Emerge

Disaster Relief Messaging Application

Project Total Documentation

Version 2

By

Luis Arevalo
008226335

Justin Passanisi
007140055

Spring 2017

SE 148 Computer Networks I

Revision History

Date	Description	Author(s)	Comments
2/10/2017- 2/20/2017	Project abstract and general architecture diagram completed	Luis Arevalo Justin Passanisi	
2/21/2017- 3/2/2017	Version 1.1 started, Sections added: Revision history, table of contents,	Luis Arevalo Justin Passanisi	
3/3/2017- 4/9/2017	Next major update. All current changes are being included in this new version 2. Adding intro and related work, and implementation status.	Luis Arevalo Justin Passanisi	Updated design diagrams
4/10/2017 - 5/24/2017	Updated implementation with setup tutorial, Updated implementation with tutorials and current work, added sections for “What we have learned and created?”	Luis Arevalo Justin Passanisi	

Emerge

Disaster Relief Messaging App

Arevalo, Luis E.
San Jose State University
San Jose, California

Passanisi, Justin D.
San Jose State University
San Jose, California

Abstract Emergency alerts have played a pivotal role in keeping both countries and states safer against the destructive forces of natural disasters. These alert systems, like the United States Emergency Broadcast System (EAS), have allowed individuals to receive notifications on potential dangers that are approaching before it is too late. Before such systems were implemented, many cities faced natural disasters with little to no response time between awareness and hazard. These systems can make the difference between an individual being outside and unaware or being inside and safe, while having time to prepare for what is incoming.

During an emergency situation, information must be spread across a large area in timely manner. Speed is a key factor in these systems, as information must arrive in seconds through multiple sources, such as phone calls, texts, and emails. Availability is another major concern within this application. During disasters, communications and power may go down at any moment and this application needs to address how to inform people adequately.Usability is another essential component, as people of all ages and all nationalities will need to utilize these services during an emergency.

This system will apply a geo-fence around cities and states, such that when a natural disaster is detected, a group of people will receive alert notifications of immediate dangers. Once the information has reached the recipients, this application will allow users to send information to other users, including text, photo, and video. This serves to allow the users to update the app with relevant news on any dangers and hazards that other users in the area should look out for. By using gps and a check system, this application will allow users to confirm their safety during and after a natural disaster and will display a list of safe users for others to see. Lastly, this application will periodically receive information packets from local news sources to provide an intuitive user interface to connect disaster reports with general users.

Keywords Emergency Notification System • Geo-fencing • Disaster Relief

February 20, 2017

Table of Contents

Chapter 1: Introduction

Emerge, short for Emergency, is the new expansion upon the current emergency alert systems in place today. Current emergency alert systems often allow users to interact with social media and technology to receive and send data concerning danger, safety, and aid. These systems provide mass bursts of notifications to areas that may be affected by dangerous emergencies. These emergencies can include fires, earthquakes, tornadoes, and more, and often can be quite deadly. By implementing these emergency broadcast systems, people are given an extra chance to find safety before it is too late. In this section, we will delve into articles pertaining to current systems, as well as current design structures that better aid this application's usability. These include some of the research documents being utilized by the Emerge team.

1.1 Literature Search

Below contains the articles pertaining to our inspiration, methods, and thinking.

Article: Introducing Safety Check

Summary: During natural disasters, it is commonly hard to reach or notify friends, family, and loved ones. Major disasters, like the Japanese earthquake and tsunami in 2011, caused many people to reach out for applications to use to check in on loved ones. Facebook was one of these sources and, through demand, decided to create a tool to let users confirm their safety. This safety check feature was later further developed to not only allow users to let others know they are safe, but to also allow users to comment and reply on safety status'. In addition, if one is in the affected area, they receive a notification to confirm their safety or update their coordinates. Between October 2014 and November 2015, Facebook was able to provide these services for eight distinct events, including the earthquakes in Nepal (Gleit, N., 2014). Facebook is gearing towards making this module a community driven tool in order to allow users to aid the alert system, such that Facebook will no longer need to be required to activate it during emergencies.

Importance: Emerge is also offering a safety check feature to allow users to confirm their safety, mark their location, and/or request for help. This Facebook feature is a prime example that can be used for referencing. Specifically this feature allows us to see potential user interface concepts and navigational concepts to ensure usability and reliability from within this section of the app.

Article: Emergency Alert System

Summary: A major service provided for emergency situations in the United States is the Emergency Alert System (EAS). This service operates through cable television systems, wireless cable systems, and satellite-digital-audio-radio service (SDARS) providers (Federal Communication Systems, n.d.). In this system, when a disaster occurs a message is broadcasted across all devices contained within the immediate area. This serves to update a mass group of people based on location that may be or are in danger of a disaster. States can also activate this service to alert its citizens to evacuate or prepare for a disaster to occur. In a

larger scale, the president can also utilize this system to send out a nationwide alert. By applying an “area of affect” broadcast, the EAS is ensured to reach those in the most danger quickly.

Importance: Taking inspiration from the EAS, Emerge will use geolocation of devices and/or accounts to provide alerts to only devices that are in the most danger. However, we will still allow users to view all disasters in the world, using EAS as a baseline, we can ultimately make our services more directed and focused. This system allows for law enforcement and medical aid to operate more smoothly during a disaster by providing focused data to affected areas.

Article: Switching Child View Controllers in iOS with Auto Layout

Summary: This article provides a tutorial showing how to do a dynamic view switch in swift using a container, buttons, child view controllers, and auto layout. In this, a container is put into a view and programmatically told what to load into the container. By doing this, it allows for a button press to change what is viewed within the container, rather than navigating to a new view entirely. The way this is set up also provides the functionality that the views stop being rendered as soon as they exit the container view.

Importance: This has increased usability and readability of both our design and workspace. By reducing four repeated segue connections from 4+ different views, or 16+ connections in total, down to only a total of four connections, we have decreased the time and effort needed to navigate through our application. This has helped in both needed computation and memory, as well as reduced the amount of time it takes to go from one screen to the next.

1.2 State-of-the-Art Summary

The current systems in place for emergency alerts, like the EAS and Facebook Safety Check, have been proven to be immensely helpful in emergency situations. The current services today have provided many features allowing users to ask for help easier, check in to make sure others know they are safe, and provide real time updates as events occur. However, these systems also have one major drawback: they require activation by a person to alert people.

In systems like these, the faster an alert is provided to people, the more chance people have to react appropriately and find safety. This is absolutely essential in designing for these systems and however fast they may be at sending out alerts, the human factor in activation will keep the industry a step behind the potential future. Emerge intends to provide a new age rendition of these systems by providing all the common services and listen to news media websites for consistent updates. By opening the channel to listening to a multitude of respectable sources, when a single event occurs and is posted anywhere within these sources, Emerge will immediately alert all users within the danger area. Using this key functionality, Emerge can

provide faster and more consistent updates. Collectively, Emerge will be the new direction of this field and will expand upon current emergency alert systems to provide remarkable improvements.

1.3 References

Federal Communications System. (n.d.). Emergency Alert System (EAS). Retrieved April 10, 2017, from https://www.fcc.gov/general/emergency-alert-system-eas

Gleit, N.(Oct. 15, 2014). Introducing Safety Check. Retrieved April 10, 2017, from https://newsroom.fb.com/news/2014/10/introducing-safety-check/

Woelmer, M. (Oct. 13, 2015). Switching Child View Controllers in iOS with Auto Layout. Retrieved from https://spin.atomicobject.com/2015/10/13/switching-child-view-controllers-ios-auto-layout/

Chapter 2: Requirements Specification

2.1 Functional Requirements

2.1.1 Use Cases

3.1.1.A. Use Case: User Sign Up	
Actors:	*GeneralUser*, *Database*
Entry Condition:	*GeneralUser* has installed the application
Exit Condition:	*GeneralUser*'s account is added into the *Database*
Level:	MMF
Quality Requirements:	Security, Usability
Steps/Flow of Events:	1. User clicks on the sign up button. 2. User fills out fields: username, password, confirm password, name, phone, and email. 3. System verifies information 4. User clicks sign up button to confirm information provided
Variations:	i. Continuing from step 3: System find fault(s) in verification process and alerts user ii. User fixes errors iii. System verifies information again, continuation of step 3 again.

3.1.1.B. Use Case: User Sign In	
Actors:	*RegisteredUser, Database*
Entry Condition:	*RegisteredUser* is not signed into the system
Exit Condition:	*RegisteredUser* is signed in the system
Level:	MMF
Quality Requirements:	Performance, Safety
Steps/Flow of Events:	1. *RegisteredUse*r enters username and password. 2. Database authenticates *RegisteredUser.* 3. System navigates user to main screen.
Variations:	1. *RegisteredUser* places in wrong password or email 2. System alerts user that the password or email is invalid.

3.1.1.C. Use Case: To Reset Password	
Actors:	*RegisteredUser, Database*
Entry Condition:	*RegisteredUser* loaded up application and is not logged in
Exit Condition:	*RegisteredUser*'s password is updated
Level:	MMF
Quality Requirements:	Safety
Steps/Flow of Events:	1. *RegisteredUse*r clicks forgot password link 2. *RegisteredUser* is navigated to Firebase password reset screen and puts in information 3. System sends alert email to user for security purposes
Variations:	n/a

3.1.1.D. Use Case: To Message Users	
Actors:	*RegisteredUser, Database*
Entry Condition:	User is registered in the firebase database
Exit Condition:	*RegisteredUser* sends message to another user
Level:	MMF
Quality Requirements:	Performance, Usability
Steps/Flow of Events:	1. *RegisteredUse*r clicks on message icon 2. System displays list of all messages the user has 3. RegisteredUser selects New Message Button 4. RegisteredUser selects recipient and fills out subject and content and clicks send 5. Message is sent to recipient with extra metadata including date, time, sender name, optional: location
Variations:	Variation 1: Not Signed In i. User is not signed in ii. System alerts user to sign in to use functionality Variation 2: Existing Message i. Continuation of step 2: User selects existing conversation between another user ii. User is displayed messages and content box for reply Variation 3: Sending Picture i: Continuation of step 4, before clicking send. User selects add picture button. ii. System displays options to use camera or library. iii. User selects option and adds photo iv. Photo is added to content v. Continuation of step 5 Variation 4: Sending Audio i: Continuation of step 4, before clicking send. User selects add audio button. ii. System displays options to use microphone or library. iii. User selects option and adds audio iv. Audio is added to content v. Continuation of step 5

	Variation 5: Sending Video/GIF i: Continuation of step 4, before clicking send. User selects add video button. ii. System displays options to use camera or library. iii. User selects option and adds video/GIF iv. Video/GIF is added to content v. Continuation of step 5

3.1.1.E. Use Case: To Confirm Safety	
Actors:	*GeneralUser, RegisteredUser, Database*
Entry Condition:	*User* selects pin button in application
Exit Condition:	*User* is registered as safe during an event
Level:	MMF
Quality Requirements:	Performance, Usability
Steps/Flow of Events:	1. *User* is take to a screen containing a list of safe users 2. *GeneralUser* selects to confirm safety and is taken to a screen to fill out name, and photo 3. User selects the confirm safety button 4. Database stores *User* into list with name, photo, date, time, and coordinates
Variations:	Variation 1: *RegisteredUser* i. Continuation of step 2: RegisteredUser's information is automatically put into fields ii. Continue on step 3.

3.1.1.F. Use Case: To Request Help	
Actors:	*GeneralUser, RegisteredUser, Database*
Entry Condition:	*User* selects pin button in application
Exit Condition:	*User* is registered as needing assistance during an event and information is saved/sent to emergency responders
Level:	MMF
Quality Requirements:	Performance, Usability
Steps/Flow of Events:	1. *User* is take to a screen containing a list of safe users 2. *User* selects needing assistance 5. System alerts user of assistance request recieved 3. Database stores *User's* location, date, and time and displays an option for user to add photo and name. 4. User puts in photo and name and selects update information button 5. System alerts user of update
Variations:	Variation 1: *RegisteredUser* i. Continuation of step 3: RegisteredUser's information is automatically put into photo and name. Skips steps 4 and 5.

2.2 Non-Functional Requirements

The following sections contain the non-functional requirements of this application.

2.2.1 Performance

The performance of Emerge is expected to hold that all functionalities take less than three clicks/actions by the user to execute. The response time between emergency occurrence and notification to the user should also occur in less than five seconds.

2.2.2 Security

Due to Emerge being account operated, it is expected that all information is encrypted during transmission. Emerge also holds location data, so it is expected that no user can view others location data, except in the confirmed safe list.

2.2.3 Reliability

Emerge is a disaster relief messaging app, therefore it is expected that Emerge is up and available 98% of the time during a year, with an allotted 2% of downtime for updates.

2.2.4 Usability

This version of Emerge is a MMF and, in so, usability is major criteria. Emerge intends to be intuitive enough for users to recognize distinct actions through buttons and menus.Placement of elements should follow Fittz law and wording should be clear and directive.

2.2.5 Supportability

Emerge's MMF version is supportable by iOS. Future releases should be supportable by Android and web.

2.2.6 Implementation

Implementation for Emerge is in Swift 3.0 using XCode on MACOS ElCapitan/MACOS Sierra.

2.2.6 Interface

User interface is a future concern of Emerge, as usability is a primary concern for the MMF version.

2.2.7 Packaging

Emerge is packaged as an iOS application directed for the apple app store.

2.2.8 Legal

When Emerge is publically available, users must agree to a Terms of Service and Privacy agreement, such that users are fully aware of the agreements set forth by the Emerge application.

Chapter 3: Determinants

This chapter contains all determinants and artifacts that may affect our process and planning throughout the lifespan of this project.

3.1 Deliverables

The deliverables for Emerge are as follows:

Date	Title	Comments
Feb 21, 2017	Project Abstract and Diagram	
Mar 5, 2017	Project Progress Check 1	
Mar 19, 2017	Project Progress Check 2	
Apr 10	Project Midterm Evaluation	Needs: 1. Documentation Draft, 2. Code and ReadMe 3. Individual Contribution

3.2 Dependencies

Emerge is a swift iOS mobile application using Firebase as its backend service. Therefore, this application depends primarily on both developers and users having access to iOS devices, including laptops, cell phones, and tablets. In order to develop using iOS, developers must have access to both an apple computer and iOS device. On the apple computer, the user must have both xcode and cocoapods installed. Lately, this project depends heavily on the accessibility of Firebase servers to connect and respond to changes in the alert system. Some non-functional requirement dependencies include usability, performance, and accessibility, as emergency broadcasts may need to occur at any moment.

3.3 Concerns

The concerns within the Emerge application are:

- Firebase compatibility with Swift 3; This may require extra time spent by the Emerge team to learn and memorize new syntax
- Internet crawl for emergency data; Data must include type of emergency, location, danger range, and date

Chapter 4: Project Architecture

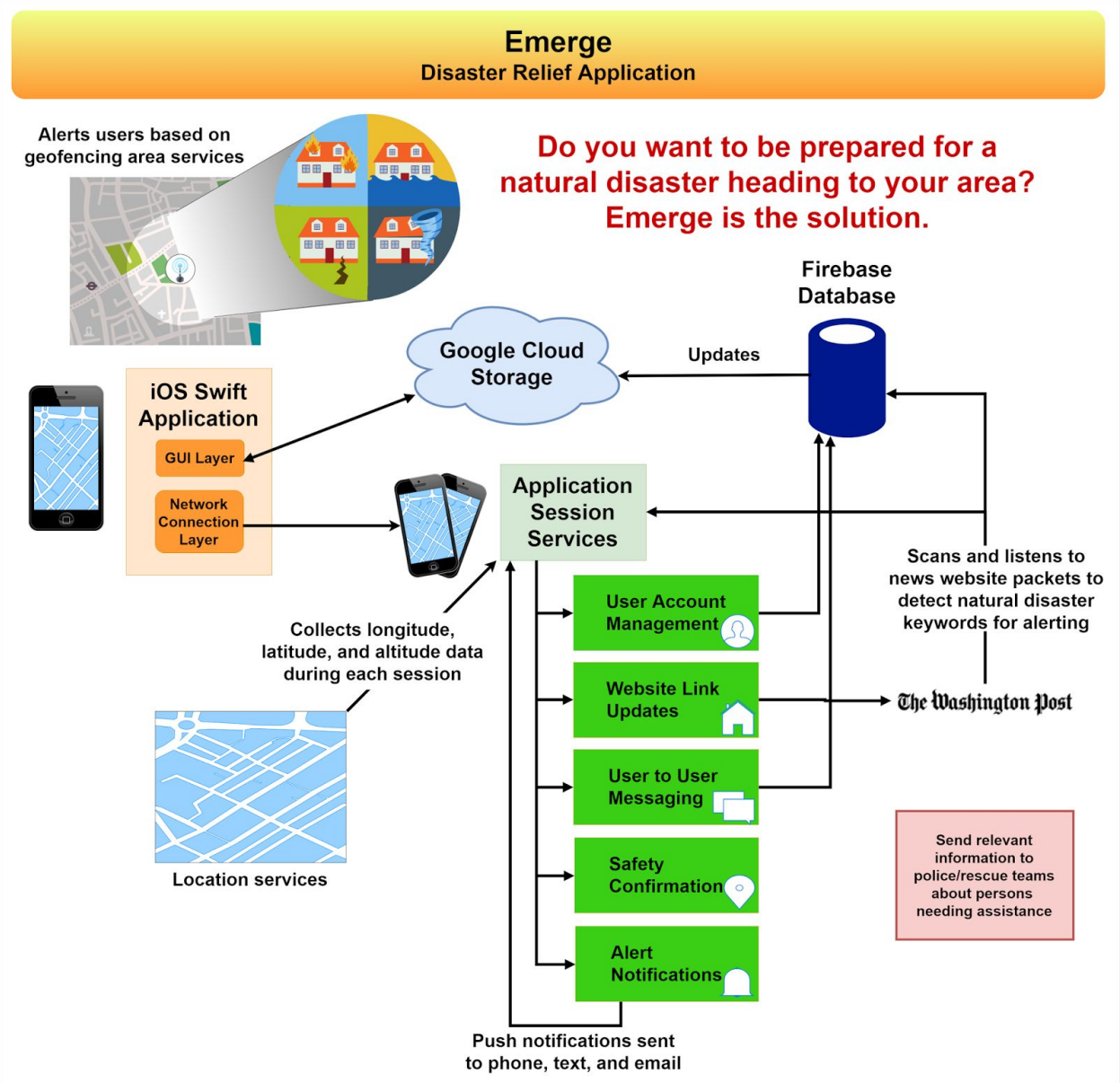


Figure 1: Architecture Diagram

Chapter 5: Design Specification

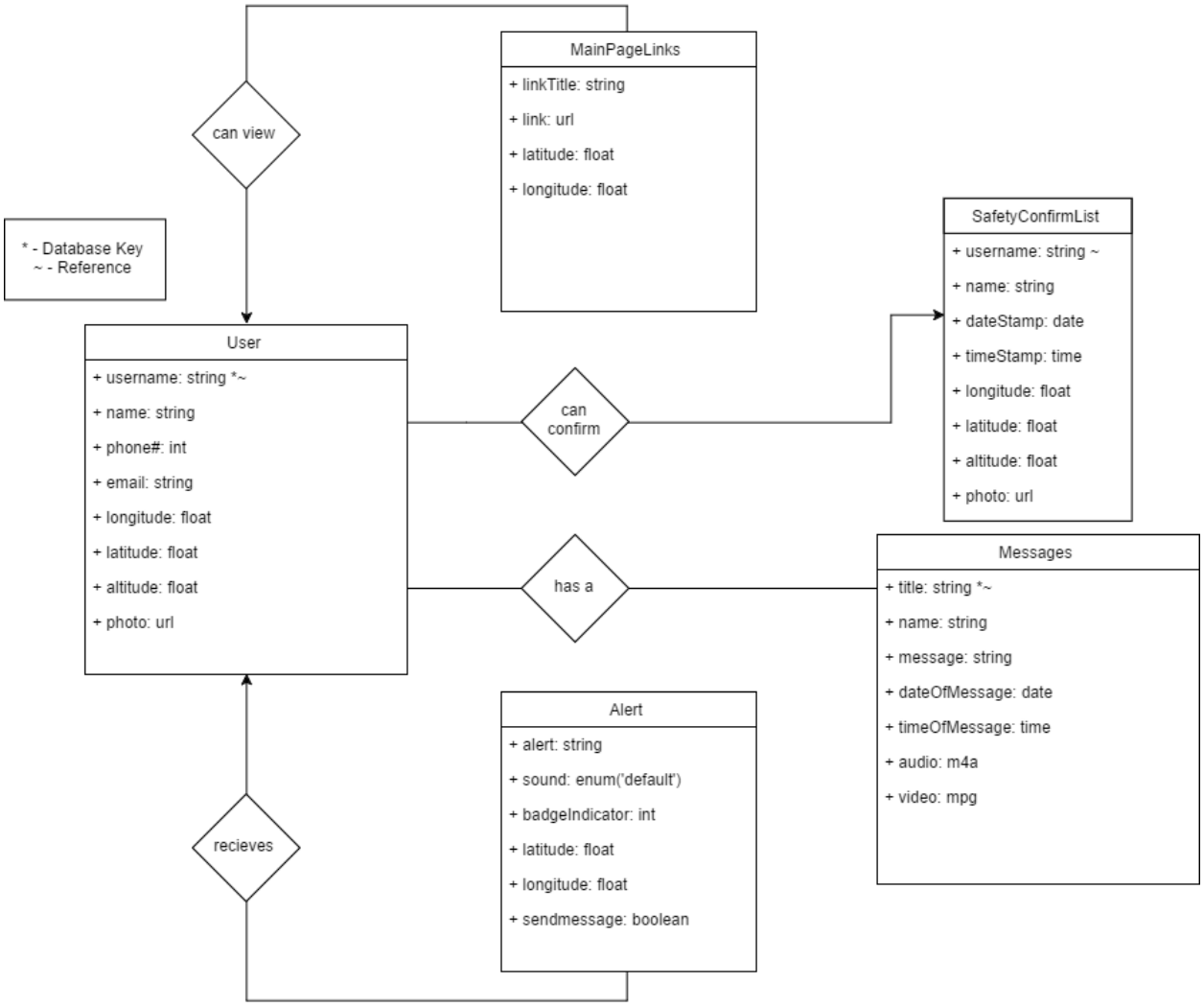


Figure 2: Database Relational Tables

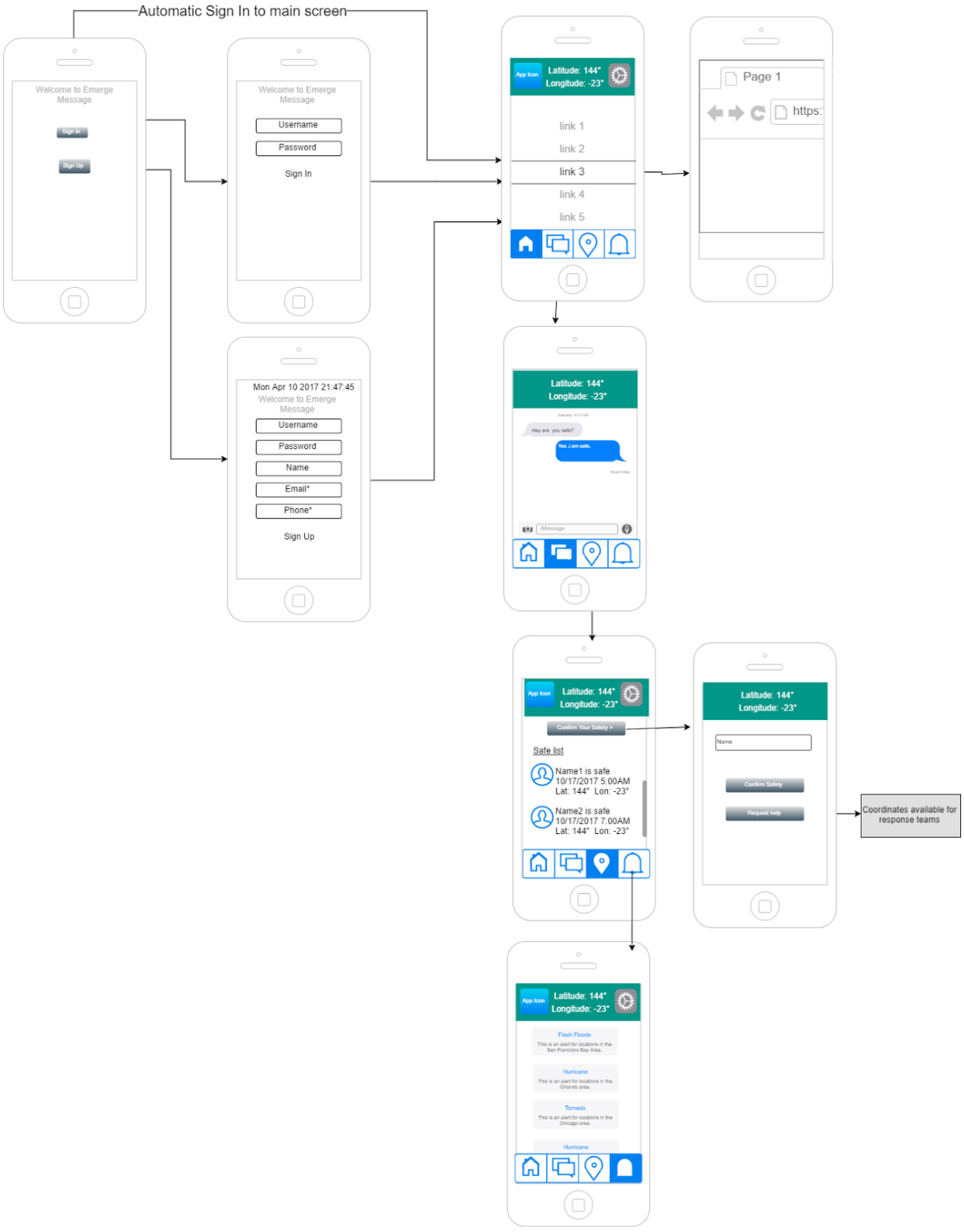


Figure 3: User Interface MockUp

Chapter 6: Implementation

6.1 Implementation SetUp Tutorial

Setup

For this application, we are implementing it for the iOS platform using Google Firebase as our backend. Contained in this section is the set-up regarding how to start this project on your own.

Initial Requirements: Apple Computer, iOS device, xCode, (included in setup) CocoaPods

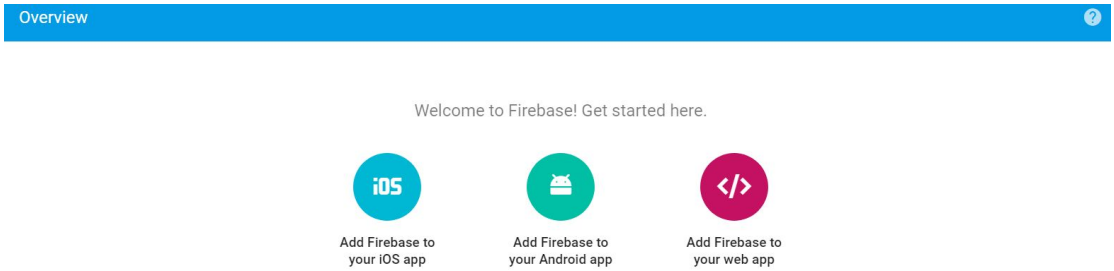
I. Install CocoaPods (Skip if already installed and updated)
Open "terminal" from finder > applications
Type **"sudo gem install cocoapods"** into your terminal, run, and let install.

II. Once CocoaPods is installed, open up xcode and create a new **swift** iOS project. In our case, we named our project "Emerge Messaging".

III. Once your project is created, go back into the terminal and write the following.
cd *your-project-directory* (navigate to your project folder)
pod init (create a pod file;
Pod file = link file to external library)
Open your newly create podfile and write:
pod 'Firebase/Core' (connects the firebase core library to app)
Back in your terminal, write:
pod install

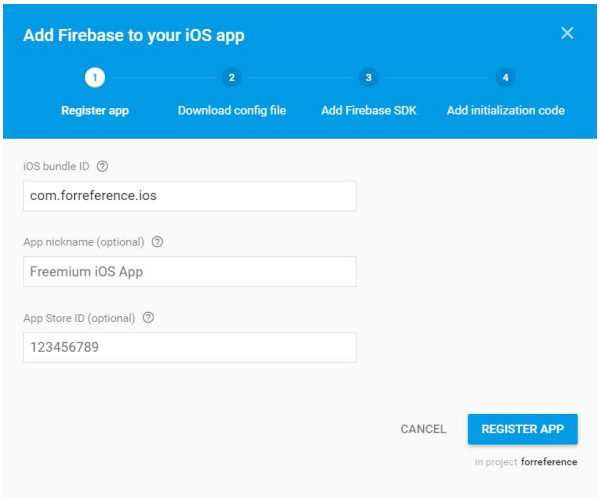
IV. After pod install completes, your project folder and files will be altered to connect to the podfile and firebase API. Close out of your xcode project, navigate to your project folder in the finder, and open the new .xcworkspace file. This is your new project file. (Note: Do NOT use your old xcode project file, as it will not be connected to the podfile and will not update your project.)

V. Now that cocoapods have been installed into your application, navigate over to
firebase.google.com
Sign up for a firebase account and in the top right corner, click "Go To Console" (if not in console already). Once inside console, you will see three options in the overview page:

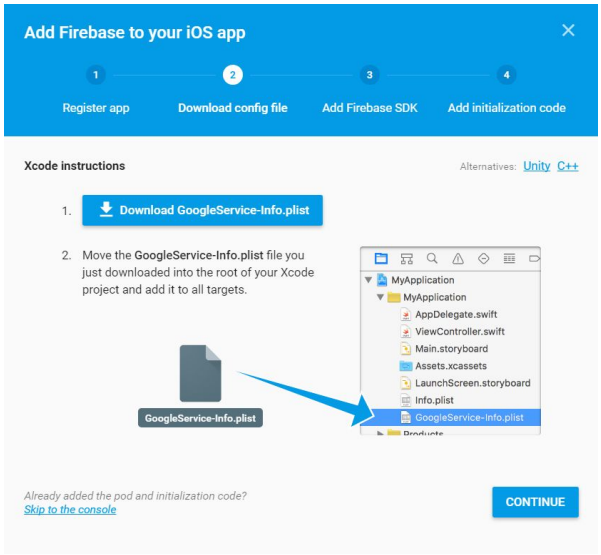


Select "Add Firebase to your iOS app and follow the four substeps below:

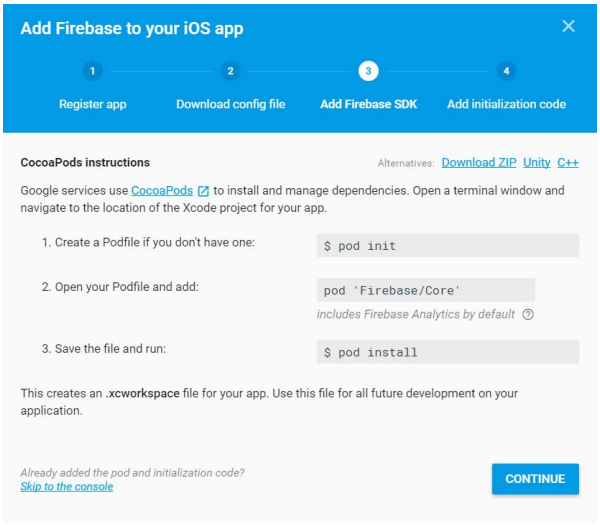
1. Input your iOS bundle ID
(leave the optional fields blank if you have no values for them. Your iOS bundleID can be made up as long as it follows the convention com.name.ios)



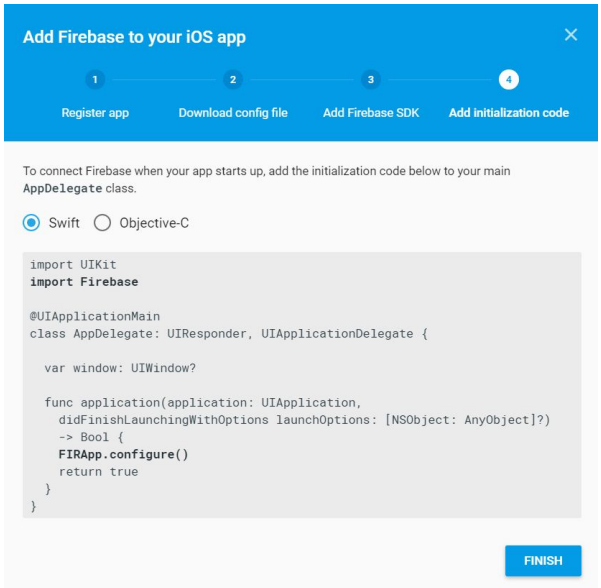
2. Download the GoogleService-Info.plist file and drag and drop the file into the directory in xcode.
(Make sure to move this into the directory inside of xcode. This ensures that the file imports correctly into the xcode project).



3.We have completed this substep above in step III. If you haven't created your podfile yet, please do so now.



4. Lastly, open your AppDelegate.swift class and input the two bolded lines into the respective areas.



At this point, you are completed with setup. Your project is now created and connected with the backend Firebase services.

6.2 Implementation Walkthrough Tutorial

Create View Tutorial

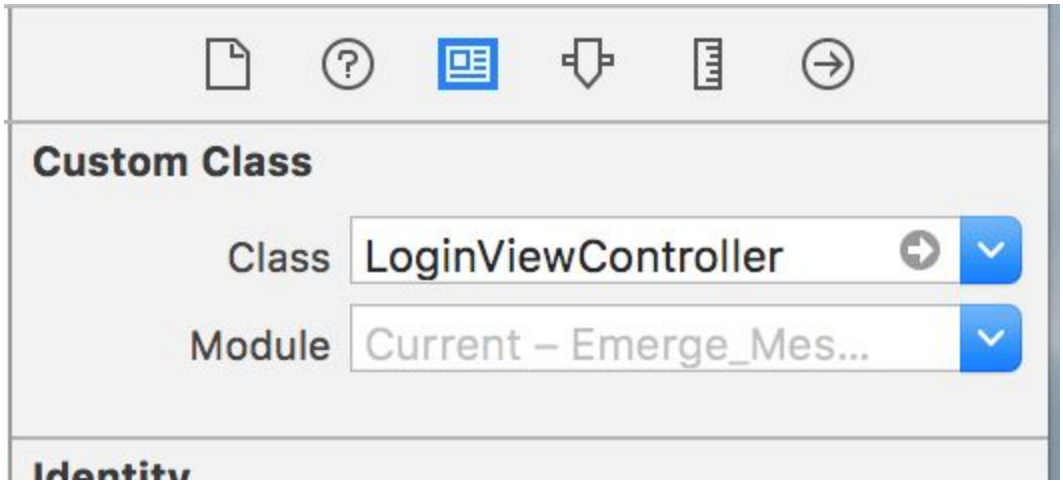
1. Create a controller view for each user interface

Initial Requirements: Swift iOS application created, cocoapods installed

1. Inside of the workspace file, click and drag a new **view controller** from the bottom right container of the main storyboard screen. Drag this new view controller onto the main story board.
2. Now create a new swift file with the corresponding view controller's name and save it.
3. Next, copy the code from the ViewController.swift into the newly created swift class and rename the class header to the view controller's name.

Ex. Click and drag from bottom right to create a view controller for login. Create a swift class called LoginViewController. Copy and paste the code over and change the line `class ViewController: UIViewController` to `class LoginViewController: UIViewController`

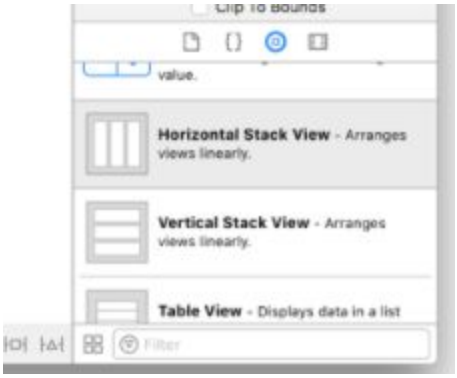
4. Once completed, the user must link the new controller in the storyboard. Select the storyboard view controller and in the right menu select the identity tab. In the drop down box for "class", select the corresponding class. In this example, this would be "LoginViewController", as seen below.



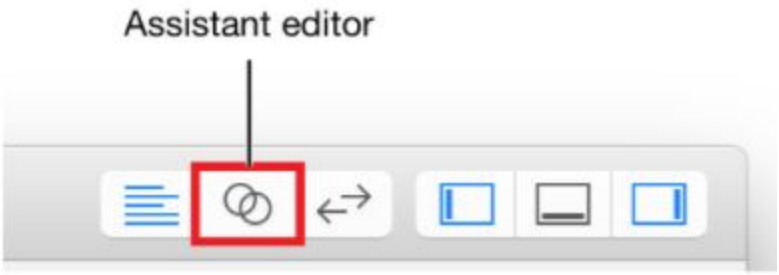
5. Repeat these steps to create new view controllers for every class. You now have successfully created new controllers and linked them to the storyboard.

Create Element Tutorial

1. Open the storyboard.
2. From the bottom right container, click and drag element to view in the storyboard.



3. Once on the storyboard, control click and drag from the element to the parent view or another element to add the constraints. These constraints include: center horizontally to element, center vertically to element, vertical and horizontal spacing, constant width or height setting, and aspect ratio.
4. With your element properly positioned on the screen, you can now connect your element to code. Open the assistant editor.



5. With the correct swift class open, control drag from the element to the code. Now you can select whether you want your element to be an outlet or an action. Outlets let you alter the element or retrieve attributes of the element, while actions let you perform functions based on user input.

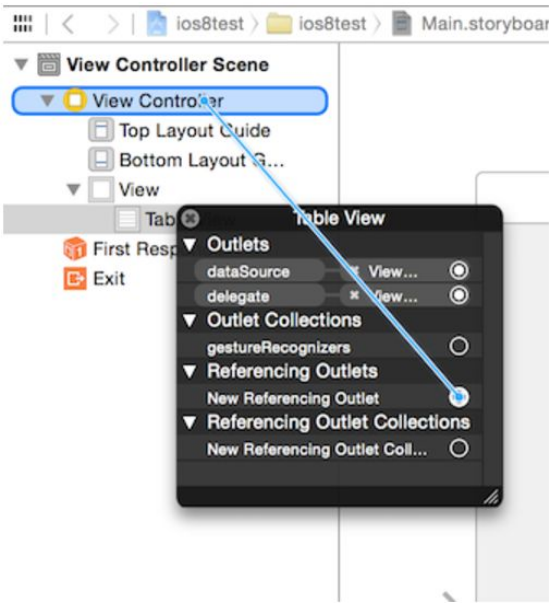
Table View Tutorial

For table views, we used two tutorials:

https://developer.apple.com/library/content/referencelibrary/GettingStarted/DevelopiOSAppsSwift/CreateATableView.html
https://www.weheartswift.com/how-to-make-a-simple-table-view-with-ios-8-and-swift/

As an overview:

1. Drag a tableview element to view controller screen in the storyboard. (See Create Element Tutorial).
2. Control drag from the tableview to the view controller. Add the tableview as both a datasource and delegate.



3. Create a new file for the cell using the following: -> CocoaTouchClass with the type TableViewCell and the name corresponding to the type of cell. Ex: EventTableViewCell.swift

4. Connect the table view cell class to the table view cell on the storyboard (See step 4 in create view tutorial).

5. With the attribute editor open and the table view cell still selected, input a cell “identifier” name for the tableviewcell.

6. In your main class using the table view, input the following code.

```
class ViewController: UIViewController, UITableViewDelegate, UITableViewDataSource {

    ...

}
```

```
class ViewController: UIViewController, UITableViewDelegate, UITableViewDataSource {

    ...

    func tableView(tableView: UITableView, numberOfRowsInSection section: Int) -> Int {
        return 0
    }

    func tableView(tableView: UITableView, cellForRowAtIndexPath indexPath: NSIndexPath) -> UITableViewCell {
        return UITableViewCell()
    }

    func tableView(tableView: UITableView, didSelectRowAtIndexPath indexPath: NSIndexPath) {

    }

}
```

7. Add elements and connect elements to your cell (See CreateElementTutorial).
8. Connect the Table View to the main view controller. (See CreateElementTutorial).
9. Input code to connect the cell to the tableView(... cellForRowAtIndexPath …) function.

```
        var cell:UITableViewCell = self.tableView.dequeueReusableCellWithIdentifier("cell") as UITableViewCel
l


        cell.textLabel?.text = self.items[indexPath.row]

        return cell
```

10. Load information into the main class to populate the cells.

(See tutorial links for more in depth instructions)

Messaging Tutorial

Implementing the messaging feature, we followed a tutorial we found online. We interpreted this tutorial and the table view tutorials to create our messaging portion of our app.

https://www.raywenderlich.com/140836/firebase-tutorial-real-time-chat-2

Firebase Tutorials

Authentication Tutorial: https://firebase.google.com/docs/auth/ios/start
Database tutorial: https://firebase.google.com/docs/database/ios/start

These tutorials include the functions and codes necessary to utilize authentication and the database for firebase.

GPS coordinates tutorial

We followed the tutorial below on implementing GPS coordinates to our application.
http://stackoverflow.com/questions/25296691/get-users-current-location-coordinates

Utilizing each of these tutorials, we constructed each class, view controller, element and backend listener to achieve user accounts, log in, sign up, web article lists, messaging, safety lists, and notification lists.

Tutorial Specific Screens

Initial home screen:
Implementing the home screen we imported Firebase to handle the user's login credentials and protect the user's information. We also connected a navigation controller so the program can switch between "SignIn" and "Sign Up" with a corresponding back button.

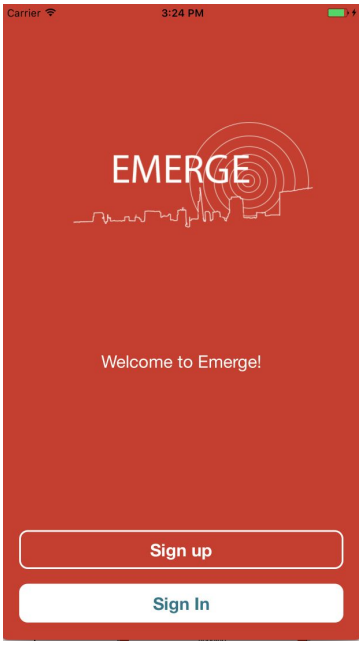
Sign In:
Sign in has 3 text fields and are connected to the code. There is also a button that is connected. Inside the code, we created a function called signInFinalize which gets the text checks if they are empty and returns an error. If it is correct Firebase checks and authenticates the user and sends them to the main screen. Once then we prepare the screen to navigate to the next screen.

Sign Up:
Sign up follows the same convention as sign in. Except has 6 text fields, for username, password, confirm password, name, email and phone number. Each were connected as outlets inside of the code. Here we gathered all the text fields checked if they are empty and error if they are. Once the user enters them correctly it checks if the password and confirm password match. If so the program continues otherwise it returns an error. As it continues, firebase takes over and sends it to the database and creates the user. Once finished the user is navigated to the main screen.

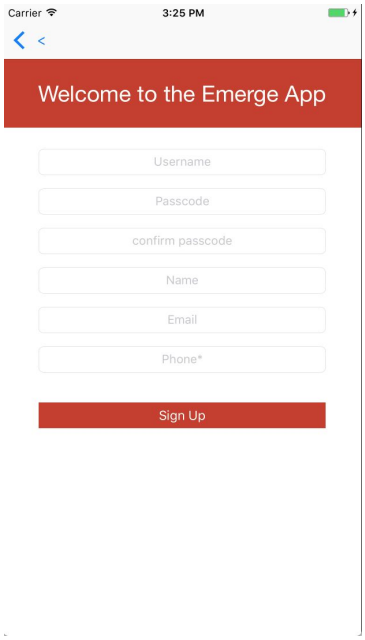
Main Screen:
The main screen contains a button for the home screen, the messaging app, location, and notifications screen. We also implemented a sub view. The subview displays the correct window when the proper button is pressed and updates the view. We also implemented getting latitude and longitude depending on the user's GPS coordinates by following a tutorial online. We had to import Corelocation library and enable NSLocationAlwaysUseUsageDescription inside the info plist file. Once the UI was complete we override the ViewDidload method to check if the user is in range of immediate danger. If so it will notify the user of the event. LocationManger gets the user latitude and longitude and sends it to Firebase to store and update as the user moves. For each button, there is method to navigate between each view depending on which was pressed, we also followed tutorials to implement this function.

Walkthrough For Each Screen

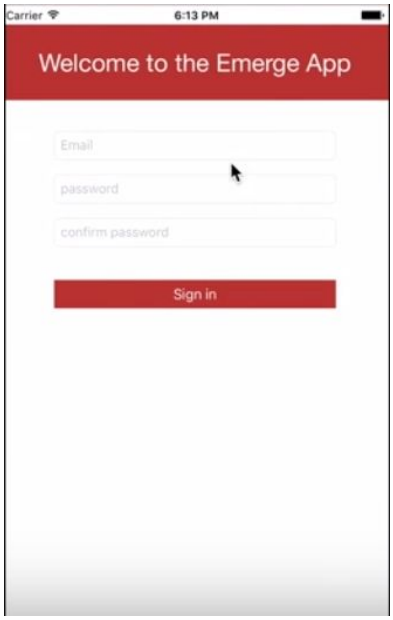
1. Upon launch, users will be taken to the initial screen to sign up or sign in to the application.



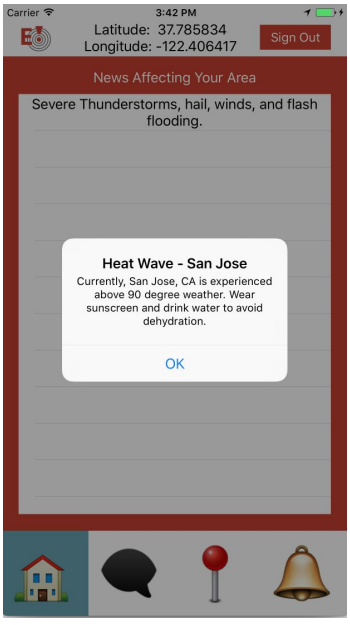
2. Clicking Sign Up, user will be prompted for username, password, confirm password, name, email, and phone number.



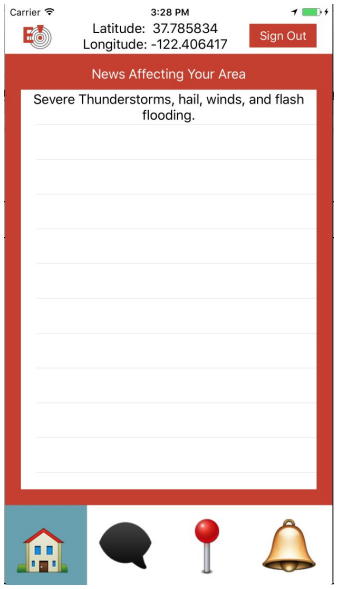
3. Clicking on sign in will prompt user for their email and password.



4. Once the user logs in or signs up, they will be directed to the main screen. Initially the application will display any notifications the user has missed.



After selecting ok, the home screen appears with a display of news for the current user's location.

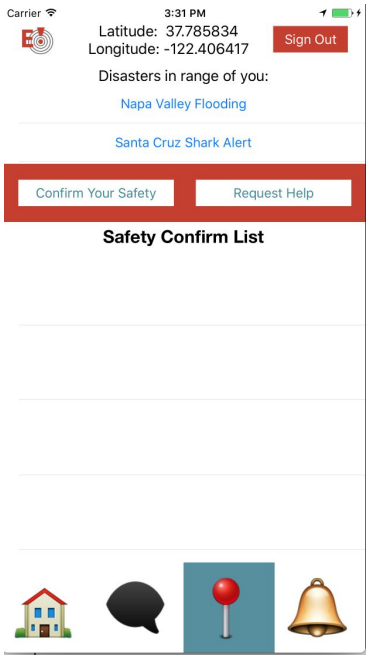


The user can select any table row in this view to open the corresponding link in the phone's internet browser.

Located at the bottom of the screen is the navigation to other functionalities within Emerge. The second option is for users to communicate through messages.

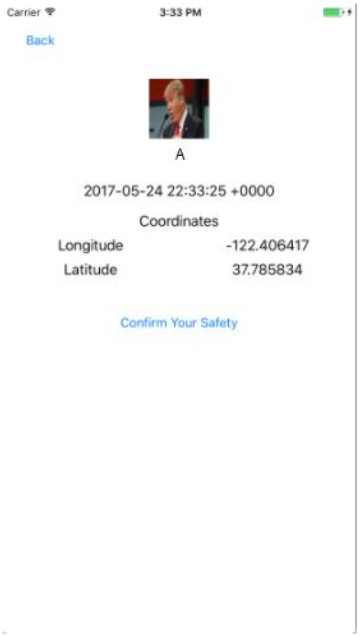
Selecting any of these messages will allow the user to view and reply to the specific message as well as delete the message conversation. Clicking on "new" in this screen will start a message conversation in which the user can input another user's email to start a message dialog.

The third option provides the safety list functionality.

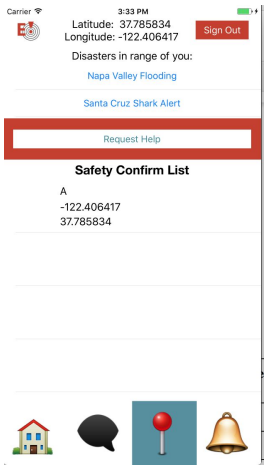


Users can select the confirm safety button to be taken to a confirm screen.

In this screen, the user's information is auto populated into the view. Therefore, the user only needs to click “confirm your safety”.

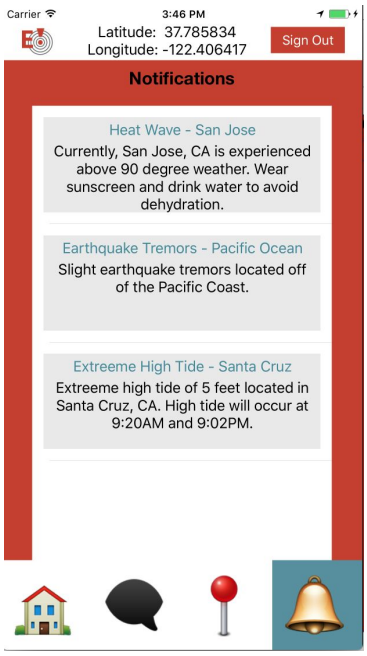


Upon returning the user can be seen updated into the list.



The user can also select request help at any time, to send their information directly to the database for forwarding to first responders. This will remove them from the confirmed safe list and give them an alert letting them know they requested help.

The last option provides the notification functionality that allows the user to view all previous notifications.



6.3 Learned and Created

Most importantly with this project we learned how to create a functional application that can be used to help others. Mobile development is still a learning process for this team and putting together this application helped strengthen our understanding over the mobile app life cycle. We have successfully created an emergency app that allows users to receive information about emergency situations that may be near, as well as lets users communicate with family and loved ones to let them know their safety, their requests for help, and collaborate their next steps in an emergency event. Even more so, this was an extra learning curve on how to utilize database normalization to achieve a faster and more responsive database queries. We created a database that follows BCNF formatting and keeps information within our data in clearly defined tables as seen in our database diagram. All together, this application provides real-time communication between host-to-host for messaging and server-to-host notifications.