Tag Buddy 

TagBuddy

A Safer Messaging App for Kids

Author: Niall Meade

Year: 3rd Year

School: Ardscoil Rís, Limerick

Date: January 2019

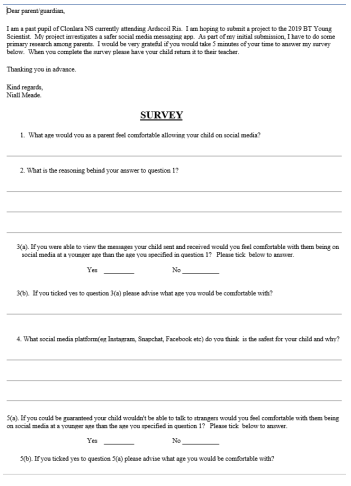
Table of Contents

Introduction

I thought of the idea for TagBuddy after an internet safety talk we had in Ardscoil Rís last year. One of the topics the speaker spoke about was how anyone could be contacted online by a stranger. With this in mind, I thought there should be a safer way of using a messaging app without the risk of being contacted by a stranger, TagBuddy was the solution. TagBuddy is a safe and secure, NFC (Near Field Communication) authenticated messaging app designed for kids. The core idea behind TagBuddy is the only way two users can message each other is by "friending" each other via NFC. This would prevent a user from messaging people they have never physically met thus making it harder for strangers to message you. In a time where child internet safety is a prominent issue this app offers a viable solution to preventing children/teens being contacted by strangers or child predators online. A study conducted by Cybersafe Ireland in 2018 found that 34% of pre-teens in Ireland are in regular contact with a stranger online. To put that in perspective if you were to walk into the average class room of 27 in Ireland today and ask the pupils if they are in regular contact with a stranger online, 9 pupils would raise their hand.

Proposal

The aim of my project is to create a safer social media/ messaging app for kids. The only way a child would be able to add a friend is by tapping their phone against their friend's phone and via NFC they would add each other. This means you would have to meet someone in real life to add someone eliminating the risk of a child talking to strangers online. So far I have conducted a survey at my local primary school asking parents various question regarding social media. The results showed that on average parents felt comfortable allowing their child online at 14.07 years old. However, after I proposed my idea of their child not being able to talk with strangers and allowed the parents to monitor all chats the age dropped to 11.43, a difference of 2.64 years. I am also working on a basic app that will send a command via from one phone to another turning on a light on the other phone.



Dear parent/guardian,

I am a past pupil of Clonlara NS currently attending Ardscoil Rís. I am hoping to submit a project to the 2019 BT Young Scientist. My project investigates a safe social media messaging app. As part of my initial submission, I have to do some primary research among parents. I would be very grateful if you would take 5 minutes of your time to answer my survey below. When you complete the survey please have your child return it to their teacher.

Thanking you in advance.

Kind regards,
Niall Meade.

SURVEY

1. What age would you as a parent feel comfortable allowing your child on social media?

2. What is the reasoning behind your answer to question 1?

3(a). If you were able to view the messages your child sent and received would you feel comfortable with them being on social media at a younger age than the age you specified in question 1? Please tick below to answer.

Yes _____ No _____

3(b). If you ticked yes to question 3(a) please advise what age you would be comfortable with?

4. What social media platform(ie Instagram, Snapchat, Facebook etc) do you think is the safest for your child and why?

5(a). If you could be guaranteed your child wouldn't be able to talk to strangers would you feel comfortable with them being on social media at a younger age than the age you specified in question 1? Please tick below to answer.

Yes _____ No _____

5(b). If you ticked yes to question 5(a) please advise what age you would be comfortable with?

Project Objectives

1. Create majority of the app as a web app because I'm most comfortable using HTML and CSS
2. Research how NFC works on Android and how the NDEF (NFC Data Exchange Format) data protocol is used when programming
3. Research current "Children's Internet Protection Act" and the recent changes made by the EU
4. Research how I can use ionic framework when creating my web app so everything scales properly depending on screen size and resolution (the web app should look the same across all devices)
5. If time allows I would like to create a parent app that allows the parent to view the messages the child has sent and received.

Timeline

End of September and October: Receive all survey results back. Finish work on basic communication via NFC between 2 android devices.

November: Complete web app with HTML and CSS

December: Bring basic communication over NFC together with the web app and troubleshoot any problems. Begin work on the report.

Late December to early January: Finish the report, make the poster and prepare for RDS presentation by making everything look more professional all around

Conclusion to proposal

Internet safety is extremely topical at the moment, especially with children. From my experience kids are expected to be on social media due to peer pressure. Some parents don't allow the kids online and this excludes them from their friends. This is one of the reasons why I think a new social media app is needed for kids that don't have the risks of Snapchat or Instagram.

Aims

1. Develop a safe/secure child-friendly messaging platform prototype.
2. Investigate existing messaging services and their anti-bullying protocols.
3. Investigate features to make messaging services more child-friendly
4. Develop a quick, secure and appealing app using asynchronous design

Experiment

Before I went about building the app, I wanted to know what the parents thought of social media and if they thought my proposed app could be useful. I asked the following questions:

1. What age would you as a parent feel comfortable allowing your child on social media

2. What is the reasoning behind your answer to question 1?

3. If you were able to view the messages your child sent and received would you feel comfortable with them being on social media at a younger age than the age you specified in question 1? If yes, what age?

4. What social media platform (eg Instagram, Snapchat, Facebook etc) do you think is the safest for your child and why?

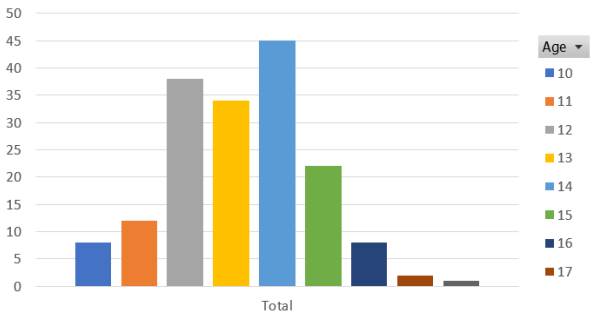
5. If you could be guaranteed your child wouldn't be able to talk to strangers would you feel comfortable with them being on social media at a younger age?

5b. If yes, what age?

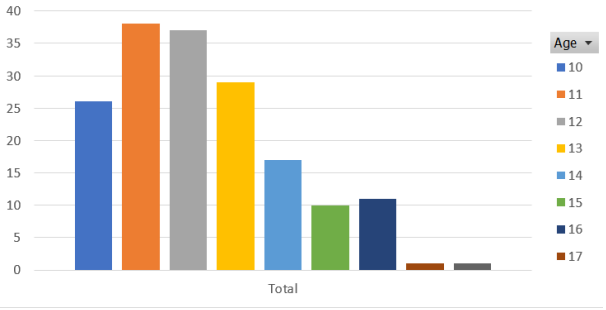
After asking these questions to 170 parents at my school, the results I found were promising. The average age a parent was comfortable allowing their child on current social media was 14 years old. However, once I proposed my idea the age dropped to 11.4, a difference of 2.6 years. I felt that this difference was significant enough to prove that the app would be used. In

the plots below, it's clear that the TagBuddy concept causes the data to significantly skew left compared to the original which is almost a symmetrical shape.

Before



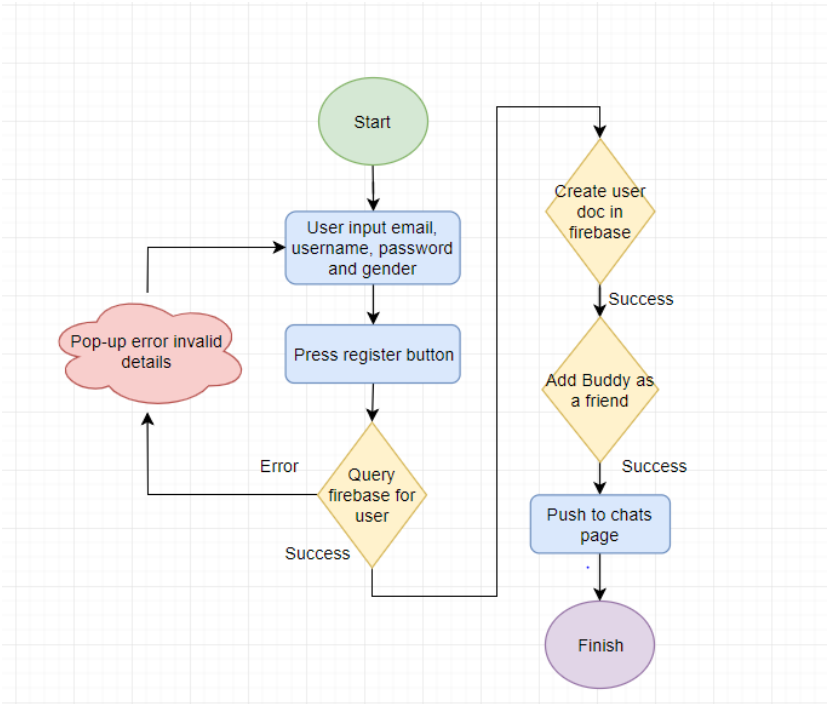
After



App Design

When I set out to start creating the app, I wanted to understand the logic behind them. To this I decided I would make some diagrams to explain them. I did some research and I found flow, sequence and Jacobson use case diagrams. Flow diagrams are used to explain how the code or app flows in terms of different states. Sequence diagrams are like flow diagrams except it's harder to show asynchronous functions such as promises or observables as the name suggests they are just sequential. Jacobson use case diagrams are used mainly to show what a user or the app can do. This helps informs functionality needed for the app and user.

Register Account



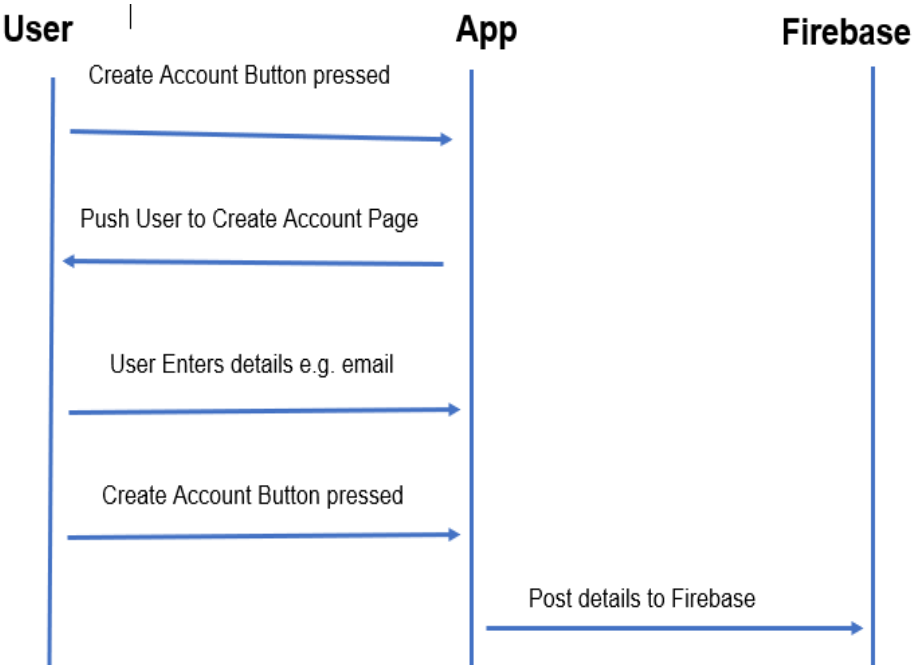
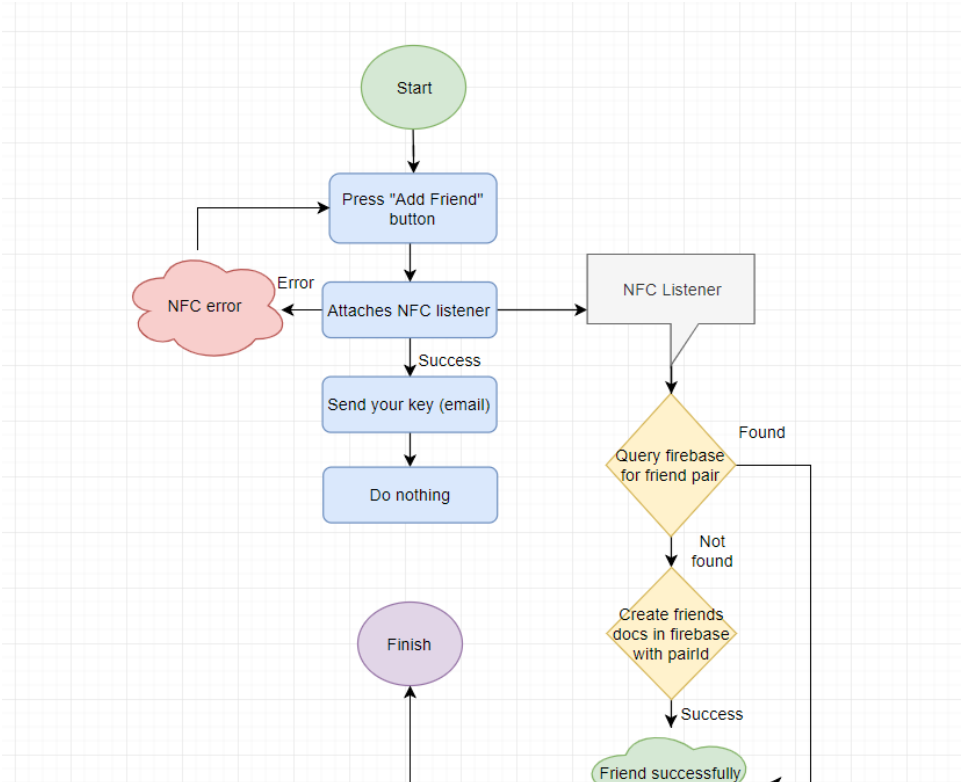


fig x. Register account sequence diagram

Login and Add a Friend



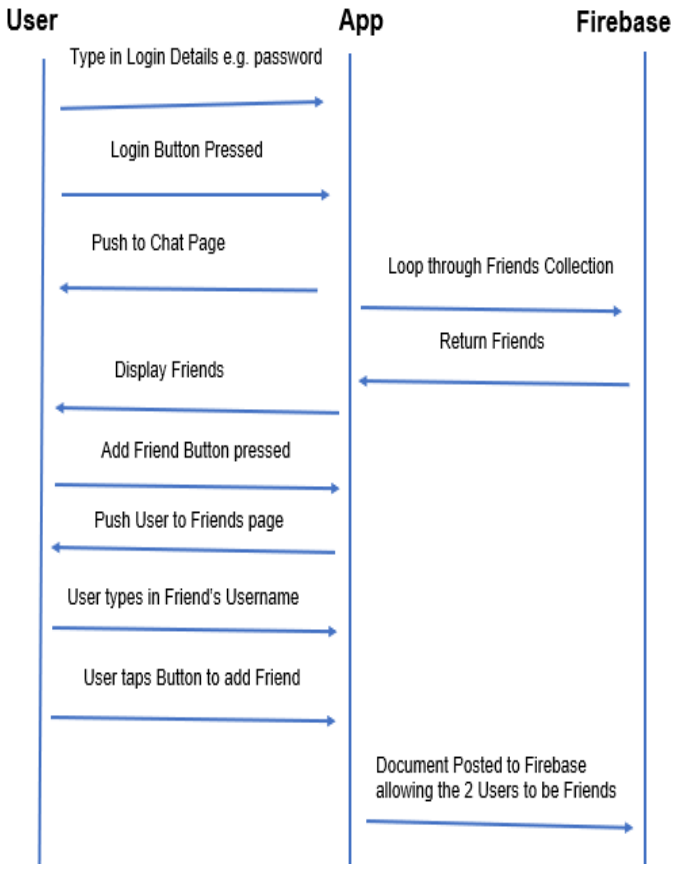
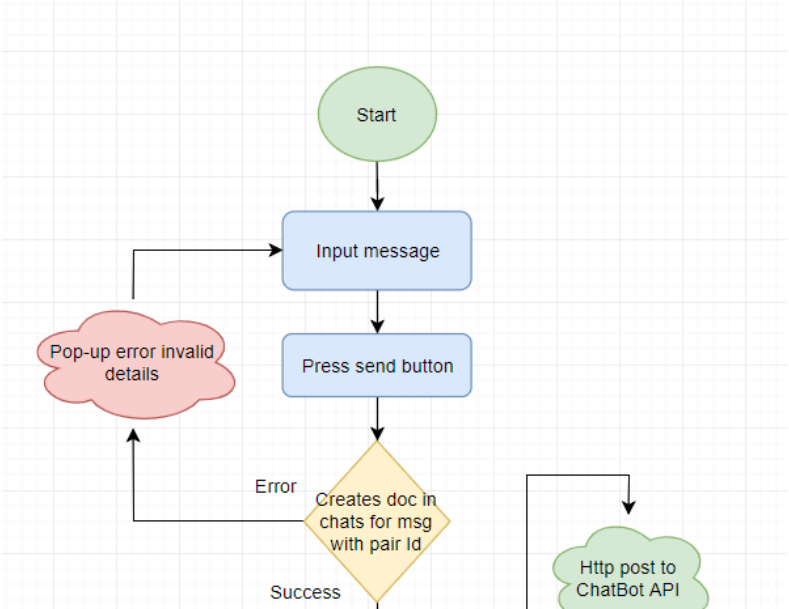


fig x. Login and add friend sequence diagram

Login and Message a Friend



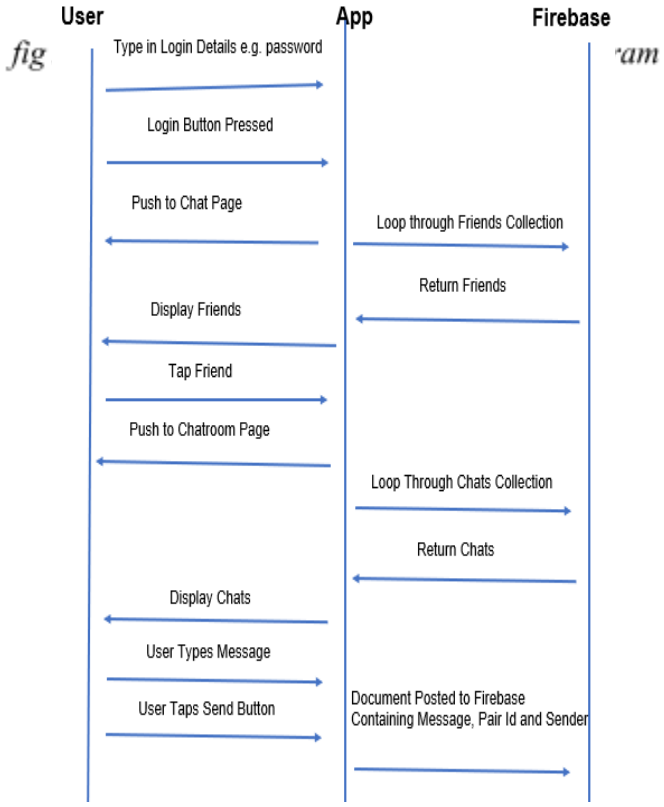
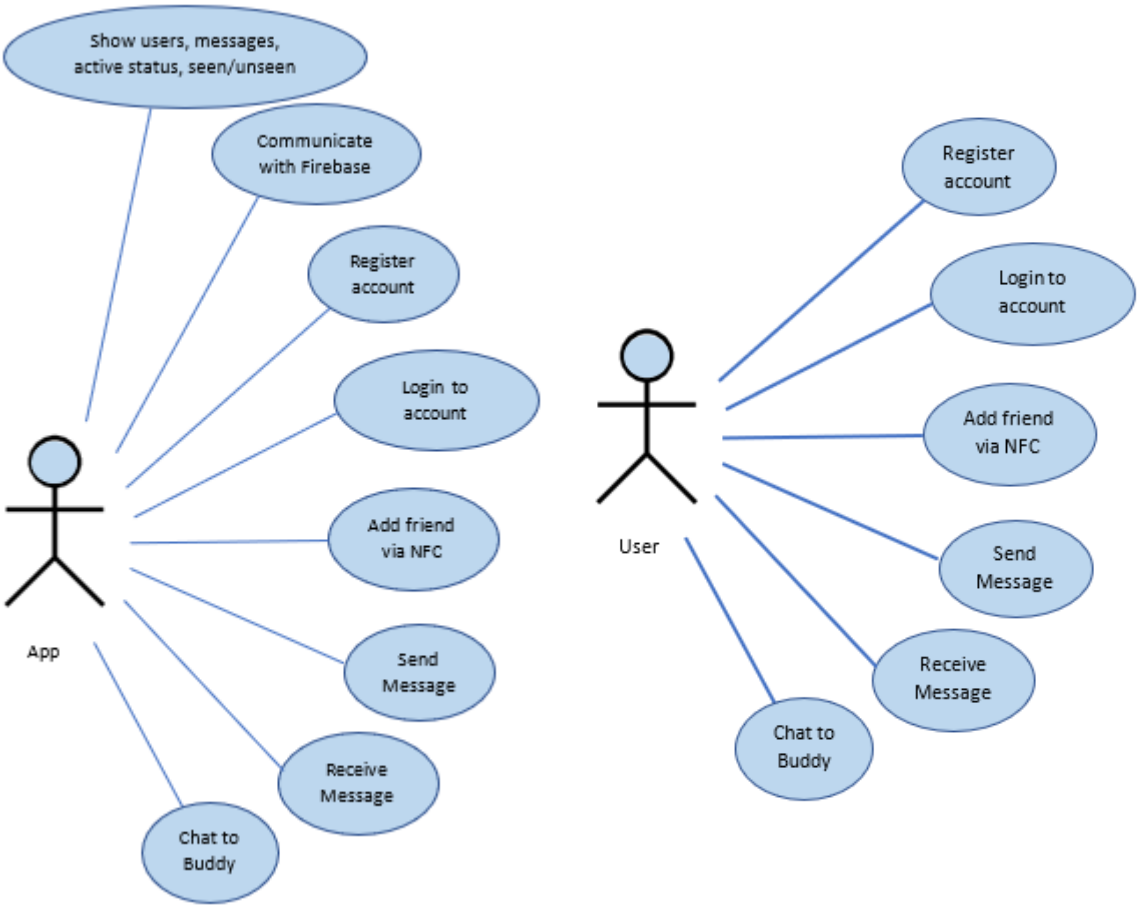


fig x. Login and message friend flow/state diagram

These diagrams helped me when it came to writing the code in my project. If I hadn't created these diagrams, I think I would have found it a lot harder to keep focused when coding. Although I feel there is lots of room for improvement in the final code it reflects the flow shown in these diagrams.

Jacobson Use Case Diagrams

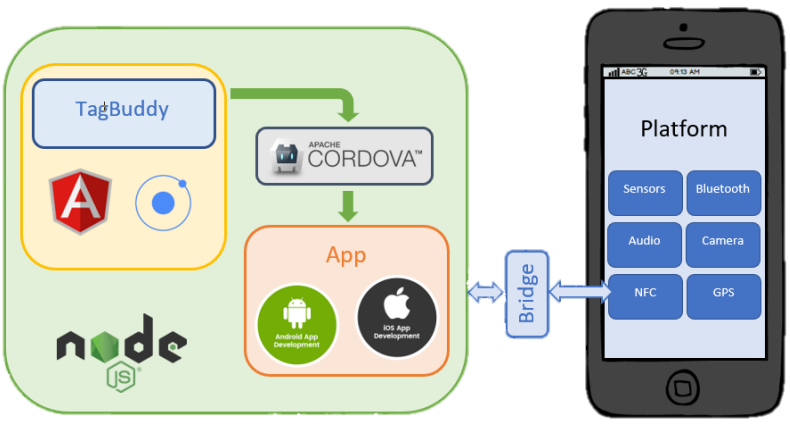


The use case diagrams helped me to understand what functionality both the user and app have when I started development. If I look at my code now nearly all the speech bubbles are represented as functions in my code. Again, this diagram helped me to focus the way I was developing my app and make sure I stayed on the right track.

Technology

Ionic, Angular and Cordova

Ionic is a free open source software development kit for building hybrid mobile apps. This means that the app will look the same and scale correctly across all platforms because it is a hybrid app. A hybrid app performs similar to a native app (designed for a specific platform e.g ios) but the code in a hybrid app consists of a mixture of a web app and a native app. Ionic also offers a variety of native plugins such as firebase, PhoneGap and HTTP which I used in the app. Ionic is based on Angular JS which is a JavaScript based open source front end web application framework. Ionic and Angular together create a web app framework which is then packaged by Cordova into an apk in my case for android. A "bridge" allows the apk/app to communicate with the phones sensors, GPS, NFC, Bluetooth etc.



Caesar Shift Cypher

With safety and security being the core idea of my project, I felt it was important to keep all data saved to my database (firebase) encrypted. To do this I used a function called "Caesar Shift". Caesar shift takes a string and encrypts it by changing each letter based on what number it comes in the alphabet. For example, if I was to shift the word "apple" by 1 it would end up being "bqqmf". This appears to be gibberish. When a user posts any messages, emails, usernames or anything else to the database it is run through Caesar Shift at a shift of key=5. When any information is called from firebase it is also run through Caesar shift this time with a shift of -5 to decrypt the information.

Firebase

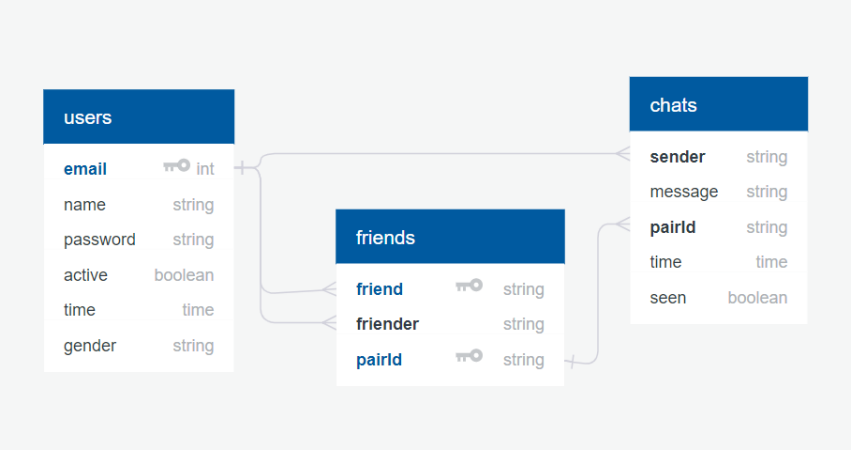
Firebase is a database/mobile and web app development platform that I will be using for my back-end. To use firebase I had to run the following commands in terminal.

1. ionic cordova plugin add cordova-plugin-firebase
2. npm install --save @ionic-native/firebase

Before I could call firebase commands I had to tell it where my firebase app was (API key). To do this I had to add this code to app.config file. After running these commands and adding the code I needed to import firebase into the project, I was then able to post and call from firebase. Within firebase you have collections, within these collections you have documents that contain fields of strings, numbers etc. In my firebase project I have 3 collections called users, chats and friends. When a new account is created it creates a new document with all your details in the users collection. The code I use is "*this.db.collection("users").doc(this.email).set(payload)",* breaking this line down, this.db is calling the firebase module, .collection(users) directs it to the collection users, and

.doc(this.email).set(payload) is creating a new document called this.email which is the users email and sets the fields in the document to the payload which is the email, username and password. When the user enters the chats page it lists all the chats they have access to or all the users they are friends with, to do this it queries the friend collection and loops through each document and returns it if the friend field is equal to "this.chatuser"(the users username). All collections are linked to each other via a key. For example, the email field in users is linked to either the friend or friender field in the friends collection or the sender field in chats collection. These keys are used to reference back to a different collection, for example when the list of friends is pulled from the friends collection on the chats page it does not know if the friend is active. So it takes the friend field from the returned friend document and queries the users collection looking for a match. It is only a match if the friend field is equal to the email field, if it is a match it returns the user.

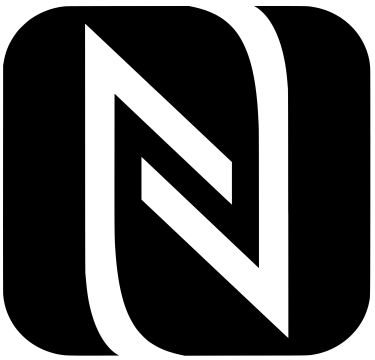
```
var config = {
  apiKey: "<API_KEY>",
  authDomain: "<PROJECT_ID>.firebaseapp.com",
  databaseURL: "https://<DATABASE_NAME>.firebaseio.com",
  storageBucket: "<BUCKET>.appspot.com",
};
firebase.initializeApp(config);
```



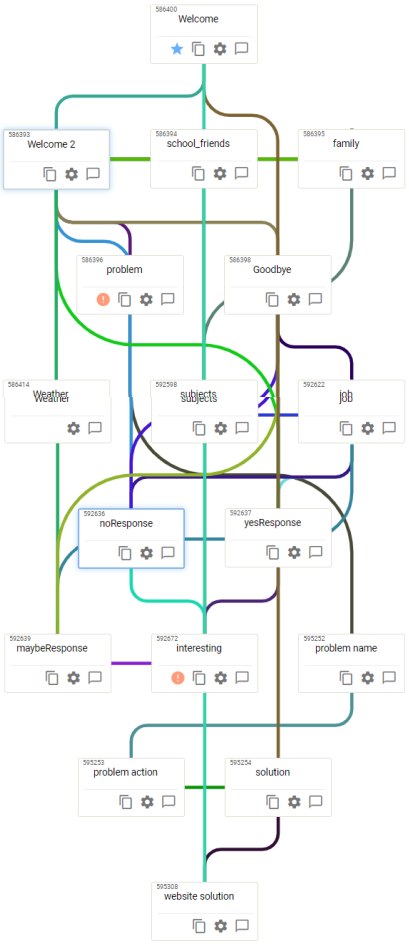
NFC

NFC stands for near field communication and is a way of two devices communicating over a short range of 0 to 5 cm. It is a set of communication protocols that enable two electronic

devices to exchange data. NFC is a form of RFID technology, but can both send and receive a signal. As the IOT (Internet of Things) becomes more and more relevant in the technology industry NFC is at the forefront of it and is in more and more phones. I decided to use NFC in my project as it is the safest way to send private user data to another device. To send and receive NFC signals I had to use the NDEF (NFC Data Exchange Format), this was put in place to allow all NFC capable devices to communicate with each other even if they were on different platforms e.g. ios and android. To implement NFC in my project I used the phonegap NFC plugin.



AI ChatBot



The core idea behind TagBuddy is to create a safer messaging platform. I felt that a chatbot could be used to talk with kids and collect information such as if the child is being bullied. The chatbot or "Buddy" would use AI (Artificial Intelligence) to conduct a conversation and ask different questions such as "How was school today?". "Buddy" has a schema as shown on the left. This means that if "Buddy" gets a certain answer such as "I have a problem", it would be directed to problem which answers back to the user with "What's the problem? Did something happen at school?". I'm using SnatchBot for this project. To communicate with Snatchbot the app runs a HTTP post (promise) (using the Ionic native HTTP API) to the Bot's endpoint hosted on SnatchBot.com with the payload as the message. It then runs HTTP get which returns the reply and posts it in the user's chatroom. "Buddy" uses NLP (Natural Language Processing) to decide how to respond to each message. NLP has 5 steps to return an answer.

Tokenization: Separates words into individual pieces which are represented as values ranging from -1 to 1

Sentiment Analysis: Studies and learns the user's experience

Normalisation: Processes text and fixes typing errors and common spelling mistakes

Named Entity Recognition: Looks for keywords such as names

Dependency Parsing: Searches for verbs, nouns, phrases, objects and subjects to discover related phrases they want to convey

Libs

Libraries are files of code that were made to be shared. Static libraries differ slightly from normal libraries because a directory to the library has to be contained in the program wishing to use it.

Prototyping

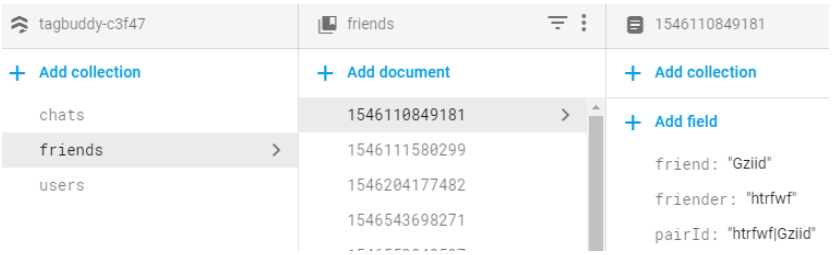
Messaging App

Before I could start on making what I thought was a safer messaging platform I first had to create the standard model at the moment and build on top of it. I started with the login and register page. The register page had to do 3 main things, check if the user's details have already been used, if yes display a pop-up message saying this account already exists. If the details haven't been used before then it should create an account by posting a document with all the users details encrypted to firebase under the users collection. The last step the register page should do is push the user to the chats page. The login user page has to do 2 things, check that the account exists, if not display a pop-up saying account doesn't exist. If the account exists the user should be pushed to the chats page and their user information is saved to storage under "chatuser". The next page the user will face is the chats page. This page has

to do 2 things, display all users except the “chatuser” and turn the list of users into a list of buttons. When one of these buttons is tapped it should send the user to a chatroom with the other user and create a unique “pair Id” which is basically the 2 users’ email addresses combined into one string which is saved to storage for use later. The chatroom page has to do 2 things, post messages to firebase and load new and old messages from both users. When a message is posted to firebase it is encrypted (Caesar shift) and a new document is created with the pair Id, message and the name of the sender, it is posted to the chats collection.

Friends

The next step was to introduce a friending system so everyone wouldn’t have permission to talk with anyone else. To do this I created a new page called the friendsAdd page. On this page there would be a field to enter a friends username and a button to add them as a friend. When this button is clicked it checks that there is a username typed in, if yes it takes the friend’s username, the user’s username and the pair Id and posts them to firebase as a document under the friends collection.



Chatbot

The next step was to look at implementing a chatbot. When someone signed up it would add the chatbot “Buddy” as a friend. When a user texts “Buddy” it runs a http post request to SnatchBot sending the message alongside the user id. A HTTP request is then run which listens for the response, the response is then posted to the chatroom with “Buddy” as the sender.

NFC

The final feature I wanted to implement was the ability to add friends over NFC. In the friendsAdd page I added a button to add friends over NFC. When this button is tapped it creates an NDEF listener. Within this NDEF listener it sends out an NFC signal with the

username of the user as the payload and listens for NFC signals of another user. If this transaction takes place successfully, a new document is added to firebase under the friends collection containing the 2 users' usernames' and the pairId.

Code

Register a User

The purpose of the regUser function is to primarily register the user. It also must do other things such as check if the email is already used. This function is called when the user hits the button to register an account. The first line of the function is an "If" statement that just checks if the user has entered their details such as email and username. If the details are entered it saves them to storage under "LoginForm" and creates a pop-up alert with a message saying please wait. It then continues on with an "if else" statement which takes the "LoginForm" email, Caesar shifts it by 5 and then loops through the user collection on firebase and returns all documents or "users" that are equal to the Caesar shifted "LoginForm" email. If the amount of users returned is more than 0 it sets the Boolean "emailOk" to false. This means that if the email is already used on another account it won't let the user create a new account under the same one, a pop-up message will be displayed saying "Hmmm it appears we have an account registered under this email". It then proceeds to do the same thing for the username. If it passes both of these checks the addUser function is called which Caesar shifts all strings in the loginForm by 5 and posts the LoginForm to firebase and adds "Buddy" the chatbot as a friend, storage is then set to "loginForm" under chatuser. The user is then displayed with a message asking if they agree to TagBuddy's terms of service and agree to be kind to others on the platform. Once they agree they are pushed to the chats page.

Login User Function

The purpose of the Login function is to log the user in and verify that the details are the same as what's on firebase. The first line is an "if" statement that checks that the email and password fields aren't empty. If they are empty a pop-up appears saying "Missing region". If they contain something it continues on and saves the details under "LoginForm" in storage and displays a pop-up saying "Please wait…". It then loops through the user collection on

firebase and returns the user where the email they have typed in is equal to the email field on firebase. If no users are returned a pop-up is displayed saying “email not registered”. Next it cross checks the password against the user but first it has to Caesar shift the firebase user fields by 5 to match up against what the user has typed in. If the two are equal to each other it sets the “chatuser” in storage to the user that was returned earlier. The user is then pushed to the chats page assuming it crosses all checks for email and password.

ngOnInit function (chats page)

The purpose of this function is to find all of the user’s friends, if they are active and if they are male or female. The function is called when the page loads. The first thing that happens is it gets “chatuser” from storage and sets it to “this.chatuser”. Next it sets up the string “tempName” by setting it to “chatuser.name” Caesar shifted by 5. Next it loops through the firebase collection friends and returns friend if the “friender” field is equal to “tempName”. All the friends that are returned are Caesar shifted -5 and displayed to the user on the chats page. Next it takes the friend field on the returned friend and loops through the users collection and if the name field is equal to friend it returns the user. This is used when the friends are displayed later because it checks if they are active and if they are a male or female. A problem I had was the page wasn’t loading instantly so I kept getting errors. I fixed this by putting part of the function in a “try{}catch(error){}” statement.

Add NFC Function

The purpose of this function is to add a friend over NFC. The function is called when the add friend via NFC button is pressed. First it adds a NFC listener and subscribes to it which listens for any messages via NFC. When it gets a message, it sets this.message to it. The user is then presented with a pop-up saying “Do you want to add this.name to your friends list”. The user can either agree or disagree. If the user agrees it calls the addUsername function which posts a new document to the friends collection on firebase.

Add Chat Function

The add chat function is called when the user hits the send button in the chatrooms page. The first thing that happens is it checks to make sure something has been typed in as the message. If there is something typed in it sets the chatpayload to the message, who the sender is, the

pair id and the time. Next it posts to firebase in the chats collection setting the name of the document to the time and the fields in the document to the payload. Next if the chatpartner is “Buddy” it runs a HTTP post with the params and the message. Next on a response to the HTTP post take the message and create a new chat document under the chats collection with “Buddy” as the sender.

Building and Testing

While I was building the app, I had to test that things were working as I went. To do this I ran “ionic serve” in the terminal which basically hosts the app as a website locally. Once I had a fully working app I decided to run it as an app on my phone. To do this I had to install the android ide and in the terminal I ran “ionic cordova run android”. I had to connect my phone and enable usb debugging. After some testing on my phone I had to build the apk to deploy the app. First I had to run this command in terminal and run through the process “keytool -genkey -v -keystore my-release-key.jks -keyalg RSA -keysize 2048 -validity 10000 -alias my-alias”. Next I ran “ionic cordova build android –release”, this built it into an apk which I had to copy and paste into TagBuddy’s root folder. Next I ran the following command to sign the apk “jarsigner -verbose -sigalg SHA1withRSA -digestalg SHA1 -keystore my-release-key.jks app-release-unsigned.apk my-alias”. The last step was to change the name of the apk file to TagBuddy. I uploaded it to google drive and made a qr code with the shareable link form google drive.

Conclusion

I have built a working prototype of TagBuddy with some key features such as the ability to add friends over NFC, a chatbot designed to be easy to talk to from a child’s perspective and basic messaging app functionality. Moving on I would like to improve on the chatbot as it is currently a basic form of AI and I would like to train it using neural networks. There are also a few bugs that I will have to work on removing. Another key aspect of my app that I feel would be useful is a parent app which would link with a child account and the parent could view all messages and friends. I believe that the concept of the TagBuddy is extremely useful and improves on an extremely topical issue of child internet safety.

Appendix

Home page

Typescript

```
import { Component, OnInit } from "@angular/core";
import {
  NavController
} from "ionic-angular";
import { Storage } from "@ionic/storage";
import { ChatsPage } from "../chats/chats";
import { RegPage } from "../reg/reg";
import { LoginPage } from "../login/login";
import { IonicPage } from 'ionic-angular';
import {
  LoadingController,
  ToastController
} from "ionic-angular";
import { AngularFirestore } from "angularfire2/firestore";
import { User } from "../../app/app.models";
import { ChatService } from "../../app/app.service";
import { appconfig } from "../../app/app.config";
import { AlertController } from 'ionic-angular';
import { getScrollData } from "ionic-angular/umd/components/input/input";

@Component({
  selector: "page-home",
  templateUrl: "home.html"
})
export class HomePage implements OnInit {
  email: string;
  loginForm: any = {};

  constructor(
    public navCtrl: NavController,
    private db: AngularFirestore,
    private chatservice: ChatService,
    private loadingCtrl: LoadingController,
    private toastCtrl: ToastController,
    private storage: Storage,
    public alertCtrl: AlertController
    ) {}

  ngOnInit() {
    this.loginForm.email = "";
    this.loginForm.password = "";
    this.storage.get("chatuser").then(chatuser => {
      if (chatuser && chatuser.email !== "") {
        this.navCtrl.push(ChatsPage);
      }
    });

  }

  loginUser() {

    if (this.loginForm.email != "" && this.loginForm.password != "") {
      this.storage.set('loginForm', this.loginForm);
```

```
      //Check if email already exists
      let myLoader = this.loadingCtrl.create({
        content: "Please wait..."
      });
      myLoader.present().then(() => {

this.db.collection("users").doc(this.chatservice.caesarShift(this.loginForm.
email,5)).ref.get().then( doc => {

          //console.log(user)
          if (!doc.exists) {
              const alert = this.alertCtrl.create({
                title: 'Hmmm!',
                subTitle: 'We can\'t seem to find the email you\'ve entered
on our database',
                buttons: ['OK']
              });
              alert.present();
                 myLoader.dismiss();
            } else {
              //User already exists, move to chats page
             let user = doc.data();
                user.email = this.chatservice.caesarShift(user.email, -5);
                user.name = this.chatservice.caesarShift(user.name, -5);
                user.password = this.chatservice.caesarShift(user.password,
-5);

                if(user.password === this.loginForm.password){
                  this.storage.set("chatuser", user);

                  let toast = this.toastCtrl.create({
                    message: "Login In Successful",
                    duration: 3000,
                    position: "bottom"
                  });
                  toast.present();
                  myLoader.dismiss();

                  this.navCtrl.push(ChatsPage);
                }
                else{
                  const alert = this.alertCtrl.create({
                    title: 'Whoops!',
                    subTitle: 'Incorrect password! Try Again ...',
                    buttons: ['OK']
                  });
                  alert.present();
                  myLoader.dismiss();
                }
            }
        }).catch(function(error) {
          console.log("Error getting document:", error);
      });;
      });
    }
    else {
      const alert = this.alertCtrl.create({
        title: 'Whoops!',
```

```
        subTitle: 'Input your email and password!',
        buttons: ['OK']
      });
      alert.present();
    }
  }

  reg(){
    this.navCtrl.push(RegPage);
  }
  getData(){


this.db.collection("users").doc(this.chatservice.caesarShift(this.loginForm.
email,5)).ref.get().then(function(doc) {
      if (doc.exists) {
          console.log("Document data:", doc.data());
      } else {
          console.log("No such document!");
      }
    }).catch(function(error) {
        console.log("Error getting document:", error);
    });
}


}
```

Html

```
<ion-content padding id="HomePage">


<div id="logoBanner">
  <img id="logo" src="assets/imgs/fullicon.jpeg">
</div>


  <div id="signInForm">
    <ion-list>

      <ion-item>
        <ion-label floating>Email</ion-label>
        <ion-input type="email" [(ngModel)]="loginForm.email"></ion-input>
      </ion-item>
      <ion-item>
        <ion-label floating>Password</ion-label>
        <ion-input type="password"
[(ngModel)]="loginForm.password"></ion-input>
      </ion-item>

      <ion-item>
        <div id="buttonDiv">
          <button id="button" ion-button large block
(click)="loginUser()">Login</button>
        </div>
      </ion-item>
```

```
    </ion-list>
  </div>

  <div style="text-align: center">
    <h5>New Here? <a style="color: rgb(0,170,189)" (click)="reg()"
href="javascript:void(0);">Create an Account</a></h5>
  </div>
</ion-content>
```

Register Page

Typescript

```
import { IonicPage } from 'ionic-angular';
import { Component} from "@angular/core";
import {
  NavController,
  LoadingController,
  ToastController
} from "ionic-angular";
import { AngularFirestore } from "angularfire2/firestore";
import { User } from "../../app/app.models";
import { ChatService } from "../../app/app.service";
import { Storage } from "@ionic/storage";
import { ChatsPage } from "../chats/chats";
import { appconfig } from "../../app/app.config";
import { AlertController } from 'ionic-angular';
/**
 * Generated class for the RegPage page.
 *
 * See https://ionicframework.com/docs/components/#navigation for more info
on
 * Ionic pages and navigation.
 */

@IonicPage()
@Component({
  selector: 'page-reg',
  templateUrl: 'reg.html',
})
export class RegPage {
  email: string;
  loginForm: any = {};
  emailOk: boolean;
  done: boolean = false;
  constructor(
    public navCtrl: NavController,
    private db: AngularFirestore,
    private chatservice: ChatService,
    private loadingCtrl: LoadingController,
    private toastCtrl: ToastController,
    private storage: Storage,
    public alertCtrl: AlertController
  ) {}


  ngOnInit() {
```

```
  }

  addUser(payload : any = {}) {
    payload.email = this.chatservice.caesarShift(payload.email, 5);
    payload.name = this.chatservice.caesarShift(payload.name, 5);
    payload.password = this.chatservice.caesarShift(payload.password, 5);
    payload.gender = payload.gender;
    payload.active = "false";
    payload.time = payload.time;
    //check for user already created

    //add the bot chat
    this.db.collection("friends").doc(payload.time.toString()).set({
      friender: payload.name,
      friend: this.chatservice.caesarShift("Buddy", 5),
      pairId: payload.name + "|" + this.chatservice.caesarShift("Buddy", 5),
    });

    return this.db.collection("users").doc(payload.email).set(payload);
  } //addUser

  regUser() {
    if (this.loginForm.email != "" && this.loginForm.name != "" &&
this.loginForm.password != "") {
      //check email and password



      this.storage.set('loginForm', this.loginForm);
      //Check if email already exists
      let myLoader = this.loadingCtrl.create({
        content: "Please wait..."
      });
      myLoader.present().then(() => {
        this.db
          .collection<User>(appconfig.users_endpoint, ref => {
            return ref.where("email", "==",
this.chatservice.caesarShift(this.loginForm.email, 5));
          })
          .valueChanges()
          .subscribe(users => {
            console.log(users);
            if(users.length != 0){
              this.emailOk = false;
            }
            else{
              this.emailOk = true;
            }

          });

          this.db
          .collection<User>(appconfig.users_endpoint, ref => {
            return ref.where("name", "==",
this.chatservice.caesarShift(this.loginForm.name, 5));
          })
          .valueChanges()
```

```
          .subscribe(users => {
            console.log(this.emailOk)
            if(users.length==0&&this.emailOk){
              this.loginForm.time = new Date().getTime();
              this.done = true;
              this.addUser(this.loginForm)
                .then(res => {
                this.loginForm.email =
this.chatservice.caesarShift(this.loginForm.email, -5);
                this.loginForm.name =
this.chatservice.caesarShift(this.loginForm.name, -5);
                this.loginForm.password =
this.chatservice.caesarShift(this.loginForm.password, -5);
                //Registration successful, go to chats page
                this.storage.set("chatuser", this.loginForm);
                myLoader.dismiss();


                const confirm = this.alertCtrl.create({
                  title: 'Tag Buddy - Terms of Use',
                  message: 'Do you agree to to be kind to others when using
and treat everyone fairly?',
                  buttons: [
                    {
                      text: 'Disagree',
                      handler: () => {
                        const toast = this.toastCtrl.create({
                          message: 'Cannot proceed until the agreement is
accepted',
                          duration: 3000,
                          position: 'bottom'
                        });
                        toast.present();
                      }
                    },
                    {
                      text: 'Agree',
                      handler: () => {
                        this.navCtrl.push(ChatsPage);
                      }
                    }
                  ]
                  });
                  confirm.present();
                return;
              })
              .catch(err => {
                console.log(err);
                myLoader.dismiss();
              });
          } else if(this.done==false){
            if(this.emailOk==false&&users.length==0){
              const alert = this.alertCtrl.create({
                title: 'Hmmm!',
                subTitle: 'It appears we have an account registered under
this email!',
                buttons: ['OK']
              });
```

```
              alert.present();
              myLoader.dismiss();
            }
            else if(users.length!=0&&this.emailOk){
              const alert = this.alertCtrl.create({
                title: 'Whoops!',
                subTitle: 'This username is already taken!',
                buttons: ['OK']
              });
              alert.present();
              myLoader.dismiss();
            }
            else if(users.length!==0&&!this.emailOk){
              console.log(users[0]);
              console.log(this.emailOk)
              const alert = this.alertCtrl.create({
                title: 'Hmmm!',
                subTitle: 'This account has already been registered ... return
to the login page and try to login!',
                buttons: ['OK']
              });
              alert.present();
              myLoader.dismiss();
            }
            }
            });
      });
    } else {
      const alert = this.alertCtrl.create({
        title: 'Whoops!',
        subTitle: 'Make sure all fields are populated before you try to
register an account!',
        buttons: ['OK']
      });
      alert.present();
    }
  }
}
```

HTML

```
<ion-header>

  <ion-navbar>
    <ion-title>Register an Account</ion-title>
  </ion-navbar>

</ion-header>
<ion-content padding id="regPage">
<div id="logoBanner">
  <img id="logo" src="assets/imgs/fullicon.jpeg">
</div>


  <div id="signInForm">
    <ion-list>
      <ion-item>
        <ion-label floating>Email</ion-label>
```

```
        <ion-input type="email" [(ngModel)]="loginForm.email"></ion-input>
      </ion-item>
      <ion-item>
        <ion-label floating>Username</ion-label>
        <ion-input type="text" [(ngModel)]="loginForm.name"></ion-input>
      </ion-item>
      <ion-item>
        <ion-label floating>Password</ion-label>
        <ion-input type="password"
[(ngModel)]="loginForm.password"></ion-input>
      </ion-item>
      <ion-item>
        <ion-label floating >Gender</ion-label>
        <ion-select [(ngModel)]="loginForm.gender">
          <ion-option value="female">Female</ion-option>
          <ion-option value="male">Male</ion-option>
        </ion-select>
      </ion-item>

      <ion-item>
        <div id="buttonDiv">
          <button id="button" ion-button large block
(click)="regUser()">Register Account</button>
        </div>
      </ion-item>

    </ion-list>
  </div>
</ion-content>
```

Chats Page

Typescript

```
import { Component, OnInit } from "@angular/core";
import { IonicPage, NavController, NavParams } from "ionic-angular";
import { AngularFirestore } from "angularfire2/firestore";
import { Storage } from "@ionic/storage";
import { appconfig } from "../../app/app.config";
import { User, Friend, Chat} from "../../app/app.models";

import { ChatService } from "../../app/app.service";
import { ChatroomPage } from "../chatroom/chatroom";
import { FriendsaddPage } from "../friendsadd/friendsadd"
interface Person {
  gender: string;
  active: string;
  message: string;
}

@IonicPage()
@Component({
  selector: "page-chats",
  templateUrl: "chats.html"
})
export class ChatsPage implements OnInit {

  availableusers: any = [];
```

```
  persons : { [id: string] : Person; } ={}
  chatuser;
  tempName: string;
  pairId: string;
  doneLeave: boolean = false;
  doneEnter: boolean = false;
  constructor(
    public navCtrl: NavController,
    public navParams: NavParams,
    private db: AngularFirestore,
    private storage: Storage,
    private chatservice: ChatService
  ) {}

  ngOnInit() {
    //Fetch other users //and active status //and icon for friends

    this.storage.get("chatuser").then(chatuser => {
      this.chatuser = chatuser;
   this.storage.get("pairId").then(pairId =>{
      this.pairId = pairId;

    })

    this.tempName=this.chatservice.caesarShift(this.chatuser.name, 5);
      this.db
      .collection<Friend>(appconfig.friends_endpoint, res => {
        return res.where("friender", "==",this.tempName);
      })
      .valueChanges()
      .subscribe(friends => {
          this.availableusers = friends.filter(friend => {
            if (friend.friender == this.tempName) {
              friend.friend = this.chatservice.caesarShift(friend.friend,
-5);
              return friend
            }
          });
          console.log(this.availableusers)
         for(let i in this.availableusers){
          this.db
        .collection<User>(appconfig.users_endpoint, res => {
          return res.where("name",
"==",this.chatservice.caesarShift(this.availableusers[i].friend, 5));
        })
        .valueChanges()
        .subscribe(users => {

            this.db
            .collection<Chat>(appconfig.chats_endpoint, res => {
            return res.where("pair",
"==",this.chatservice.caesarShift(this.availableusers[i].pairId, 5));
          })
        .valueChanges().subscribe(chats => {
        try{
          let msg = "No messages yet"
          if(chats.length!=0){
```

```
            msg =
this.chatservice.caesarShift(chats[chats.length-1].message, -5);
          }
          console.log(msg)
           this.persons[this.availableusers[i].friend] =
{gender:users[0].gender, active:users[0].active, message:msg}
          }
          catch(error){
            //do nothing
          }
          });

        });
      }
      });
      console.log(this.persons)
    });



  }

  ionViewDidLeave() {
    //set active to false
  console.log("Not active")

this.db.collection("users").doc(this.chatservice.caesarShift(this.chatuser.e
mail,5)).update({
      active: "false",
    }) .catch(error => console.log(error));


  }


  ionViewDidEnter() {
    // console.log("Enetred Page")
    // //set active to true


this.db.collection("users").doc(this.chatservice.caesarShift(this.chatuser.e
mail,5)).update({
      active: "true",
    }).catch(error => console.log(error));
  }

  goToChat(chatpartner) {

    this.pairId = chatpartner.pairId;
    console.log(this.pairId);

    this.chatservice.currentChatPairId =  this.pairId;

    this.chatservice.currentChatPartner = chatpartner;

    this.navCtrl.push(ChatroomPage);

  } //goToChat
```

```
  addFriend(){
    this.navCtrl.push(FriendsaddPage);
  }



  getImg(userName){
    if(userName==="Buddy"){
      return "assets/imgs/buddy.png"
    }
    else{
      try{
    return "assets/imgs/"+this.persons[userName].gender+".png"
      }
      catch(error){
        return "assets/imgs/male.png"
      }
    }
  }

  getActiveStatus(userName){
    if(userName==="Buddy"){
    return "assets/imgs/active.png"
  }
  else{
    try{

    if(this.persons[userName].active==="true"){
      return "assets/imgs/active.png"
    }
    else{
      return "assets/imgs/notActive.png"
    }
  }catch(error){
    return "active"
  }
}
}

getMostRecentMsg(userName){
  if(userName==="Buddy"){
    return "Hi, I'm Buddy ðŸ‘‹ ðŸ˜ŽðŸ¤–, Please tell me your name so we can
be friends!! ðŸ¤©"
  }
  else{
  try{
  return this.persons[userName].message;
  }
  catch(error){
    return ""
  }
}
}

}
```

HTML

```
<ion-header>

  <ion-navbar>
    <ion-title>{{chatuser?.name}}</ion-title>

  </ion-navbar>

</ion-header>


<ion-content class="chatsPage" padding>

  <ion-list>
      <ion-list-header>
        Friends
      </ion-list-header>
      <ion-fab top right style="padding-right:1%;">
          <button ion-fab mini color=light (click)="addFriend()"><ion-icon
name="add" ></ion-icon></button>
      </ion-fab>


    <ion-item  *ngFor="let friend of availableusers"
(click)="goToChat(friend)">
      <ion-avatar item-start>
        <img src='{{getImg(friend.friend)}}'>
      </ion-avatar>
      <h2>{{friend.friend || "Anonymous"}} <img style="width:10px;
height:10px; "src='{{getActiveStatus(friend.friend)}}'></h2>
      <p>{{getMostRecentMsg(friend.friend)}}</p>
    </ion-item>
  </ion-list>

</ion-content>
```

Chatroom Page

Typescript

```
import { Component, OnInit, ViewChild } from "@angular/core";
import { IonicPage, NavController, NavParams } from "ionic-angular";
import { AngularFirestore } from "angularfire2/firestore";
import { User, Chat } from "../../app/app.models";
import { appconfig } from "../../app/app.config";
import { ChatService } from "../../app/app.service";
import { Storage } from "@ionic/storage";
import { HTTP } from '@ionic-native/http';

/**
 * Generated class for the ChatroomPage page.
 *
 * See https://ionicframework.com/docs/components/#navigation for more info
on
 * Ionic pages and navigation.
 */
interface Seens{
  seen: string;
```

```
  from : string;
}
@IonicPage()
@Component({
  selector: "page-chatroom",
  templateUrl: "chatroom.html"
})
export class ChatroomPage implements OnInit {
  setMsg: string
  seens :{[msg:string]: Seens;} ={}
  chats: any = [];
  chatpartner = this.chatService.currentChatPartner;
  chatuser;
  message: string;
  req : string;
  load :boolean
  jsonStr: string;
  responseStr: string;
  chatPayload: Chat;
  intervalScroll;
  @ViewChild("content") content: any;

  constructor(
    public navCtrl: NavController,
    public navParams: NavParams,
    private db: AngularFirestore,
    private chatService: ChatService,
    private storage: Storage,
    private http: HTTP
  ) {}

  ionViewDidLoad() {
    console.log("ionViewDidLoad ChatroomPage");

  }
  ionViewDidUnload() {
    this.content.scrollToBottom(300);
  }
  //scrolls to bottom whenever the page has loaded
  ionViewDidEnter() {
     //300ms animation speed

     console.log("didenter")
     setTimeout(() => {
      this.content.scrollToBottom(300);
      console.log(this.chatuser.email)

this.db.collection("users").doc(this.chatService.caesarShift(this.chatuser.em
ail,5)).update({
      active: "true",
    }).catch(error => console.log(error));
   }, 1000);//may have to be increased if wifi is slow in rds

  }

  ionViewWillLeave() {
    console.log("willleave")
  }
```

```
    //set active to true


/*
  scrollToBottom() {
    setTimeout(() => {
      this.content.scrollToBottom();
    });
  } */

  ngOnInit() {
    //get active status
    this.storage.get("chatuser").then(chatuser => {
      this.chatuser = chatuser;
      this.db
      .collection<Chat>(appconfig.chats_endpoint, res => {
        return res.where("pair", "==", this.chatService.currentChatPairId);
      })
      .valueChanges()
      .subscribe(chats => {
        //this.availableusers = users;
        //console.log(chats);
        for(var i = 0; i<chats.length; i++){
        this.chats[i] = chats[i];
        this.chats[i].message =
this.chatService.caesarShift(this.chats[i].message,-5)
        this.chats[i].sender =
this.chatService.caesarShift(this.chats[i].sender,-5)
        this.chats[i].pair =
this.chatService.caesarShift(this.chats[i].pair,-5)

        if(this.chats[i].sender!=this.chatuser.email){
          console.log("got here")
          console.log(this.chats[i].sender)
          console.log(this.chatuser.email)

          this.seens[chats[i].message] = {seen:"true",
from:this.chats[i].sender};

this.db.collection("chats").doc(this.chats[i].time.toString()).update({
            seen: "true",
          }).then(()=>{
            this.content.scrollToBottom(300);
        }).catch(error => {console.log(error)});
      }
      else{
        this.seens[this.chats[i].message] = {seen:this.chats[i].seen,
from:this.chats[i].sender};
        }
      }
  });
  this.content.scrollToBottom(300);
    });


      console.log("done loading content")
```

```
      this.content.scrollToBottom(300);
  } //ngOnInit



  addChat() {
    if (this.message && this.message !== "") {
      console.log("Chatuser email")
      console.log(this.chatuser.email)
      this.chatPayload = {
        message: this.chatService.caesarShift(this.message,5),
        sender: this.chatService.caesarShift(this.chatuser.email,5),
        pair: this.chatService.currentChatPairId,
        seen: "false",
        time: new Date().getTime()
      };

this.db.collection("chats").doc(this.chatPayload.time.toString()).set(this.ch
atPayload)
        .then(() => {
          //Clear message box
          this.message = "";

          //Scroll to bottom
          this.content.scrollToBottom(300);
        })
        .catch(err => {
          console.log(err);
        });

        if(this.chatpartner.friend === "Buddy"){
          console.log("BOT");
          let url =
'https://snatchbot.me/channels/api/api/id40400/appTagBuddy/apstag?user_id='+t
his.chatuser.email;
          let params = {
            "message": this.message,
          };
          let headers = {};
          console.log(params);
          this.http.setDataSerializer("json");
          this.http.setHeader("*", "Accept", "application/json");
          this.http.setHeader("*", "Content-Type", "application/json");
          this.http.post(url, params, headers)
          .then(data => {

            console.log(data.toString());
            let obj = JSON.parse(data.data);
            let strResponse = ""
            for(let i in obj.messages){
              strResponse = strResponse + obj.messages[i] + "\n";
            }
            this.chatPayload = {
              message: this.chatService.caesarShift(strResponse,5),
              sender: this.chatService.caesarShift("Buddy",5),
              pair: this.chatService.currentChatPairId,
              seen: "true",
```

```
              time: new Date().getTime()
            };

this.db.collection("chats").doc(this.chatPayload.time.toString()).set(this.ch
atPayload)
              .then(() => {
                //Clear message box
                this.message = "";

                //Scroll to bottom
                this.content.scrollToBottom(300);
              })
              .catch(err => {
                console.log(err);
              });
          })
          .catch(error => {

            console.log(error.status);
            console.log(error.error); // error message as string
            console.log(error.headers);

          });

        }
    }
  } //addChat

  isChatPartner(senderEmail) {
    try{
      return !(senderEmail != this.chatuser.email);
  }
  catch(e){
    return;
  }
  } //isChatPartner

  scrollToBottom(){
    setTimeout(() => {
     this.content.scrollToBottom(200);
  }, 1000);
  }

  isLastMsgSeen(){
    try{
    if(this.seens[this.setMsg].from==this.chatuser.email){

if(this.seens[this.setMsg].seen=="true"||this.chatpartner.email=="Buddy"){
        return "seen"
      }
      else if(this.seens[this.setMsg].seen=="false"){
        return "not seen"
      }
      else{
        return ""
      }
    }
  }
```

```
  catch(error){

  }
  }

  setLastMsg(msg){
    this.setMsg = msg;

  }
}
```

HTML

```
<ion-header>

  <ion-navbar>
    <ion-title>{{chatpartner.friend}}</ion-title>
  </ion-navbar>

</ion-header>


<ion-content #content padding id="chatPage">

  <ion-list>
    <ion-item *ngFor="let chat of chats | sort:'time'" class="chat" text-wrap
[ngClass]="{'chat-partner' : isChatPartner(chat.sender)}">
      {{chat.message}} {{setLastMsg(chat.message)}}
    </ion-item>
    <div style="text-align:right"><h6>{{isLastMsgSeen()}}</h6></div>
  </ion-list>

</ion-content>

<ion-footer>
  <ion-toolbar>
    <ion-row>
      <ion-col col-10>
        <ion-input type="text" [(ngModel)]="message"
(click)="scrollToBottom()"placeholder="Enter Message...."></ion-input>
      </ion-col>
      <ion-col col-2>
        <button id="button" ion-button block (click)="addChat()">
          Send
        </button>
      </ion-col>
    </ion-row>


  </ion-toolbar>
</ion-footer>
```

FriendsAdd Page

Typescript

```
import { Component } from '@angular/core';
```

```
import { IonicPage, NavController, NavParams, ToastController} from
'ionic-angular';
import { Chat } from "../../app/app.models";
import { Storage } from "@ionic/storage";
import { ChatService } from "../../app/app.service";
import { appconfig } from "../../app/app.config";
import { Firebase } from '@ionic-native/firebase';
import firebase, { storage } from "firebase"

import { ChatsPage } from "../chats/chats";
import { OnInit } from "@angular/core";
import { ModalController } from "ionic-angular";
import {
  AngularFirestoreDocument,
  AngularFirestore,
  AngularFirestoreCollection
} from "angularfire2/firestore";

import { User, Friend } from "../../app/app.models";
import { Observable } from "rxjs/Observable";



import { ChatroomPage } from "../chatroom/chatroom";

import { diPublic } from "@angular/core/src/render3/instructions";
import { checkAndUpdateElementInline } from
'@angular/core/src/view/element';
import { from } from 'rxjs';
import { firestore } from 'firebase';
import { AlertController } from 'ionic-angular';
import { NFC, Ndef } from '@ionic-native/nfc';
import { Message } from '@angular/compiler/src/i18n/i18n_ast';
/**
 * Generated class for the FriendsaddPage page.
 *
 * See https://ionicframework.com/docs/components/#navigation for more info
on
 * Ionic pages and navigation.
 */

@IonicPage()
@Component({
  selector: 'page-friendsadd',
  templateUrl: 'friendsadd.html',
})
export class FriendsaddPage {

  chatPayload: Chat;
  email: string;
  name: string;
  chatuser: string;
  friendsStorage: string;
  private chatService: ChatService;
  //private db: AngularFirestore
  Friend;
  friendertime:number;
  friendtime: number;
```

```
pairId: string;

user: AngularFirestoreCollection<User>;
//private test: AngularFirestoreDocument<Test>;

firebase: Firebase

constructor(
  private storage: Storage,
  private nfc: NFC,
  private ndef: Ndef,
  private chatservice: ChatService,
  private db: AngularFirestore,
  public navCtrl: NavController,
  public navParams: NavParams,
  private toastCtrl: ToastController,
  public alertCtrl: AlertController
  ) { };

ngOnInit() {
  this.storage.get("chatuser").then(chatuser => {
  this.chatuser = chatuser.name;
  });
}


addUsername(){
  console.log(this.chatuser);
  this.pairId = this.name + "|" + this.chatuser;
  this.storage.set('pairId', this.pairId);

  this.chatuser = this.chatservice.caesarShift(this.chatuser, 5);
  this.pairId = this.chatservice.caesarShift(this.pairId,5);
  this.name = this.chatservice.caesarShift(this.name, 5);


  this.db
      .collection<Friend>(appconfig.friends_endpoint, res => {
        return res.where("pairId", "==",this.pairId);
      })
      .valueChanges()
      .subscribe(users => {
        if(users.length == 0){
          let time = new Date().getTime();
          this.db.collection("friends").doc(time.toString()+"_1").set({
            friender: this.chatuser,
            friend: this.name,
            pairId: this.pairId
          })

          this.db.collection("friends").doc(time.toString()+"_2").set({
            friender: this.name,
            friend: this.chatuser,
            pairId: this.pairId
          })
        }
      });
```

```
 let toast = this.toastCtrl.create({
        message: "Friend Added",
        duration: 3000,
        position: "bottom"
       });
       toast.present();
       this.navCtrl.push(ChatsPage);
}

//Runtime error get nfc working!!!
addNFC(){
  // listen for friend
  // send out credentials
  this.nfc.addNdefListener(() => {
     console.log('successfully attached ndef listener');
     this.name = "success"
  }, (err) => {
    console.log('error attaching ndef listener', err);

  }).subscribe((event) => {
    console.log('received ndef message. the tag contains: ', event.tag);
    console.log('decoded tag id', this.nfc.bytesToHexString(event.tag.id));
    this.name =
this.nfc.bytesToString(event.tag.ndefMessage[0]["payload"]).substring(3);
    const confirm = this.alertCtrl.create({
      title: 'Add this friend?',
      message: 'Do you want to add '+this.name+' to your friends list?',
      buttons: [
        {
          text: 'Disagree',
          handler: () => {
            confirm.dismiss();
            console.log('Disagree clicked');
          }
        },
        {
          text: 'Agree',
          handler: () => {
            this.addUsername();
            confirm.dismiss();
            console.log('Agree clicked');
          }
        }
      ]
    });
    confirm.present();
  });
  let message = this.ndef.textRecord(this.chatuser.toString());
  this.nfc.share([message]);
}
  ionViewDidLoad() {
    console.log('ionViewDidLoad FriendsaddPage');
  }
}
```

HTML

```
<!--
```

```
  Generated template for the FriendsaddPage page.

  See http://ionicframework.com/docs/components/#navigation for more info on
  Ionic pages and navigation.
-->
<ion-header>

  <ion-navbar>
    <ion-title>Add Friend</ion-title>
  </ion-navbar>
</ion-header>

<ion-content padding id="addPage">
<div id="logoBanner">
  <img id="logo" src="assets/imgs/fullicon.jpeg">
</div>



<h4 style="text-align: center; padding-left: 5%; padding-right: 5%;
color:rgb(16, 28, 56);"><i>"Hi <b>{{chatuser}}</b>  to add a friend simply
press the button below then hold your phone back-to-back with your friends
phone."</i></h4>

<ion-list style="padding-top: 5%">
    <ion-item>
      <ion-label floating >Friend's username</ion-label>
      <ion-input   type="text" [(ngModel)]="name" placeholder = "Friend's
username"></ion-input>
    </ion-item>

    <ion-item>
        <button id="button" ion-button large block
(click)="addUsername()">Add Friend </button>
      <button id="button" ion-button large block (click)="addNFC()">Add
Friend via NFC</button>
    </ion-item>

</ion-list>

</ion-content>
```