is divided into two mobile applications, one for the employer, and one for the employee.

Employers:

- Load in app credits to their account - used to pay others.
- Post jobs, look at applications, hire applicants, and pay the employee.

Employee:

- Search for jobs (including distance and location filters)
- Apply for jobs, and get paid for their work.

## Learning Goals

- Two team members had no experience in web development
  - Wanted to learn how to design a RESTful API
  - Database integration in the back end
  - Wanted to learn how to use premade APIs to ease redundant development (e.g. Firebase)
  - Deployment of application for  non-local use
- No members had experience in React Native
  - Wanted to try a new framework for building an application
- No members had experience with continuous integration
  - Wanted to learn how to use Travis CI limiting pushes to the repo that resulted in something not working

Firebase React Travis CI

## Intended Deliverables

## Deliverable 1

For deliverable 1, we intended to have the minimum viable product finished for our applications. This included:

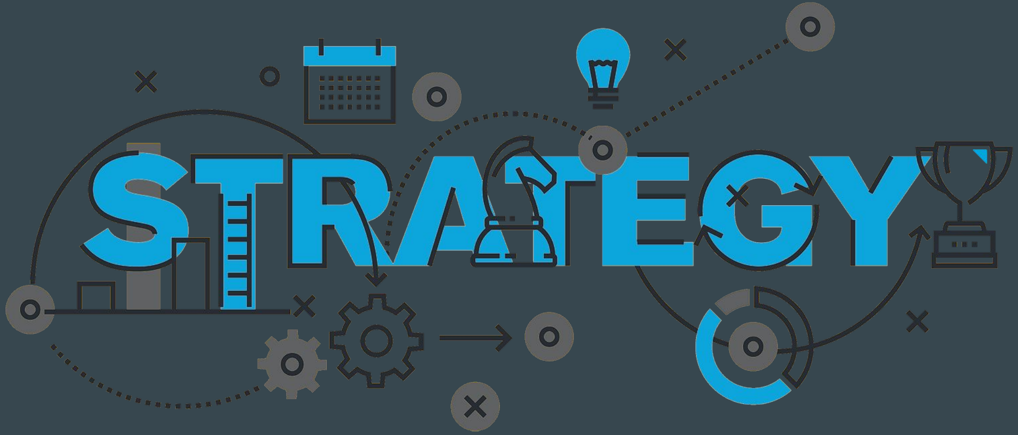
- Ability for employer/employee to login
- Ability for employer/employee to register
- Ability for employer to add jobs
- Ability for employee to search jobs
- Ability for employer/employee to edit account details

## Deliverable 2

For deliverable 2, we intended to get our main features that differentiate our app up and running. This included:

- Ability for employee to search jobs listed in order of nearest job
- Ability for employee to limit search to jobs under a given radius from them
- Ability for employee to apply to a job
- Ability for employer to hire an applicant for given job
- Ability for employer to pay employee credits for given job

## Project Strategy

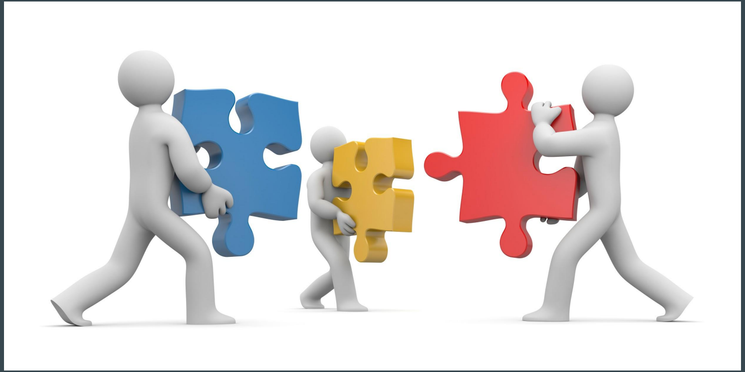


## Project Strategy

We split our group into two separate groups: Backend and Frontend.

We were aiming to allow each member to be focused into the areas where their respective goals lied.

Backend:
- Ryan
- Raymond
- Patrick



Frontend:
- Tim
- Shreyash

## Project Strategy

Frontend wanted to:

- Implement login, register, and edit profile pages
  - For both employers and the employees.
- Implement posting jobs, hire applicants and pay employees.
  - For the employers.
- Implement viewing the jobs, apply for the jobs and receive pay from employers.
  - For the employees

## Project Strategy

Backend wanted:

- RESTful API
  - to create, remove, update, and delete users/jobs  from the database.
- Implement all necessary searching and filters
  - Lower the workload for the frontend.

## Architecture

## Backend Architecture

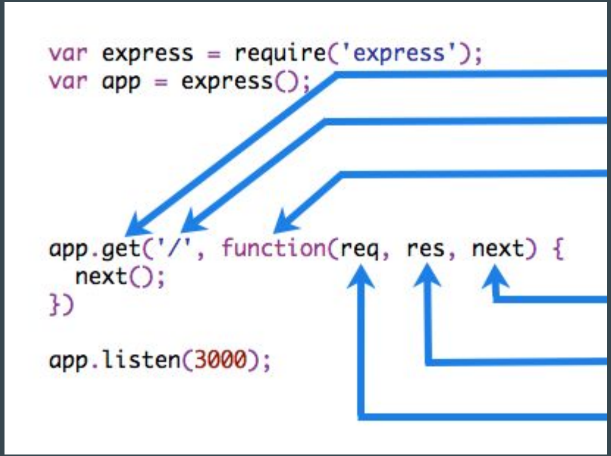
Firebase:

- Supported by Google
- Database for Users & Jobs
- NoSQL real-time cloud database
    - Possibility of multiple users accessing database at once.

## Backend Architecture

More backend:

- Node.js
- Express.js
  - Node.js framework
- Simplified HTTP requests
- Firebase functions
  - Plethora of functions made backend programming much easier

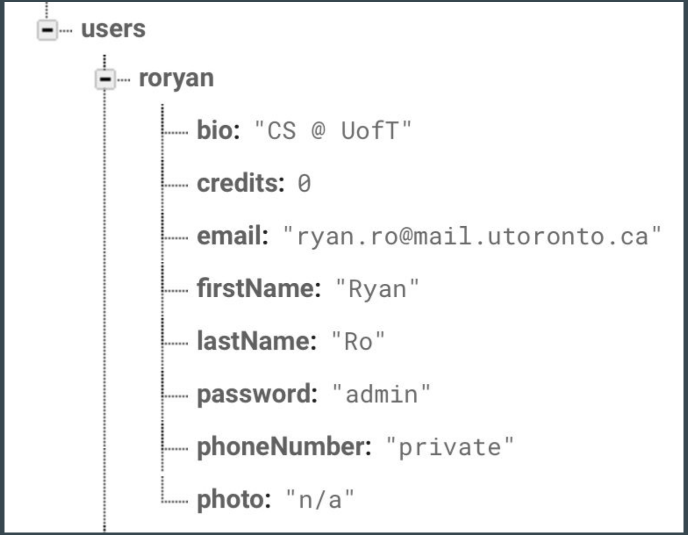


```
var express = require('express');
var app = express();

app.get('/', function(req, res, next) {
  next();
})

app.listen(3000);
```

## Backend Architecture

Our two endpoints in the backend were users and jobs.

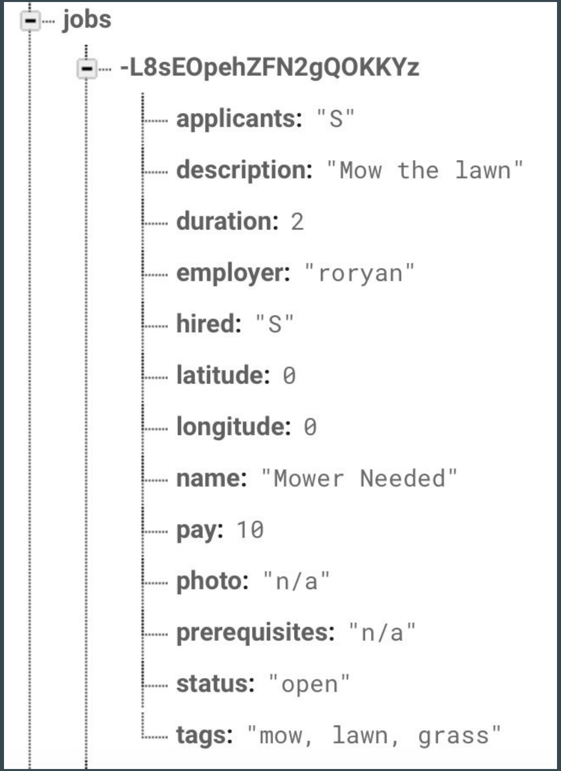
User object example:



```
users
  roryan
    bio: "CS @ UofT"
    credits: 0
    email: "ryan.ro@mail.utoronto.ca"
    firstName: "Ryan"
    lastName: "Ro"
    password: "admin"
    phoneNumber: "private"
    photo: "n/a"
```

## Backend Architecture

Job object example:



```
jobs
  -L8sEOpehZFN2gQOKKYz
    applicants: "S"
    description: "Mow the lawn"
    duration: 2
    employer: "roryan"
    hired: "S"
    latitude: 0
    longitude: 0
    name: "Mower Needed"
    pay: 10
    photo: "n/a"
    prerequisites: "n/a"
    status: "open"
    tags: "mow, lawn, grass"
```

## Backend Architecture

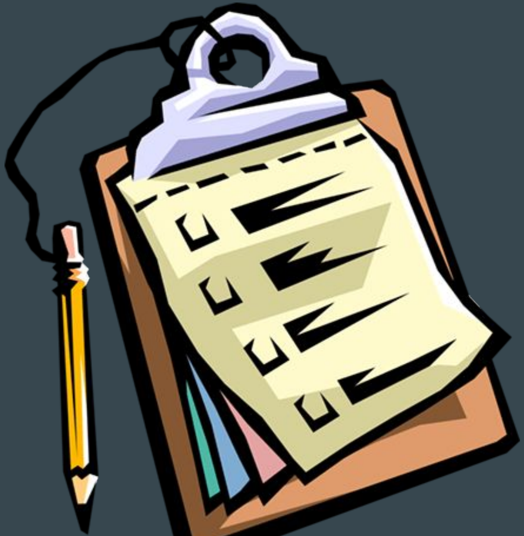
The API calls had the following parameters

- GET /users : Gets all users
  - username : If specified gets user with given username
- GET /jobs : Gets all jobs that are not closed
  - jobID : If specified gets job with given jobID
  - search : If specified gets jobs with given search string in its name or tags
  - employer : If specified gets jobs by a given employer
  - order : If specified orders the jobs gotten by given order (supports location, and pay)
  - latitude, longitude, km : If specified filters only the jobs gotten that are in the given km radius

## Backend Architecture

Other calls (POST, PUT, DELETE):

- Relatively generic
- Taking in data to post/update/delete
- Multitude of checks
    - Does the user or job exist?
    - Was this username already taken?
    - Etc...
- Return the respective codes (2xx, 4xx)



## Frontend Architecture

We wanted a mobile Application for both iOS and Android. A member for each platform would spread us too thin

Solution? React Native.

- Compiles mobile web applications
- We can specify choose between both iOS and Android

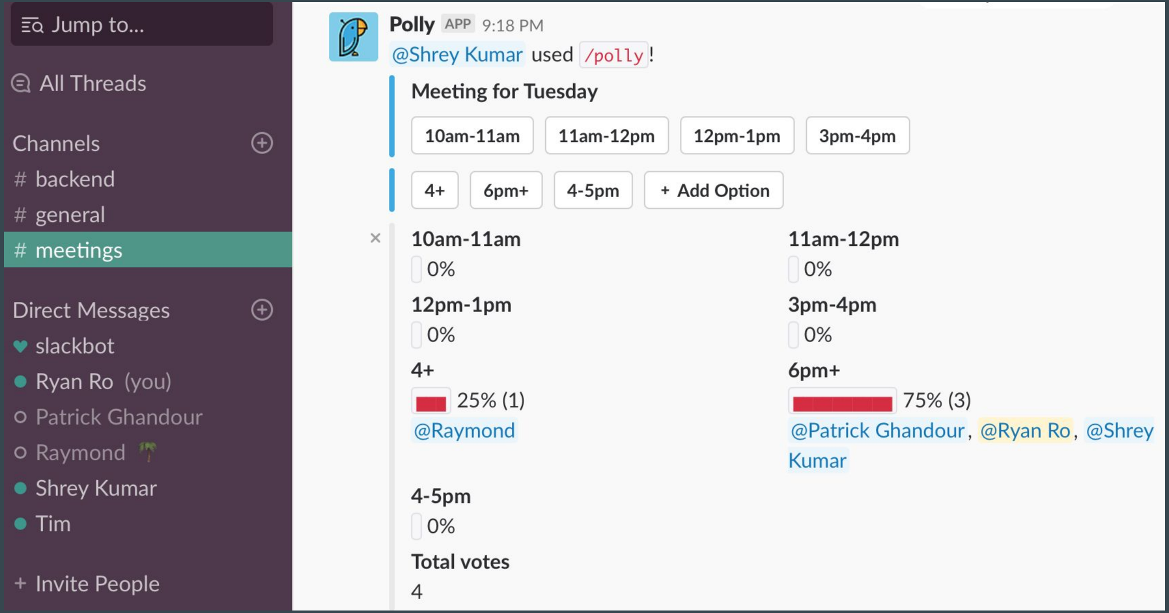
With React Native, our applications called our detailed backend API, and rendered the response or status from those calls.

React Native

# Collaboration

## Meetings

For our meetings, we polled our members, to see which time worked out for us all, and then met at the respective time in BA3200, to discuss our project.



Jump to...
All Threads
Channels
# backend
# general
# meetings
Direct Messages
slackbot
Ryan Ro (you)
Patrick Ghandour
Raymond
Shrey Kumar
Tim
+ Invite People

Polly APP 9:18 PM
@Shrey Kumar used /polly!

Meeting for Tuesday

10am-11am | 11am-12pm | 12pm-1pm | 3pm-4pm
4+ | 6pm+ | 4-5pm | + Add Option

10am-11am
0%

11am-12pm
0%

12pm-1pm
0%

3pm-4pm
0%

4+
25% (1)
@Raymond

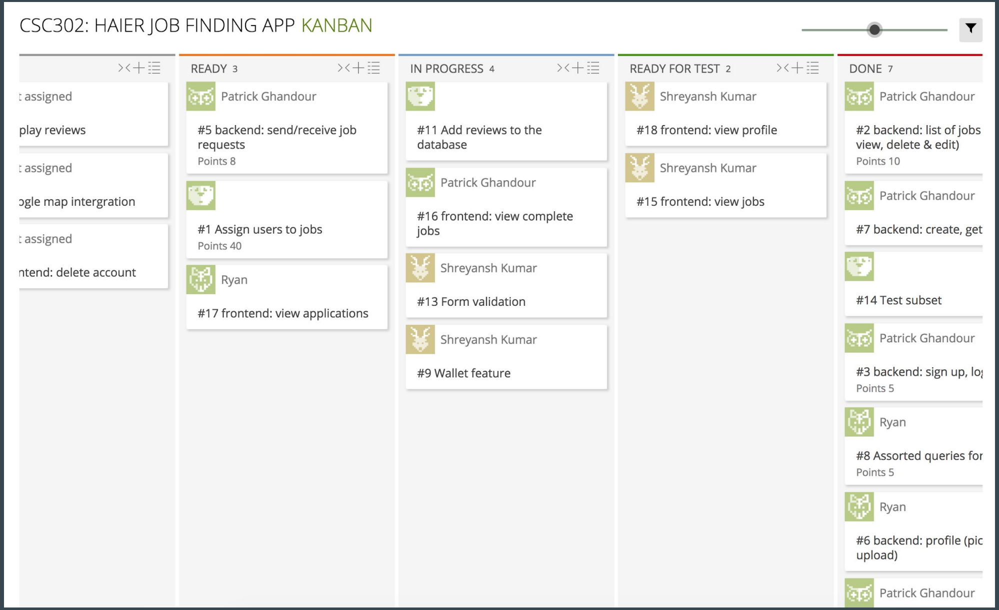
6pm+
75% (3)
@Patrick Ghandour, @Ryan Ro, @Shrey Kumar

4-5pm
0%

Total votes
4

## Meetings

In the meetings we discussed what needed to be done, and who was going to approach which task and how they were going to approach it. This was in sync with our Kanban board on taiga.io.



CSC302: HAIER JOB FINDING APP KANBAN

| | READY 3 | IN PROGRESS 4 | READY FOR TEST 2 | DONE 7 |
|---|---|---|---|---|
| : assigned / play reviews | Patrick Ghandour / #5 backend: send/receive job requests / Points 8 | #11 Add reviews to the database | Shreyansh Kumar / #18 frontend: view profile | Patrick Ghandour / #2 backend: list of jobs view, delete & edit) / Points 10 |
| : assigned / ɔgle map intergration | #1 Assign users to jobs / Points 40 | Patrick Ghandour / #16 frontend: view complete jobs | Shreyansh Kumar / #15 frontend: view jobs | Patrick Ghandour / #7 backend: create, get |
| : assigned / ntend: delete account | Ryan / #17 frontend: view applications | Shreyansh Kumar / #13 Form validation | | #14 Test subset |
| | | Shreyansh Kumar / #9 Wallet feature | | Patrick Ghandour / #3 backend: sign up, lo; / Points 5 |
| | | | | Ryan / #8 Assorted queries for / Points 5 |
| | | | | Ryan / #6 backend: profile (pic upload) |
| | | | | Patrick Ghandour |

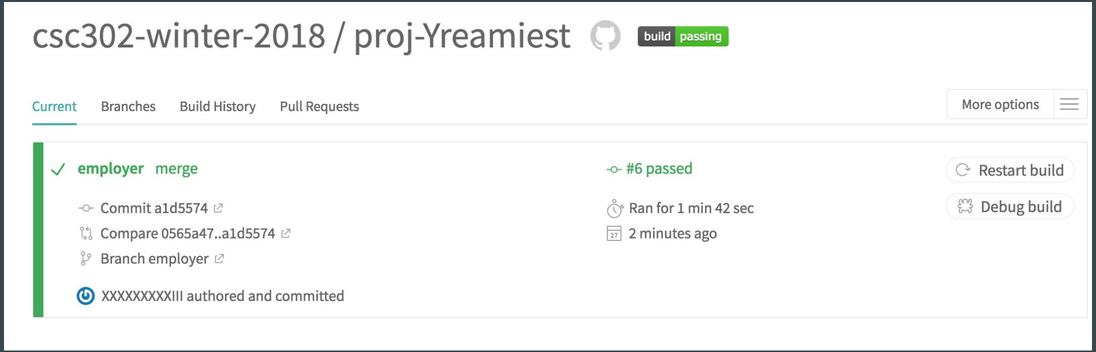
## Kanban

- The Kanban process is efficient for our group
  - Each member, can at their own time of preference, add, take, and accept tasks that either they have come up with on their own, or have come up during a meeting
- Since we are still students, we would like some exposure to the other areas of software development,
  - With Kanban, members can take tasks they are interested in
- If a member could not make a meeting, with a Kanban board it is easy to visualize and see, what they have missed, and what needs to be done
- The tasks were sized so each task should not take too long, and each member who gets assigned or takes the task, knows exactly what to do

## Testing

## Testing: Backend

- Created a set of tests using Mocha covering every possible API call
- Used Travis for continuous integration
- Made sure every pull request passed the tests
- Before merging another member who is familiar with the code reviewed the changes


csc302-winter-2018 / proj-Yreamiest

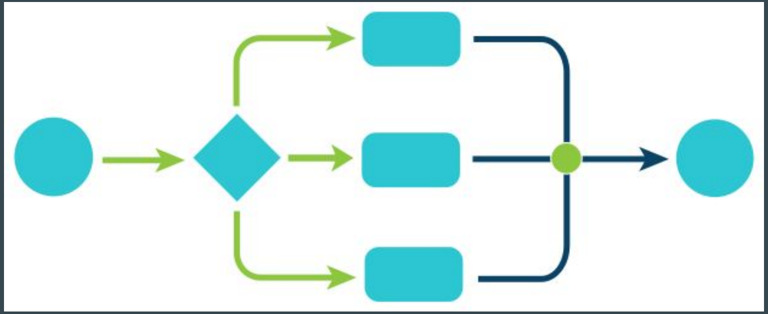
✓ employer merge

Commit a1d5574
Compare 0565a47..a1d5574
Branch employer
XXXXXXXXXIII authored and committed

#6 passed
Ran for 1 min 42 sec
2 minutes ago

## Testing: Frontend

- Tested specific workflows, e.g.
  - Adding a job
  - Accepting a request for a job
- Tested for data concurrency
  - Made sure frontend and backend data were consistent



## Features

## Features: Employee App

- Request jobs after viewing jobs from job list
- Filter jobs by location, search, employer
- Order jobs by pay and distance
- View jobs applied for, accepted, in progress, and completed
- When accepted for a job, view location, employer contact information, and location
- Set job to in progress

## Features: Employer App

- Add/Modify/Delete jobs
    - Includes description, pay, time, location, etc.
- View profile of user who applied to the job
    - May accept or decline application
- View all previously listed jobs
    - Also all accepted job application
- Mark job request as complete
    - Pays the employee

## Demo

## Goals

## Goals: Learning

- Learned how to use react native to create Android and iPhone apps (With no prior experience)
- Two group members who had no experience in web development
    - Learned how to create a RESTful API
    - Learned how to use a noSQL database
- Develop an app through Kanban
- Continuous integration

## What went well and what didn’t

What went well:

- Project Management
  - Kept separate branches on GitHub
  - Communicated about changes & kept in contact through Slack
- Product Design
  - Backend and Frontend were able to communicate easily
  - Everything we initially planned was finished in time for today’s demo

What didn’t:

- Did not allocate enough manpower to the frontend.
- Entire group meetings
  - The group rarely got everyone together because of conflicting schedules
  - Not a big issue since we were split into 2 smaller teams and can meet separately
  - But, the teams were isolated and didn’t know what the other team was up to.

Fin$_{\text{sigh}}$