SWEN 325 Assignment 1 Report

Jacob Robson

Introduction - OneOff

For this assignment me and Tom Brown decided we were to create a mobile application which provides a platform for connecting employers with employees. This platform would provide an online medium to list, and apply for one off jobs in New Zealand.

Architecture

The architecture of the application consists of three different layers: •

Presentation Layer - contains UI components as well as the components processing them.

Business Layer - composed of workflows, business entities and components.

Data layer - comprises data utilities, data access components and service agent.

Ionic applications are built with Cardova, which is a means of packaging html/css/js into apps that can run on mobile and desktop platforms. The first two layers, presentation and business are enforced by this file structure, as the ionic UI components are constructed in the html, while the business logic is implemented in the typescript files. The third layer is represented as an external component which is firebase, an online authentication and real-time database tool.

Presentation layer:

This is the layer contains the ionic components and the visualisation of the program. This is mostly constructed in the html files. In ionic when you create a page it creates a css, html, and two typescript files. The html is where one creates the ionic components used for the UI. The css is where the components and pages get styled. It is important to have this visual layer consistent across the whole app, as it is the means as to which your data

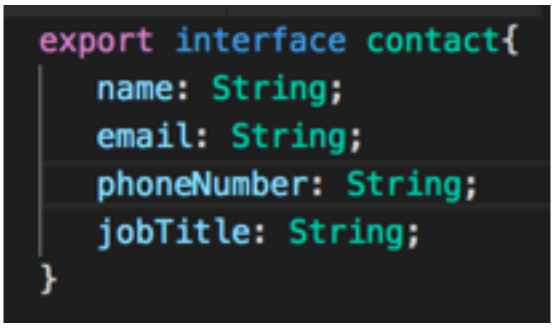
is displayed and retrieved. We chose a colour scheme from a palette and used this to style our pages.
We used the tab ionic component to group the functionality of our apps. We have three main tabs, each tab is further split into two ionic segments. Profile tabs which has a Profile page, and a contacts page. Then we have a home page where users can view listed jobs, either in map format, or as a list. And then finally we have a jobs tab. Where users can view jobs applied for in one segment, and jobs listed in another.

This layer is also where data is inputted from. A consideration whilst designing this layer is choosing the correct data format. Using data validation ensures ‘bad data’ does not meet the business application. So with our for we tried to ensure number inputs where used for numbers and strings were used for strings.

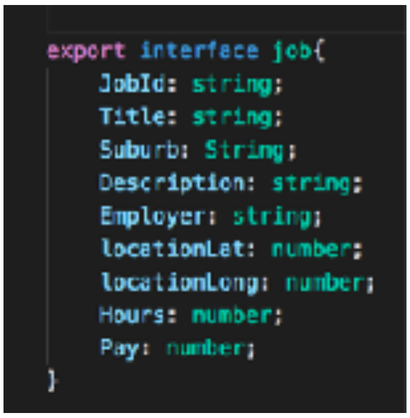
Business layer:

This layer is made up from the ts files. It is responsible for the logic and functionality of the application. It acts as the mediation between the database and the UI. For our app, this layer controlled the navigation of pages. Also it is responsible for storing an updating objects/data that the html reads.

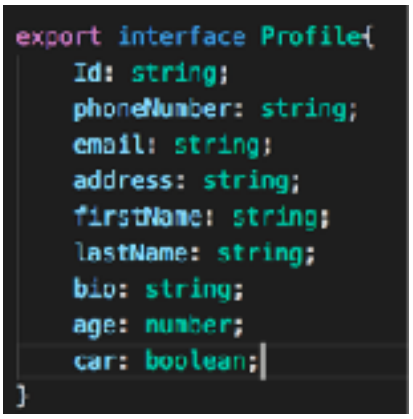
Our business layer also declares the main objects we store in the database. These objects can be found in the models folder. In our application we have 3 main models; contact, profile and a job. These are represented as JSON objects, and are created and manipulated by the various class pages, such as editJobPage and newJobPage.



```
export interface contact{
    name: String;
    email: String;
    phoneNumber: String;
    jobTitle: String;
}
```



```
export interface job{
    JobId: string;
    Title: string;
    Suburb: String;
    Description: string;
    Employer: string;
    locationLat: number;
    locationLong: number;
    Hours: number;
    Pay: number;
}
```



```
export interface Profile{
    Id: string;
    phoneNumber: string;
    email: string;
    address: string;
    firstName: string;
    lastName: string;
    bio: string;
    age: number;
    car: boolean;
}
```

Data layer:

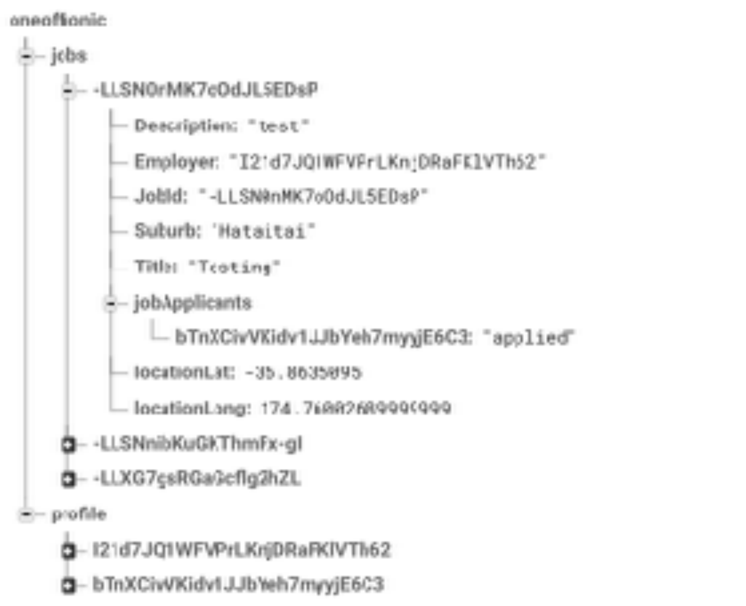
This layer is responsible for data CRUD , which is data creation, data retrieval, data updating, and data deletion. For this app we used firebase database to store our data, however only after getting halfway through the project did we realise that our architecture could be improved.

For a mobile app we should try separate these layers, however we have found that we designed our business, and data layers so that they merge into one. We have calls to our firebase database throughout the app pages. But what we should have done, is had a DB object which was responsible for all the CRUD functions. Then we could have made calls to this object from the business layer. This would have succeeded in separating business and data retrieval logic.

This is a big flaw in the design of our application, and for assignment two we will try separate theses concerns for a more traditional design.

External component - firebase:

Console of firebase showing how data is stored:



```
oneoflonic
  jobs
    -LLSNOrMK7cOdJL5EDsP
      Description: "test"
      Employer: "I2'd7JQIWFVPrLKnjDRaFKlVTh52"
      JobId: "-LLSNOnMK7oOdJL5EDsP"
      Suburb: "Hataitai"
      Title: "Testing"
      jobApplicants
        bTnXCivVKidv1JJbYeh7myjE6C3: "applied"
      locationLat: -35.8635095
      locationLong: 174.76882689999999
    -LLSNnibKuGKThmFx-gl
    -LLXG7gsRGaGcflg2hZL
  profile
    I21d7JQ1WFVPrLKnjDRaFKlVTh62
    bTnXCivVKidv1JJbYeh7myyjE6C3
```

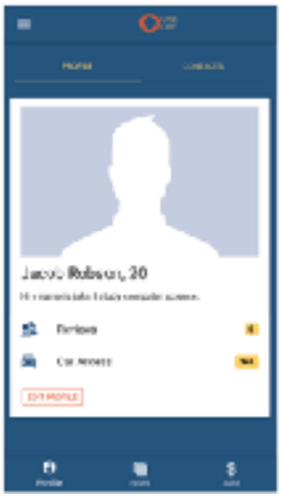
Fire base is a comprehensive mobile development platform, which supports authentication and real time data storage and manipulation. The real time database is a noSQL database which stores JSON objects. It provides developers with flexibility and is really easy to set up. It also has huge storage size potential and is server-less. The real time console is also very helpful- we can easily visualise what/how and where our data is being stored in our database.

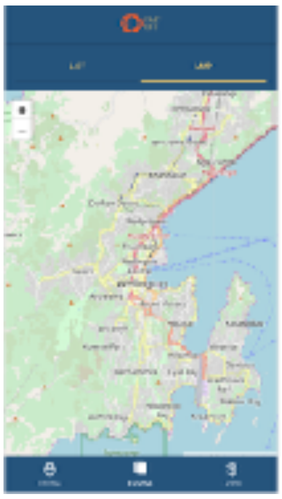
However, throughout the development of the app the disadvantages of using firebase also became evident. Its querying can be quite hard to understand, and there are a limited amount of tutorials online. Also because it stores data as JSON objects, and not in tables which also means there is not foreign keys. This makes cascading deletes and updates quite complicated, and also makes the integrity of the database quite vulnerable. Also because of the JSON objects, the data can form quite complicated trees which are hard to navigate and search.

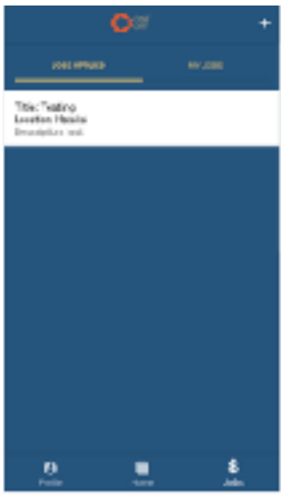
Unfortunately we integrated this component into our app architecture incorrectly. Most pages that utilise either data creation, data reading, data updating, or data deletion access and make call queries directly to the firebase database through there respective ts files. Instead, what we should have done is had a database management system class, which would have been responsible for all the querying of the database. And then we could make calls to the class from each of the pages. Not only would this have saved a lot of duplicate code, it also would have improved the overall architecture of the application. As the three main layers of the mobile application would have looser coupling and have more separate concerns.

UX decisions:

UX decision - Tab layout:







When we first started discussing the design of the app we were trying to decide the best way to separate the employee logic from employer logic. At first we thought it would be better to split the whole application in two, and have the users log in as an employer or log in as an employee/applicant.

However we then decided it would be better to have a tab format which we could use to seperate the groups of logic of the application. We decided it would be best to have three main tabs. These tabs are then split into two segments which assist with separating content relative to its respective tab.

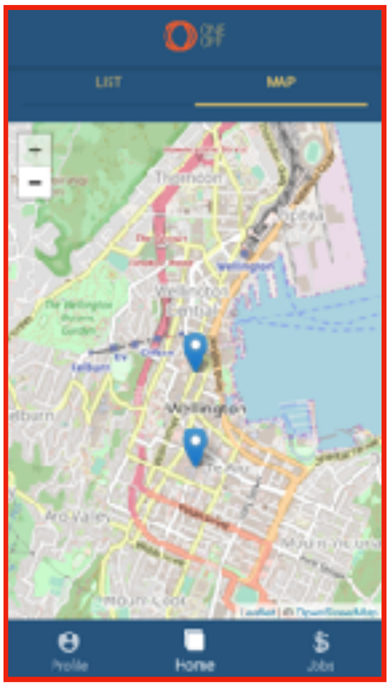
The first tab is the profile tab (On the left). The first segment of the tab is the profile of the currently logged in user. From here a user can edit their profile. The second segment is the contacts segment where employees contact details get added after they have been accepted for jobs.

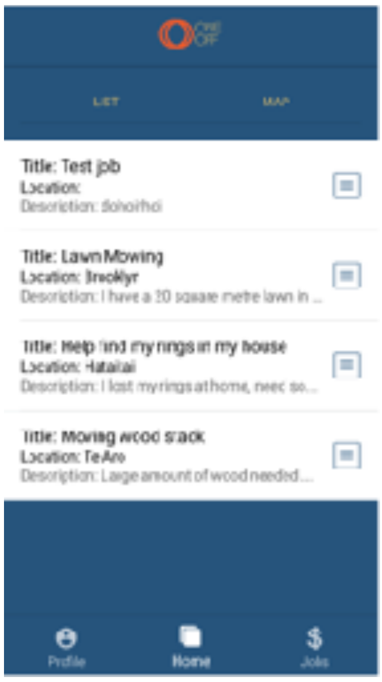
The second tab is the home tab (In the middle). This is where users can view jobs listed in their area. This are of logic is separated again so that it can be viewed geographically, in the map segment. Or in list format on the other segment. From here users can view and apply for jobs.

The final tab is the jobs tab (On the right). From here users can view/edit/delete jobs they have listed in the my jobs segment. Users can also list jobs from this tab. In the second segment, jobs applied, users can view the application status of jobs they have applied for themselves.

This separation of concerns via the tab component helped overcome the initial problem we had of separating employee and employer logic as now the two have been merged together. Users are now able to to easily navigate to different functions of the app as both an employer and employee.

UX decision - Displaying jobs.





Title: Test job
Location:
Description: dohoirhoi

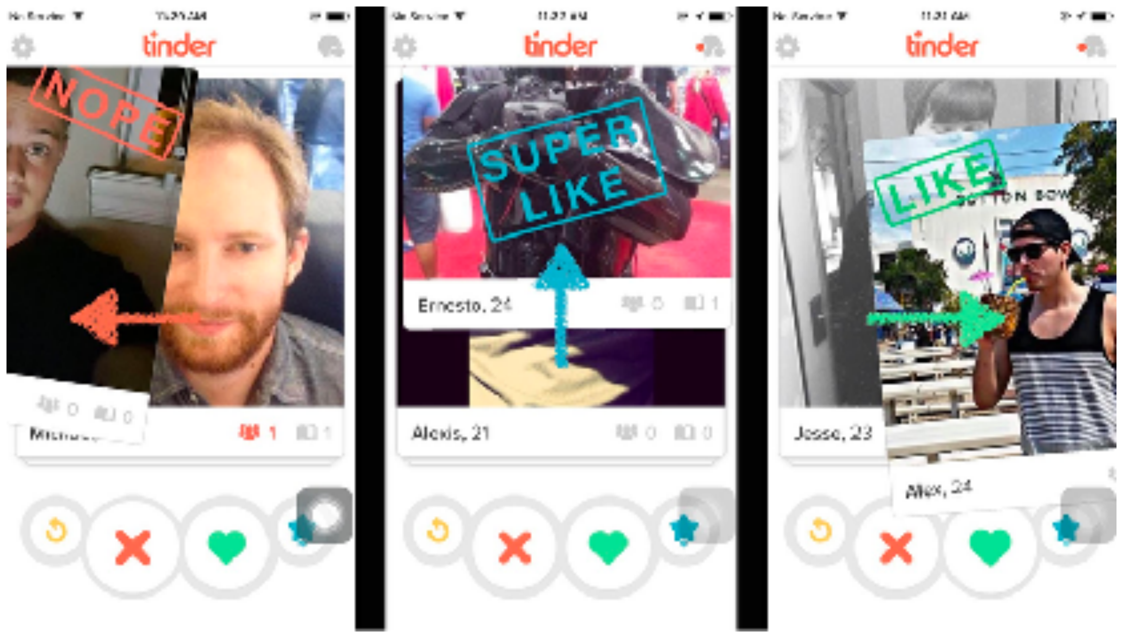
Title: Lawn Mowing
Location: Brooklyn
Description: I have a 30 square metre lawn in ...

Title: Help find my rings in my house
Location: Hataitai
Description: I lost my rings at home, need so...

Title: Moving wood stack
Location: Te Aro
Description: Large amount of wood needed ...

The reason we decided to make this app was because we thought that the existing applications, such as student job search and trademe jobs missed the gap for one off jobs. The two applications provide a platform similar to ours, however OneOff is more directed at casual, DIY type users. We believe that the user experience should reflect this. So when it can to viewing & applying for jobs we wanted to make it as easy and as casual as possible.

Initially we thought tinder cards would be a good idea:



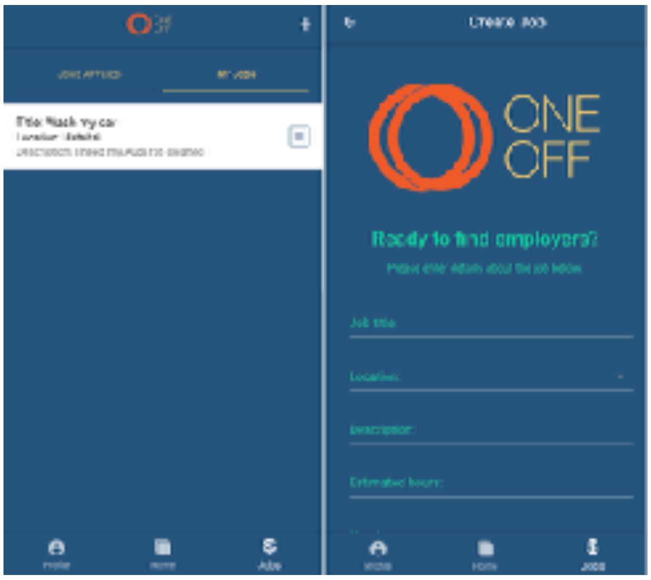
Although swiping left and right is a nice and easy way to sort through data. We thought that having a geographic visualisation of the jobs would be more beneficial to our types of users. Student job search and trademe jobs do not offer this visualisation. And we thing that it is the best way for One off applicants to quickly view, and apply jobs which are close to them in order to make money as quick as possible.

UX decision - Button layout & consistency

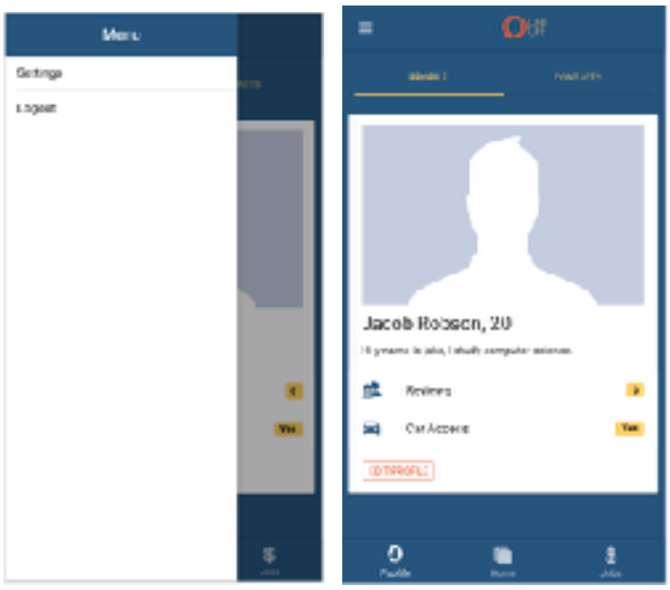
We decided from the outset that it would be better to plan our pages and the navigation of them, instead of diving in to development straight away. We knew from SWEN 303 that button layout and consistency, was essential for a a good user experience.

We decided that we would have 3 tabs to group and represent different functionality, as previously discussed. Then we thought the segments would help aid navigation even further and add to the consistency to the layout.

Then we also needed navigation to two more pages - settings and create a job. We decided to add to buttons which would mirror each other. These buttons could also be put into the tab which represents the same branch of functionality. These buttons would be in the tool bar and provide a natural flow to the app.



On the right is the far right tab, which is the jobs tab. In the top right corner we have the list job button, which pushes the user to a create job page. Which they can easily press back from to pop them back to the jobs page. Or they can creat a job which also pops them back to the jobs page.



On the left is the far left tab, which is the profile tab. In the top left corner we have the menu button, which displays the menu on the left side of the screen. A user can easily swipe back to the right to reveal the profile again.

The nature of this navigation is consistent and mirrored. Users will try something they have done before. We could have had both theses buttons in the same place(both on the left or both on right). However because of the tab layout we thought that this would be a more natural and aesthetically pleasing way to do it.

Ionic framework review:

Ionic - introduction

This was me and toms first time developing any sort of mobile or web application. We started out with creating the initial mandatory program. This was easy enough to develop and gave us quite a good understanding of the how ionic ionic worked as a framework.

We were both pleasantly surprised as to how much ionic did for a developer. I initially had thought we were going to do a lot more work in customising the components like you would with say swing, however they actually only need a little bit of styling in order to make the app look like it has been developed professionally.

An issue we did have whilst developing the app was the online documentation and tutorials. Although the actual documentation of ionic has many visual examples alongside the code that produces theses examples, we still struggled with recreating these examples with the functionality we wanted and in the context of our app. The online video tutorials helped us here, but many of these were created on older versions of the framework. So we had to do a lot of fiddling around, in order to create the app we envisioned.

Ionic - Pros

1. Hosting on local server
    1. Being able to develop the app from a local serve was a huge advantage of ionic. It meant that we did not nee to run the app through an emulator, nor did we have to manually download it on one of our devices.. We were still able to see what this app would look on various mobile devices throughout the developer mode on google chrome.
2. Cross Platform App Development
    1. One of the biggest advantages which developers talk about with ionic, is its ability to run on all platforms; web, IOS and android. The ionic framework compiles the application in a way that presents each operating system the same app. However the apps copnenets are offered to to the operating system in way which it can understand just with slightly different styles.
3. Documentation
    1. The documentation on the website is hugely helpful and there are countless different ways to produce different functionalities. There is also visual example which are presented alongside these examples. This is hugely beneficial to a developer as one does not need to trial different components, to see if they work for their app. In addition to this the forum has many solutions to common bugs and errors which developers run into.
4. External Libraries
    1. The are many open source external libraries which are very easy to include in you program. Even when the built in components do not provide the functionality you need,

chance are there is a library which someone has developed to provide this functionality. We almost used one for the tinder cards discussed previously, however our UX decision did not need this. The 3rd party components increase the power of this framework exponentially.

Ionic - cons

1. Large Changes Over Updates

    1. One of the biggest issues we had was tutorials and other information being created on the older versions of the framework. Especially at the start of the app when were trying to get external components such as firebase, working it was very hard to be able to query the database as all the tutorials we were using were on an older framework. Because the older versions had such different structure, we had to triangulate info from the new documentation and the old tutorials ignorer to get the desired functionality.

2. Low Detailed Error Messages

    1. The most frustrating part of ionic was its error messages. With syntax issues the errors were fine. However when the logic of the app was wrong, it was very hard to find the source. I think this was due to the framework structure as there were so many layers to the application, that errors could occur somewhere, and the messages would have you believe they were somewhere else. The error stack trace were usually far too long to understand. This made it very debugging very frustrating and time consuming.

3. Dependencies Not Automatically Installed and inconsistent compiling.

    1. This was very frustrating as sometimes when pulling from the repo, I would get errors and think that tom had pushed a broken version of the app. However usually it was because I hadn't run the npm install command before serving. It seems that when serving this app, it would be better if the framework automatically checked for dependency updates and downloaded them accordingly.
    2. Also the program sometimes did not compile consistently with the updated changes, due to importing new modules etc. Sometimes the saved code hadn't updated the build files, and the system would throw errors. After serving again the program would randomly work.

Ionic - conclusion.

I was pleasantly surprised with how rapidly and easy it was to develop apps in ionic. It took a lot of getting used to and was quite hard to start development having never worked with html or javascript. Once I got a feel for the framework, and had a base to work from the general feel for the program became easy to understand. Navigating from page to page, creating new components, understanding how the classes worked, slowly became a lot easier. I feel if I were to re develop the app now it would tae me a third of the time. I thoroughly enjoyed working on this app, and was pleased with the final product we produced. I am looking forward to trying out new frameworks, as now I have something to compare it to.