PING APP

SOFTWARE DESIGN DOCUMENT

December 2022

TEAM MEMBERS
THUY UYEN MY TRAN
ELVIS LOR
BRANDON HUYNH
SHEE HER
KYLE KILCREASE

Contents

Section 1 - Introduction

- Ping App is a mobile application that will automatically send the user's current location and text messages to the user's trusted contact list that they can either add themself or sync it from their phone contact list.
- Normally, when you go outside by yourself, or when you are in danger, you can only call or text one person at a time from your phone. Moreover, you have to go through the steps such as taking out the phone; type in the phone number, or go through your contact list to find the right person to call/text; then call, or type your text messages one by one.
- With the Ping App, the users can save a lot of time by just going to the app, pressing the PING button, and a text message that includes their current location will be sent to multiple people at a time.

1.1 Purpose

The purpose of this document is to define the architecture, system design, and the progress that the Ping App is implemented in order to understand the expectation of functionalities of the app. Furthermore, this document also provides the information, details, graphs, and structures of the app.

1.2 Scope

This Software Design Document will thoroughly explain and provide the description of the software and the software system to be built based on the expected functionality.

1.3 Estimates

#	Description	Hrs. Est.
1	UX/UI Design	3
2	Front End Implementation	7
3	Database Setup	30
4	Login/Signup Feature	10
5	Login/Signup with Google/Facebook	4
6	Sync Phone Contacts	12
7	Android Permissions	8
8	User Profile Photo Uploading	15
9	Allow Sharing Location	20
10	Allow Users to Change Their Password	NA
11	Allow Users to View History on Recent Pings	NA
12	PING Button Functions Implementation	17
	TOTAL	126

Section 2 - Use Cases

2.1 Actors

2.1.1 Public Users

- The app is designed for everyone to use, especially students, elderly, people who normally go out by themselves, and people who are susceptible to emergency
- The users will be able to login, signup, allow location, use Ping Button, add/edit contacts, see history of recent Pings, change password, update profile, and view other user's location when they hit the Ping button.

2.1.2 Administrative Users

- The administrative users can edit the database that contains the information of the public users. Moreover, they are also able to manage, update, and make necessary improvements to the database.

2.2 List of Use Cases

2.2.1 Public User Use Cases

- Login/Signup
- Allow location (Yes/No)
- Use Ping button
- Add/Edit contacts
- See history of recent Pings
- Change password
- Update profile
- View other user's location on the app when they hit the Ping button

2.2.2 Administrative Use Cases

- Manage database
    - Delete user accounts
    - Manage user accounts
    - Delete data
    - Manage data
    - Manage User Email account login
    - Delete User Password

2.3 Use Case Diagrams

2.3.1 Public User Use Case Diagram

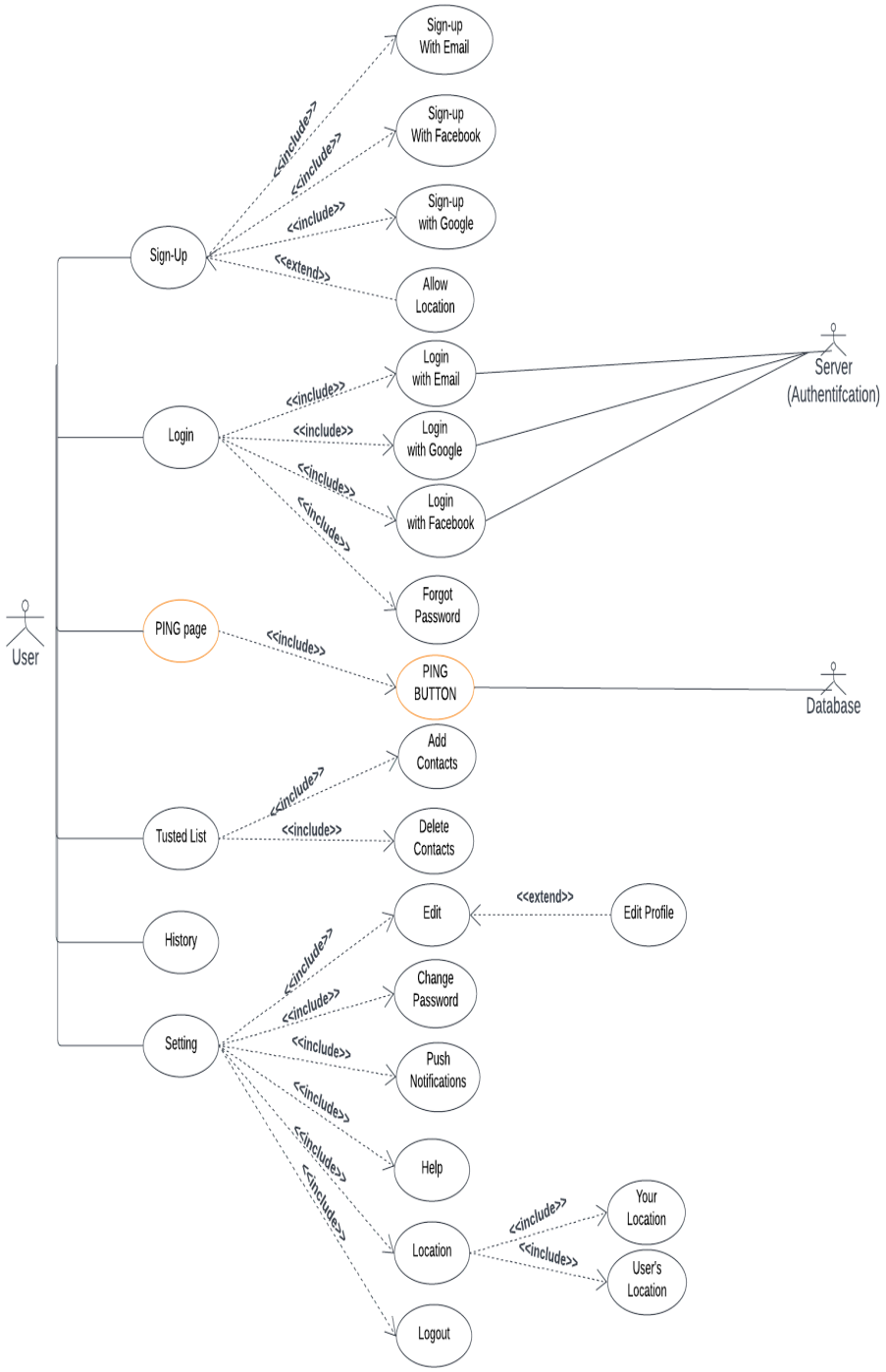


Figure 1: Public User Use Case Diagram

2.3.2 Administrative User Use Case Diagram

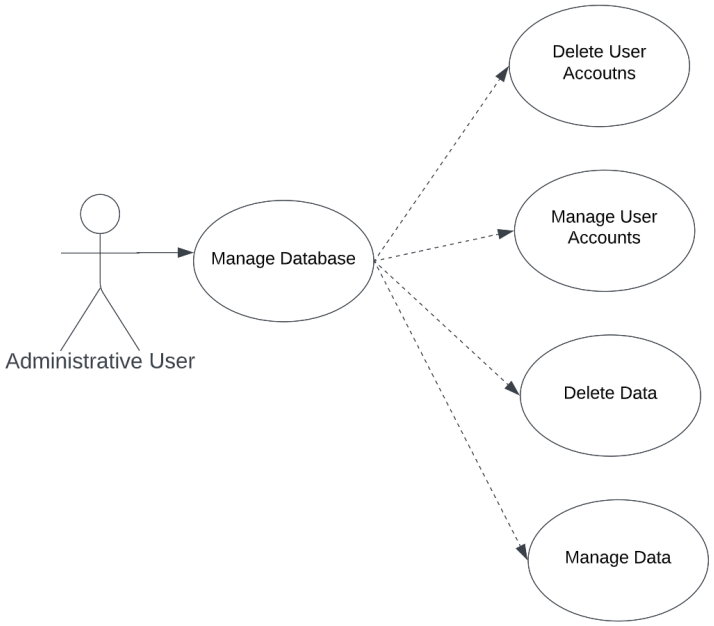


Figure 2: Administrative User Use Case Diagram

Section 3 - Design Overview

3.1 Introduction

This section will provide the design overview of:

- System architecture
- System interfaces
    - User interface
    - Software interface
- Constraints and assumptions

3.2 System Architecture

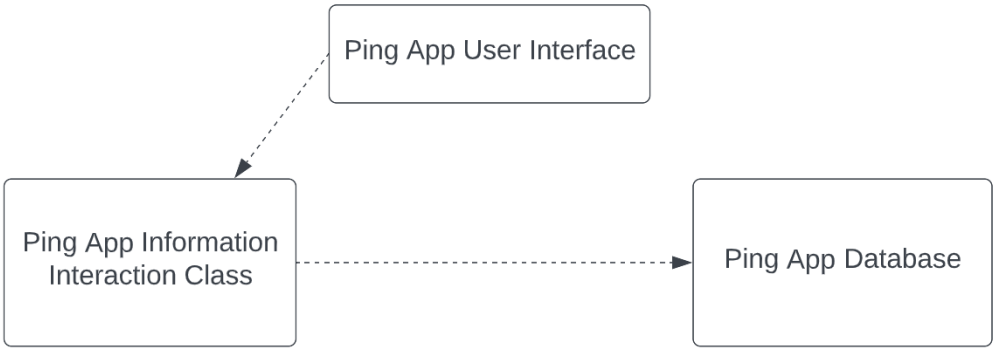


Figure 3: System Architecture

3.3 System Interfaces

3.3.1 User Interface

3.3.1 User Interface Design Overview

The user interface is designed with Figma, and the main theme colors are light green, white, and light orange. The figure below shows how the application interface design will look like:

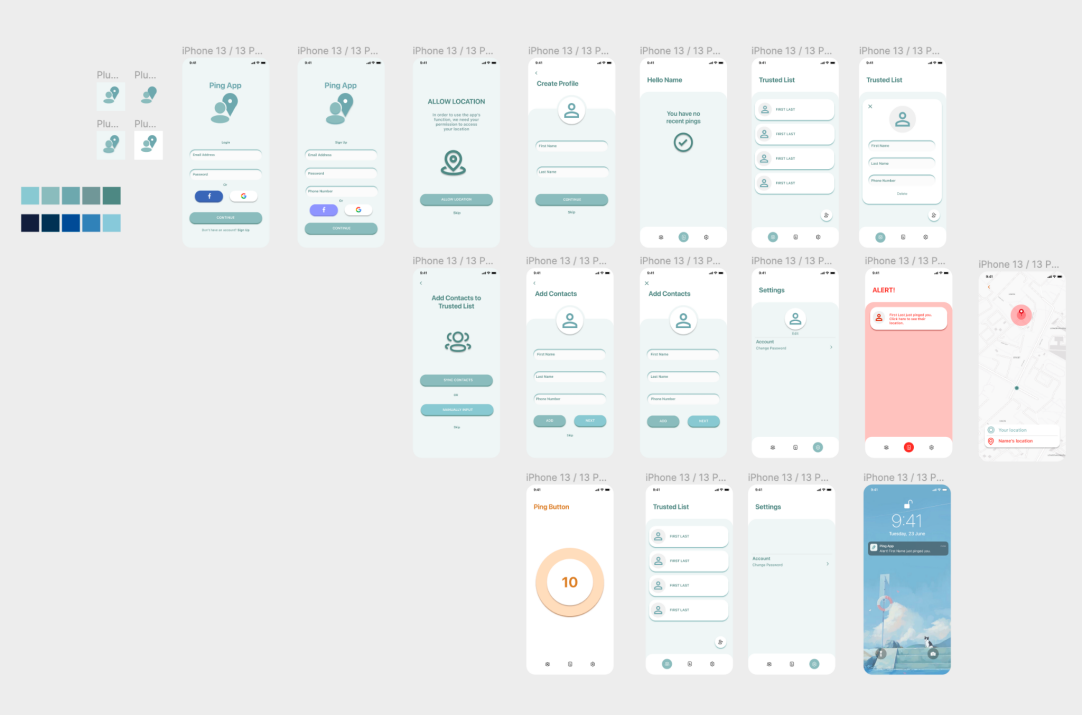


Figure 5: User Interface Design

3.4.1 User Interface Navigation Flow

The user interface navigation shows how the users can navigate through the application. The users are able to use all the navigation functions based on how they want. They can also take actions on the application, which will communicate to the database using the software interface. Figure 6 shows the navigation flow of the user interface.

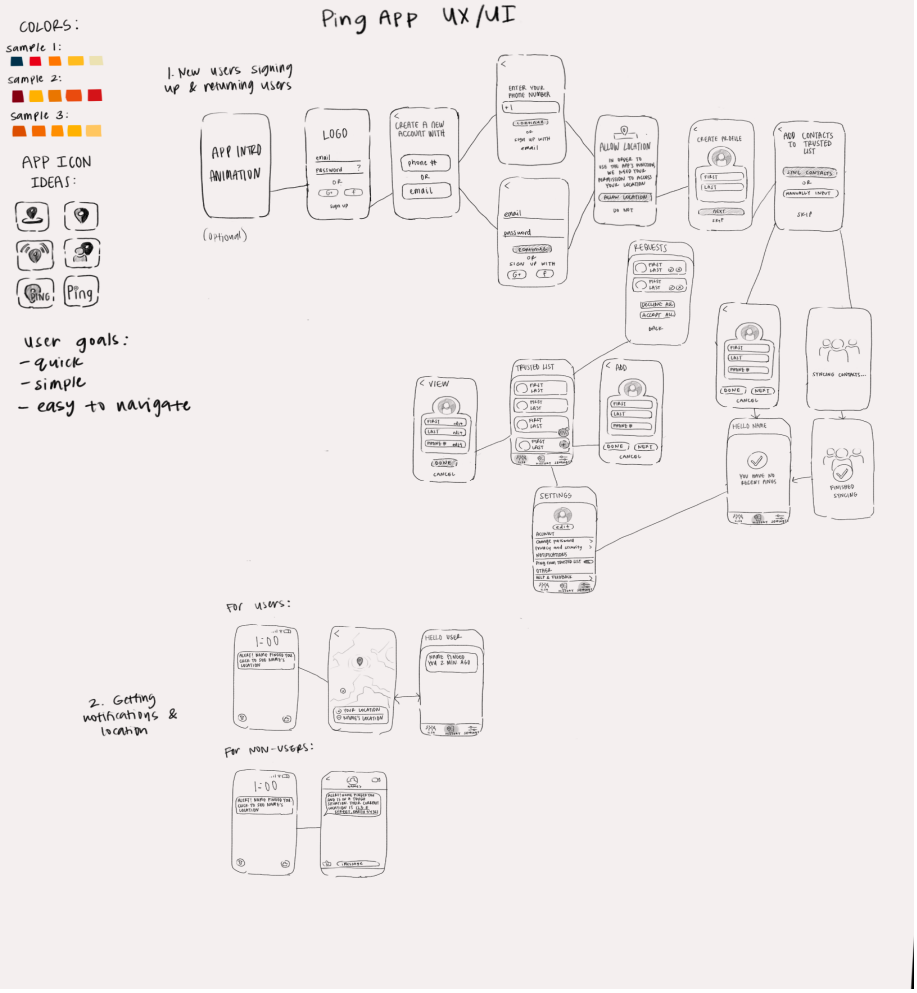


Figure 6: User Interface Navigation Flow

3.3.2 Software Interface

The software interface pulls data from the database, then pushes it to the user interface where it is supposed to be displayed.

3.4 Constraints and Assumptions

3.4.1 Assumptions

- Users either have phone's cellular connection or wifi connection in order to use the app
- The app has minimal to zero bugs
- The location of the user when being sent to other users is accurate

3.4.2 Constraints

- A few of the constraints that I was dealing with along the way is getting everything to work together correctly. I was having trouble mixing what I have already developed to work well with the other components of the group. One such constraint was getting the map to display and work correctly with the SMS so I ended up turning that into its own little feature in the setting of the app. **-Elvis**
- Some constraints I faced include my limited knowledge of Angular, Ionic, and Github, and html/css. I mostly had trouble understanding all of these including how to turn the UX/UI design into html/css. In addition, since we're using Ionic, I had to make sure that the frontend implementations were the same for both ios and android. **-Shee**
- For my side of the project, I dealt with similar constraints of learning how to integrate my code with the rest of the team, as well as relearning things about Ionic and Angular over the course of the semester. The framework, while easier to work with for multi-platform apps, has its own set of particularities as well that provided a challenge for implementing and integrating the features for the app. **-Kyle**
- Some constraints I had during the project was learning the Ionic framework and how the different libraries worked for Ionic. Many of these libraries were created for older versions of Ionic and had very little or confusing documentation. So whenever I came across a problem or bug using these libraries, it took a lot of time debugging them so that they would work properly. **-Brandon**

Section 4 - Object Description

4.1 Objects

4.1 Objects

4.1.1 User Interface

File Name: history.page.html	
Description: This page is to display the tab history and its content. This page does not have any inputs or outputs. The purpose of this page is to display recent history. If there is nothing to show then it will display as is.	
Tags	**Tag Description**
ion-content	Content area for other tags to control scrollable area.
div class = "card"	Container for holding information and other tags.
div class= "none"	Container called "none" to hold information to be displayed if there are no

	recent pings. Tags within this container include an image and text.
ion-avatar	Circular component to wrap an image.
	<ion-content> History You have no recent pings <ion-avatar> <img src="assets/images/circle-checkedcheck.png"> </ion-avatar> </ion-content>

File Name: main.page.html	
Description: This page is used to display the main tab, which is where the button is at. This page has user interactions.	
Tags	**Tag Description**
ion-content	Content area for other tags to control scrollable area
div	This container is for holding the ping button. It includes a button and progress tab for the button functionality. The progress is used to show how long the user held the button for and for how long they should hold in order to send an alert.
	<ion-content> Hello User <button class="button" appHoldable (holdTime)="holdButton($event)">Hold</button> <progress [value]="progress" max="100"></progress> </ion-content>

File Name: settings.page.html	
Description: One of the pages to display the setting of the app. The page includes user interactions where they can edit their profile, get more information about the app (Help), logout, and change push notification settings.	
Tags	**Tag Description**
ion-content	Content area for other tags to control scrollable area
div class = “card”	Class container called card to hold items.
div class= “top”	Class container called top to hold profile pic of user and an edit link to a page where they can edit their profile.
ion-avatar	Circular component to wrap user’s profile for display.
ion-item button detail= “true”	An clickable item that contains text and placed in a list. Detail= “true” is for displaying a right arrow icon on an item.

```
<ion-content>
  <h1>Settings</h1>
  <div class="card">
    <div class="top">
      <ion-avatar>
        <img alt="Silhouette of a person's head" src="assets/images/person.png" />
      </ion-avatar>
      <span [routerLink]="['/edit']">Edit</span>
      <hr>
    </div>

    <ion-item button detail="true">
      <ion-label>Change Password</ion-label>
    </ion-item>
    <ion-item button detail="true">
      <ion-label>Push Notifications</ion-label>
    </ion-item>
    <ion-item button detail="true">
      <ion-label>Help</ion-label>
    </ion-item>
    <ion-item button detail="true" [routerLink]="['/login']">
      <ion-label>Logout</ion-label>
    </ion-item>
    <!--will be deleted later-->
    <ion-item button detail="true" [routerLink]="['/location']">
      <ion-label>Location</ion-label>
    </ion-item>
  </div>
</ion-content>
```

File Name: trusted-lists.page.html	
Description: One of the tabs for displaying the trusted list of the users. It displays an image and their names in their individual cards. Once clicked on a specific person, their information will show up, where the user can edit the person's name and number. It also includes a button where they can manually add people. The people on the user's list can be deleted by swiping left.	
Tags	**Tag Description**
ion-content	Content area for other tags to control scrollable area
ion-list lines= "none"	For displaying a list without lines.
ion-item-sliding *ngFor= "let item of contactItems$ | async"	Sliding container that can be swiped to delete for each component and to display all the trusted lists.
ion-items-options side="end"	For the item to be swiped to the left to display options such as delete.
app-contact-card [item]= "item"	Component holding trusted list information.
ion-item-option (click)= "removeFromList(item)"	For the item to be deleted, click on it when the options are displayed after swiping left.
ion-fab vertical= "bottom" horizontal= "end" slot= "fixed"	A circular button on the right button side of the page that is in a fixed position. Once clicked, it will take the user to the add page to manually add contacts.

```
<ion-content>
  <h1>Trusted List</h1>
  <div class="card">

<ion-list lines="none">
  <ion-item-sliding *ngFor="let item of contactItems$ | async">
    <ion-item>
      <app-contact-card [item]="item"></app-contact-card>
    </ion-item>

  <ion-item-options side="end">
    <ion-item-option (click)="removeFromList(item)">
      <ion-icon name="trash-outline" size="large" color="danger"></ion-icon>
    </ion-item-option>
  </ion-item-options>
</ion-item-sliding>
</ion-list>


  </div>



  <ion-fab vertical="bottom" horizontal="end" slot="fixed">
    <ion-fab-button [routerLink]="['/add-contacts']">
      <ion-ripple-effect></ion-ripple-effect>
      <ion-icon name="person-add-outline"></ion-icon>
    </ion-fab-button>
  </ion-fab>
</ion-content>
```

File Name: home.page.html	
Description: This page displays the tabs for the app. It displays all four tabs that can take the users to different pages based on their needs.	
Tags	**Tag Description**
ion-content	Content area for other tags to control scrollable area
ion-tabs	Navigation component and a container for individual tab components.
ion-tab-bar	A UI component that contains all the tab buttons.


```
<ion-content>
  <ion-tabs>
    <ion-tab-bar slot="bottom">

      <ion-tab-button tab="trusted-lists">
        <ion-icon name="people-outline"></ion-icon>
      </ion-tab-button>

      <ion-tab-button tab="history">
        <ion-icon name="list-outline"></ion-icon>
      </ion-tab-button>

      <ion-tab-button tab="main">
        <ion-icon name="radio-button-off-outline"></ion-icon>
      </ion-tab-button>

      <ion-tab-button tab="settings">
        <ion-icon name="settings-outline"></ion-icon>
      </ion-tab-button>

    </ion-tab-bar>
  </ion-tabs>
</ion-content>
```

File Name: contact-card.component.html	
Description: For displaying how the information of a person from the trusted list looks like. It is a component that is used in the trusted list tab.	
Tags	**Tag Description**
div class= “card” [id]= “item.id”	Class container to hold a person's information and their id. The container will be displayed based on the id number of the person according to the data from the database.
ion-avatar slot= “start”	Circular component to wrap a person's image.
ion-label	Contains the person’s first and last name passed from the database.
ion-modal #modal trigger= “{{ item.id }}	A component that pops up when the user clicks on the “card”. It displays the person’s information and an option for the user to edit it.
ng-template	Defines a template content
form	Contains the person’s information in an editable format of a form.

```
<div class="card" [id]="item.id">

      <ion-item lines="none">
        <ion-avatar slot="start">
          <img [src]="item.image"/>
        </ion-avatar>
        <ion-label>{{item.firstName}} {{item.lastName}}</ion-label>
      </ion-item>



    <ion-modal #modal trigger="{{ item.id }}">
      <ng-template>
        <ion-icon class="close" name="close-outline" (click)="modal.dismiss()"></ion-icon>

        <ion-content>

          <ion-avatar class="profile">
            <img [src]="item.image">
          </ion-avatar>

          <form>
           <input type="text" name="first" value="{{ item.firstName }}" required/>
           <br><br>
           <input type="text" name="last" value="{{ item.lastName }}" required/>
           <br><br>
           <input type="tel" name="phone" pattern="[0-9]{3}-[0-9]{3}-[0-9]{4}" value="{{ item.phone }} " required/>
           <div class="btm">
            <span>Save</span>
              </div>
             </form>
           </ion-content>
         </ng-template>
       </ion-modal>

     </div>

```

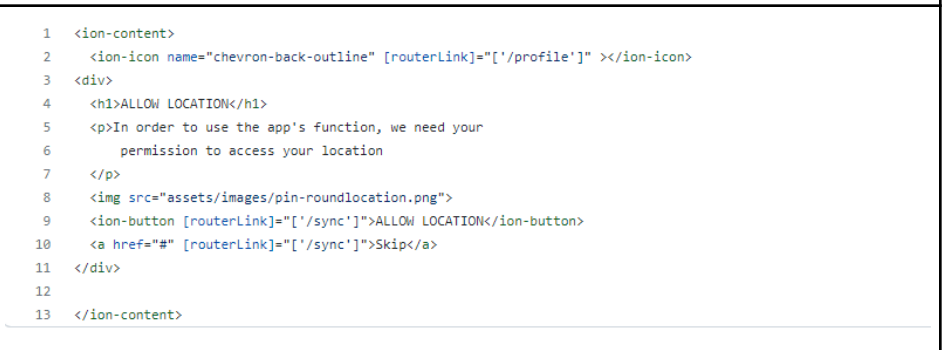
File Name: add-contacts.page.html	
Description: Display the page for manually adding contacts into the trusted list. This page appears when the user clicks on the plus button on the trusted list tab.	
Tags	**Tag Description**
header	Header contains an arrow which takes the user back to the trusted list tab.
div class = “card”	Container to hold information for adding contact.
form	Contains items for user input.
div class= “btns”	Container for buttons for users to make a choice between add (add to contact list) and next (add another person).

```
<ion-content>
  <header>
    <ion-icon name="close-outline" [routerLink]="['/home/trusted-lists']" ></ion-icon>
    <h1>Add Contacts</h1>
  </header>
  <div class="card">
 <!-- <img src="assets/images/person.png" /> -->


    <form>
      <input class="hidden-file-input" multiple="false" (change)="setPicture($event)"
        #pictureInput type="file" accept="image/*">
      <img id="profilePic" src="assets/images/person.png" (click)='pictureInput.click()'/>
      <br><br>

      <input type="text" name="first" placeholder="First Name" required/>
      <br><br>
      <input type="text" name="last" placeholder="Last Name" required/>
      <br><br>
      <input type="tel" name="phone" pattern="[0-9]{3}-[0-9]{3}-[0-9]{4}" placeholder="Phone Number" required/>
    </form>

    <div class="btns">
      <ion-button class="add">
        Add
      </ion-button>
      <ion-button class="next">
        Next
      </ion-button>
     </div>
  </div>
</ion-content>
```

File Name: add-now.page.html	
Description: This page displays the page for users to manually input their trusted list. This page is used for new users when they sign up.	
Tags	**Tag Description**
form	Contains items for user input.
div class= “btns”	Container for buttons for users to make a choice between add (add to contact list) and next (add another person).

```
<ion-content>
  <ion-icon name="chevron-back-outline" [routerLink]="['/signup']" ></ion-icon>

  <h1>Add Contacts</h1>
  <div class="card">

    <form>
      <input class="hidden-file-input" multiple="false" (change)="setPicture($event)"
        #pictureInput type="file" accept="image/*">
      <img id="profilePic" src="assets/images/person.png" (click)='pictureInput.click()'/>
      <br><br>

      <input type="text" name="first" placeholder="First Name" required/>
      <br><br>
      <input type="text" name="last" placeholder="Last Name" required/>
      <br><br>
      <input type="tel" name="phone" pattern="[0-9]{3}-[0-9]{3}-[0-9]{4}" placeholder="Phone Number" required/>
    </form>

    <div class="btns">
      <ion-button class="add">
        Add
      </ion-button>
      <ion-button class="next">
        Next
      </ion-button>
     </div>
     <span class="last">
      <a href="#" [routerLink]="['/home/main']">Skip</a>
     </span>
  </div>
</ion-content>
```

File Name: allow-location.page.html	
Description: This page is used for new users who just signed up. It displays the page to ask for permission for accessing their location.	
Tags	**Tag Description**
ion-content	Content area for other tags to control scrollable area
div	Contains items for display such as an image, text, and buttons.
	

```
<ion-content>
  <ion-icon name="chevron-back-outline" [routerLink]="['/profile']" ></ion-icon>
<div>
  <h1>ALLOW LOCATION</h1>
  <p>In order to use the app's function, we need your
      permission to access your location
  </p>
  <img src="assets/images/pin-roundlocation.png">
  <ion-button [routerLink]="['/sync']">ALLOW LOCATION</ion-button>
  <a href="#" [routerLink]="['/sync']">Skip</a>
</div>

</ion-content>
```

File Name: edit.page.html
Description: This page is for users to edit their profile. It displays the edit page and is located in the setting tabs.

Tags	Tag Description
div class= “card”	Class container card to hold items for displaying information.
form	Contain items for user inputs.
div class= “btn”	Class container to hold button.
	

```
<ion-content>
  <ion-icon name="chevron-back-outline" [routerLink]="['/home/settings']" ></ion-icon>
  <h1>Edit Profile</h1>
  <div class="card">

    <img src="assets/images/person.png" />

    <form>
      <input type="text" name="first" placeholder="First Name" required/>
      <br><br>
      <input type="text" name="last" placeholder="Last Name" required/>
      <br><br>
      <input type="email" name="eml" placeholder="Email Adress" required/>
      <br><br>
      <input type="tel" name="phone" pattern="[0-9]{3}-[0-9]{3}-[0-9]{4}" placeholder="Phone Number" required/>
      <br><br>
    </form>

    <div class="btn">
      <ion-button>
        SAVE
      </ion-button>
     </div>
  </div>
</ion-content>
```

File Name: location.page.html	
Description: This page is for displaying the location.	
Tags	**Tag Description**
ion-icon class= “arrow” name= “chevron-back-outline” [routerLink]= “"['/home/history']"	Displays an arrow icon to take the user back.
div class= “card”	Class container to hold items such as a card for displaying user current location.


```
<ion-content>

  <ion-icon class="arrow" name="chevron-back-outline" [routerLink]="['/home/history']" ></ion-icon>

  <div class="card">
    <div class="item">
      <span class="user"><ion-icon name="person-outline"></ion-icon>  Your location</span>
      <hr>
       <span class="locate"><ion-icon name="location-outline"></ion-icon>  Person's location</span>
    </div>
  </div>


</ion-content>
```

File Name: login.page.html	
Description: This page displays the login page for the app. Users can interact with this page and input their information if they have an account, or sign up. Users are given multiple choices to make in their page.	
Tags	**Tag Description**
form	Contains items for user input.
div class= “socials”	Container for holding items such as different social medias the user wished to login with.
div class= “bottom”	Container for button and another option if user forgets password.

```
<ion-content>
  <div class="top">
    <h1>Ping App</h1>
    <img src="assets/images/iconapp1.png">
  </div>
 <span>Login</span>
 <form>
  <input type="email" name="eml" placeholder="Email Adress" required/>
  <br><br>
  <input type="password" name="pwd" placeholder="Password" required/>
  <span class="pwd">Forgot Password?</span>
 </form>
 <span class="or">Or</span>

 <div class="socials">
  <ion-button class="btn-fb">
    <ion-icon name="logo-facebook"></ion-icon>
  </ion-button>
  <ion-button class="btn-go">
    <ion-icon name="logo-google"></ion-icon>
  </ion-button>
 </div>

 <div class="bottom">
  <ion-button class="btn-continue" [routerLink]="['/home']">CONTINUE</ion-button>
 </div>
 <span class="last">Don't have an account?<a href="#" [routerLink]="['/register']">Sign Up</a></span>
</ion-content>
```

File Name: profile.page.html

Description: This page is used for new users who just sign up. The page displayed a form for users to input in their information.	
Tags	**Tag Description**
ion-content	Content area for other tags to control scrollable area
form	Contains items for user input.
div class= “btn”	Containers for buttons to continue on to the next page.
span class= “last”	Span to link the user to the next page without inputting any information, allowing them to skip.

```
<ion-content>
  <ion-icon name="chevron-back-outline" [routerLink]="['/register']" ></ion-icon>
  <h1>Create Profile</h1>
  <div class="card">

    <img src="assets/images/person.png" />

    <form>
      <input type="text" name="first" placeholder="First Name" required/>
      <br><br>
      <input type="text" name="last" placeholder="Last Name" required/>
      <br><br>
      <input type="tel" name="phone" pattern="[0-9]{3}-[0-9]{3}-[0-9]{4}" placeholder="Phone Number" required/>
      <br><br>
    </form>

    <div class="btn">
      <ion-button [routerLink]="['/allow-location']">
        CONTINUE
      </ion-button>
     </div>
     <span class="last">
      <a href="#" [routerLink]="['/allow-location']">Skip</a>
     </span>
  </div>
</ion-content>
```

File Name: register.page.html	
Description: This is the sign up page for new users. It displays the information that users need to input inorder to continue using the app.	
Tags	**Tag Description**
ion-content	Content area for other tags to control scrollable area
form	Contains items for user input
div class= “socials”	Container for holding items such as different social medias the user wished to login with.
div class= “bottom”	Container for button for the user to continue onto the next page.

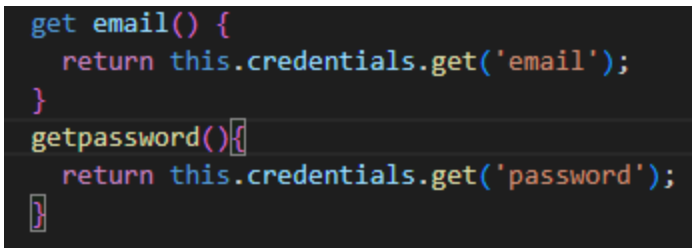
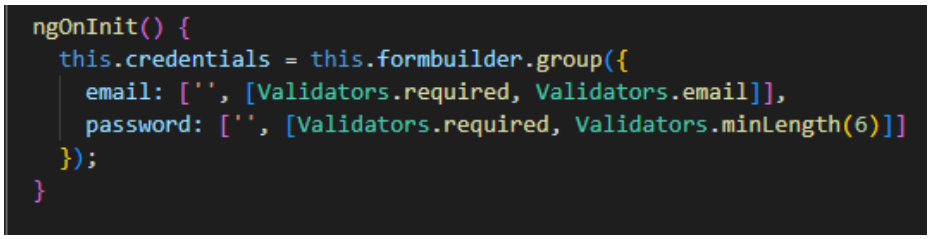
	

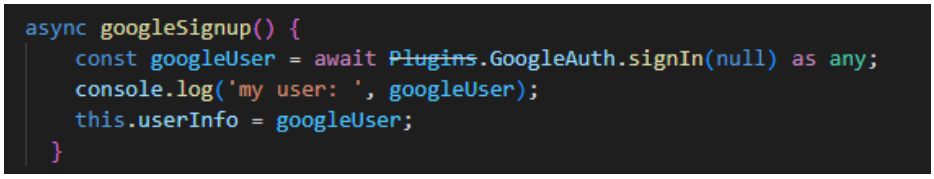
```
<ion-content>
  <div class="top">
    <h1>Ping App</h1>
    <img src="assets/images/iconapp1.png">
  </div>
  <span>Sign Up</span>
  <form>
   <input type="email" name="eml" placeholder="Email Adress" required/>
   <br><br>
   <input type="password" name="pwd" placeholder="Password" required/>
  </form>
  <span class="or">Or</span>

 <div class="socials">
  <ion-button class="btn-fb">
    <ion-icon name="logo-facebook"></ion-icon>
  </ion-button>
  <ion-button class="btn-go">
    <ion-icon name="logo-google"></ion-icon>
  </ion-button>
 </div>
 <div class="bottom">
  <ion-button class="btn-continue" [routerLink]="['/profile']">CONTINUE</ion-button>
 </div>
 <span class="last">
  <a href="#" [routerLink]="['/login']">Cancel</a>
 </span>
</ion-content>
```

File Name: sync.page.html	
Description: This page is only shown to new users who just sign up. It displays the information needed from the user before continuing.	
Tags	**Tag Description**
ion-content	Content area that includes items such as image, text, and buttons.
	

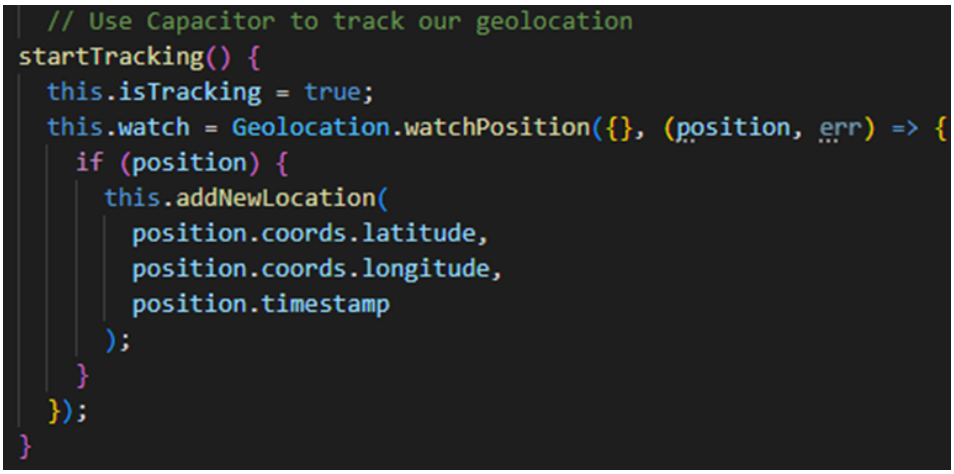
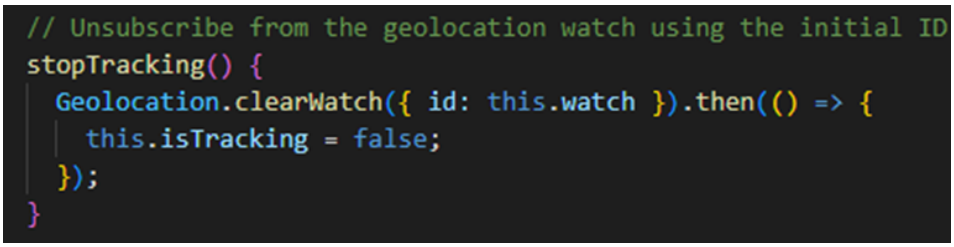
```
<ion-content>
  <ion-icon name="chevron-back-outline" [routerLink]="['/allow-location']" ></ion-icon>
  <div>
    <h1>Add Contacts to Trusted List</h1>
    <img src="assets/images/people.png">

  <ion-button>SYNC CONTACTS</ion-button>
 <span>OR</span>
 <ion-button class="other" [routerLink]="['/add-now']">MANUALLY INPUT</ion-button>
 <a href="#" [routerLink]="['/home']">Skip</a>
  </div>
</ion-content>
```

File Name: login.page.ts	
Description: The login page is where the user information is stored into the cloud database such as the email, password, and phone number. Now if they have a google email account they can sign in as well.	
Function	**Functions Descriptions**
getemail() and getpassword()	Generate a dialog box that asks for the user's email and password.

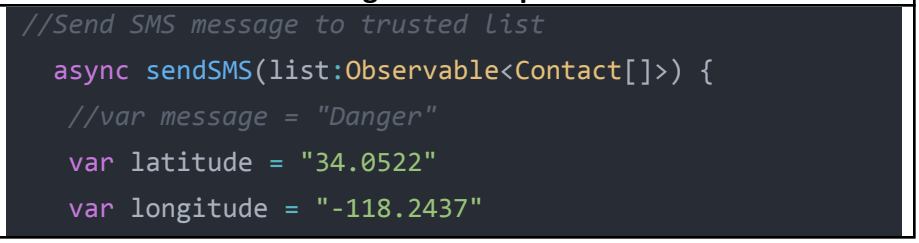
	
ngOnInIt()	This checks the validation of the users email and password to see if the meet the system minimum requirement.a
	
async register()	This allows a new user to create and register their information to the systems of the application. This then store the information into the database
	
async login(), **async showAlert()**	This is when the user has already registered and to return an sign in as a user drawing the credential from the database that is in place. Once that is done a pop up will show that everything's okay and you can move forward

	
async googleSignup	This dialog box allows the user to sign in if they have a google account.
	

File Name: home.page.ts	
Description: This the home page that will display handles for the different functions for the Google Map Geolocation. This includes getting the longitude and latitude position of your current location. Then with the help of the html file it will display your current location on the screen.	
startTracking()	This allow the user to track their current movement and get the coordinate for a more real time location, if they want to use this to send the location instead.

	
stopTracking()	This stops the tracking and mapping of your current location.
	

4.1.2 Software Interface

File Name: sms.serivice.ts	
Description: This is a service that handles the different functions for the native SMS messaging. This includes getting android permissions for SMS, checking for SMS permissions, and sending out the SMS with the user's location.	
Function	**Function Description**
sendSMS(list:Observable<Contact[]>)	This function takes in a list of contacts, creates a google map link with a specific longitude/latitude, and sends the list the link individually with a 2 second delay between each message.
	Program Description
	

```
    var link =
"https://www.google.com/maps/search/?api=1&query=" +
latitude + "%2C"+ longitude //Google map url + lat + comma
+ long
  //  var message = "Hello \n" +
"https://www.google.com/maps/search/?api=1&query=47.595151
8%2C-122.3316393"
    var message = "Current Location \n" + link
    var trustedNumbers = [];
   //Send to multiple numbers
    const sleep = (ms) => new Promise(r => setTimeout(r,
ms)); //Need delay between each send to process sent
messages
    const grabTrustedList = {
     //Grabs trusted list
     next: (myContactList: Contact[]) => {
       for (var myContact of myContactList) {
         //Name of contact
         console.log("Obrsever Subscribe: " +
myContact.displayName)
         for (var myNumber of myContact.phoneNumbers){
            // Grab mobile phone numbers
            if (myNumber.label == "mobile"){
              console.log("Observer Subscribe: ",
myNumber.number);
              trustedNumbers.push(myNumber.number)
            }
         }
        }
    },
    error: (err: Error) => console.error('Observer got an
error: ' + err),
    complete: () => console.log('Observer got a complete
notification'),
  };
   //this.contacts.subscribe(grabTrustedList);
   list.subscribe(grabTrustedList);
```

<table>
<tr><td></td><td>

```
   //Contact each trusted user's
      for(var i = 0; i < trustedNumbers.length; i++) {
       //Arrays don't work, iterate through for loop and
send 1 message at a time
       //this.sms.send(trustedNumbers,message)
       this.sms.send(trustedNumbers[i],message)
       await sleep(2000); //delay each message
      }
     console.log("Completed SMS messaging")
  }
```

</td></tr>
<tr><td>getSMSPermission(): Promise<void></td><td>Function Description</td></tr>
<tr><td></td><td>Requests for the app to have SEND_SMS android permissions. Users can accept or deny.</td></tr>
<tr><td></td><td>Program Description</td></tr>
<tr><td></td><td>

```
     //Request for SMS permissions
async getSMSPermission(): Promise<void> {
         console.log("Requesting SMS permissions...")

this.androidPermissions.requestPermission(this.androidPerm
              issions.PERMISSION.SEND_SMS)
                                  }
```

</td></tr>
<tr><td>async SMSAlert()</td><td>Function Description</td></tr>
<tr><td></td><td>Alerts the user if the app doesn’t have SMS permissions enabled</td></tr>
<tr><td></td><td>Program Description</td></tr>
<tr><td></td><td>

```
            //Alert for missing SMS permission
                        async SMSAlert() {
      const alert = await this.alertController.create({
                        header: 'Ping Denied',
       subHeader: 'App Permission Missing: SMS Access',
             message: 'Try again after enabling',
                         buttons: ['OK'],
                              });
```

</td></tr>
</table>

<table>
<tr><td></td><td>

```
        await alert.present();
    }
```

</td></tr>
<tr><td>checkSMSPermission(list)</td><td>

Function Description

Checks for SMS permissions of app. Requests permissions if the app doesn’t have them or sends messages to the trusted list if it does.

Program Description

```
//Checks SMS permissions for app
checkSMSPermission(list) {

this.androidPermissions.checkPermission(this.androidPermissions.PERMISSION.SEND_SMS).then(
    result => {
  //Send messages if SMS permissions are enabled
  if (result.hasPermission == true) {
   console.log("Has SMS permissions... sending sms")
   //Sends SMS to trusted list
   this.sendSMS(list);
  }
 //Request for SMS permissions if disabled
  else {
   console.log("Does Not have SMS permissions...")
   this.SMSAlert();
   this.getSMSPermission();
  }
 },
 err =>
this.androidPermissions.requestPermission(this.androidPermissions.PERMISSION.SEND_SMS)
 );
}
```

</td></tr>
</table>

File Name: contacts.serivice.ts

Description: This is a service that handles the different functions for the contact syncing. This includes getting android permissions for phone contacts, checking for contact permissions, grabbing the phone contacts and adding them to the trusted list

Function	Function Description
getContactPermissions(): Promise<void>	Request for app to have READ_CONTACTS android permissions. Users can accept or deny.
	Program Description

```
//Invokes android permissions for Contacts
async getContactPermissions(): Promise<void> {
        console.log('button clicked');
        Contacts.getPermissions();
                }
```

getContacts(): Promise<void>	**Function Description**
	Grabs contacts from the user's phone. Sorts the results by name and stores them into the trusted list.
	Program Description

```
//Grab phone contacts
async getContacts(): Promise<void> {
   Contacts.getContacts().then(result => {
      console.log('result is:' , result);
      const phoneContacts: Contact[] = result.contacts;
      //Sort phone contacts by display name
      phoneContacts.sort((a, b) =>
a.displayName.localeCompare(b.displayName))
      this.contacts = of(phoneContacts);
   });
  }
```

addTrustList(name:string,number:string)	**Function Description**
	Takes in a name and number as parameters and creates a contact. Inserts new contact into the trusted list and sorts by name.
	Program Description

```
//Add a contact to trusted list
async addTrustList(name:string,number:string) {
```

```
    //Creates a new Contact
    //var addedName = "Adam Added"
    var addedName = name
    var numberType = "mobile"
    //var addedNumber = "5599876543"
    var addedNumber = number

    const newPhonenumber: PhoneNumber = {
      label: numberType,
      number: addedNumber
    }
    //Create a new phonenumber and email address
    let contactList: PhoneNumber[] = [newPhonenumber];
    var emailList: EmailAddress[];
    var newContact: Contact = {
      contactId: "",
      displayName: addedName,
      phoneNumbers: contactList,
      emails: emailList,
      photoThumbnail: "",
      organizationName: "",
      organizationRole: "",
      birthday: ""
    }
    //Insert new contact into trusted list
    const insert = {
      //Grabs trusted list
      next: (myContactList: Contact[]) => {
        //Inserts new contact into trusted list
        myContactList.push(newContact)
        //Sort Contact list by display name
        myContactList.sort((a, b) =>
a.displayName.localeCompare(b.displayName))
      },
      error: (err: Error) => console.error('Observer got an
error: ' + err),
```

```
      complete: () => console.log('Observer got a complete
notification'),
    };
    this.contacts.subscribe(insert);

  }
```

removeTrustList(targetName: string,targetNumber:string)	**Function Description**
	Searches the trusted list using a target name and number. If a user exists, removes them from the trusted list
	Program Description

```
 //Remove a contact
  async removeTrustList(targetName:string,targetNumber:string) {
  //  var targetName = "Adam Added"
    //var targetNumber = "5599876543"
    var index = 0;
    //Looks for target in trusted list and removes them
    const remove = {
      next: (myContactList: Contact[]) => {
        for (var myContact of myContactList) {
        //Looks for a matching name in trusted list
         if (myContact.displayName == targetName){
          console.log(myContact.displayName + " found at index:
" + index)
          //Looks for matching mobile number
            for (var myNumber of myContact.phoneNumbers){
            //Mobile phone numbers
             if (myNumber.label == "mobile" && myNumber.number
== targetNumber){
              myContactList.splice(index, 1); // 2nd parameter
means remove one item only
              console.log(myNumber.number + " found at index: "
+ index);
              return
             }
          }
          index++
```

```
        }
        console.log("No trusted contact with info: " +
targetName + " " + targetNumber);
      },
      error: (err: Error) => console.error('Observer got an
error: ' + err),
      complete: () => console.log('Observer got a complete
notification'),
    };
    this.contacts.subscribe(remove);
  }
```

contactAlert()

Function Description

Alerts the user if the app doesn't have Contact permissions enabled

Program Description

```
    //Alert for missing contact permission
    async contactAlert() {
        const alert = await this.alertController.create({
          header: 'Contact Syncing Denied',
          subHeader: 'App Permission Missing: Contact Access',
          message: 'Try again after enabling',
          buttons: ['OK'],
        });

        await alert.present();
      }
```

checkContactsPermission()

Function Description

Checks for Contact Access permissions of app. Requests permissions if the app doesn't have them or syncs contacts to the trusted list if it does.

Program Description

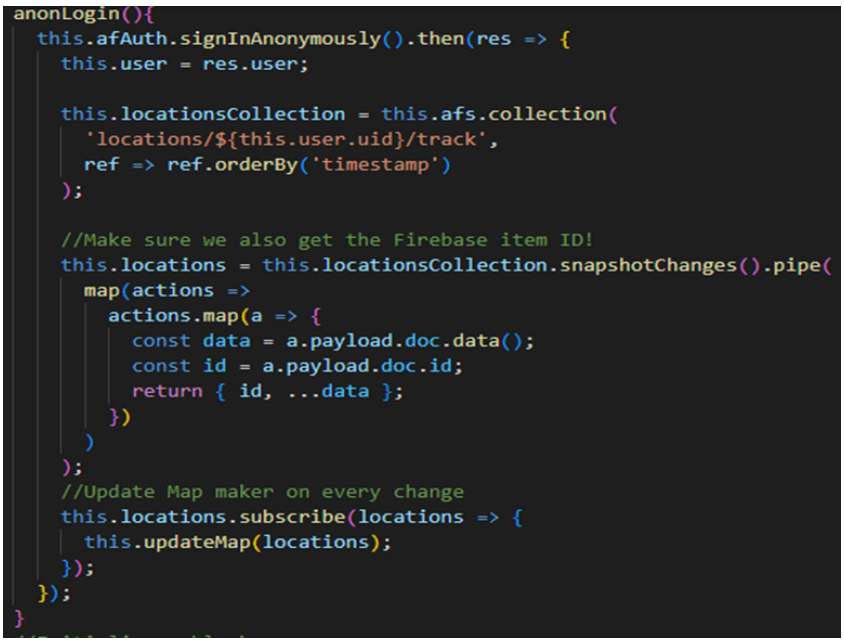
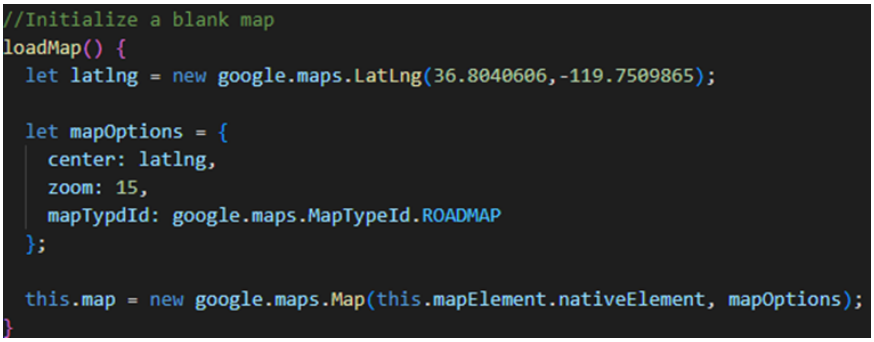
```
  //Checks contact permissions for app
checkContactsPermission() {

this.androidPermissions.checkPermission(this.androidPermissions.
PERMISSION.READ_CONTACTS).then(
```

```
      result => {
                  //Send messages if SMS permissions are enabled
                  if (result.hasPermission == true) {
                    console.log("Has Contact permissions...
grabbing contacts")
                    this.getContacts();
                  }
                  //Request for SMS permissions if disabled
                  else {
                    console.log("Does Not have Contact
permissions...")
                    this.contactAlert();
                    this.getContactPermissions();
                  }
                },
      err =>
this.androidPermissions.requestPermission(this.androidPermission
s.PERMISSION.SEND_SMS)
    );
  }
```

logTrustList()	**Function Description**
	Logs the current trusted list for testing purposes
	Program Description

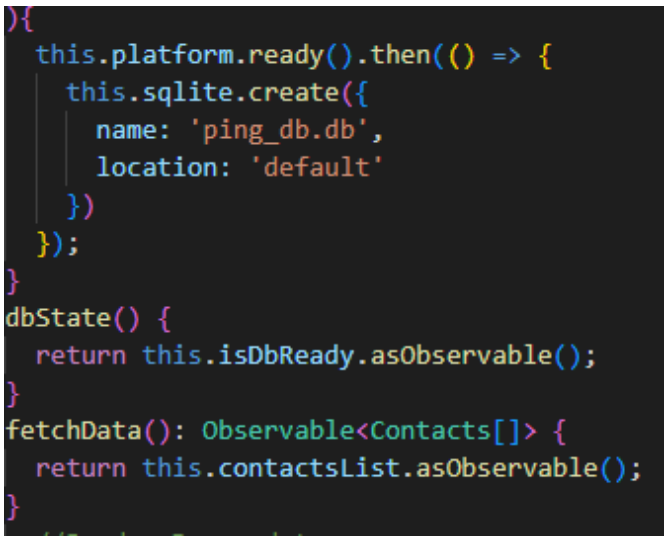
```
  //Log contact list for testing purposes
  async logTrustList() {
   const myObserver = {
     //Grabs contact list from phone
     next: (myContactList: Contact[]) => {
       for (var myContact of myContactList) {
         //Name of contacts
        console.log("Observer Subscribe: " +
myContact.displayName)
         for (var myNumber of myContact.phoneNumbers){
           //Mobile phone numbers
           if (myNumber.label == "mobile"){
             console.log("Observer Subscribe: ",
myNumber.number);
           }
         }
       }
     },
     error: (err: Error) => console.error('Observer got an
error: ' + err),
     complete: () => console.log('Observer got a complete
notification'),
   };
   this.contacts.subscribe(myObserver);
 }
```

File Name: home.page.ts	
Description: This the home page that will display handles for the different functions for the Google Map Geolocation. This includes getting the longitude and latitude position of your current location. Then with the help of the html file it will display your current location on the screen.	
Functions	**Function Descriptions**
anonLogin()	**This is a set up for anonymous signing in a user to use the map feature.**

	
loadMap()	**This lets the user see a map that is generated onto your screen. The map is going to be generated from the user's current location.**
	

4.1.3 Database Access

File Name: database.service.ts	
Description: This is the database set up for the application. This is where the user information is collected in store into the cloud database. This will allow the database to get the contact of the user whether it be a single or multiple contacts to store it into the database. It will also allow the database to update and delete any contact as well.	
Functions	**Functions Descriptions**
constructor()	This will initialize the start of the database to access for storing all of the information.

	
getFakeData()	This is a list of dummy data that is rendered to show user that the database is operational.This also allow use to inject data into the applications.
	
getContacts() and getContact()	These are the functions that the database uses to gather and generate the information and print it out.

<table>
<tr><td></td><td>

```
getContacts(){
  return this.storage.executeSql('SELECT * FROM contactData', []).then(res => {
    let items: Contacts[] =[];
    if(res.rows.length > 0) {
      for(var i =0; i < res.rows.length; i++){
        items.push({
          id: res.rows.item(i).id,
          person_name: res.ros.item(i).person_name,
          phone_num: res.row.item(i).phone_num,
          email: res.row.item(i).email
        });
      }
    }
    this.contactsList.next(items);
  });
}
```

```
//Grabing on contact
getContact(id): Promise<Contacts> {
  return this.storage.executeSql('SELECT * FROM contactData WHERE id = ?', [id]).then(res => {
    return {
      id: res.rows.item(0).id,
      person_name: res.rows.item(0).person_name,
      phone_num: res.rows.item(0).phone_num,
      email: res.rows.item(0).email
    }
  });
}
```

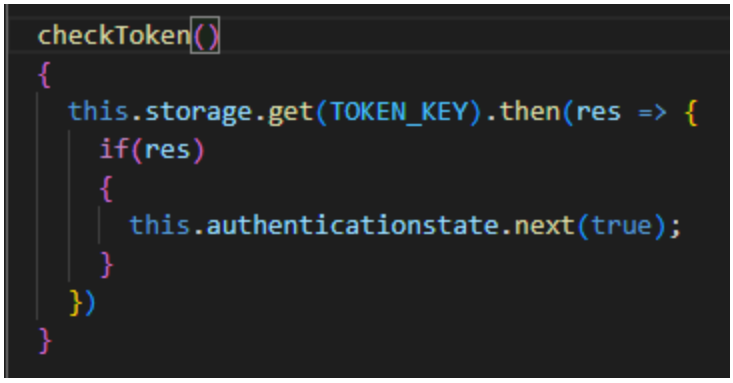
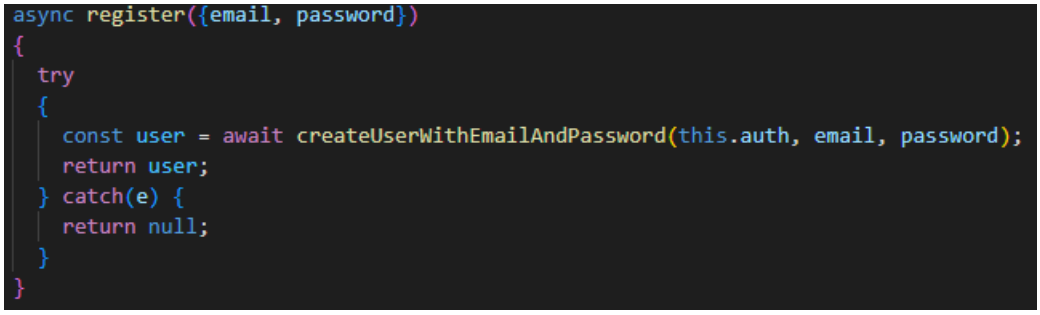
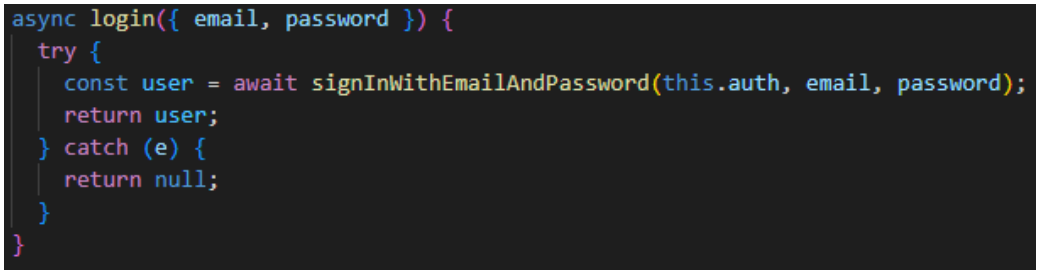
</td></tr>
<tr><td>addContact()</td><td>This allow for the adding of contact into the database.</td></tr>
<tr><td></td><td>

```
//Add
addContact(person_name, phone_num, email){
  let data = [person_name, phone_num, email];
  return this.storage.executeSql('INSERT INTO contactData (person_name, phone_num, email) VALUES (?
  .then(res => {
    this.getContacts
  })
}
```

</td></tr>
<tr><td>upDate() and delete()</td><td>This allows for the updating or deleting of contact from and to the database.</td></tr>
<tr><td></td><td>

```
//Update
updateContacts(id,contact: Contacts){
  let data = [contact.person_name,contact.phone_num, contact.email];
  return this.storage.executeSql('UPDATE contactData SET person_name = ?, phone_num = ?, email = ? WHERE id = ${id}', data)
  .then(data =>{
    this.getContacts();
  })
}
//Delete
deleteContacts(id){
  return this.storage.executeSql('DELETE FORM contactData WHERE id = ?',[id])
  .then(_=>{
    this.getContacts();
  })
}
```

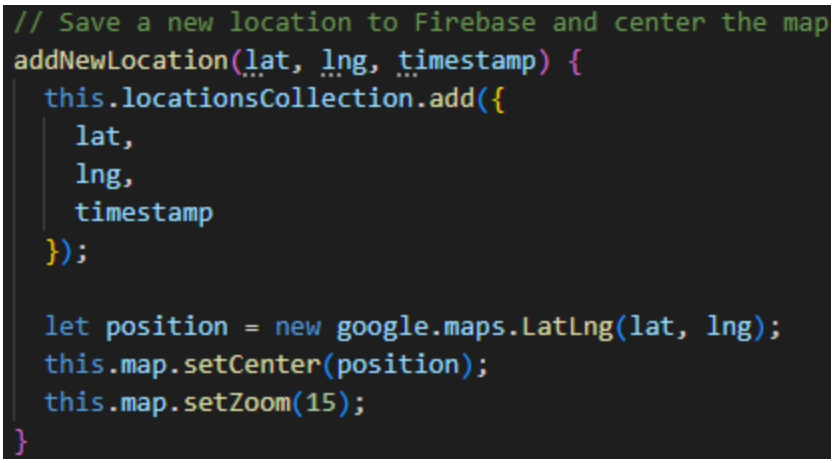
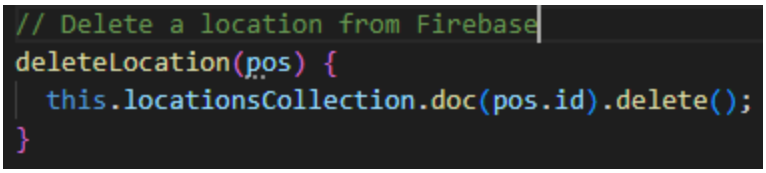
</td></tr>
</table>

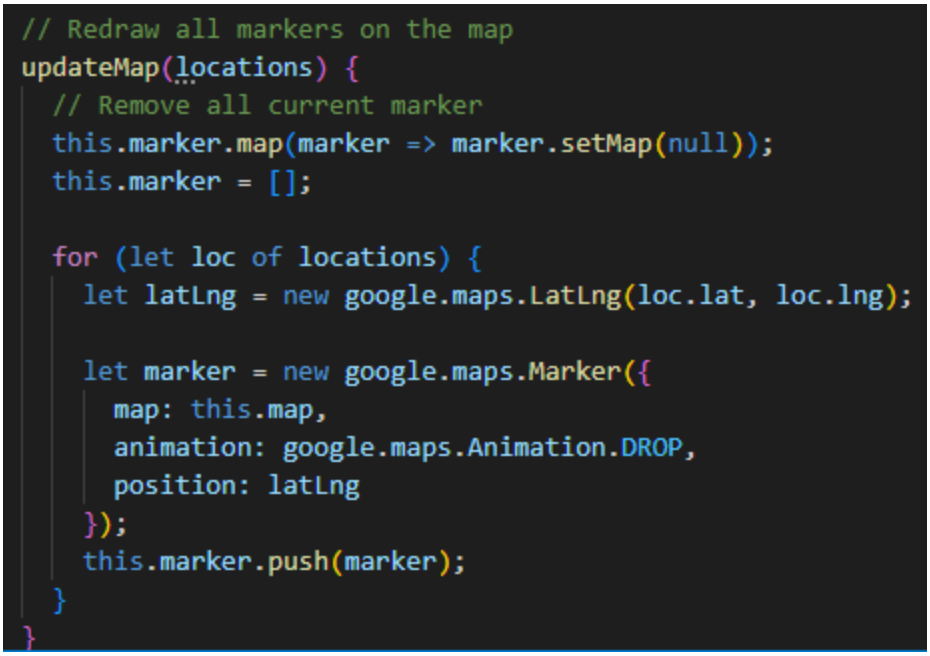
File Name: auth.service.ts	
Description: The authentication service is the security of the app. It allows the user to login and to ask for permission access to the users information.	
Functions	**Functions Description**
checkToken()	This checks a user specific token that is assigned to them that allow access to the app. The token is stored in the cloud database.
	
async register()	This is the setup of the registration to the database. So, when they register into the app this catches that information and is then stored into the database.
	
async login()	This is the setup of the login to the database. So, when they login into the app this catches that information and is then stored into the database.
	

```
checkToken()
{
  this.storage.get(TOKEN_KEY).then(res => {
    if(res)
    {
      this.authenticationstate.next(true);
    }
  })
}
```

```
async register({email, password})
{
  try
  {
    const user = await createUserWithEmailAndPassword(this.auth, email, password);
    return user;
  } catch(e) {
    return null;
  }
}
```

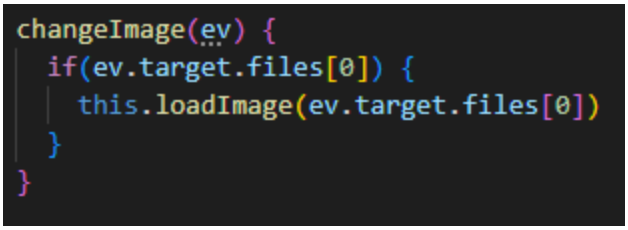
```
async login({ email, password }) {
  try {
    const user = await signInWithEmailAndPassword(this.auth, email, password);
    return user;
  } catch (e) {
    return null;
  }
}
```

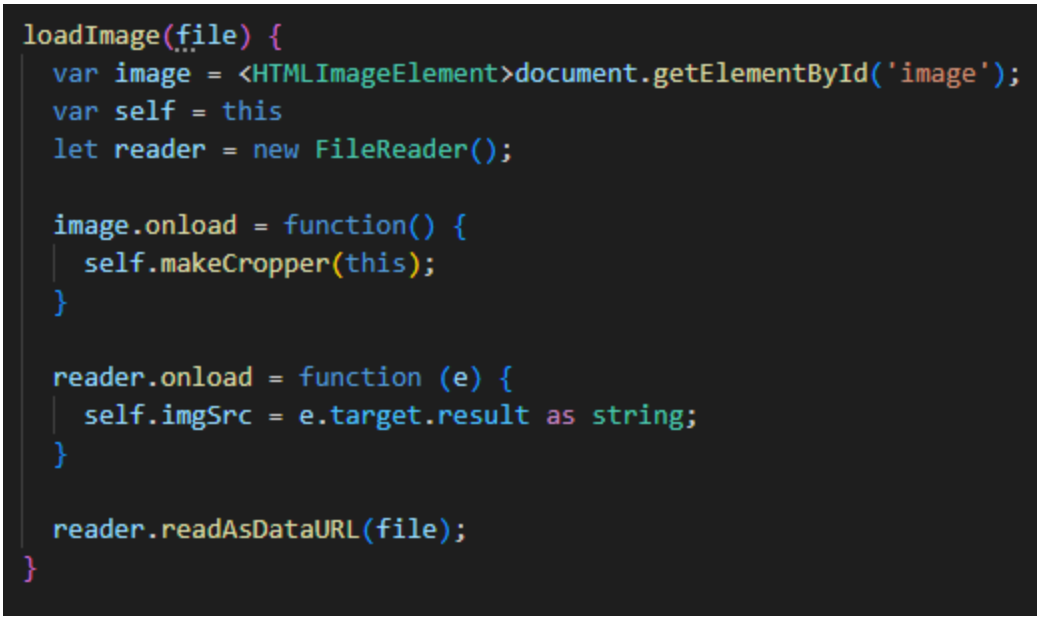
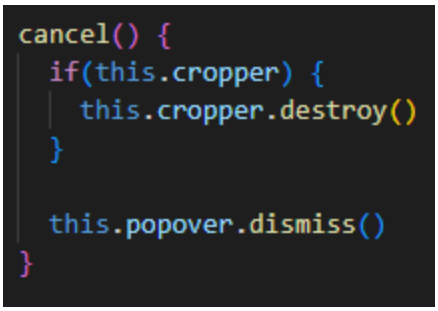
File Name: home.page.ts	
Description: This the home page that will display handles for the different functions for the Google Map Geolocation. This includes getting the longitude and latitude position of your current location. Then with the help of the html file it will display your current location on the screen.	
addNewLocation()	**This saves the user's previous location on to a cloud database.**
	
deleteLocation()	**This deletes any of the user's previous location from the database.**
	
updateMap()	**Reset the map of the user's and it also redraws the marker on the map and it also removes all current markers.**

```
// Redraw all markers on the map
updateMap(locations) {
  // Remove all current marker
  this.marker.map(marker => marker.setMap(null));
  this.marker = [];

  for (let loc of locations) {
    let latLng = new google.maps.LatLng(loc.lat, loc.lng);

    let marker = new google.maps.Marker({
      map: this.map,
      animation: google.maps.Animation.DROP,
      position: latLng
    });
    this.marker.push(marker);
  }
}
```

File Name: image-cropper.component.ts	
Description: This component can be used to handle user input for image files, and will allow the user to select a portion of the image to crop for use as a profile picture. This component makes use of Ionic's popover system in order to display the component on top of any other page.	
Functions	**Function Description**
makeCropper()	This function will create a new Cropper object, which handles all the internal logic for cropping images.

	
changeImage()	This function allows the user to upload a new image, and will swap out the previous image being processed by the cropper with a new one.
	
loadImage()	This function takes in a file and prepares a FileReader. This class allows the use of an onload event in order to set the cropper's target as soon as the image finishes being read.

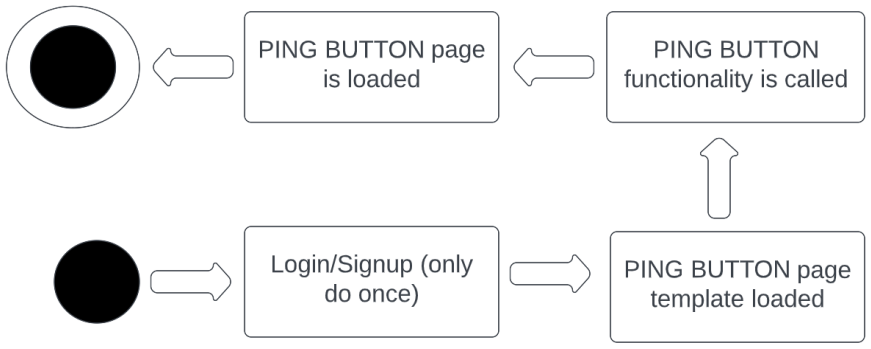
	
cancel()	This function handles destroying the cropper object and dismissing the popover so the page underneath is usable again.
	
done()	This function is called when the user submits their desired crop settings. By returning the file and the cropData, the application can know exactly how to crop the image for the user, as well as have access to the preview image for use in displaying on the page. This function also needs to dismiss the popover and destroy the cropper object.

```
done() {
  var returnData = {
    'sourceImage': {
      'file': this.imgFile,
      'cropData': this.cropper.getData(true)
    },
    'previewImage': this.cropper.getCroppedCanvas()
  };

  if(this.cropper) {
    this.cropper.destroy()
  }

  this.popover.dismiss(returnData)
}
```

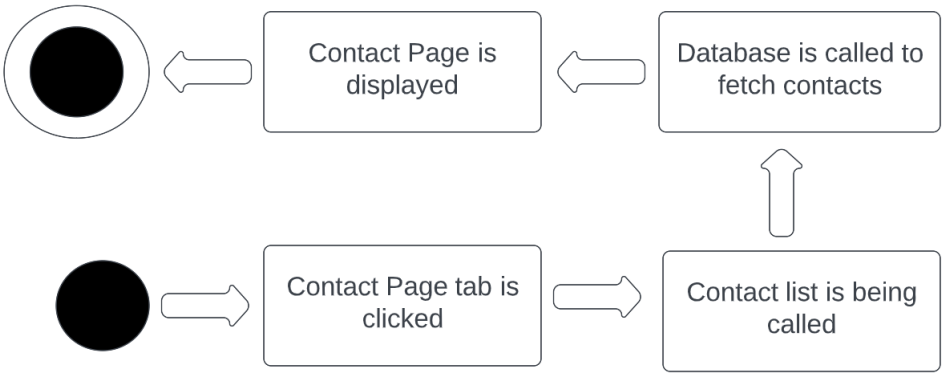
Section 5 - Dynamic Model

5.1 Sequence Diagram

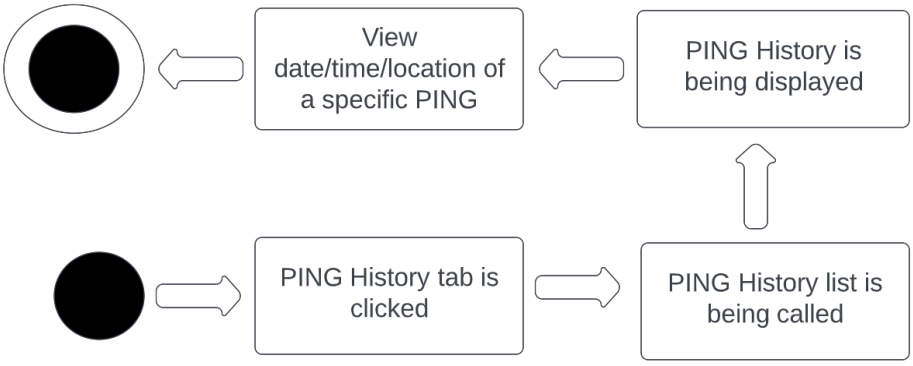
5.1.1 Ping Page



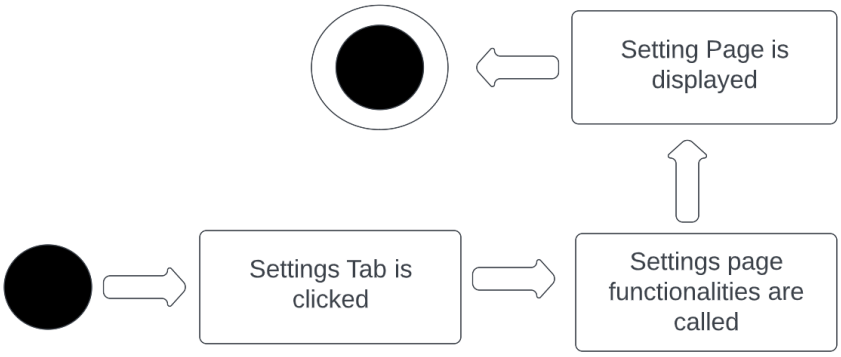
5.1.2 Contact Page



5.1.3 PING History Page



5.1.4 Settings Page



Section 6 - Non-Functional Requirements

6.1 Performance Requirements

- The application should have a smooth navigation experience for the users
- The PING button functionality should work as expected
- The application should send out text messages in a timely manner
- The map location link should be more than 80% accurate with fetching the user's current location
- The database must be up to date and working while the application is running.

6.2 Design Constraints

- Once the PING button is pressed, text messages are sent one by one to each user on the contact list as an array instead of sending messages at the same time to the user; however, the timing gap between each message sent is small.