Joseph Lloyd 10177148

Joe.lloyd.22.24@gmail.com

Skripto 5000 final project report

DBS Higher Diploma in Science in Computing (Software Development)

24/07/2015

Supervisor: Clive Gargan

# Abstract

Scripto 5000 is an Apache Cordova app using the Ionic frame work. It is an instant messaging app powered by Node JS and Socket IO on Google Cloud. It uses its own API to make calls to a google cloud MySQL Database.

# Acknoledgments

I would like to Acknowledge Clive Gargan, Gary McNeill, Gina O’Brien, all of my lecturers at DBS and the Stack Overflow Community for their contributions towards this project.

Contents

[Abstract 1](#_Toc425510533)

[Acknoledgments 1](#_Toc425510534)

[Introduction 4](#_Toc425510535)

[Interim Recap 4](#_Toc425510536)

[Aims 4](#_Toc425510537)

[Aproche 4](#_Toc425510538)

[Background 4](#_Toc425510539)

[context 4](#_Toc425510540)

[anticipated benefits 5](#_Toc425510541)

[typical users 5](#_Toc425510542)

[development methods 5](#_Toc425510543)

[relevant existing software 5](#_Toc425510544)

[Litriture Review 5](#_Toc425510545)

[Documentation 5](#_Toc425510546)

[Visual Studio Apache Cordova 5](#_Toc425510547)

[Ionic 6](#_Toc425510548)

[Angular JS 6](#_Toc425510549)

[Socket IO 6](#_Toc425510550)

[Node JS 6](#_Toc425510551)

[Google Cloud 7](#_Toc425510552)

[Git & GitHub 7](#_Toc425510553)

[Other Litriture 7](#_Toc425510554)

[Stack Overflow 7](#_Toc425510555)

[W3Schools 8](#_Toc425510556)

[Code School Angular Tutorials 8](#_Toc425510557)

[Project discussion 8](#_Toc425510558)

[Tech used 8](#_Toc425510559)

[Apache Cordova 8](#_Toc425510560)

[Ionic 8](#_Toc425510561)

[Google Cloud 9](#_Toc425510562)

[Socket IO 9](#_Toc425510563)

[Git bash & git hub 9](#_Toc425510564)

[Walk Through 10](#_Toc425510565)

[Login 10](#_Toc425510566)

[Create Account & Edit Account 11](#_Toc425510567)

[Profile 12](#_Toc425510568)

[Contacts 13](#_Toc425510569)

[Chats 14](#_Toc425510570)

[Messaging 15](#_Toc425510571)

[Walk Though Summary 15](#_Toc425510572)

[Front-End Details 16](#_Toc425510573)

[Ionic 16](#_Toc425510574)

[Angular JS 16](#_Toc425510575)

[Back-End Details 17](#_Toc425510576)

[Compute Engine (Skripto5000’s web API) 17](#_Toc425510577)

[Database & Data Transfer 18](#_Toc425510578)

[MySQL 18](#_Toc425510579)

[JSON 18](#_Toc425510580)

[Version Control & backups 18](#_Toc425510581)

[Hardware & Debugging 18](#_Toc425510582)

[Security 19](#_Toc425510583)

[Firewall 19](#_Toc425510584)

[Login 19](#_Toc425510585)

[Session Variable 19](#_Toc425510586)

[Results & evaluation 20](#_Toc425510587)

[Testing 20](#_Toc425510588)

[Conclution & Future Work 20](#_Toc425510589)

[Bibliography 21](#_Toc425510590)

# Introduction

Scripto 5000 uses a large array of different technologies to perform its instant messaging. It currently falls short of what I had anticipated when I wrote the scope. Instant messaging from a phone app proved much more challenging than I had predicted.

## Interim Recap

At the time of the Interim report I had the basic framework built with Apache Cordova and Ionic working in tandem and deploying to my mobile phone. I had laid out a chart detailing different aspects of the app along a timeline. Unfortunately I overestimated the speed at which I could learn and implement new technologies and new app features. For example I expected the Node JS and web API to take no longer than a day but in reality this was an ongoing process that took closer to 2 weeks.

Because of the extra time spent on each task the final project only has its basic functionality. Please see appendix 1.0 the original Gantt chart now filled in with the details of how long each task took.

## Aims

After realising that I had scoped the project incorrectly I changed the Aim of the app to produce an instant messenger with basic functionality using cloud tech.

The aim is to show the use of new technology Ionic, Angular JS, Apache Cordova and Node JS. I also wanted to show the use of Cloud technologies in the form of the Google Cloud platform. I intend to show a proficiency with databases and data conversion. Finally I aim to show an aptness for version control via Git and GitHub

## Aproche

Almost all of the technology used in the project was not covered on the course. Because of this I have taken extra time to research learn and practice with things like AngularJS and Ionic. I even made several smaller projects to test basic functionality. I spent a large portion of time researching and testing tech like Socket IO and learning how Google Cloud would work with my project.

A key part of the development process was making sure I had a backup. This was achieved by committing my work to git hub every time I made some progress. By keeping all my code On GitHub I knew I had a safe place for backup.

# Background

There are many instant messaging apps out there already building Scripto5000 was a difficult challenge that required an entirely new skill set. Instant messaging is something a large majority of people use on a regular basis, because of this there is a large demand for applications that allow people to instantly communicate with each other.

## context

The app is developed in Cordova meaning it is cross platform and so fast to spread across different operating systems like windows iso and android. This way the app won’t discriminate based on the device the person uses. You can contact your Iphone friends and your Samsung friends or Nokia friends all with one app.

## anticipated benefits

The idea is that you can send messages to any other user of the app so long as you have their phone number. There is an easy to use contacts section where you can enter a number and find your friends using the app.

## typical users

Anyone who can work apps like snap chat, Facebook messenger or Whatsapp would be a potential user. Generally from ages 16-50 since everyone needs to keep in contact with each other, or people who would like an easy way to contact people across platform since everything is fed into the same database no matter what operating system your phone uses.

## development methods

The development methodology used for the project is the agile methodology. Using agile means that I created usable pieces of software within a few weeks of one another. Areas did take longer than anticipated because of the learning curve. The project does feel like its first iteration along the agile process, with its basic functionality in place.

## relevant existing software

The work environment is visual studio community 2013 with several plugins. The Apache Cordova add-on for VS2013 created the core of the project. To build a project that looks and feels good I have used the Ionic framework that is built on AngularJS.

The Node JS server and SQL database created with Google Cloud were very important pieces of software used to build Scripto5000. They built the entire backend of the project and allowed for the more complex parts of the app to work.

# Litriture Review

When building Scripto5000 I used many sources of information and documentation since the project is built with such a large array of different technologies. This section is broken down between documentation and other forms of information that I used.

## Documentation

This section will review the official source documentation for the tech used throughout the project

### Visual Studio Apache Cordova

Source: <https://www.visualstudio.com/en-us/features/cordova-vs.aspx>

The Visual Studio documentation provided me we simple instructions on how to get set up and begin using Apache Cordova in Visual Studio. It then provided me with the necessary details for installing and using the built in nu-get packages that unlock extra functionality.

It was informative but heavy. There were some outdated methods and things that didn’t match up to the current version of Visual Studio Community 2013 edition. Whilst it was a fountain of knowledge I did find myself looking for other online tutorials to do things that should have been easily shown in the Visual Studio documentation. (Microsoft, 2015)

### Ionic

Source: <http://ionicframework.com/docs/>

The Documentation for Ionic was the most clear and concise material I read in relation to the project. Each new HTML tag the ionic framework added is first explained with example code and then emulated on the right hand side to show what it looks like in practice. This was very helpful as it allowed me to build the app using the new Ionic tags confidently.

Because if Ionics explicit documentation I actually did not have to use any other source to get information on how to use the tags. (Drifty Co, 2015)

### Angular JS

Source: <https://docs.angularjs.org/guide>

The Angular JS documentation is extensive giving each area (Controllers, Scopes, Security etc.) its own page that details every aspect of it with links to examples on git hub. Each page normally starts off with the concept and some small bit of theory then it goes on to explain how angular handles the topic. It also supplies us with detailed examples of code on the page starting off relatively simple and getting more complex as we move down.

One thing I really appreciated about the Angular JS documentation was that it tells you when some element is depreciated and gives links to what should be used in the angular framework to replace its functionality. (Google, 2015)

### Socket IO

Source: <http://socket.io/docs>

I found the Socket IO documentation to be very simplistic which made it hard to achieve the more difficult messaging in the app. Because of its simplicity I was however able to follow part of the documentation and create a very basic chatroom in under an hour. I invited some friends to join and test out since I had it hosted on my Node JS instance. This was fun but also a very important part of the Socket IO learning process. Whilst it was great to build something so fast and see how the basics worked it still left me hunting online for more detailed/complex examples and explanations. (MIT/Community, 2015)

### Node JS

Source: <https://nodejs.org/documentation/>

The Node JS documentation Provides allot of detail on how to set up the server but I didn’t use it for this. Instead I used the Node JS documentation to help me trouble shoot errors I was getting as it gives details on how to make http connections and make requests. The documentation is very large and seems to cover each topic in only a few sentences not giving too much detail. This did lead me to search sites like Stack Overflow for more extensive answers to my queries allot of the time. (Node, 2015)

### Google Cloud

Source: <https://cloud.google.com/docs/>

The Google Cloud documentation provided me with the necessary steps for basic setup. There were some basic tutorials that I followed through to help me understand the Google Cloud platform. The steps to Implementing the SQL database with google cloud were explained in detail and I had no problems setting that up. The documentation regarding Node JS was limited as they didn’t have instructions on how to connect it to a SQL database and I had to look elsewhere to find that information.

Over all I found the examples to be pretty specific and only useful if you planned on doing what they had set up in their tutorials. On the other hand the built in documentation as you go through setup was very informative and made basic set up pretty strait forward. (Google, 2015)

### Git & GitHub

Source: <https://git-scm.com/doc> & <https://help.github.com/index.html>

The Git documentation was really just used as a reference point along with the setup of my GitHub repository. This documentation is clear and strait to the point, both sites make it easy to find information on any part of this version control process. They highlight BASH commands and expected outputs which made Git very strait forward to use.

As a side note since I’ve used Git a number of times before I was only referencing this material as opposed to learning a new concept from it. So in regard to Git & GitHub as reference material they were informative and easy to search for any query I had. (Git, 2015)

## Other Litriture

This section will review literature used that was not source documentation.

### Stack Overflow

Source: <http://stackoverflow.com/>

The site Stack Overflow provided me with many answers to questions I had during the development process. Some were questions I asked and some were already up there from the community. As a source Stack Overflow is only as reliable as the person making the post whether it be a question comment or answer. But because of its large and well respected community I feel that it is one of the most important sources I used in the project. When well respected and highly rated members of the community discus a problem the information can be better than the most detailed documentation. It shows real world application as well as all of the years of experience people have in their given field. Because of the reputation system it is easy to distinguish new people from veterans meaning that you can always pick a reliable answer if one exists.

Over all Stack Overflow helped me tease out some serious problems that the documentation just didn’t cover with professionals that had years of experience. On the other hand there is the issue that anyone with an account can answer a question. Because of the reputation system I don’t see this as a problem since you can disregard low ratted users answers/comments. (Comunity members, 2015)

### W3Schools

Source: <http://www.w3schools.com/js/default.asp>

The W3Schools site provided me with some great information in regards to JavaScript. It’s very readable with interactive examples. The interactive examples are really what separates this site from others as a source of information. Example code is nearly never a mirror of your exact situation, this is why the interactive code on W3S is so useful as a source. I was able to bend and modify how things were used such as how to iterate through an array of JSON data and return a result.

W3Schools provided me with some very basic understanding of new concepts that I hadn’t come across during my time on the course and provided me a testing ground for code snippets that I wanted to modify. (w3schools, 2015)

### Code School Angular Tutorials

Source: <http://campus.codeschool.com/courses/shaping-up-with-angular-js/intro>

I found a series of videos that lead you through all of the basic concepts of Angular JS. They cover the creation of a whole single page web app using Angular JS. These videos come with tests afterwards, they explain in detail the way Angular JS directives work alongside the controllers and factories to give a real responsive site.

I watched these comprehensive videos many times. They gave me an idea of what I can and can’t do with Angular JS. They also manage to explain some more difficult Angular JS concepts with relative ease making them a really important source for learning. (Code School, 2015)

# Project discussion

Details of how Skripto5000 was developed and how it works.

## Tech used

Below is a list of all the different types of technologies, languages and tools that went into the making of the Skripto5000 messaging app. Most of which were learned as part of the project.

### Apache Cordova

Apache Cordova allows access to a phones native functions via plugins added to the config.xml file. Things such as local storage and the phones camera. Because of this an app can be designed with Apache Cordova and then deployed to different platforms. Since I used the version of Apache Cordova that comes with Visual Studio I was limited to deploying to Windows, ISO or Android. (Microsoft, 2015)

### Ionic

Ionic is a front-end SDK for developing mobile apps with web technologies. This means that the majority of the app is written with HTML, CSS and JavaScript (Angular JS). The learning curve here is how the new Ionic HTML tags interact with the Apache Cordova tags to bring mobile functionality and look to the app. (Drifty Co, 2015)

#### angular js

Angular JS was by far the most difficult thing to learn on the project. It is used alongside Ionic and JQuery to extend the HTML language. It provides a client side MVC framework that lets the app take JSON data and feed it to the user depending on the request made. It also uses Controllers that help set up the page and deals with the login. The app is a single page application that loads different tabs based on requests by the user, these requests are all handled by the user. (Code School, 2015)

### Google Cloud

Because of the nature of the application I knew it had to be deployed on the cloud for the purposes of testing. To test the basic functionality it required the use of one or two phones that had the app installed and sent and received data via the cloud. This is why I chose google cloud to host the app. (Google, 2015)

#### GoogLe clouds sql database

I used a SQL database stored on google cloud to manage all of the data storage since I knew this data would have to be supplied to each user on the app. I created the Database on the cloud and managed it with MySQL Workbench where I could locally insert data or build tables and relationships and then save the changes to Google Cloud SQL database. (Google, 2015)

#### Google clouds compute engine (Node JS)

Node JS was necessary for data conversion originally but as I learned more about its functionality it became one of the most important parts of the app. It was on Google Clouds Node instance I created the web API by using the express.js framework. This gives the app its most complex functionality. (Google, 2015)

### Socket IO

I used Socket IO my web API and in the Angular JS controllers to send and receive messages instantly. This is done by getting my Node JS server file to look for incoming sockets and then sending the message onto the user that is listening on that channel. (MIT/Community, 2015)

### Git bash & git hub

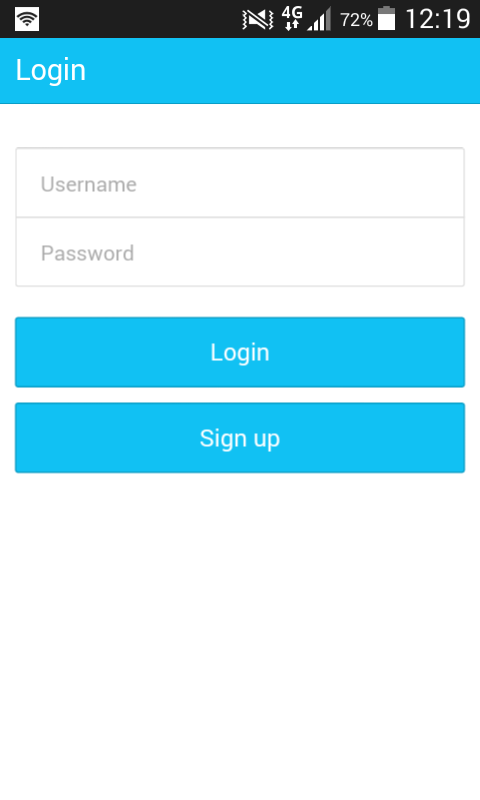
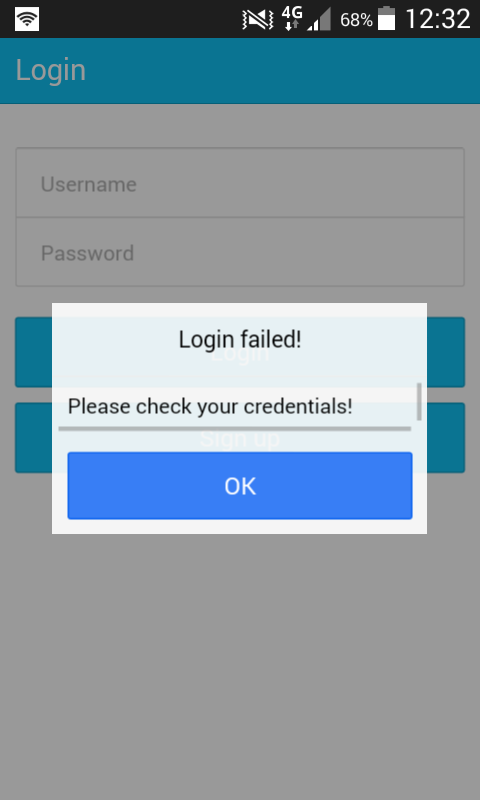
For version control and code backup I user the GIT born again shell to upload my code to GIT hub where its east see changes made to code and review things that I have changed since the last time I uploaded code. GIT was one of the most essential tools used through the development process. (Git, 2015)

## Walk Through

This section describes with screenshots what each part of Scripto5000 does.

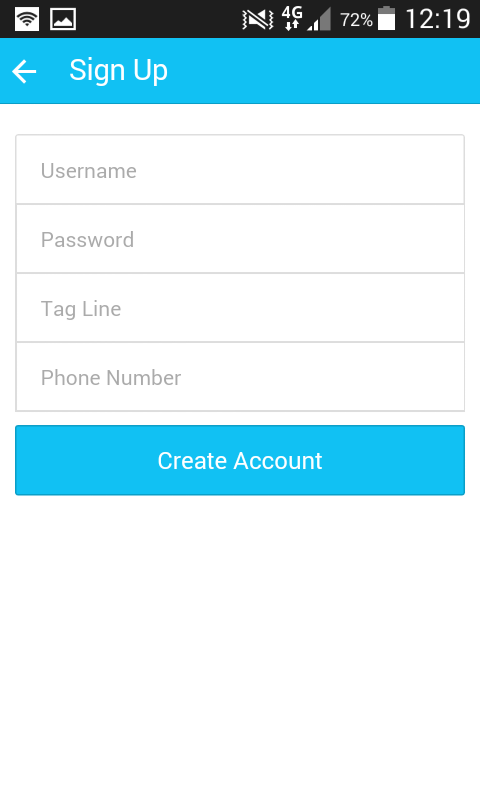
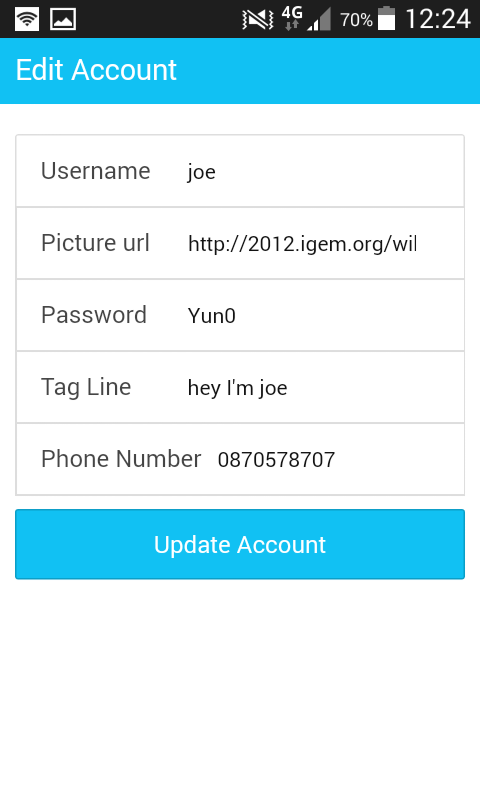
### Login

The login screen is very simple but clean shown on the left. The entry field for password is protected with the standard password input to stop someone seeing your password if they look over your shoulder. After inputting the password and username you are taken to the profile page but if you input incorrect credentials you are presented with the popup shown on the right. Then you are prompted to enter your password and username again.

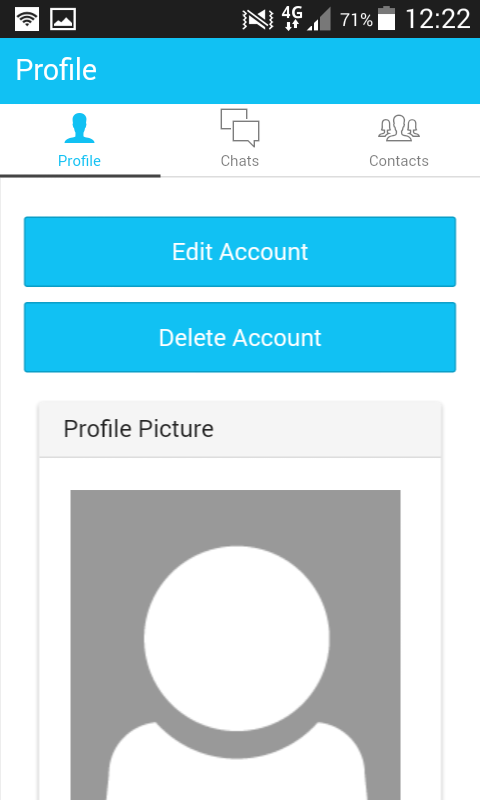
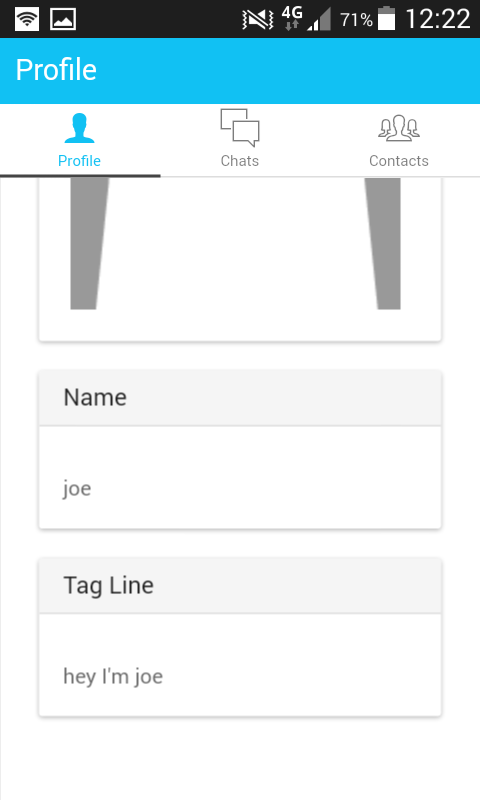
### Create Account & Edit Account

If you select the “Sign up” button below “Login” you are taken to the create account page, here you can register your details for an account as seen on the left. If you wish to edit your account you can go to your profile and press the “Edit Account” button. It will populate a set of fields with your account details and let you modify them then save them. Seen on the right.

### Profile

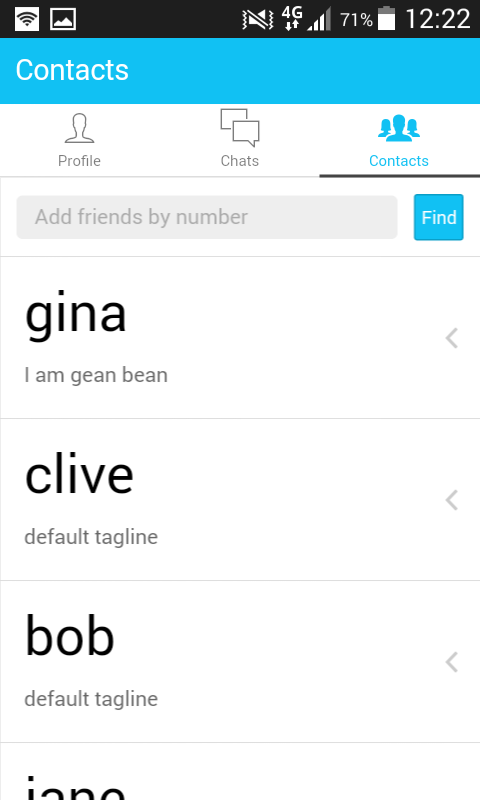
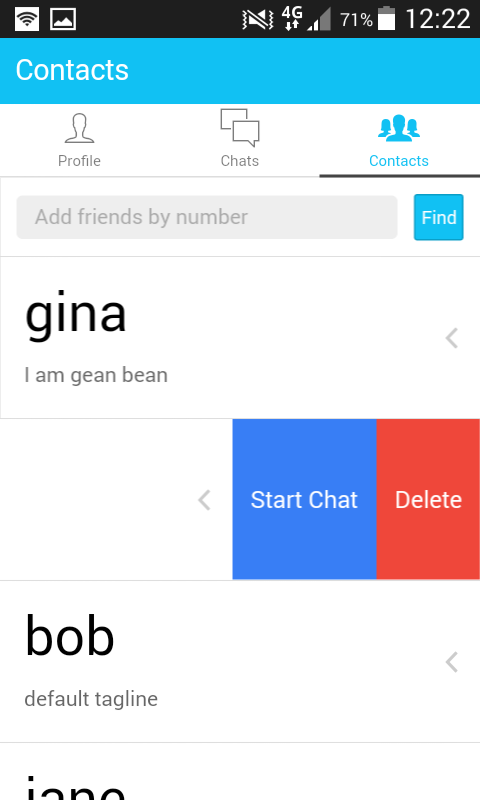
The profile page lets a user see his/her current profile picture name and tagline. As well as viewing the user profile there are two buttons, one to delete the account and one that leads to the edit page shown above.

### Contacts

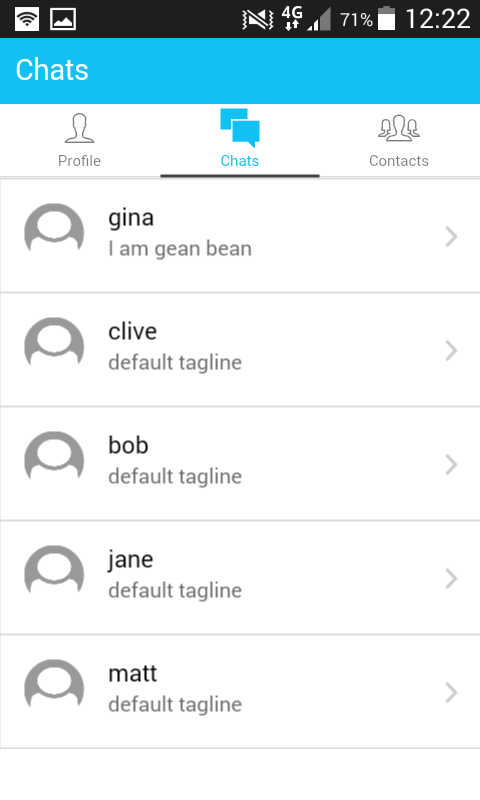
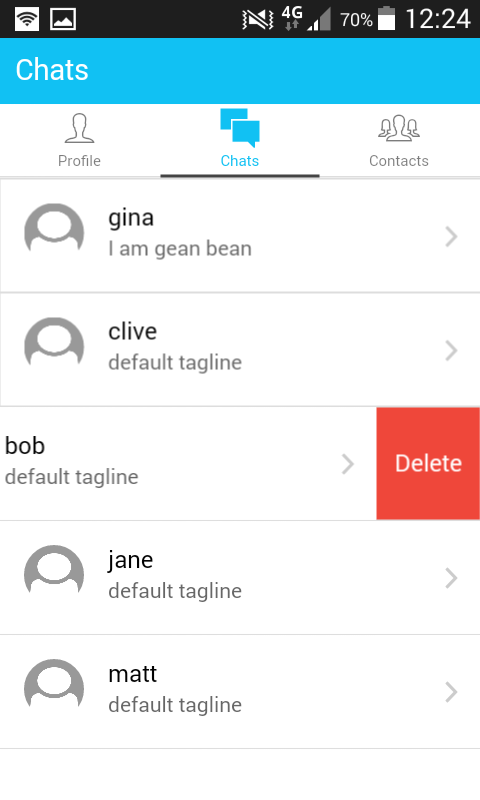
The contacts page shows a list of all you friends, it also allows you to add a new friend via a phone number at the top. The number entered is checked in the database and then if it is found the user is asked if they are sure they want to add this person as a friend. If yes is selected then a new friendship is created in the database and the next time the contacts page is loaded your new friend will appear in the list of contacts.

In the contacts list you can also delete a friend or add them to your chats. This is shown on the right, you can select a user and swipe right to reveal a menu. The menu is animated and slides out with the users movement.

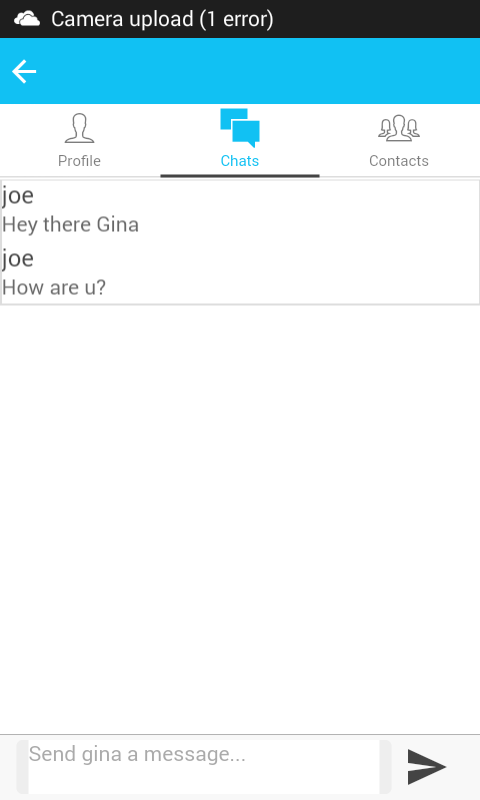
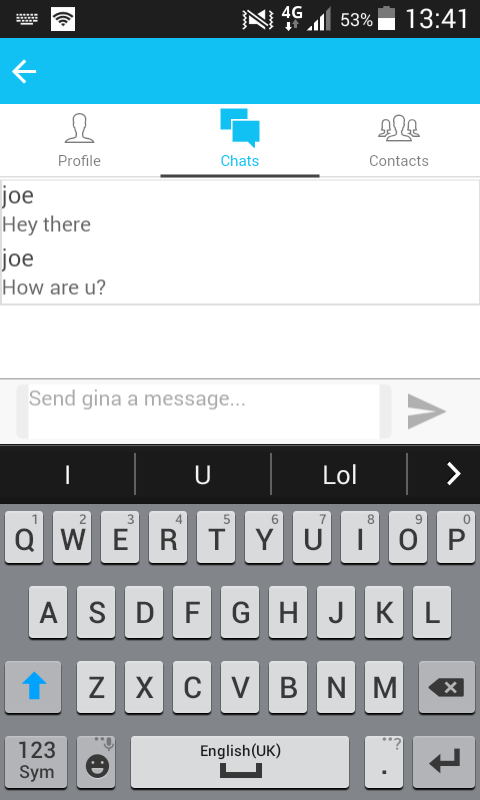
### Chats

In the chats tab you can see everyone who you have engaged in a chat with, it is from here you can select a person to send a message. You can also slide the chat to the left, similar to the contacts page, to delete a chat.

### Messaging

The final page in the application is the messaging page. The input is a footer so it always sits at the bottom of the page. New massages are instantly sent and received by the users connected. The messages are just appended to the list that can be viewed here.

### Walk Though Summary

The walk through shows each page and what it can do on a basic level. This give a good overview of how the app works but does not replace a hands on look as these still images don’t show animations or live message sending like a hands on test with the app can.

## Front-End Details

The frontend was built with Ionic as I mentioned in the Tech used section.

### Ionic

The Ionic SDK gives the HTML new tags that it uses to make the HTML more expressive. Ionic brought so much functionality to the table as well as a really clean look to the whole app with animations and native looking menus and buttons. But there was a learning curve that came with it. Ionic added so many new tags to the HTML it was nearly like learning a new language.

Ionic pages in the app are all rendered into the main page by the app.js file that controls the routes and tells the app what content to load based on what the user requests. The thing that makes Ionic so powerful as an SDK is its coupling with Angular JS.

### Angular JS

Angular JS is really where all of the “magic” happens in the front end. There are three core files that were populated over the course of the project, they were “app.js”, “services.js” and “controller.js”.

#### app.js

The “app.js” is where the “Ionic Platform” is initiated and the “State Provider” is built it is also where Angular JS is declared and links to the “services.js” and “controllers.js” are made. Ionic Platform lets the app know that it uses the Ionic SDK. State Provider lets the app dictate what templates to load and what controllers to attach to each template. The final thing that app.js does is initiate the session variable, this is the variable that is gotten from the database when a user logs in and is unique to each user. (Angular, 2015)

#### services.js

The “services.js” file contains a series of services and factories. The Angular JS documentation describes a service as “Angular services are substitutable objects that are wired together using dependency injection (DI). You can use services to organize and share code across your app.” (Angular, 2015). For the purposes of Scripto5000 there was only one service, a login service that accepted a password and username. The service then sends a request to the web API and loops through the results trying to find a match. Finally it returns a “promise” either true or false and sets the session variable accordingly.

The other blocks of code in the “services.js” file are factories. According to the Angular documentation “The Factory recipe constructs a new service using a function with zero or more arguments (these are dependencies on other services). The return value of this function is the service instance created by this recipe.” (Angular, 2015). All of the factories built for Scripto5000 make some sort of custom call to the Node JS web API. Most of the calls send or receive JSON data, an example of this would be the ‘User’ factory. The ‘User’ factory sends a request to the web API with one argument the session object. It then gets a response of JSON data containing all the profile information about the user.

A more complex factory would be the one used to wrap around the Socket IO requests. This factory, just like the others, makes a request to the web API. A different function in the same factory receives the request and then returns the value to any controller where it was instantiated.

Over all services help maintain clean reusable code. It’s like creating a series of functions that can be called in any controller that has the name of the factory as one of the function arguments. This was a pretty big learning curve and didn’t become apparent until towards the end of the project. As such in the final code there are things in a few of the controllers that should be refactored to factories. (Angular, 2015)

#### controllers.js

The Angular JS documentation define a controller as “a JavaScript constructor function that is used to augment the Angular Scope” (Angular, 2015). In Scripto5000 each controller is linked to a route from the app.js file as mentioned above. When the route is loaded a controller is loaded along with it. Controllers manipulate the Angular scope like making a button perform an action or capturing some JSON data.

In skripto5000 there is a controller for each page. The most basic thing each of these controllers does is declare some scope object and send a request to the web API through a factory. The scope object is then populated with the JSON data that the factory receives. In the view for the page the scope object is shown to the user. For instance the url for the profile image is scoped and put into an image tag in the view so as to make it appear as an image.

Controllers also provide a way to take data and send it to the database. When editing an account or making a new one data is captured with a button press that is part of a controller’s scope. This data can then be fed to a factory. The factory sends the JSON to the web API to be stored. (Angular, 2015)

## Back-End Details

The back-end of Skripto5000 uses two Google Cloud technologies, the compute engine and the SQL database. The database has its own section on data transfer later so this section will mainly focus on the compute engine.

### Compute Engine (Skripto5000’s web API)

The instance of Google Cloud Compute engine has Node JS installed as well as some other modules required to run Skripto5000

* Express
* MySQL
* Socket IO
* VIM (to make modifying the server.js easier)

There are two main files that were written entirely on the command line that are essential for skripto5000 to run, these are package.js and server.js. The package.js file just helps to set up the environment, it has all the correct versions of the modules needed to run the server.js file.

The server.js file is probably the most important file in the entire project. The server.js file is run on the Node JS instance with the command “node server.js”.

The server.js file is skripto5000’s web API. It starts off by setting variables that it uses throughout for express, database connection and Socket IO. It then uses its connection variable to make a connection to the Google Cloud SQL database. The API has been set up with console.log() messages that are printed out when actions are complete. In the case of database connection a line is printed to the console that says “connected to database”, this way the instance of Node JS can be easily monitored.

After seeing the message for database connectivity there are two other messages that are printed out. They represent the ports that the app is listening on. For basic API calls Scripto5000 is set up to listen on port 3000, these calls deal with SQL calls to the database that return some JSON data to the user. The other port Skripto5000 listens on is 3001. Skripto5000 uses this port exclusively for Socket IO calls. Any message that is sent by a user is picked up on port 3001 and emitted back to the users from this port. The message is also printed to the console with user id’s so data can be tracked when looking at the console.

The server.js file was the most complex and difficult part of the project, trying to make a connection to the database and then feed the correct data to the user with no debugging help from the console was a really big challenge especially when combined with the fact that Node JS Socket IO and Express were never covered on the course. (Google, 2015)

## Database & Data Transfer

I deal with two types of data in the project, JSON and SQL data. Converting between the two proved a challenge but with the help of my web API I was able to quite effectively transfer data from the MySQL database to the app.

### MySQL

The MySQL database was hosted on Google Cloud and was built with MySQL Workbench. I created a connection to Google Cloud SQL from the MySQL Workbench and manipulated data locally then pushed it up to the cloud after it was saved. (Oracle, 2015)

### JSON

Angular JS is very compatible with JSON data, as to be expected since they are both JavaScript. My web API was able to make SQL queries to the database and return some data. It would then take that data and convert it to JSON and send it to the app requesting it. This is the principal the entire app works off from the login request to the contact list. Each page full of data that is pulled from the database and turned to JSON and then served to the user.

## Version Control & backups

For version control I used Git BASH to commit to an online repository on GitHub. This was essential to keep code safe and backed up as well as seeing what I’ve changed over the course of project. This is a link to all the commits I made to the project.

<https://github.com/joeLloyd/Scripto5000/commits/master>

I only made commits when there was something of value added to the project. As can be seen by the commit dates I didn’t commit between 28th of May to the 26th of June, this is because I spent near 3 weeks working on Google Cloud which was outside of the main code (building my web API).

For the web API, I made local copies of the server.js file and managed my database with the local tool MySQL Workbench. Both of these were deployed on the cloud but had local backups.

## Hardware & Debugging

Setting Skripto5000 up took some time, the Apache Cordova plugin for visual studio had to be installed first and then the Ionic nu-get package along with the Ionic template that gave the basic file structure to build Skripto5000. This was all done on windows in visual studio. Part of the Apache plugin adds an emulator for the phone, this is called the ripple emulator but I found it to be unreliable and clunky. Sometime was put into getting Skripto5000 onto a real mobile device, driver for the device had to be installed and the developer mode had to be unlocked on the phone.

For a real test ground two “Samsung galaxy core primes” and a “Samsung galaxy young” were used. Skripto5000 could be directly built on the phones and appears as a normal android app on the phone after it is built. Because the app could be built and stored on the phone testing the instant messaging was fairly strait forward. It was time consuming as even one small change that was made had to be built to both of the devices.

The debugging process was more difficult since there is no console output on the phone. The app would have been next to impossible to debug without breakpoints. Using the breakpoint in the correct places it was possible to see the value of different variables in a “frozen” state. The best example of this process in the project was the first time JSON data was being pulled from the Node JS server. Putting a break point on the scope object receiving the data I was able to see that it was correctly being pulled to the app. With this discovery I was able to focus on a different part of the project to fix the problem.

## Security

This section will talk about the security of the app mostly in relation to the weaknesses in the system.

### Firewall

As part of my Google Cloud setup I had to configure special fire wall rules to allow users to make JSON requests to my web API. This was a protection put on my Instance of Node JS. I created the Firewall rule to allow any IP address to make a request to the instance of Node JS so they could access my web API.

The instance of MySQL was handled differently, I allowed only two addresses to access the database. The first was the IP address of my Node JS instance so as it could make calls to the database when an app requested it. The second was my own local IP address so as I could access the data from home via MySQL Workbench.

### Login

The login verification for the app is done in a vulnerable place. After setting up the basics of the app and the Node JS instance I moved onto the login function. This took some time since it was the first real use of Angular JS and the Node JS instance. When I finally got it to work I had built an insecure login since it calls passwords and usernames and iterates through them in the app instead of just returning a promise from the server.

Passwords are stored and read as strings from the database, I did not have the time to encrypt passwords before sending them to the database with Angular JS as the login took me longer than anticipated.

I do now know how to create a server side login and plan on doing this as a first task after the hand up as future work.

### Session Variable

The session variable is held in a $rootScope variable that is unique to each user and comes from the database. It should be stored as a local storage on the phone with the ng-cookie directive from Angular JS. I did originally attempt to store the session with ng-cookie but I didn’t get results quick enough and knew I needed to move onto much more important functionality. I used $rootScope from Angular JS to store the session object because just like a cookie it would be accessible everywhere I needed it.

# Results & evaluation

Skripto5000’s development resulted in a clean but basic instant messenger. The learning results were huge as well, I feel comfortable working with Google Cloud, Node JS, Angular JS, Ionic and Apache Cordova after using them for the past three months.

Skripto5000 was somewhat of a disappointment since I didn’t reach my original goal of creating destructible messages or text message encryption. It was much more difficult building an instant messenger than I had originally thought. Especially when deploying it to the cloud instead of running it with a local host.

## Testing

Angualr JS is designed to be easily testable by writing some unit testing controllers. Unfortunately because of time constraints the code was not run through any unit testing. Since this was my first Angular JS project I felt that it was more important to understand the basics of the code instead of add the testing variable from the beginning since writing my own tests would have proved a challenge in itself.

Another option would have been to write “browser-bots” with selenium testing. These are sets of simple commands written in Java that navigate through the project and produce a pass or fail result based on the task they have been given.

The testing done for the project was purely manual due to the time constraints.

# Conclution & Future Work

In conclusion Skripto5000’s development was challenging but rewarding and I see it as a good piece of software with potential to be much more. Whilst I didn’t meet all the goals that I set in the beginning I feel that I’ve learned a huge amount and some totally new skills.

I plan on using Skripto5000 as my portfolio piece to show any potential employers and as such I will be adding to Skripto5000 and continue to pay for my Google Cloud account. The reason I feel Scripto5000 is a good portfolio piece is that it has a well-built front-end and a well-structured back-end.

Future work I plan on doing is as follows

* Refactor code from controllers to factories
* Build a set of unit tests for all the Angular JS code
* Add functionality for multi user chat
* Build a cleaner chat area with “chat bubbles”
* Send image messages
* Send recorded messages
* Create destructible messages

That’s really the plan for the next 3 months, getting all of the above added to the project will allow me to release Skripto5000 on to the play store for free download.

# Bibliography

Angular. (2015). *Angular: Developers Guide: Services*. Retrieved from docs.angularjs.org: https://docs.angularjs.org/guide/services

Angular. (2015). *Angularjs: Developer Guide: Providers*. Retrieved from docs.angularjs.org: https://docs.angularjs.org/guide/providers

Angular. (2015). *AngularJS: Developers Guide: Controllers*. Retrieved from docs.Angularjs.org: https://docs.angularjs.org/guide/controller

Code School. (2015). *Angular JS Tutorial*. Retrieved from Code School: https://www.codeschool.com/courses/shaping-up-with-angular-js

Comunity members. (2015). *stackoverflow.com/questions*. Retrieved from stackoverflow.com: http://stackoverflow.com/

Drifty Co. (2015). *Ionic Documentation*. Retrieved from Ionic.com: http://ionicframework.com/docs/

Git. (2015). *Git Doc*. Retrieved from GitHub: https://github.com/

Google. (2015). *Angualr js: Documentation*. Retrieved from angularjs.org: https://docs.angularjs.org/guide

Google. (2015). *cloud.google.com/nodejs*. Retrieved from google.cloud: https://cloud.google.com/nodejs/

Google. (2015). *Google Cloud Computing*. Retrieved from cloud.google.com: https://cloud.google.com

Microsoft. (2015). *Visual Studio Tools For Apache Cordova*. Retrieved from visualstudio.com: https://www.visualstudio.com/en-us/features/cordova-vs.aspx

MIT/Community. (2015). *Socket IO docs*. Retrieved from socketio.com: http://socket.io/docs.com

Node. (2015). *Node JS Manual & Documentation*. Retrieved from nodejs.org: https://nodejs.org/api/

Oracle. (2015). *mysql doc*. Retrieved from dev.mysql.com: http://dev.mysql.com/doc/

w3schools. (2015). *JavaScript Tutorial*. Retrieved from W3Schools: http://www.w3schools.com/js/