# 

Bail Safe

project report | NDSU - CSCI 415

Jacob O'Reilly | Jordan Meidinger | Nathan Marcotte

December 17, 2019

Contents

[Introduction 2](#_Toc27472268)

[Design 2](#_Toc27472269)

[Team Dynamics | Project Management 4](#_Toc27472270)

[Development/implementation 5](#_Toc27472271)

[Front End 5](#_Toc27472272)

[Hardware/Back End 6](#_Toc27472273)

[Future Advancements 8](#_Toc27472274)

[Key Learnings 8](#_Toc27472275)

# Introduction

As our team was brainstorming ideas we could pursue for this project, Jacob O’Reilly mentioned his interest in mountain biking and some of the dangers associated with it. Often when solo ridding bike trails, there is the danger of being injured or knocked unconscious with no one around to assist. We envisioned a solution which would be able to detect when the rider is injured or needs help and alert predefined emergency contacts with their GPS coordinates. Utilizing concepts learned from our Networking and Parallel Computation course we were able to develop a solution using both an Android Phone and a Raspberry Pi with Bluetooth communications between them. Bail Safe recognizes when a user is in danger by placing a sensor in the bike seat connected to a Raspberry Pi. The Raspberry Pi connected to the phone gathers GPS coordinates and sends SMS messages in emergencies.

We want to give peace of mind to trail riders by alerting their friends and family in emergencies.

# Design

As we thought about existing apps and technology we use in our own lives, we agreed good design was important. Not only should our app look good, but it should also function in a way that is consistent with the way our future users expect it to and be easy to learn. If an interface is difficult to learn, users will most likely abandon our solution. According to ZipWhip.com “nearly 21 percent of consumers abandon newly downloaded apps after just one use and 77 percent never use an app again 72 hours after installing it” (Wilson, 2019). This is an important factor to consider in app development as the goal is to have users continuously interact with the app.

Before staring development on the app, we first developed both a user interface (UI) design mockup and user experience (UX) flow (see Figure 1 and 2). All functionality of the UX flow is implemented within the UI mockups.

Here are the key features described in both diagrams:

* First time using app/setup
  + Allow permissions for SMS Messages, Contacts, and Location
  + Pair Android Phone to Raspberry Pi via Bluetooth
  + Select emergency contacts from phone contacts
  + Set default timer value for when alert should be sent after detecting user is no longer on bike seat
* Normal functionality of app
  + Start riding – starts count down timer to send emergency alert if bike rider is not detected on bike seat within this time period.
  + Stop activity – stops current timer. This can be used for short rests where the rider is not on the bike.
  + Timer countdown – a new app screen will appear when the timer is close to being expired with the option to cancel alert. This is helpful for when the rider forgets to turn off timer but is not in an emergency.
  + Recall alert – if the user accidently permits an alert to be sent when there is not an emergency, they can send a follow up message to their contacts indicating they are not in danger.
* Side menu available on any screen
  + Edit emergency contacts
  + Edit timer value
  + Connect to bike seat
  + Start riding

A screenshot of a cell phone

Description automatically generated

Figure 1 UI Design

A picture containing screenshot

Description automatically generated

Figure 2 UX Flow

# Team Dynamics | Project Management

As our group came together to solve this particular problem of solo bike riding, we realized we had a very unique team. Between our three group members, we have a wide range of skills including strong development skills, hardware, and user interface/user experience design. Because of the variety of different skills, we were all able to take lead in different parts of the project:

**Jacob O’Reilly:** App Side Development

**Jordan Meidinger:** Hardware + Raspberry Pi Development

**Nathan Marcotte:** UI/UX Design + Communications

To help facilitate all of the many pieces of this project we used Trello (Figure 3) to help manage all of our tasks and assign them to different group members. This helped for a variety of reasons. We were all able to quickly check who was working on what task and see how the timing of each individual task lined up with other dependent tasks. For example, because our project included both the Raspberry Pi and an Android Phone, we had to make sure each side of the project was timed right so we could setup the Bluetooth communications between the devices.

A close up of a mountain

Description automatically generated

Figure 3 Trello Board

# Development/implementation

## Front End

The front end of the BailSafe app is an Android app built on the Kotlin language. We chose Kotlin so that we could have a native Android app, making it easier to use Bluetooth services which is a major part of our project. The front end consists of a menu that allows you to connect to a Bluetooth device, which in this case is the Raspberry Pi backend. Once connected to the Raspberry Pi both the backend and front end start an infinite loop. The front end’s loop is consistently listening for the backends’ message. Every second the backend sends a “one” or a “zero” based on if the button is being held down or not. The front ends variable gets updated with whatever is being sent over from the Raspberry Pi.

When you decide you're going to start your ride, you will click the “start ride” button. The button will change from “Start Ride” to “End Ride”. You can then click the button again when you want to end the ride and it will switch back to “Start Ride”. Once the ride is started, the variable that has the value of the message being received from the back end determines if the timer in the bottom right corner of the screen is started or not. The value “one” means that the rider is sitting on the seat which is holding down the switch. This will just set the timer to the time that has been determined by the user but will not start counting down. Once the rider is off the seat the variable will be changed and the timer will start to count down. If the rider was to get back on the seat before the timer gets to zero, then the variable will be set back to “one” and the timer will go back to the set time and listen for the variable to change to “zero” again. If the rider were to get off the bike seat, they can hit end ride and it will turn off the timer. If the rider falls off the seat and doesn’t click the button to stop the ride or get back on the bike seat, the timer will count down to zero and then send a text to emergency contacts selected by the user. This text consists of a message saying that you have fallen and your current GPS coordinates. At this point it is up to the emergency contact to handle the emergency correctly.

We had some stretch goals that we were unable to get to. We ended up hard coding cell phone number values for the emergency contacts and same with setting the timer’s time value. In the future, we want to add a menu for selecting emergency contacts from your contacts. We also want to have the user input the message they want to send and setting the time value based on how much time they need. For example, if you’re a mountain biker you may be off the seat more often than if you are on a road bike. Here you would probably want a longer time, so the message won’t get sent when you’re still riding the bike. Another feature we wanted to add is that once the timer gets close to zero by a minute or so, an alarm will go off, letting you know the timer is about to go off and it would give you the option to swipe to cancel the alarm and end the ride. This would help prevent false alarms.

A screenshot of a cell phone

Description automatically generated

Figure Bail Safe App

## Hardware/Back End

The back end and hardware of this project is implemented using a Raspberry Pi and a physical switch. The operating system we choose to use on the Raspberry Pi is called Raspbian which is a Debian-based computer operating system for the Pi. One of the first things we needed to do when setting up the Pi was making sure we had the updated version of Python as Raspbian only uses Python 2.6. Once we had the current version of Python, we could get to work on researching some information on how the Pi uses its pins to communicate with the Python script we were running. The package we needed for the Python script to work with the pins on the Raspberry Pi is the GPIO (general-purpose input/output) package. The pins on the Raspberry Pi are along the top edge of the board and the necessary connections include: power, input pin, and ground. The power is used to send 3.3 V through the wiring and bread board where it is connected to a switch and resistors. The current is then sent to the ground for any overflow and the rest of the current is passed into the input pin. The input pin is read in on the script as a “one”, as the current is going through the pin. When the circuit is turned off by releasing the switch, the current stops reaching the input pin and reads “zero”. Finally, in the script we created an if statement when the input is active and the rider is on the seat, and when there is no current the rider is off the seat.

Once we were able to setup the pins/switch with the Raspberry Pi, we could send this information via Bluetooth to the phone. In order to send items over Bluetooth we needed to get another python package to communicate with the app we developed. The package we used was Pybluez, but the import on the file is just Bluetooth. The first thing we needed to do is set up a sever socket that uses RFCOMM, (Radio Frequency Communication) which is a simple set of transport protocols, made on top of the L2CAP protocol, providing emulated RS-232 serial. The socket binds to any available port and starts listening for any connections. The script also needs to create services so we can send and receive data when a device connects. The socket waits until the app connects on the phone. Once connected, a for loop is started until a disconnect or error occurs. In the loop the Pi sends via Bluetooth a zero for no rider, and a one for a rider on seat. Lastly, when a disconnect occurs the script cleans up the ports and turns off the pins to the GPIO.

A picture containing indoor, wall

Description automatically generated

Figure 5 Button Connected to GPIO on Raspberry Pi

# Future Advancements

While our team is satisfied with the progress we have made on this project, there is so many possibilities to expand the applications of this technology. We first started this project developing an emergency notification system for bike riders, but we know there are so many other similar sports which can benefit by improving user safety. Some of these related sports include ATVs, snowmobile, jet skis, and dirt biking. While expanding to other sports there is also the opportunity to improve our technology to incorporate wearables like the Apple Watch and integrated smartphone sensors to improve our ability to sense if a user is injured or in danger. Not only would the smartwatches provided additional data, they can also be used to improve the user experience by providing ease of use for interacting with our app. Often when on the trails, smartphones are tucked away in pockets or backpacks, making them difficult to access quickly. This is the perfect use case for wearables as they are easily accessible on someone’s wrist.

Another, key area of focus for future improvements is our user experience (UX) design. Because our app/solution will be used in emergencies it is important the overall experience is as seamless as possible. For example, one easy improvement is to modify our text messages sent to emergency contacts by including a direct link to Google Maps or real time location sharing, instead of just the GPS coordinates. Although this seems like a minor feature, small details like this can save time when there is an emergency.

# Key Learnings

While the main objective of this project was to reinforce our knowledge of Networking and Parallel Computation concepts, our team learned much more! We were first able to learn more about Kotlin, our programming language of choice for app side development. On the backend we learned a ton more about Python and how it handles opening ports and accepting incoming Bluetooth connections. The Raspberry Pi was amazing to work with when setting up the GPIO and will be our go to for future projects as it’s easy to learn and has well outlined documentation.

*“Next time you ride, think Bail Safe!”*

Works Consulted

Wilson, M. (2019, 2 12). *The Majority of Consumers Won’t Download an App to Communicate With You*. Retrieved from ZipWhip: https://www.zipwhip.com/blog/app-download-statistics-reveal-why-people-dont-download-apps/