GNG2101 Report: Deliverable F

Prototype II

Submitted by

[A03, Team 15]

[Jonathan Florus, 300178425]

[Laura Godfrey, 300082954]

[Mashal Joyaa, 300082650]

[Bonnie Lin, 300162118]

[Quinn Murnaghan, 300168522]

Nov 4, 2021

University of Ottawa

Abstract

This report highlights the details involved with creating the second prototype for the personal safety application. The team developed and tested the different subsystems of the personal safety application to create a mostly functional prototype II. The prototype was then presented to the client in a client meeting. The client provided feedback for the prototype which was then used to create the next steps needed. These next steps are demonstrated through alterations and advancements made to the prototype presented to the client. These alterations and advancements are tested in order to meet the target specifications outlined in Deliverable B.

Table of Contents

List of Figures

List of Tables

List of Acronyms

Acronym	Definition
API	Application Programming Interface
App	Application
AVD	Android Virtual Device
iOS	iPhone Operating System
SMS	Short Message Service
UI	User interface

1 Introduction

The client requires a personal safety application in order to notify their clients in the case of an emergency. The team has gone through the steps preceding the prototype testing phase in the iterative design process. The team has taken note of the client's needs, created conceptual designs, and created prototypes subjected to prototype testing and client feedback. For the second client meeting, the team clarified some of the client needs as well as presenting a basic design of the homepage to the client. The team then used the feedback to improve the design. The team also began to build the main subsystems of the application, such as the emergency messaging system, the turn-off function, the activity feature, the notification system, and the check-in function. For the third client meeting, the team showed most of the different subsystems in order to receive additional feedback. The team continued with prototype testing and compared the results with the target specification.

2 Third Client Meet

2.1 Client Feedback

The team presented the client with the updated prototype designs. This included the subsystems that were completed as well as ideas and concepts for the subsystems that were not completed at the time of the meeting. The homepage UI subsystem that was shown to the client was an updated prototype from prototype I. The client reacted positively to the design. She liked the light against dark colour scheme that replaced the original red, green, and blue colours that were used in the first prototype. She again emphasized her content with the drop down menu having all the features she wanted access to. The team also presented the emergency text subsystem to the client and how it would work. The client was fine with either SMS emergency texts or in app messaging emergency texts. Furthermore, the page for the turn-off subsystem was shown to the client and she reacted positively to it. She liked that there was a confirmation pop-up message for both turning off and on the check-in feature. The turning off confirmation message has a text input field in order for the user to give written confirmation to turn off the check-in feature. This was something the client had asked for and she liked that the team was able to implement it. However, the team did discuss with the client that there was not much literature on getting the written confirmation to actually work. She suggested instead a verbal confirmation, but overall said that a written confirmation would make a big difference; however, if it is not possible then it is not possible. The hiking subsystem was presented to the client and she said it looked great and had no other feedback. The notification subsystem was the final subsystem presented. The notification subsystem had not been completed at the time of the meeting. However, the team member in charge of this subsystem had a screenshot of the type of notification they wanted to do and presented that to the client. After the client was asked what

they would like included in the notifications, she suggested that it be displayed that there is a check-in coming up, how long until the check-in, and at what time the check-in would be (for instance, “Upcoming Check-In in 15 minutes at 8:30am”).

2.2 Updated Design

Due to the client greatly liking how the subsystem prototypes looked so far, the team decided to not change the majority of the subsystem designs. The team decided instead to try to fix parts of the subsystems that would allow them to function better and improve on the current designs without making major changes. The notification system proves to be a complicated challenge due to it needing to work outside of the app. However, the early stages of this subsystem were completed after the client meeting. The information in the notifications don’t reflect currently what the client suggested, however, this subsystem is able to send a notification.

3 Prototypes

3.1 Emergency Texting System

3.1.1 Prototype: SMS Messenger

This prototype was a simple sms messenger with a U.I to ensure all parts were functioning correctly. The first box is where the phone number of the desired recipient is inputted and the second is where the message is put. Once the user presses the send message button the phone’s systems request permission to access SMS communications on the device. Due to this working via SMS, any phone that is connected to a telephone carrier will be able to receive messages from this application.

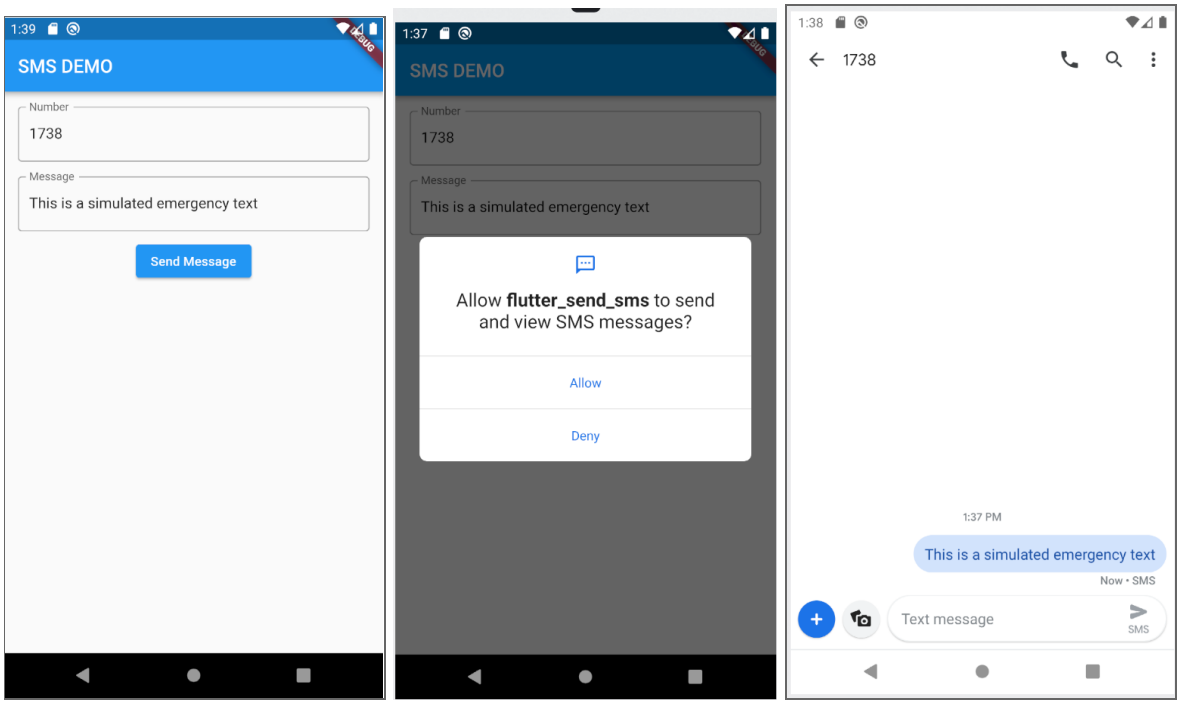


***Figure 3.1.1**: Messaging System*

Test Objective	**Target Specification to be compared with**	**How the Prototype attempts to meet the specification**	**Results**
Ability to request SMS permissions from the operating systems	None	Android specific code was written to access the operating system's SMS permissions	When the send button is pressed, the permissions are successfully requested
Can take send message to phone numbers	Default operating system messaging app	Sends a message via SMS to any inputted phone number	Successfully sends messages to inputted phone numbers

***Table 3.1.1**: SMS Messenger Test Objectives*

3.1.2 Prototype: User Input Bypass

The goal of this test was to create a bypass that would allow for a delayed message to be sent. This was done due to apple devices not having the ability to create a message without user input. As a result, a solution was creating a delayed message that could be canceled. This prototype simply checked to see if the concept of a delayed message was possible by creating a 180 second timer. After 180 seconds the message was then sent successfully.

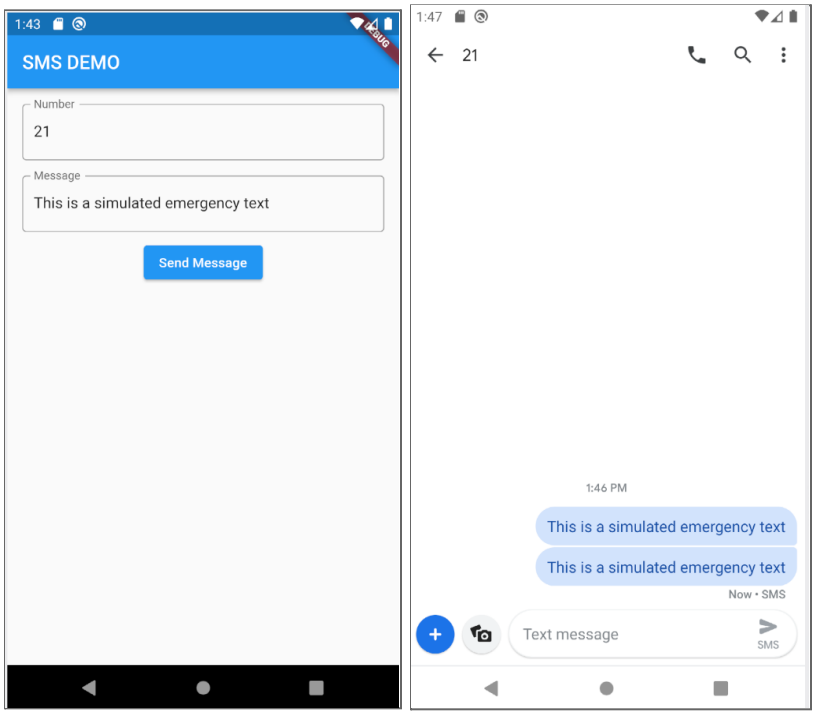


***Figure 3.1.2**: User Input Bypass Test*

***Table 3.1.2**: User Input Bypass Test Objectives*

Test Objective	**How the Prototype attempts to meet the specification**	**Results**
Ability to produce a delayed SMS message	A timer was coded so that as it ran the SMS code would be unable to run. Once the timer runs out the code for the SMS message is run.	The delayed SMS was sent properly
Ability to work on IOS	The goal was to transfer the libraries functions onto iOS	Unsuccessful, it appeared that the library has cross platform support. Unfortunately, if it did it no longer supports the current version of IOS

3.1.3 Google Services

After a number of unsuccessful attempts to use the code on the IOS platform an alternative solution was needed. The end goal is to create a messaging system that works via wifi using google services. This prototype does not include many U.I elements since the majority of

the code consists of backend development. This prototype tested the ability for a user to sign into an application with their google account. After signing in, they are brought to the home page where they have access to all the functionality of the application. On the top right, there is an icon which will be used for signing out the user. When signed out they are brought to a screen so that they can log in with a different email address.

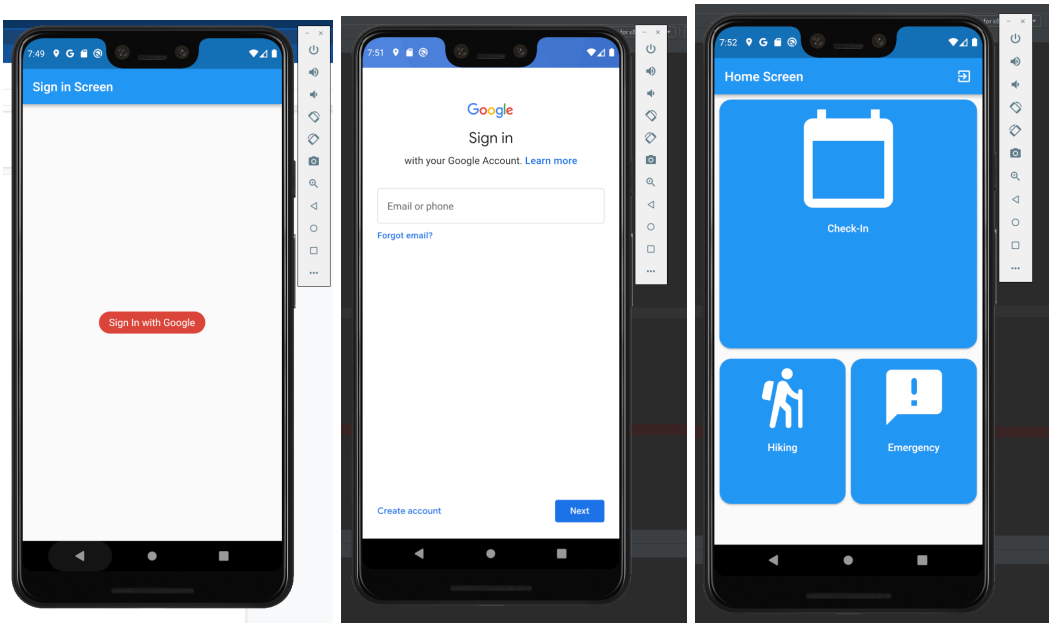


***Figure 3.1.3**: Google Services*

Test Objective	**How the Prototype attempts to meet the specification**	**Results**
Ability to sign into Google	Using a combination of the Firebase API, Google Services API, and flutter repositories a sign in system was built.	One is able to successfully sign in with google using their preferred email address.
Ability to Sign out of Google	Using the same libraries to create the sign in feature, a signout function was also created.	One can sign out using the signout button on the homepage
Opens homepage once signed in	Upon signing in the sign in page is popped out of the U.I stack and replaced with the homepage window.	Once signed in the application automatically opens up the homepage.

***Table 3.1.3**: Google Services Test Objectives*

3.2 Turn Off Function

The turn off function went through many tests to ensure it works the best it can as of right now. The app is planning to start with the Check-In Feature being turned off. The Turn Off Check-In Feature page with the feature turned off can be seen in Figure 3.3.1.

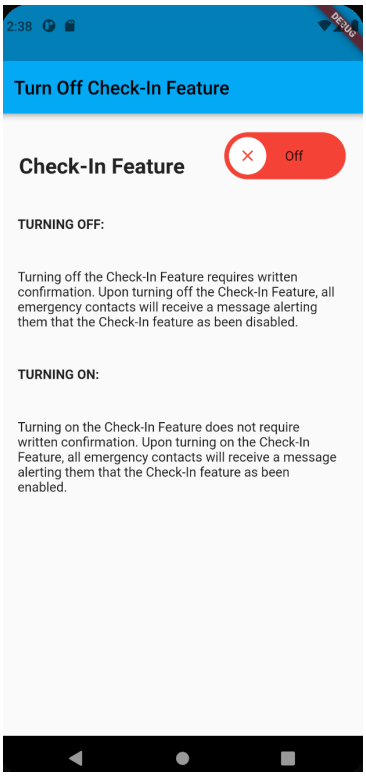


Figure 3.2.1: *General Home Page when the Check-In Feature is Turned Off*

This means that each time the Check-In Feature is turned on, it will ask what time the Check-In time should be at. This part of the subsystem prototype has not been completed yet and will be completed in the future. Currently, there is a confirmation pop-up message which asks the user to confirm that they'd like to turn on the Check-In Feature. This pop-up message can be seen in Figure 3.3.2. Currently, the confirmation and cancel buttons do not actually work, they require more time to function properly.

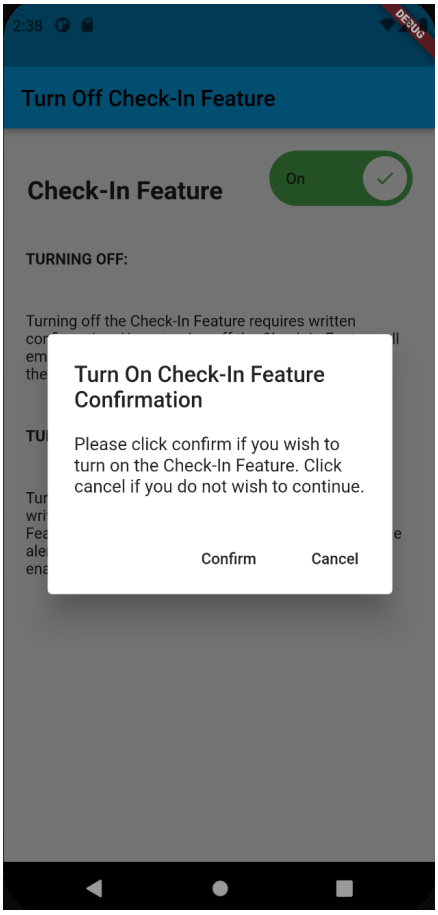


Figure 3.2.2: *Confirmation Pop-Up When the User Attempts to Turn the Check-Ins Back On*

Once the Check-In Feature is turned on it remains on indefinitely until the user turns it off. When the Check-In Feature is turned on, a message will be sent to the emergency contacts alerting them that the Check-In Feature is turned on and to be aware that they may receive emergency texts from the user if they do not check-in on time etc. The emergency messaging for the subsystem has not been implemented yet. How the page will look when the Check-In Feature is turned on can be seen in Figure 3.3.3. This page is identical to the page when the Check-In Feature is turned off in Figure 3.3.1, with the exception of the switch button being different.

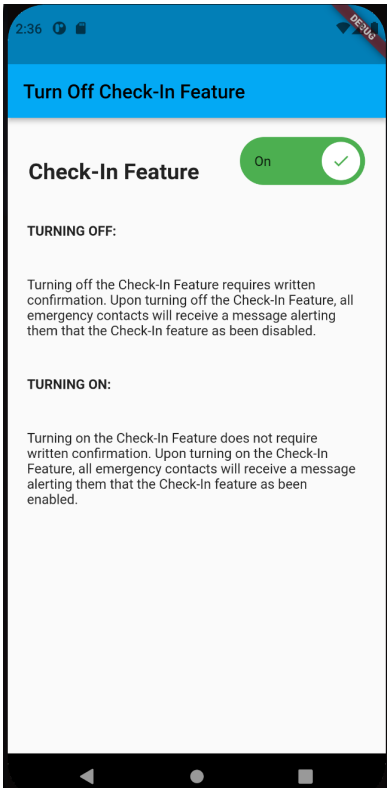


Figure 3.2.3: *General Page of the Turn Off Function*

If the user chooses to turn off the Check-In Feature for whatever reason after the Check-In Feature has been turned on, they will receive another confirmation pop up message. This confirmation message will ask for a written confirmation from the user in order to avoid missing potential health episodes that could push a button but not write a word properly. This message can be seen in Figure 3.3.4. Once the Check-In Feature has been turned off, it can remain off for any length of time before being turned back on. When the Check-In Feature is turned off, the emergency contacts will be sent a message alerting them that the user has turned off the Check-In Feature and that they do not need to be prepared to receive an emergency message if the user does not check-in etc. The emergency messaging for the subsystem has not been implemented yet. Currently, the written confirmation text field, the confirmation button, and cancel button do not actually work, they require more time to function properly.

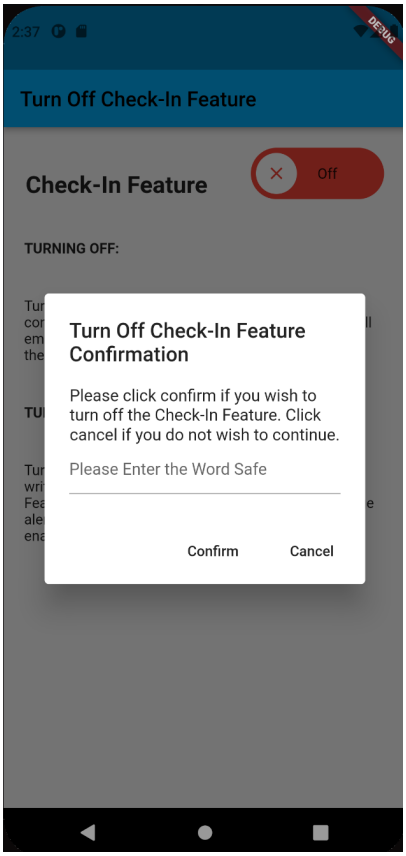


Figure 3.2.4: *Confirmation Pop-Up When the User Attempts to Turn Off the Check-In Feature*

A feature that is quite important to implement but is often forgotten until later in the process is to save the state of the app. This means that if the Check-In Feature is turned on and the user exits the app or closes the app or turns off their phone, when they reopen the app, the Check-In Feature will still be turned on. This applies for the Check-In Feature being turned off as well. The feature of saving the state of the app was implemented into this prototype and was tested multiple times by closing the Android Virtual Device (AVD) application multiple times in different states. Each time when the AVD was reopened, the switch button remained in the same state as it was when the AVD was closed. There is no way to prove this worked through screenshots, however, we were able to obtain a screenshot of the AVD message saying it was saving the state. This screenshot can be seen in Figure 3.3.5.

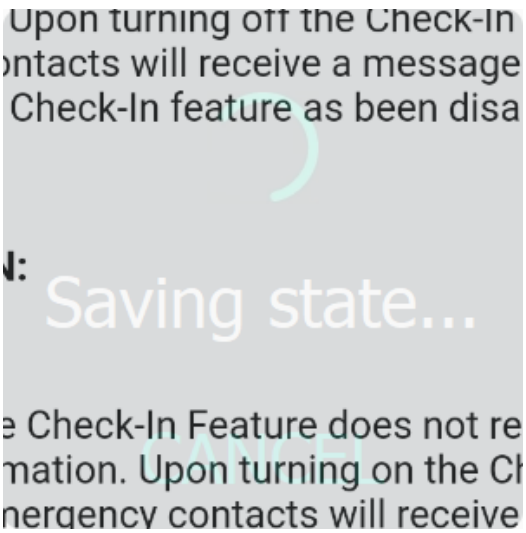


Figure 3.2.5: *Saving the State of the Switch Button*

Test Objective	**Target Specification to be compared with**	**How the Prototype attempts to meet the specification**	**Results**
Test the application to ensure it saves its state after user closes application	Application has turn-off feature	Included code to have the application save its state when the user closes the application	Application was able to save state
Test the switch button to make sure it can be in the off and on position	Application has turn-off feature	Included code that made sure the button would switch between on and off and would indicate if it was on or off based on the colours and text of the switch button	Switch button was able to turn off and on and user can tell if the switch button is on or off based on the colours and text of the button
Test the switch button to make sure a confirmation pop-up message appears when turning off and on	Application has turn-off feature	• Included code that when the switch button was pressed to turn on, a confirmation pop-up message appears asking the user to confirm or	• Switch button turning on enabled a pop-up message to appear asking the user to confirm or cancel turning on the Check-In

		cancel turning on the Check-In Feature. ● Included code that when the switch button was pressed to turn off, a written confirmation pop-up message appears asking the user to input the word "Safe" and then hit the confirm button to turn off the Check-In Feature or hit cancel to exit the pop-up message.	Feature ● Switch button turning off enabled a written confirmation pop-up message asking the user to write the word "Safe" in the text input box. The user has the option to hit confirm or cancel to proceed.
Test the written confirmation input text field	Application has turn-off feature	There is currently no code implemented to make the written confirmation input text field work properly	The subsystem fails this test
Test the confirm and cancel buttons in both pop-up messages	Application has turn-off feature	There is currently no code implemented to make the confirm or cancel buttons override the switch button turning off or on	The subsystem fails this test
Test the emergency message being sent to the emergency contacts when the switch button is turned off or on	Application has turn-off feature	There is currently no code implemented to make the switch button turning off or on trigger a message being sent to the emergency contacts	The subsystem fails this test
Test the Check-In time setting feature when the switch button is switched to on	Application has turn-off feature	There is currently no code implemented to make a pop-up message appear to set a time for	The subsystem fails this test

		the Check-In Feature when the switch button turns on	
Test if its simple to use	Application is user friendly	There is a limited number of buttons and steps to use the feature properly. We had a person from outside the team use the turn off feature and they said it was user friendly.	The subsystem is user friendly
Test if the the Check-In Feature can be turned off or on for extended period of time	Application has turn-off feature	There is no code in place to limit the amount of time the app can be in each state. Therefore, it can be in either state indefinitely.	The subsystem passes this test

Table 3.2.1*: Turn-Off Function Test Objectives*

3.4 Hiking Feature

The hiking feature is intended for the user to be able to set an estimated time of completion when they are doing any activity, such as hiking. The user may type in their location before beginning the activity; thus, in the case of an emergency, the application will use the written location and include it in the emergency text that will be sent out to the user's contacts. The user can pause or reset the timer. Once the timer runs out and the user has not stopped the timer manually, the application will ask the user if they are in need of assistance. If the user does not respond, the application will let the user know that it will be sending out a message to their contacts. Figure 3.4.1 displays the overall design of the subsystem with the location input at the top, the timer in the middle, and the pause, stop, and reset buttons at the button. Figure 3.4.2 displays how the user will be able to modify their estimated time of completion for the activity, as well as a preview of the code written to run it.

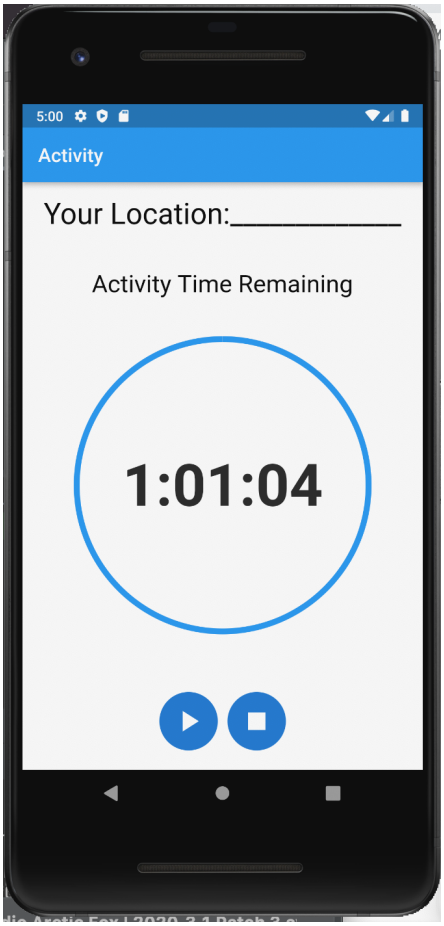


***Figure 3.4.1**: Overall Display of the Hiking System*

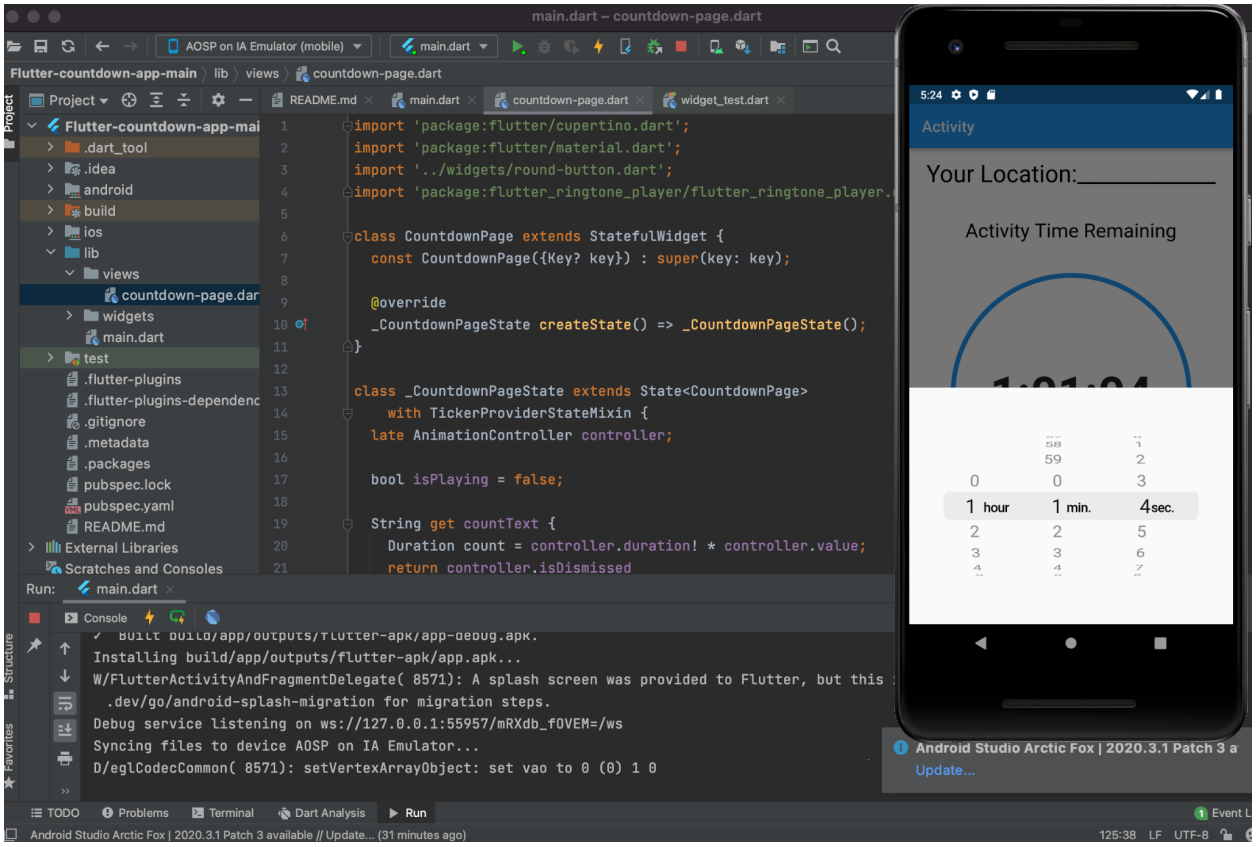


***Figure 3.4.2**: Customizable Timer of the Hiking System and Corresponding Code*

For this prototype, the test objectives focused on ensuring that the timer worked, as well as figuring out how users respond to the overall design. In further prototypes, the team will begin implementing and testing features such as allowing the user to input their location, contacting the written location to the emergency text system, and enabling a check-in if the user does not finish the activity within the allotted time frame.

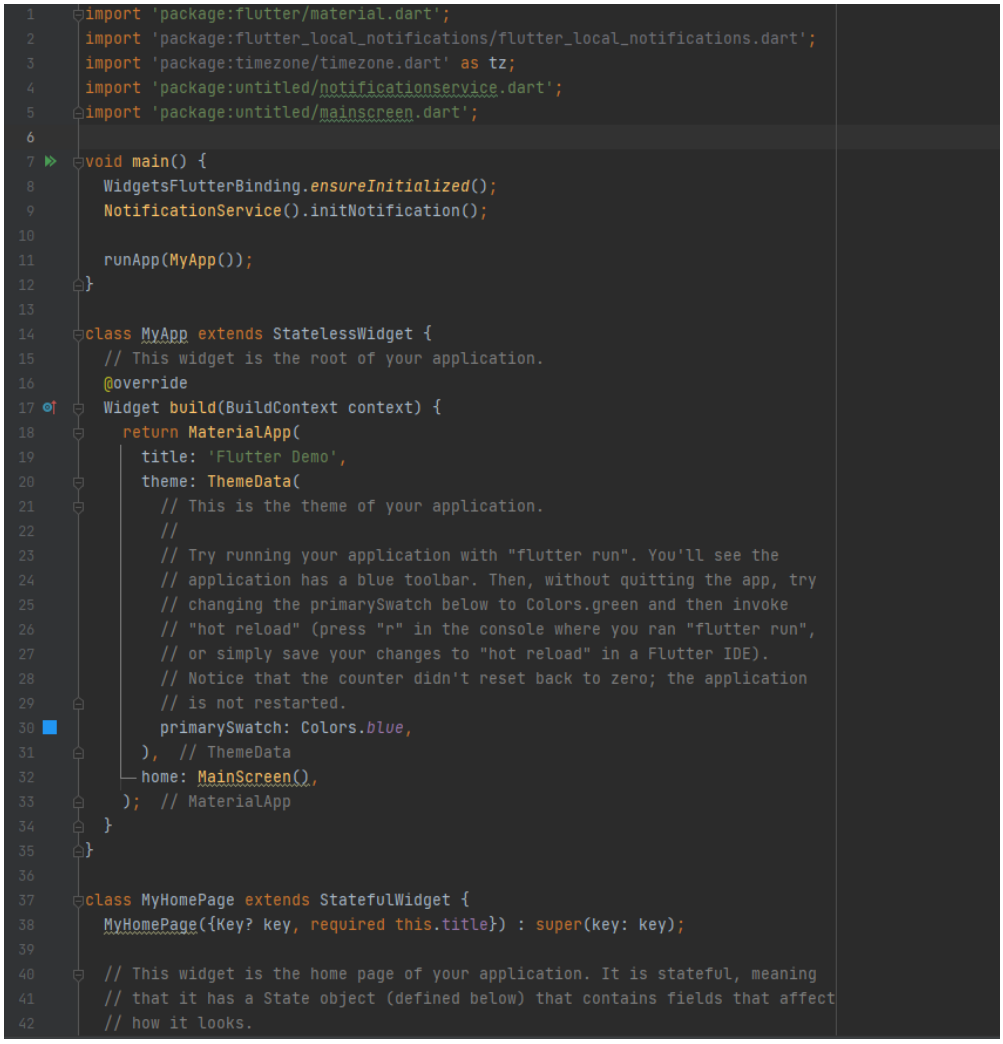
Test objective	**Target specification to be compared with**	**How the prototype attempts to meet the specification**	**Results**
Test if the timer can countdown	The application has a timed activity feature	The prototype included code to display a timer within the hiking feature	The timer was able to count down properly. The application also displayed a circle around the timer which would disappear

			corresponding with how much time is left.
Test if the timer can be paused and resumed	The application has a timed activity feature	The prototype included code to display a start button that changes to a pause button when the timer has started	When the start button had been activated, it was able to begin the countdown and turn into the pause button. The pause button was able to pause the timer when activated.
Test if the timer can be stopped and reset	The application has a timed activity feature	The prototype included code to display a stop button that will stop the timer	The stop button resets the timer upon being activated.
Test if the layout is easy to understand	The application is user-friendly	The subsystem has minimal features and is similar to many other timers to ensure the user can recognize the different features	Feedback was given by an individual sought out by the team. The individual deemed the system as easy to use and minimalistic. However, they did not expect the stop button to reset the timer. The stop button was assumed by the user to pause the timer rather than reset it.
Test if the time on the timer is customizable	The application has a timed activity feature	The team implemented a code in order for the timer to be customizable for the user to input their desired activity time	The timer was able to be changed by tapping on the time display. The application was able to open a display area where the user can scroll through to input the number of hours, minutes, and seconds in the timer, which was then able to be displayed on the screen.

Table 3.4.1*: Hiking System Test Objectives*

3.5 Notification System

The notification system is still being developed. Currently, push notifications are able to be emulated for the application, with the framework for timed interval notifications (the check-in reminders for the functionality of the app) laid out for development. The following are previews of section of the code written to implement the notification system:

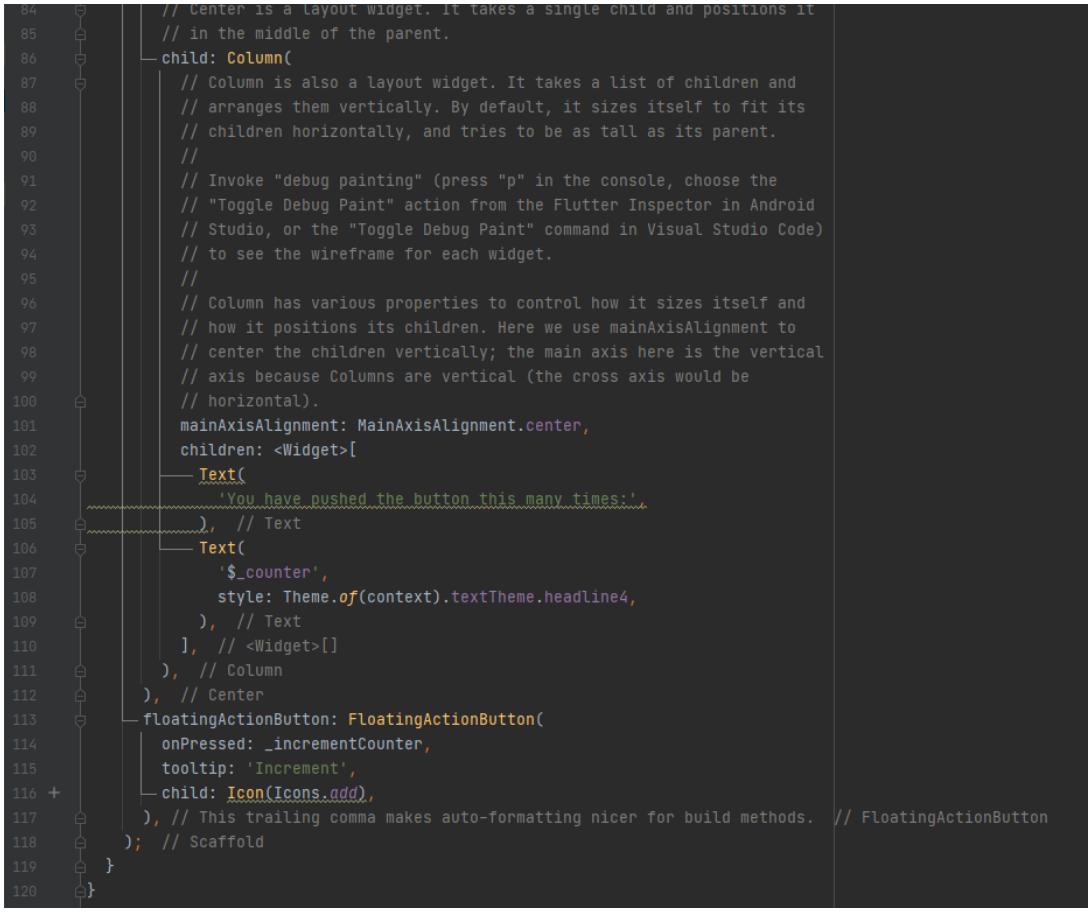


```dart
import 'package:flutter/material.dart';
import 'package:flutter_local_notifications/flutter_local_notifications.dart';
import 'package:timezone/timezone.dart' as tz;
import 'package:untitled/notificationservice.dart';
import 'package:untitled/mainscreen.dart';

void main() {
  WidgetsFlutterBinding.ensureInitialized();
  NotificationService().initNotification();

  runApp(MyApp());
}

class MyApp extends StatelessWidget {
  // This widget is the root of your application.
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Flutter Demo',
      theme: ThemeData(
        // This is the theme of your application.
        //
        // Try running your application with "flutter run". You'll see the
        // application has a blue toolbar. Then, without quitting the app, try
        // changing the primarySwatch below to Colors.green and then invoke
        // "hot reload" (press "r" in the console where you ran "flutter run",
        // or simply save your changes to "hot reload" in a Flutter IDE).
        // Notice that the counter didn't reset back to zero; the application
        // is not restarted.
        primarySwatch: Colors.blue,
      ),  // ThemeData
      home: MainScreen(),
    );  // MaterialApp
  }
}

class MyHomePage extends StatefulWidget {
  MyHomePage({Key? key, required this.title}) : super(key: key);

  // This widget is the home page of your application. It is stateful, meaning
  // that it has a State object (defined below) that contains fields that affect
  // how it looks.
```

***Figure 3.5.1**: Part 1 of main.dart Section Code of The Notification System*

```dart
  // This class is the configuration for the state. It holds the values (in this
  // case the title) provided by the parent (in this case the App widget) and
  // used by the build method of the State. Fields in a Widget subclass are
  // always marked "final".

  final String title;

  @override
  _MyHomePageState createState() => _MyHomePageState();
}

class _MyHomePageState extends State<MyHomePage> {
  int _counter = 0;

  void _incrementCounter() {
    setState(() {
      // This call to setState tells the Flutter framework that something has
      // changed in this State, which causes it to rerun the build method below
      // so that the display can reflect the updated values. If we changed
      // _counter without calling setState(), then the build method would not be
      // called again, and so nothing would appear to happen.
      _counter++;
    });
  }

  @override
  Widget build(BuildContext context) {
    // This method is rerun every time setState is called, for instance as done
    // by the _incrementCounter method above.
    //
    // The Flutter framework has been optimized to make rerunning build methods
    // fast, so that you can just rebuild anything that needs updating rather
    // than having to individually change instances of widgets.
    return Scaffold(
      appBar: AppBar(
        // Here we take the value from the MyHomePage object that was created by
        // the App.build method, and use it to set our appbar title.
        title: Text(widget.title),
      ),  // AppBar
      body: Center(
        // Center is a layout widget. It takes a single child and positions it
```

Figure 3.5.2: *Part 2 of main.dart Section Code of The Notification System*



```
        // Center is a layout widget. It takes a single child and positions it
        // in the middle of the parent.
        child: Column(
          // Column is also a layout widget. It takes a list of children and
          // arranges them vertically. By default, it sizes itself to fit its
          // children horizontally, and tries to be as tall as its parent.
          //
          // Invoke "debug painting" (press "p" in the console, choose the
          // "Toggle Debug Paint" action from the Flutter Inspector in Android
          // Studio, or the "Toggle Debug Paint" command in Visual Studio Code)
          // to see the wireframe for each widget.
          //
          // Column has various properties to control how it sizes itself and
          // how it positions its children. Here we use mainAxisAlignment to
          // center the children vertically; the main axis here is the vertical
          // axis because Columns are vertical (the cross axis would be
          // horizontal).
          mainAxisAlignment: MainAxisAlignment.center,
          children: <Widget>[
            Text(
              'You have pushed the button this many times:',
            ), // Text
            Text(
              '$_counter',
              style: Theme.of(context).textTheme.headline4,
            ), // Text
          ], // <Widget>[]
        ), // Column
      ), // Center
      floatingActionButton: FloatingActionButton(
        onPressed: _incrementCounter,
        tooltip: 'Increment',
        child: Icon(Icons.add),
      ), // This trailing comma makes auto-formatting nicer for build methods.  // FloatingActionButton
    ); // Scaffold
  }
}
```

Figure 3.5.3*: Part 3 of main.dart Section Code of The Notification System*

The code is designed with the feature of adding timed notifications in mind. The main purpose of these notifications is to serve as a physical (on screen) reminder of upcoming check-ins. This will allow the user of the app to know how long they have until a scheduled check-in, and when it is. Additional code was developed for the testing of this prototype. This is the prototype of this feature at its current stage:

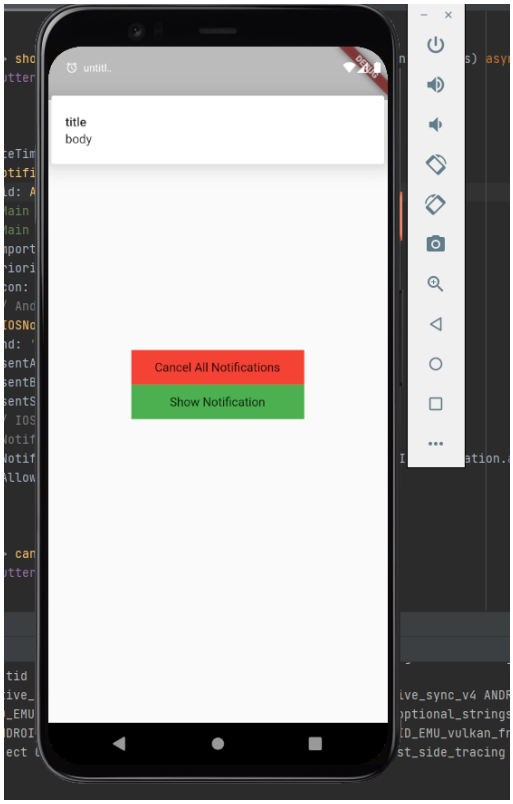


***Figure 3.5.4**: Sample Notification using the Notification System*

The client explained they would like the following message to be used as the template for check-in notifications: “Upcoming Check-In in 15 minutes at 8:30 AM!”. This message lets the user of the app know how long they have until their upcoming check-in, and a reminder of the set time for the check-in. The code must now be adjusted to be able to provide these scheduled notifications, with the content of the notification in the code being pulled from the saved data of the user’s check-in setup. By creating comprehensive notifications, with a simple and organized layout, the ultimate goal of the app is to ensure user safety is being accomplished.

Test objective	**Target specification to be compared with**	**How the prototype attempts to meet the specification**	**Results**
Test if the notification shows up.	The application has a notification alert system.	The prototype included code to display a notification	The notification showed up as desired.
Test if the notification redirects the user to the application upon pressing.	The application has a notification alert system that redirects the user to the application upon pressing.	The prototype included code to redirect the user to the application upon pressing.	The notification, when pressed by the user, redirected them to the application as desired.

***Table 3.5.1**: Notification System Test Objectives*

4 Conclusion and Future Work

After getting confirmation from the client that the team was on the right track, the group continued on the same plan for the development of the app. The group is progressing well with the development of the individual subsystems. While most of them are incomplete, the group has a clear idea of how to implement the necessary code. During the upcoming weeks, the team will be taking on the final challenge of transferring the code that has been written for android and having it function on iOS. While most of the application should function immediately, there are some libraries that require iOS specific code in order for the app to work as expected. The team anticipates having a fully functional product completed for design day that not only the client can use, but anyone in need of a personal safety application.