

GNG 2101

Design Project User and Product Manual

Per Diem - Personal Safety Application

Submitted by:

Team D1.5 - Code Bros

Maya Benhamou, 300173416

Riley de Gans, 300170104

Ekene Ebenebe, 8440519

David Todd, 300194033

Teo Vogel, 300215037

10th of April 2022

University of Ottawa

Table of Contents

Introduction	1
Overview	2
Conventions	3
Cautions & Warnings	3
Getting started	4
Configuration Considerations	4
User Access Considerations	5
Accessing/setting-up the System	5
System Organization & Navigation	5
Home Page	5
Exiting the System	5
Using the System	5
Emergency Contacts	6
Custom Check-In	7
Dexterity Test	8
Troubleshooting & Support	9
Error Messages or Behaviors	9
Special Considerations	9
Maintenance	10
Support	10
Product Documentation	11
Components	12
BOM (Bill of Materials)	12

Equipment list	14
Instructions	14
Testing & Validation	14
Conclusions and Recommendations for Future Work	18
Bibliography	19
APPENDIX I: Design Files	20

List of Figures

Figure 1 - Log In Screen

Figure 2 - Welcome Screen

Figure 3 - Emergency Contacts Screen

Figure 4 - More Options Screen

Figure 5 - Dexterity Test Screen

List of Acronyms and Glossary

Table 1. Acronyms

Acronym	Definition
API	Application Programming Interface
SMS	Short Message Service
UI	User Interface
iOS	iPhone Operating System

1 Introduction

This User and Product Manual (UPM) provides the information necessary for the elderly and disabled individuals to effectively use the Per Diem personal safety application and for prototype documentation. Perdiem was developed for an elderly client that needed a personal safety application to act as a proxy between them and specified individuals to confirm their daily well being. After several meetings, the clients' needs were fully understood and put into consideration during the prototyping stage. The latest prototype presented to the clients includes the following features: daily check in notifications, missed check in notifications, user account registration, a dexterity test, and Emergency contacts setup.

This document highlights the design problem and the final prototype, as well as instructions to easily replicate this project. For the latter, a basic understanding of object oriented languages is an asset but not a requirement. Steps to set up the application for a user, utilize the functionality of the app, and troubleshoot possible errors while using Perdiem are covered in this document.

This document is intended to be used by app developing enthusiasts, interested target users curious to explore the backend action, as well as future project groups looking to take the app from its current stage to the next level.

With respect to user data security and privacy considerations, there has been no encryptions for security on the backend, hence it is advised to avoid exposing the application to sensitive information. Security of app data is dependent on the user device, so for testers and enthusiasts, an emulator is advised.

2 Overview

Often, elderly or disabled people that live alone face health or related risks and are unable to call for help due to their limitations, which further escalates the possible risks in these scenarios. It is important to have someone that checks in on these people to make sure that they are safe, however, certain factors such as service costs or even personal preference presents a challenge in ensuring the safety of these at risk people. The personal safety app (Perdiem), resolves this issue both for users looking to avoid the service costs as well as users that do not want routine in-person visits.

Our client needed an application that prompts the user to check in daily at a specified time, within a specified check in window. This app is required to send emergency messages to each of three registered emergency contacts in a specified order when daily check in is missed. Before the emergency message is sent out in the case of a missed check in, the app is required to perform a dexterity test within a specified time. This app is also required to function on iOS devices (an iPad in the case of our client). Our client mentioned that similar apps that answered their needs were either exclusive to the United States or did not function on their device operating system (OS), Our application presents a functionality as well as cross platform utility advantage.

The first page a new user view on the app requests an email which becomes their user id. After registration, the next page includes just two buttons, one for the emergency contacts and the other to access the settings for the application.

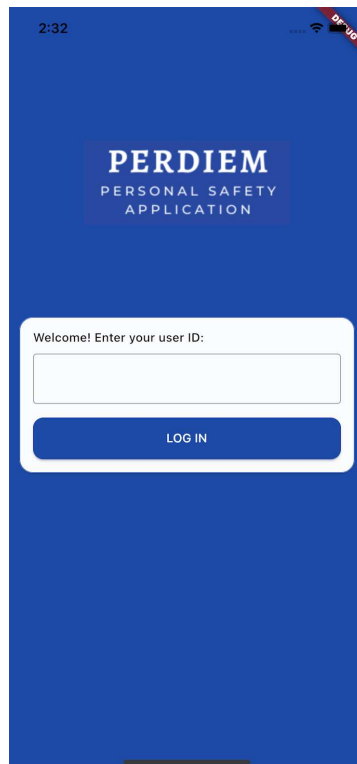


Figure 1 - Log In Screen

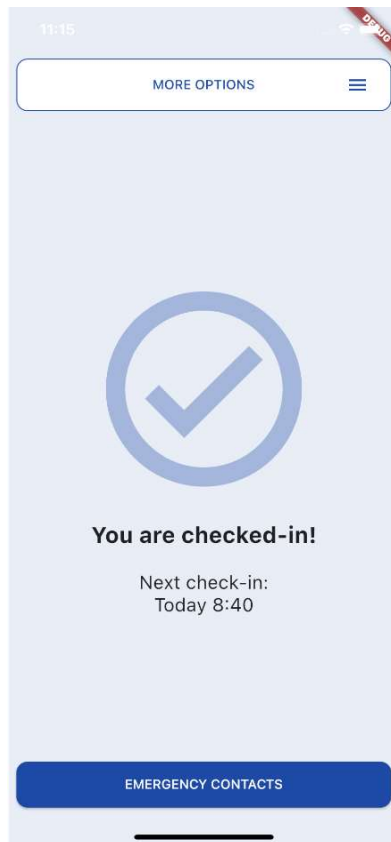


Figure 2 - Welcome Screen

2.1 Conventions

No special conventions used in this document.

2.2 Cautions & Warnings

Warning: the system must always be connected to WI-FI, this goes for use of the app and testing of the back end code.

Warning: the app cannot prevent any injuries or harm caused to the user, the developers of the app take no responsibility for any medical emergencies that the user experiences while missing a check-in. Always contact the proper authorities and medical personnel when an emergency is experienced, do not rely solely on the app's communication with emergency contacts.

3 Getting started

3.1 Configuration Considerations

To configure the back end server of the application the following steps must be taken in order:

1. Download VS Code [here \(click link\)](#)
2. Download Github [here \(click link\)](#)
3. Create a Github account [here \(click link\)](#)
4. Download the current backend code from Github [here \(click link\)](#) or from our MakerRepo [here \(click link\)](#) and open the folder in your VS Code.
5. Create a Twilio account [here \(click link\)](#)
6. Follow the steps in the Twilio registration to receive a sender phone number and verify the receiver phone numbers that will be used.
7. Create a Firebase Account [here \(click link\)](#)
8. Subscribe to the Firebase Blaze Plan [here \(click link\)](#)
9. Create a Firebase Project by clicking “Go to Console” once you have registered.
10. Configure Twilio with your Firebase console under the Extensions menu.
11. Configure the front end application with your Firebase project as explained [here \(click link\)](#)
12. Follow the steps [here \(click link\)](#) to configure your VS Code with Firebase, Firestore, Firebase Emulators, Node.js and Firebase CLI as well as deploy scheduled functions to the server.
13. Edit the code as needed under the index.js file in VS Code, consult the documentation at [here \(click link\)](#) and the comment provided by the developers before any and all changes
14. To perform non-destructive testing on your database consult the steps [here \(click link\)](#) to deploy your functions to a local emulated database, this will also save the time it takes to deploy the functions to the server when doing repetitive debugging.
15. To report a bug or request maintenance of the code contact redacted@gmail.com (Riley’s personal email)

To configure the front end project the following steps must be taken in order:

- 1 Download VS Code [here \(click link\)](#)
- 2 Download Github [here \(click link\)](#)
- 3 Create a Github account [here \(click link\)](#)
- 4 Download the current front end code from Github [here \(click link\)](#) or from our MakerRepo [here \(click link\)](#) and open the folder in your VS Code.
- 5 Get the Flutter SDK [here \(click link\)](#)
- 6 Follow the steps [here \(click link\)](#) to configure your VS Code with Flutter

3.2 User Access Considerations

Given that the product in question is software, the only limitation on usability is whether or not a potential user has access to a device which can download and run it. The application is intended for use by those of advanced age, or those who live alone in their homes and want to feel a sense of security. However, despite this, it could theoretically be used by anybody. For a user to be listed as an emergency contact currently they must be verified through Twilio with a verification code texted to them, this can be bypassed once full use of Twilio has been paid for.

3.3 Accessing/setting-up the System

All the user needs to do to use the product is download the application from the store, and then launch it from their home screen. As of now, there is no function which allows the user to make an account for themselves, so there is no requirement for the user to sign in to the application every time they want to use it. To set up their daily check-in all the user needs to do is select a time of day to check-in. In the future once development is finished ideally the user would be able to select the length of the check-in window and the number/frequency of check-in prompt notifications.

3.4 System Organization & Navigation

3.4.1 Home Page

Upon entering the app the user is given two options, emergency contacts where their contacts can be changed or more options where the logistics of the check-in can be changed (ie. check-in time, buffer period, frequency/number of check-in prompt notifications). As well on the homepage the user gets a message that says “You are checked-in!”.

3.5 Exiting the System

To exit the app simply click the home button as you would any other app. When the app is exited check-ins will continue as configured. To permanently exit the app and all of its features uninstall the app from your device.

4 Using the System

The following subsections provide detailed, step-by-step instructions on how to use the various functions or features of the Per Diem application.

4.1 Emergency Contacts

The user can input custom emergency contacts to the app. This information includes the name of the contact, as well as their number. The user can perform this action by selecting the 'Emergency Contacts' option on the menu, and then typing in the aforementioned info. Every action performed related to contacting these people is entirely automatic, and the user is present in no part of the process.

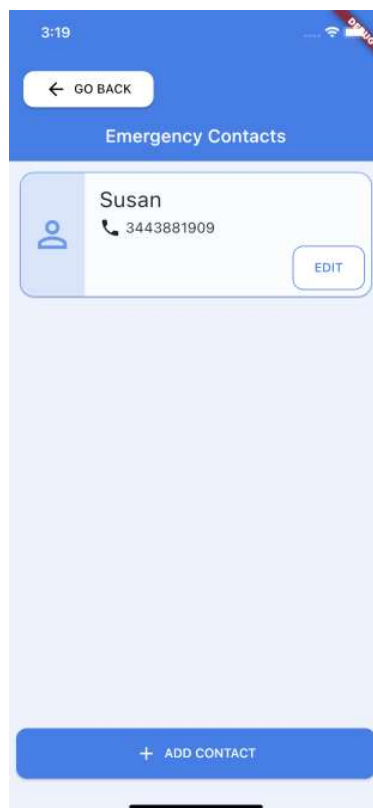


Figure 3 - Emergency Contacts Screen

4.2 Custom Check-In



Figure 4 - More Options Screen

The user can input a time period of their choosing to be the time during the day over which the application will send push notifications asking them to check-in. This time frame can be set by clicking the 'Custom Check-In' option on the menu, and then selecting the wanted frame.

Another function of this section allows the user to set the frequency of notifications sent. This determines the time between notifications in the previously set time period. There is an additional function that allows for setting a buffer period, which is a period of time after the set check-in period during which the application will refrain from texting the emergency contacts.

4.3 Dexterity Test

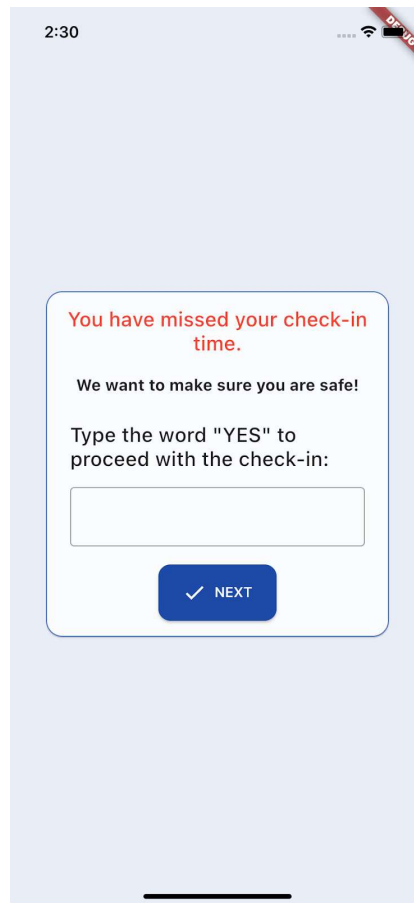


Figure 5 - Dexterity Test Screen

When a check-in is missed for the first time, after a short period the application will request the user to perform a very simple dexterity test. This manifests in the form of a text box asking for the user to type in the word 'yes'. If the user fails to complete this test successfully, emergency contacts are alerted.

5 Troubleshooting & Support

When errors are encountered while editing the backend code the simplest solution is to revert back to the original function code as provided in the setup. If this does not solve the issue you have altered a file out of the scope of the code that is irreversible. To fix this issue, start by checking for updates to all software and use the command “-npm audit fix” in the command prompt terminal of your project. If this doesn’t solve it, uninstall all of the installed software in the setup, clear your recycling bin and restart the setup procedure.

To solve errors in the front end the best thing to do is also to revert your changes. This is easy to do with the versioning control system provided by Git.

5.1 Error Messages or Behaviors

Any errors encountered while using the app will not present themselves as error messages, rather as incorrect or incomplete functionality of certain features. To solve these issues record the context of the errors (ex. Check-in time, check-in window, time of day, user information etc) then emulate the app source code through VSCode and test different values for each variable recorded. There is no simple solution to errors, many can only be resolved by extensive debugging that should only be conducted by a qualified individual. In its current state the app validates all values so no runtime errors or issues should be encountered, but if they are please report them as described in the support section.

The backend is configured in such a way that errors will only occur when the values originally written by the app do not align with the values read by the scheduled function. This will occur when the user enters the Firestore database and manually changes values or deletes documents/collections. To rectify this issue will usually be out of the skillset of the user and even the developer without full context so the simplest solution is to never manually change values unless the consequences have been thoroughly analyzed by a qualified individual. Allow any and all changes in the database to be done by the app and scheduled functions, these features can accomplish any of the changes you would want to make manually.

5.2 Special Considerations

No special considerations not already mentioned in the error messages and product documentation sections

5.3 Maintenance

Given that the prototype is software, no maintenance is necessary on the part of the user. With relation to the backend of the app, the only activities which could be considered maintenance would be related to licensing and updates.

Some examples of things which would fall under maintenance would be bug fixes, monitoring performance, ensuring continued functionality through changes in OS, and updating the user interface where needed. Routine updates are also made to the coding languages and software that the app and backend are based on, but generally these will not affect functionality.

5.4 Support

For back end support contact Riley at redacted@gmail.com (Riley's personal email) and for front end support contact Teo at redacted@gmail.com (Teo's personal email). Also consult all of the links given in the setup procedure, especially <https://firebase.google.com/docs/> which will provide the full documentation and examples for all Firebase features. When contacting for back end support please send a screenshot of how your database is organized, the link to where your function code can be downloaded and screenshots of any error messages encountered. For front end support please explain the exact issue, provide screenshots of your settings page and provide full context (ex. Check-in time, check-in window, time of day, user information etc)

6 Product Documentation

Twilio was chosen because of the ease of integration with Flutter and Firebase as well as the lack of simple to use competitors.

Firebase was chosen for its volume of documentation and reliability as a database as well as function scheduler. Other databases could be used but will not be compatible with the backend code in its current form. If another database is chosen it must be able to schedule functions and store the required data as well as be compatible with Flutter. The connection between Twilio and the Database can be made with Flutter as a middleman, so Twilio compatibility is not required for the database, but it severely complicates coding the new functions and requires changes in the app code. Non-destructive performance testing for Firebase and Twilio can be done using the Firebase Emulators as described in the setup as well as deploying to the actual server, although this can damage the data stored.

The scheduled function was built using pseudo code that basically would iterate over all the users in the database, determine whether each was in emergency or not, then using the Twilio API to send a text message to each of their stored contacts. Considerations when recreating or changing this code are how the data is organized (ie. the structure or hierarchy). If the data is organized differently the function will not be able to access the correct information. Another consideration to be taken is the edge case where the users check-in start time and check-in end time are on different days, for example check-in from 11pm to 2am. An example of an issue would be the current time is 1am, therefore the user is in their check-in window, but the current time is less than 11pm, making it also appear like it is before the check-in time. The current version of the code accounts for this, but before making changes we suggest drawing out all the of the possible scenarios to have a complete understanding of the edge case.

Flutter was chosen for the front end given that it's a very modern and fast way to develop cross-platform apps. That means that with only one code we can create an app for Android and iOS, that previously would have been two different projects. It's also a very well documented and easy to use framework and the apps that turn out from using it are very performant (they run fast). This framework has a declarative UI philosophy and uses the Dart programming language. For common developing scenarios you can access the Cookbook [here \(click link\)](#) and for taking advantage of packages from other developers access the Dart packages site [here \(click link\)](#).

This project uses five external packages:

- [flutter_local_notifications](#) for sending check-in and dexterity test notifications
- [shared_preferences](#) for storing user data
- [firebase_core](#) & [cloud_firestore](#) for communicating with the back-end
- [flutter_launcher_icons](#) for generating the app icon

6.1 Components

6.1.1 BOM (Bill of Materials)

Table 1: Description of all materials used throughout the project.

Personal Safety Application			
Material	Use/Description	Cost (\$)	Source (link)
Software			
GitHub Repository	Internet hosted version control, for sharing and updating group code	0	Link
Git	Local Version Control System	0	Link
Discord	Team communication through video/voice calling and group chat	0	Link
VS Code	Source-code editor used for developing flutter code	0	Link
Flutter SDK	Cross-platform development kit for creating code that works on IOS and Android	0	Link
Node.js SDK	Cross-platform back-end JavaScript runtime environment	0	Link
Firebase server hosting	A computer in the cloud to handle queries and storing data	0 (during dev/testing) prices increase around 1000 users	Link
Twilio SMS API	An API for delivering SMS messages (for emergency contacts)	0 (during dev/testing) prices increase around 1000 users	Link
Firebase database	A database to store user data	0 (during dev/testing) prices increase around 1000 users	Link
Google Workspace	Cloud-based collaboration and	0	Link

(Drive, Docs, Slides, Sheets)	productivity tools for completing deliverables		
Zoom	Cloud-based teleconferencing and chat for client meetings	0	Link
Stack Overflow	Question and Answer Forum for collecting information about coding features in Flutter	0	Link
Wrike	Collaborative project management and planning	0	Link
Figma	Graphics prototyping tool for Windows and IOS development	0	Link
Canva	Custom graphic design platform for presentations, documents and UI mockups	0	Link
Lucid Chart	Collaborative drawing platforms for charts and diagrams related to UI and concept development	0	Link
TestFlight	IOS app for testing on end user devices without approval through the Appstore.	0	Link

6.1.2 Equipment list

1. Computer that can run VS Code for development and testing
2. Stable WI-FI Connection (ie. Router, Switch, AP, Ethernet etc.)
3. End User Test Device (Any Android or IOS Device)

6.1.3 Instructions

To prepare the app on an Android or IOS device simply go to the devices app store and download the Per Diem app. For more complex instructions with respect to app code and backend setup see the configuration considerations.

6.2 Testing & Validation

Test 1 - UI Font Size/Colour

The quantitative test performed was testing different colour combinations and font sizes for different levels of eyesight. This would test whether different color combinations can meet the target specification of using size 20 font at reasonable reading distance / varying eyesight strengths and evaluate what the best colour combination and font size would be using measurable tests. Ideally the colour combination and font size chosen could be read at any of the distance measured, ensuring that even for the most visually impaired users the app is still accessible. To test this, random strings of letters with decreasing font size were read from the UI prototype on Canva at increasing distances from 0-10 feet, increasing at increments of 2 feet, to simulate eye strength decreasing. A new string was randomly generated each time a distance was tested, using font sizes from 6-32 decreasing in increments of 2. The tester stepped back to the distance being tested and read as many letters as they could, the font size of the smallest letter they could read correctly was recorded. This test was repeated for each of the colour combinations; green font on white background, black font on white background, blue font on white background and white font on black background. Results were recorded in a table then graphed as a line graph from a scatter plot to see how each combination performed. As seen below the results show that a more reasonable minimum font size target specification would be 32 instead of 20. At this size any text could be seen from most distances and levels of visual impairment. As far as colour combinations, white text on black background performed the best, with black text on white background coming in second and blue and green text on white background producing around the same results in third. The expected results for this test were that black on white would perform the best, followed by green on white, then blue on white, finally white on black. Given these results black on white was chosen for the first prototype UI and white on black was chosen for the fail-safe check-ins.

Max Readable Font Size vs Distance Away from Screen for Different Colour Combinations

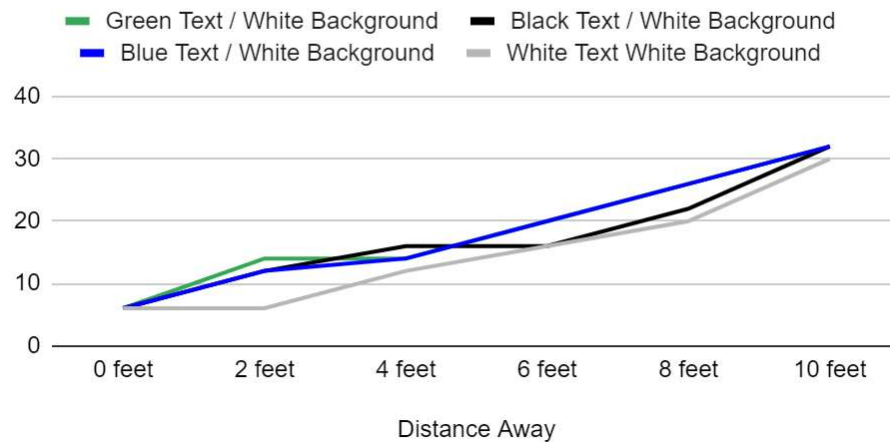


Figure 1 - Max Readable Font Size at Different Distance for Different Colour Combinations Plotted for Analysis as Line Graphs from Scatter Plots

Test 2 - SMS API and Firestore Database Cloud Function Testing

For this test a low fidelity, specific prototype was made to test whether the emergency text length had any impact on the delay time from when the message was sent to when the message was received by the end user. This test was conducted using the Firestore database already created for the app, with the Twilio SMS API extension implemented and a rudimentary cloud function coded on Node.js. The way this prototype works is that in our database there are two collections, messages and users. Under the messages collection, the extension works so that each time a document is created with a “to” (phone number) and “body” field a message will be sent using the contents of the document. The users collection is organized such that each user is assigned a randomly generated ID and has fields for emergency state (true/false), name, emergency contacts and check-in time. The basic cloud function is written so that each time the emergency state of the user is changed, a test text message will be sent to a verified tester’s phone number. Messages of different lengths were used as the test text and the time was measured between when the user’s state was updated to the time the text was received, then plotted below for analysis. As seen in the

graph, there is no correlation between text length and delay time. This allows us to create a new target specification for the maximum length of the emergency text. No maximum is imposed by considerations for delay time, but a maximum of 160 characters is imposed by the SMS system. If the message exceeds 160 characters it will be divided into multiple messages which will increase costs related to the Twilio and Firebase services.

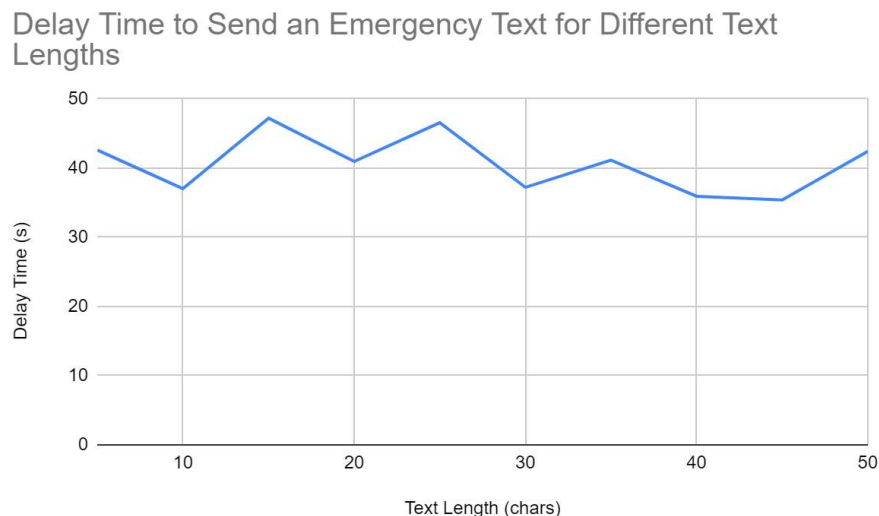


Figure 2 - Delay Time to Send an Emergency Text for Different Text Lengths Plotted for or Analysis as a Line Graph from a Scatter Plot

Test 3 - UI Configuration and Battery Testing

For this test the UI from the previous prototype was improved upon in Flutter to create a medium fidelity, general prototype. This prototype was used to deploy the app as an apk to Android users and through TestFlight to IOS users for functionality and usability testing. This prototype was also used to test battery consumption as this was brought up as a concern by the client in the second and third meetings. The test was performed by leaving the app open for specific amounts of time in both the dark (black background, white text) and light (white background, black text) modes, then the battery consumption was recorded at each time interval and plotted below for analysis. Dark mode consumed 0.08% battery per minute while light mode consumed only 0.06%. This shows us that technically light mode is more efficient, but there is no considerable difference, similar to the findings with readability. This means that the choice

between these modes will be based on final feedback from the client on aesthetics and usability, as both modes perform very similarly across two different tests.

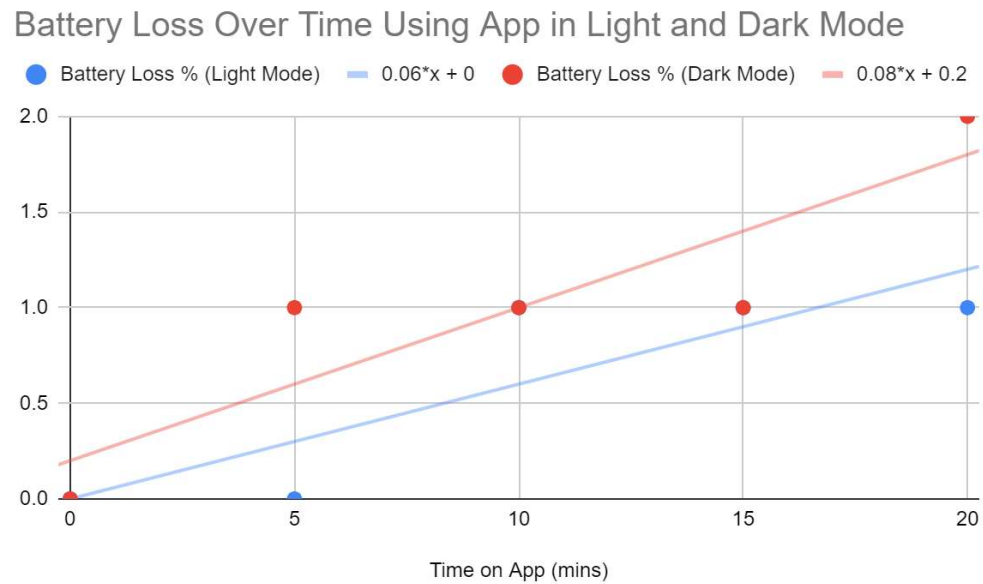


Figure 3 - Battery Loss Over Time for Different UI Configurations Plotted for Analysis as Linear Fits from Scatter Plots

7 Conclusions and Recommendations for Future Work

An important skill sharpened over the course of this project was time management. For all of us, this was a very intensive semester, so it was imperative that we used our time effectively. To facilitate this, we accommodated team members which were having particularly difficult weeks by lessening their work loads. This system worked well for us, up until the prototyping phase, where it was decided that the coding and documentation would need to be handled by different members.

From this, a lesson was learned in communication. Given that certain members of the team did not have the same level of understanding of certain aspects of the application as those who were working on it, weekly meetings were necessary to facilitate a cohesiveness between the deliverables and product.

Given more time to work on this project, the first functionality which would be implemented would be a pause check-in function. This was abandoned relatively deep into development of the application due to how unexpectedly complex it was to create. This function, although unnecessary to the products overall function, would avoid potential annoyance for the user in the form of unwanted, or unneeded check-ins. For example, if the user were with family, and did not feel the need to be checked in on, they could pause it for however long they were with them, and it would automatically turn back on once the timer ran out.

Another function which was requested by the client was a hike mode. Although the overall description of this function given by the client was quite vague, it essentially boiled down to a function which would, when activated, be a mini check in on the user. Instead of it being only a daily check-in there would be another one at the time given by the user to when she would get back from her hike or any outdoor activity. This is an added safety measure to make sure that during the hike if something would happen the users' emergency contacts would be made aware and contact the authorities if necessary. As well as, adding an additional GPS tracker to help locate the user to their emergency contacts if they don't check within a period of the predicted time of their activity.

8 Bibliography (Web Ressources)

<https://code.visualstudio.com/download>

<https://desktop.github.com/>

<https://github.com/>

https://github.com/rdegans/GNG2101_Personal_Safety

<https://makerepo.com/MayaBen/1147.gng-2101-d15-perdiem-personal-safety-app>

<https://www.twilio.com/login>

<https://console.firebase.google.com/u/0/>

<https://firebase.google.com/pricing>

<https://firebase.google.com/docs/web/setup>

<https://firebase.google.com/docs/functions/get-started>

<https://firebase.google.com/docs>

<https://firebase.google.com/docs/emulator-suite>

<https://docs.flutter.dev/get-started/install>

<https://docs.flutter.dev/get-started/editor?tab=vscode>

APPENDICES

9 APPENDIX I: Design Files

Table 3. Referenced Documents

Document Name	Document Location and/or URL	Issuance Date
MakerRepo	https://makerepo.com/MayaBen/1147.gng-2101-d15-perdiem-personal-safety-app	April 10th, 2022