Fall Detection

Group12: Boris Bagryanskiy, Kezia Butcher, Talal Almutairi, Ryan Hoke

# Abstract:

The fall detection device is a wearable device that detect falls and reports each fall to the user via an application to check if the user is unconscious or needs medical attention. The users should fill out their information in the application and their caretakers’ information. When the device detects a fall, the user will be notified via the application, which will have a timer that will count down, and if the user doesn’t respond during that time the application will contact their caretaker if they have one, in the case the user doesn’t have a caretaker or the caretaker doesn’t respond it will stimulate a call to the EMS (Emergency Medical Service).

**Table of Contents**

[***Abstract: 2***](#_gjdgxs)

[***Introduction***](#_30j0zll) *4*

[***Design Achievements:***](#_1fob9te) *4*

[***Hardware:***](#_3znysh7) *5*

[**Complete schematic**](#_2et92p0) 5

[**Complete physical layout of all boards**](#_tyjcwt) 6

[**List of materials**](#_3dy6vkm) 7

[**CD of Spec Sheets for all parts used**](#_1t3h5sf) 8

[***Complete Software design:***](#_2s8eyo1) *13*

[***Complete source code listing:***](#_17dp8vu) *14*

[***Test Results:***](#_3rdcrjn) *28*

[***Safety Precautions:***](#_26in1rg) *32*

[***Reflections:***](#_lnxbz9) *32*

[***User Manual:***](#_35nkun2) *33*

[***Maintenance:***](#_1ksv4uv) *34*

Introduction:

One of the common things that have been and is still killing people for a long while is falling. Falling may cause concussions or even deadly injuries if the injured people don’t get the right care or immediate attention. Even a simple fall may get really dangerous for elderly, especially when they live by themselves and have no one to take care of them. In order to solve this problem, building an affordable fall detection device was our goal, it’s main purpose is to help these people get them immediate medical attention to avoid the consequences of these unfortunate accidents which will lead to save their lives and in the same time they don’t pay a huge price, unlike apple watch which has a starting price of $400.

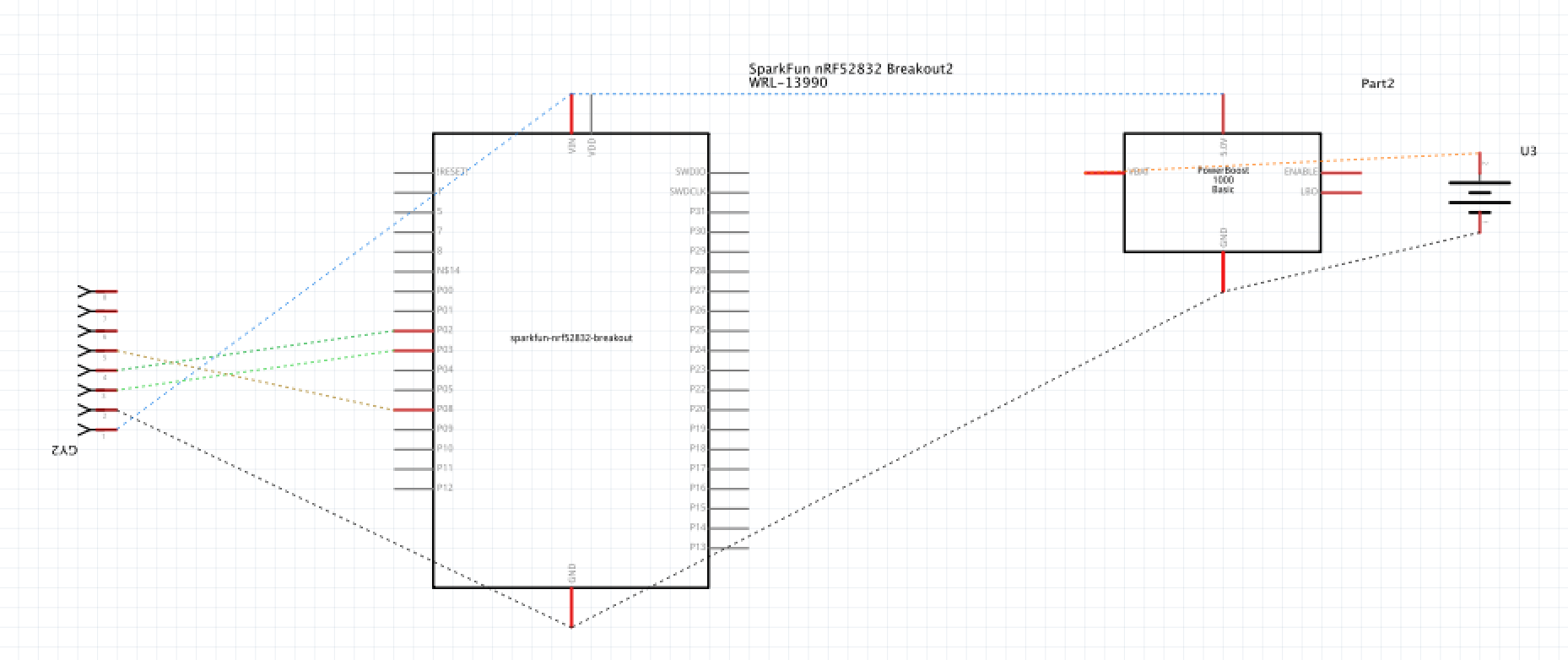
# Design Achievements:

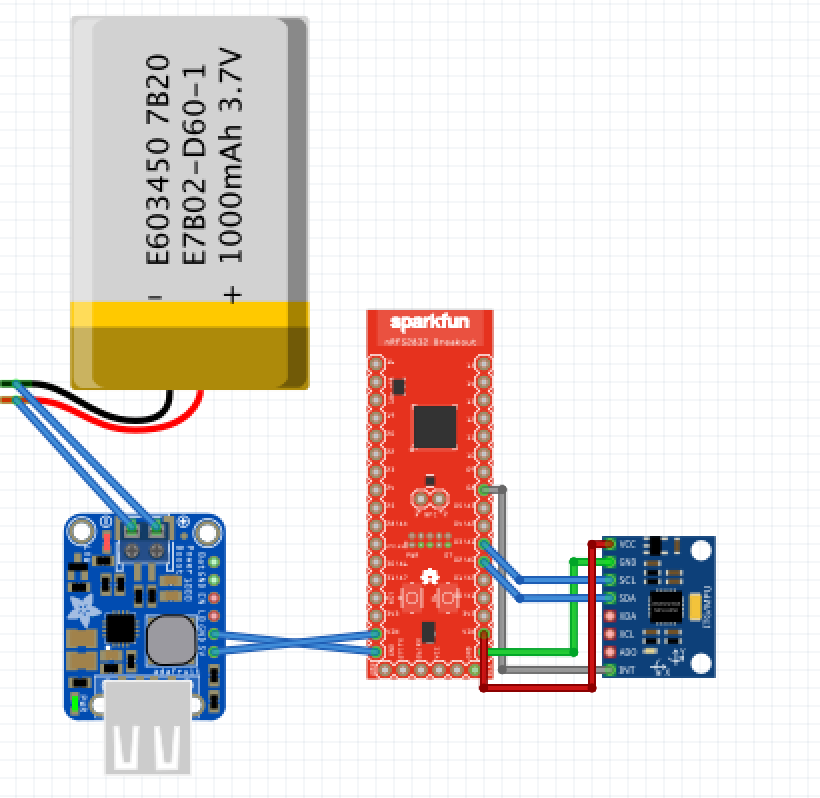
The group were able to build a wearable device that detects falls and report them to the user, the caretaker and simulate a call to the EMS. The wearable device is made to be worn on the belt du to ts bulkiness. The fall detection device has a triple axis accelerometer which will sense the static force of the acceleration. Also, sparkfun nrf52832 breakout which distributes the power to the other components, it supports the LBE (Low Bluetooth Energy) and provides a low power wireless communication.The group also used the powerboost charger 1000 which is used to charge the battery and to deliver the power from the battery to the nrf52832 breakout.

In addition, the group made an application which uses the Bluetooth to get connected to the fall detection device which receives fall notifications and reports these falls to the caretaker or even stimulate a call to the EMS if the user doesn’t respond, the caretaker or the EMS will receive the notifications of the incident through Wi-Fi. The application has an easy interface for elderly to set up and to get connected to the device. Another thing the group added to the application is a button to press to insure the user is fine, the reason for having to press the button for a while is to prevent the user from falling on the device and pressing the button by mistake. However, the group couldn’t make a new schematic for the sparkfun nrf52832 breakout board because the manufacturer didn’t have a detailed schematic for this part.

# Hardware:

## Complete schematic



Complete physical layout of all boards

## 

## 

## 

## 

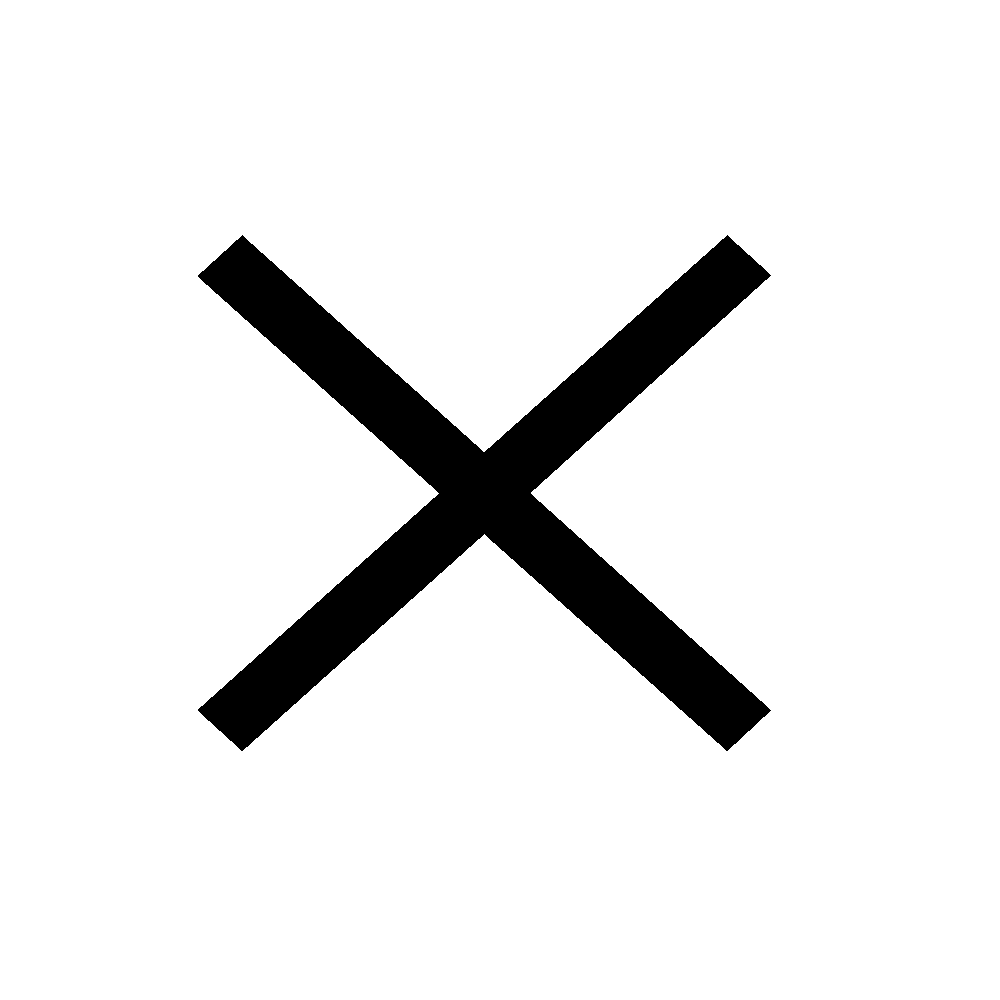
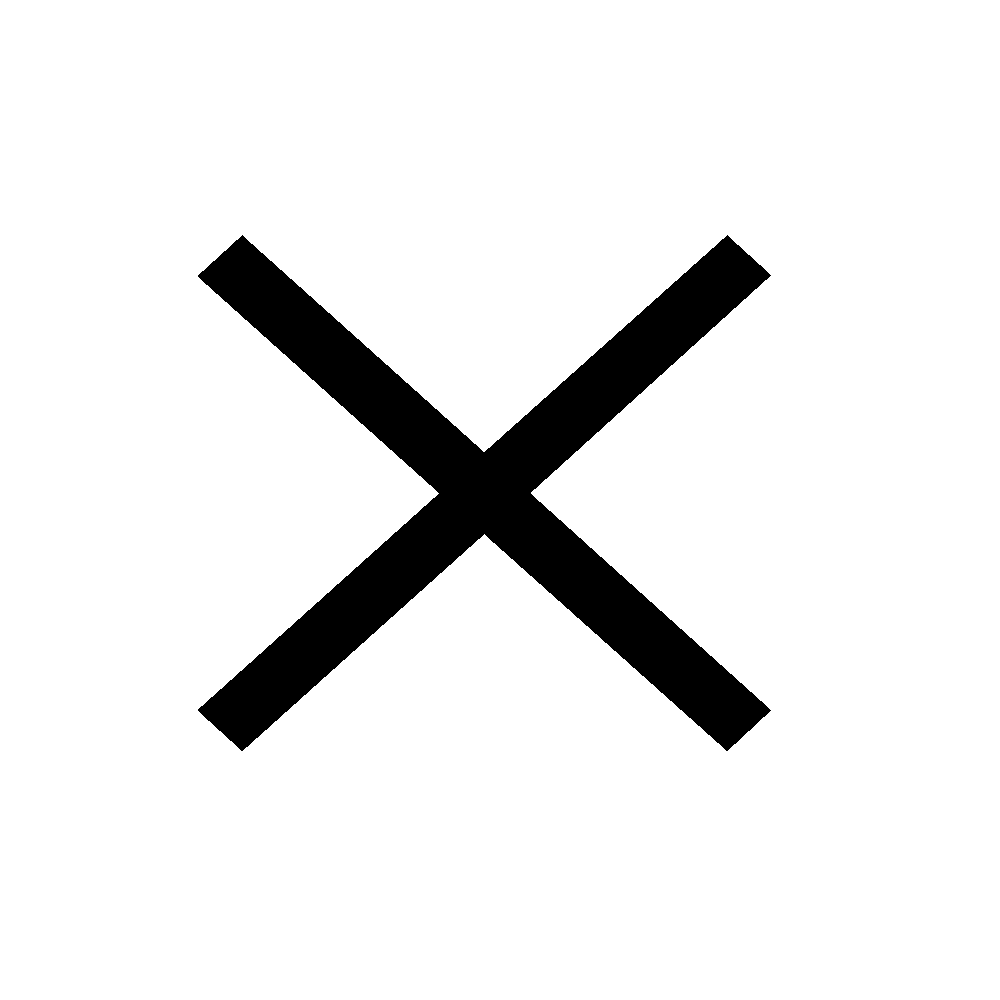
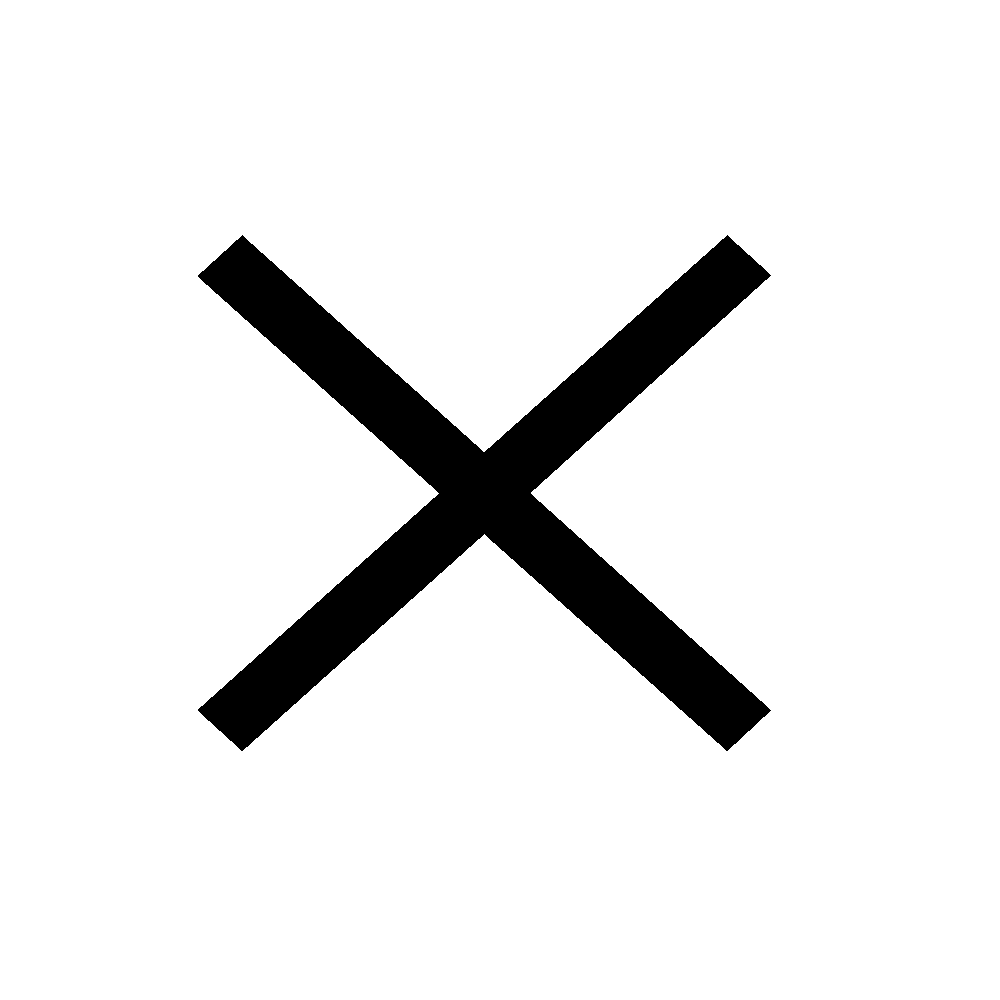
## 

## 

## 

## 

## List of materials

1. Power booster
2. Adafruit 1000 charger, rechargeable 5 V Li-Po USB boost @ 1A-1000C
3. Prototyping board
4. Protoboard; perforated.
5. SIM&NAT
6. 140 male-to-male, 140 male-to-female Dupont wire, 140 female-to-female cables, each 30 cm/11.8 inch for Arduino or Raspberry Pi 2/3.
7. Artistic wire
8. 20-gauge tarnish-resistant wire.
9. USB connector
10. Sparkfun microB USB breakout.
11. USB to serial converter
12. SparkFun Beefy 3 FDTI Basic Breakout.
13. Break-away headers
14. SparkFun Break Away Header.
15. Battery
16. SparkFun lithium-ion battery
17. Accelerometer and gyroscope assembly
18. MPU-6050 triple-axis accelerometer and gyroscope breakout board.
19. Particle sensor
20. SparkFun MAX30105 particle sensor breakout.
21. Bluetooth and wireless controllers
22. SparkFun 2.4 GHz nRF52832 module breakout.

12. Housing component

a. 3D printed housing box

## CD of Spec Sheets for all parts used

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **BRAND** | **PRODUCT** | **PART #** | **TITLE** | **LINK** |
| SparkFun | Accelerometer-Gyroscope module | Invensense MPU-6050 | 3-axis accelerometer + 3-axis gyroscope integrated in one IC that produces a 6-axis and I2C port to facilitate full 9-axis Motion-Fusion Algorithms | https://www.sparkfun.com/products/11028 |
| Multiprotocol radio system on chip (SOC) module | nRF52832 | Microcontroller with Bluetooth low energy (BLE) support and Nordic’s proprietary ultra-low power wireless communication | https://www.sparkfun.com/products/13990 |
| Beefy 3 FDTI USB to serial converter module | FT231X USB to serial IC | 3.3 V serial board compatible with all Arduino models and clones | https://www.sparkfun.com/products/13746 |
| Battery | Li-polymer battery | 1000 mAh with standard 2-pin JST-PH connector; pin spacing of 2 mm | https://www.sparkfun.com/products/13813 |
| Particle sensor module | Max30105 | Particle detection, distance sensing, photon detection, and pulse characteristics sensor compatible with 5 V and 3.3 V microcontrollers | https://www.sparkfun.com/products/14045 |
| USB connector | Micro B-type  USB connector module | Individual USB terminals broken out into labeled contacts for easy soldering compared to the connector itself | https://www.sparkfun.com/products/12035 |
| Break-away headers | Straight headers | 40-pin array that may be cut to any size | https://www.sparkfun.com/products/116 |
| SIM&NAT | Jumper wires | Connecting ribbons | 120 pieces of ribbon cables divided in 40 male-to-male, 40 male-to-female, and 40 female-to-female for Raspberry Pi signal jumpers | <https://www.amazon.com/gp/product/B06XRV92ZB/ref=ppx_yo_dt_b_asin_image_o02_s00?ie=UTF8&psc=1> |
| Adafruit | Power booster | 3.7V Lilon/Lipoly battery | Portable and rechargeable power supply with an output of up to 1 A. | <https://www.amazon.com/Adafruit-PowerBoost-1000-Charger-Rechargeable/dp/B01BMRBTH2/ref=sr_1_fkmrnull_1?crid=353L6TS5IQ6ZR&keywords=powerboost+1000+charger+-+rechargeable+5v+lipo+usb+boost+1a+-+1000c&qid=1555556770&s=electronics&sprefix=powerboost%2Carts-crafts%2C144&sr=1-1-fkmrnull> |
| Perfboard | Prototyping board | Perforated double-sided boards | Perforated 6.2” by 4.5” with .042” hole diameter, .1” grid and 2200 solder pads | <https://www.amazon.com/gp/product/B0193W4FRE/ref=ppx_yo_dt_b_asin_title_o01_s00?ie=UTF8&psc=1> |
| Artistic wire | copper wire | AWD-20-BC-06YD | 20-gauge bare Non-Tarnish copper wire, 6 yards | <https://www.amazon.com/gp/product/B003UDC5UM/ref=ppx_yo_dt_b_asin_image_o06_s00?ie=UTF8&psc=1> |

# Complete Software design:

The hardwear of the device changed due to lack of availability of schematics. Originally after the device was built with the components from sparkfun there was going to be a custom PCB built to do the same detecting as the first one. However the custom PCB was unable to be created due to lack of already existing schematics from companies.

Another component that has been modified from the original plan is the housing component. Originally the device was supposed to be worn on the wrist like a watch with links similar to watch links going around the wrist. The links would have spaces inside of them for storage of little batteries. Also, the motive for the device to be worn on the wrist was to be able to have a heart rate monitor track your pulse on your wrist. However, we did not get the heart rate monitor working, did not get the small batteries, and the device was too large to be worn on the wrist. Also if worn on the wrist fast movement of the arm up and down caused difficulties differentiating between a fall and a fast movement of the arm. We decided to move the device to be worn on the belt which works very well.

Another aspect of the original design that was problematic is that the app is built with React Native and Expo. Expo does not have a BLE (bluetooth low energy) API, but they are in the process of creating one. We do have a workaround this for now until Expo comes out with BLE API. The workaround plan is to have the bluetooth from the device synched with a computer. The data gets sent to the computer which will update the database, which then gets sent to the app. In the near future when Expo releases a proper bluetooth library, the bluetooth can be incorporated in it properly.

# Complete source code listing:

**React JS Code for the App:**

**app.js:**

import React from 'react';

import ApiKeys from './components/Auth/ApiKeys.js';

import \* as firebase from 'firebase';

import { Ionicons } from '@expo/vector-icons';

import { Platform, StatusBar, StyleSheet, Text, View } from 'react-native';

import { Container, Content, Header, Form, Input,

Item, Button, Label } from 'native-base';

import { AppLoading, Asset, Font } from 'expo';

import { Provider, connect } from 'react-redux';

import { createAppContainer, createStackNavigator } from 'react-navigation';

import LoginScreen from './components/Screens/Login';

import SignUpScreen from './components/Screens/SignUp';

import MainScreen from './components/Screens/Main';

import AlertScreen from './components/Screens/Alert';

import { store } from './reducers/index';

class Home extends React.Component {

static navigationOptions = {

title: 'LineLife',

};

render() {

return (

<Container style={styles.container}>

<Button style={{marginTop: 10}}

full

rounded

success

onPress={() => this.props.navigation.navigate('Login')}

>

<Text style={{color:'white'}}>Login</Text>

</Button>

<Button style={{marginTop: 10}}

full

rounded

success

onPress={() => this.props.navigation.navigate('SignUp')}

>

<Text style={{color:'white'}}>Sign Up</Text>

</Button>

</Container>

);

}

}

// Connect the screens to Redux

let HomeContainer = connect(state => ({ count: state.count }))(Home);

// Create our stack navigator

const RootStack = createStackNavigator({

Home: {screen: HomeContainer },

Login: {screen: LoginScreen },

SignUp: {screen: SignUpScreen },

Main: {screen: MainScreen },

Alert: {screen: AlertScreen},

});

// And the app container

const Navigation = createAppContainer(RootStack);

// Render the app container component with the provider around it

export default class App extends React.Component {

constructor(props) {

super(props);

this.state = {

isLoadingComplete: false,

isAuthenticationReady: false,

isAuthenticated: false,

};

// Initialize firebase...

if (!firebase.apps.length) { firebase.initializeApp(ApiKeys.firebaseConfig); }

firebase.auth().onAuthStateChanged(this.onAuthStateChanged);

}

onAuthStateChanged = (user) => {

this.setState({isAuthenticationReady: true});

this.setState({isAuthenticated: !!user});

}

render() {

return (

<Provider store={store}>

<Navigation />

</Provider>

);

}

}

const styles = StyleSheet.create({

container: {

flex: 1,

justifyContent: 'center',

alignItems: 'center',

backgroundColor: '#ecf0f1',

padding: 8,

},

paragraph: {

margin: 24,

fontSize: 18,

fontWeight: 'bold',

textAlign: 'center',

},

});

**-------------------------------------------------------------------------------------------------------------------------------**

**login.js:**

import React from 'react';

import { Text, TextInput, View, StyleSheet, Image,

Alert } from 'react-native';

import { Button, Container, Content } from 'native-base'

import \* as firebase from 'firebase';

import stil from '../../assets/styles/main.js'

import { connect } from 'react-redux';

export default class Login extends React.Component {

constructor(props) {

super(props);

this.state = {

email: '',

password: '',

name: '',

caretaker: '',

};

}

static navigationOptions = {

title: `Login`,

};

onLoginPress = () => {

firebase.auth().signInWithEmailAndPassword(this.state.email, this.state.password)

.then( async(user) => {

await firebase.database().ref(`users/${user.user.uid}/displayName`).once('value',

(snapshot) => { this.setState({ name: snapshot.val() }) });

await firebase.database().ref(`users/${user.user.uid}/careTaker`).once('value',

(snapshot) => { this.setState({ caretaker: snapshot.val() }) });

this.props.navigation.navigate('Main', { name : this.state.name , caretaker: this.state.caretaker} );

}, (error) => { Alert.alert(error.message); });

}

render() {

return (

<Container style={styles.container}>

<TextInput style={styles.input}

value={this.state.email}

onChangeText={(text) => { this.setState({email: text}) }}

placeholder="Email"

keyboardType="email-address"

autoCapitalize="none"

autoCorrect={false}

/>

<View style={{paddingTop:10}} />

<TextInput style={styles.input}

value={this.state.password}

onChangeText={(text) => { this.setState({password: text}) }}

placeholder="Password"

secureTextEntry={true}

autoCapitalize="none"

autoCorrect={false}

/>

<Content style={{width:100}}>

<Button style={{marginTop: 10}}

full

rounded

success

onPress={this.onLoginPress}

>

<Text style={{color:'white'}}>Login</Text>

</Button>

</Content>

</Container>

);

}

}

const styles = StyleSheet.create({

container: {

flex: 1,

justifyContent: 'center',

alignItems: 'center',

backgroundColor: '#ecf0f1',

padding: 50,

},

input: {

width: 200,

height: 40,

borderWidth: 1,

padding: 4,

}

});

**-------------------------------------------------------------------------------------------------------------------------------**

**Signup.js:**

import React from 'react';

import { Text, TextInput, View, StyleSheet, Image, Alert } from 'react-native';

import { Button, Container, Content } from 'native-base'

import \* as firebase from 'firebase';

import { connect } from 'react-redux';

export default class SignUp extends React.Component {

constructor(props) {

super(props);

this.state = {

name: '',

email: '',

password: '',

passwordConfirm: '',

phoneNumber: '',

careTaker:'',

};

}

static navigationOptions = {

title: `SignUp`,

};

onSignupPress = async() => {

if (this.state.password !== this.state.passwordConfirm) {

Alert.alert("Passwords do not match");

return;

}

var flag = true;

await firebase.database().ref().child('users').orderByChild("displayName").equalTo(this.state.careTaker).once('value', snapshot => {

if (!snapshot.exists()) {

Alert.alert("Enter A Valid and Recognized Caretaker");

flag = false;

}

});

if (flag){

await firebase.auth().createUserWithEmailAndPassword(this.state.email, this.state.password)

.then((res) => {

var uid = firebase.auth().currentUser.uid;

firebase.database().ref().child('users/' + uid).set({

displayName: this.state.name,

email: this.state.email,

Password: this.state.password,

PhoneNumber: this.state.phoneNumber,

careTaker: this.state.careTaker,

numberOfFalls: 0,

})

}, (error) => { Alert.alert(error.message); });

} else {

return;

}

firebase.auth().signInWithEmailAndPassword(this.state.email, this.state.password)

.then(() => { this.props.navigation.navigate('Main', { name : this.state.name }); }, (error) => { Alert.alert(error.message); });

}

render() {

return (

<Container style={styles.container}>

<TextInput style={styles.input}

value={this.state.name}

onChangeText={(text) => { this.setState({name: text}) }}

placeholder="Name"

autoCapitalize="none"

autoCorrect={false}

/>

<View style={{paddingTop:10}} />

<TextInput style={styles.input}

value={this.state.email}

onChangeText={(text) => { this.setState({email: text}) }}

placeholder="Email"

keyboardType="email-address"

autoCapitalize="none"

autoCorrect={false}

/>

<View style={{paddingTop:10}} />

<TextInput style={styles.input}

value={this.state.password}

onChangeText={(text) => { this.setState({password: text}) }}

placeholder="Password"

secureTextEntry={true}

autoCapitalize="none"

autoCorrect={false}

/>

<View style={{paddingTop:10}} />

<TextInput style={styles.input}

value={this.state.passwordConfirm}

onChangeText={(text) => { this.setState({passwordConfirm: text}) }}

placeholder="Password (confirm)"

secureTextEntry={true}

autoCapitalize="none"

autoCorrect={false}

/>

<View style={{paddingTop:10}} />

<TextInput style={styles.input}

value={this.state.phoneNumber}

onChangeText={(text) => { this.setState({phoneNumber: text}) }}

placeholder="Phone Number"

autoCapitalize="none"

autoCorrect={false}

/>

<View style={{paddingTop:10}} />

<TextInput style={styles.input}

value={this.state.careTaker}

onChangeText={(text) => { this.setState({careTaker: text}) }}

placeholder="Care Taker Name"

autoCapitalize="none"

autoCorrect={false}

/>

<View style={{paddingTop:10}} />

<Content style={{width:100}}>

<Button style={{marginTop: 10}}

full

rounded

success

onPress={this.onSignupPress}

>

<Text style={{color:'white'}}>Sign Up</Text>

</Button>

</Content>

</Container>

);

}

}

const styles = StyleSheet.create({

container: {

flex: 1,

justifyContent: 'center',

alignItems: 'center',

backgroundColor: '#ecf0f1',

padding: 50,

},

input: {

width: 200,

height: 40,

borderWidth: 1,

padding: 4,

}

});

**-------------------------------------------------------------------------------------------------------------------------------**

**main.js:**

**import** React from 'react';

**import** { Text, TextInput, View, StyleSheet, Image} from 'react-native';

**import** { Button, Container, Content } from 'native-base'

**import** \* as firebase from 'firebase';

**import** { connect } from 'react-redux';

**import** { setPersonType } from '../../reducers/index'

**import** { setCountdown } from '../../reducers/index'

**const** mapStateToProps = (state) => {

**return** {

countdown: state.countdown,

personType: state.personType,

};

}

**const** mapDispatchToProps = (dispatch) => {

**return** {

setCountdown: (text) => { dispatch(setCountdown(text))},

setPersonType: (text) => { dispatch(setPersonType(text))},

};

}

**class** Main **extends** React.Component {

constructor(props) {

**super**(props);

**this**.state = {

numFalls: "0",

}

**var** count = 0

**const** user = firebase.auth().currentUser.uid

firebase.database().ref(`users/${user}/numberOfFalls`).on('value', (snapshot) => {

**this**.setState({ numFalls: snapshot.val() })

**if** (count > 0){

**this**.props.navigation.navigate('Alert')

}

count++

});

}

**static** navigationOptions = ({navigation, navigationOptions }) => {

**const** { navigate } = navigation;

**return** {

title: navigation.getParam('name', 'Error') ,

headerLeft: **null**,

headerRight: (

<Content style={{right: 10}}>

<Button style={{padding:5}}

transparent

onPress={ () => {

firebase.auth().signOut().then(() => {

navigate('Home'); },

(error) => { Alert.alert(error.message);

});

}

}

>

<Text style={{color:'green'}}>Log Out</Text>

</Button>

</Content>

),

}

}

render() {

**return** (

<View style={styles.container}>

<Text>Number **of** Falls</Text>

<Text>{**this**.state.numFalls}</Text>

<Text>{**this**.props.personType}</Text>

<Text>{**this**.props.countdown}</Text>

</View>

);

}

}

**const** styles = StyleSheet.create({

container: {

alignItems: 'center',

justifyContent: 'center',

padding: 24,

},

});

**export** **default** connect(mapStateToProps, mapDispatchToProps)(Main);

**-------------------------------------------------------------------------------------------------------------------------------**

**Alert.js:**

import React from 'react';

import { Text, TextInput, View, StyleSheet, Image, AppRegistry,

Animated,

TouchableWithoutFeedback, Linking } from 'react-native';

import { Button, Container, Content } from 'native-base'

import \* as firebase from 'firebase';

var ACTION\_TIMER = 2000;

var COLORS = ['rgb(255,255,255)', 'rgb(111,235,62)'];

const url = `tel:${3049891098}`;

export default class Alert extends React.Component {

constructor(props) {

super(props);

this.state = {

pressAction: new Animated.Value(0),

textComplete: '',

buttonWidth: 0,

buttonHeight: 0,

timer : 15,

}

setInterval(() => {

if (this.state.timer == 0) {

Linking.canOpenURL(url)

.then((supported) => {

if (supported) {

return Linking.openURL(url)

.catch(() => null);

}

});

}

this.setState(previousState => (

{ timer: previousState.timer - 1 }

))

}, 1000);

}

componentWillMount = () =>{

this.\_value = 0;

this.state.pressAction.addListener((v) => this.\_value = v.value);

}

static navigationOptions = {

title: 'Alert',

};

getProgressStyles = () => {

var width = this.state.pressAction.interpolate({

inputRange: [0, 1],

outputRange: [0, this.state.buttonWidth]

});

var bgColor = this.state.pressAction.interpolate({

inputRange: [0, 1],

outputRange: COLORS

})

return {

width: width,

height: this.state.buttonHeight,

backgroundColor: bgColor

}

}

getButtonWidthLayout = (e) => {

this.setState({

buttonWidth: e.nativeEvent.layout.width - 6,

buttonHeight: e.nativeEvent.layout.height - 6

});

}

handlePressIn = () => {

Animated.timing(this.state.pressAction, {

duration: ACTION\_TIMER,

toValue: 1

}).start(this.animationActionComplete);

}

handlePressOut = () => {

Animated.timing(this.state.pressAction, {

duration: this.\_value \* ACTION\_TIMER,

toValue: 0

}).start();

}

animationActionComplete = () => {

var message = '';

if (this.\_value === 1) {

message = 'You held it long enough to fire the action!';

this.props.navigation.navigate('Main', { name : this.state.name , caretaker: this.state.caretaker} );

}

this.setState({

textComplete: message

});

}

render() {

return (

<View style={styles.container}>

<View>

<Text>ALERT: FALL DETECTED!</Text>

</View>

<View>

<Text>Countdown to EMS: {this.state.timer}</Text>

</View>

<TouchableWithoutFeedback

onPressIn={this.handlePressIn}

onPressOut={this.handlePressOut}

>

<View style={styles.button} onLayout={this.getButtonWidthLayout} >

<Animated.View style={[styles.bgFill, this.getProgressStyles()]} />

<Text style={styles.text}>HOLD ME IF YOU ARE OKAY!</Text>

</View>

</TouchableWithoutFeedback>

</View>

);

}

}

var styles = StyleSheet.create({

container: {

flex: 1,

flexDirection: 'column',

alignItems: 'center',

justifyContent: 'center'

},

button: {

padding: 20,

borderWidth: 4,

height: 100,

borderColor: '#111'

},

text: {

backgroundColor: 'transparent',

color: '#111'

},

bgFill: {

position: 'absolute',

top: 0,

left: 0

}

});

**-------------------------------------------------------------------------------------------------------------**

**Arduino code for device:**

//This Code is a modified version of Jeff Rowberg's open source MPU6050 implementation

//His code is used to set up the MPU6050 and read data from the accelerometer/gyroscope

//Our code is the algorithm that is used to detect the falls via the data

/\* ============================================

I2Cdev device library code is placed under the MIT license

Copyright (c) 2012 Jeff Rowberg

Permission is hereby granted, free of charge, to any person obtaining a copy

of this software and associated documentation files (the "Software"), to deal

in the Software without restriction, including without limitation the rights

to use, copy, modify, merge, publish, distribute, sublicense, and/or sell

copies of the Software, and to permit persons to whom the Software is

furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in

all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR

IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,

FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE

AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER

LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,

OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN

THE SOFTWARE.

===============================================

\*/

#ifndef \_BV

#define \_BV(n) (1<<(n))

#endif

// I2Cdev and MPU6050 must be installed as libraries, or else the .cpp/.h files

// for both classes must be in the include path of your project

#include "I2Cdev.h"

#include <unistd.h>

#include "MPU6050\_6Axis\_MotionApps20.h"

//#include "MPU6050.h" // not necessary if using MotionApps include file

// Arduino Wire library is required if I2Cdev I2CDEV\_ARDUINO\_WIRE implementation

// is used in I2Cdev.h

#if I2CDEV\_IMPLEMENTATION == I2CDEV\_ARDUINO\_WIRE

#include "Wire.h"

#endif

// class default I2C address is 0x68

// specific I2C addresses may be passed as a parameter here

// AD0 low = 0x68 (default for SparkFun breakout and InvenSense evaluation board)

// AD0 high = 0x69

MPU6050 mpu;

//MPU6050 mpu(0x69); // <-- use for AD0 high

#define INTERRUPT\_PIN 11 // use pin 2 on Arduino Uno & most boards

#define LED\_PIN 7 // (Arduino is 13, Teensy is 11, Teensy++ is 6)

#include <SPI.h>

#include <BLEPeripheral.h>

bool blinkState = false;

// MPU control/status vars

bool dmpReady = false; // set true if DMP init was successful

uint8\_t mpuIntStatus; // holds actual interrupt status byte from MPU

uint8\_t devStatus; // return status after each device operation (0 = success, !0 = error)

uint16\_t packetSize; // expected DMP packet size (default is 42 bytes)

uint16\_t fifoCount; // count of all bytes currently in FIFO

uint8\_t fifoBuffer[64]; // FIFO storage buffer

// orientation/motion vars

Quaternion q; // [w, x, y, z] quaternion container

VectorInt16 aa; // [x, y, z] accel sensor measurements

VectorInt16 aaReal; // [x, y, z] gravity-free accel sensor measurements

VectorInt16 aaWorld; // [x, y, z] world-frame accel sensor measurements

VectorFloat gravity; // [x, y, z] gravity vector

float euler[3]; // [psi, theta, phi] Euler angle container

float ypr[3]; // [yaw, pitch, roll] yaw/pitch/roll container and gravity vector

//Our variables

int minNum = 100000;

int maxNum = -100000;

int maxChange = 0;

int count = 0;

int maxCount = 0;

int falldetected = 0;

int fallCount = 0;

int timer = 0;

//For BLE

const char \* localName = "FallDetectionLL";

BLEPeripheral blePeriph;

BLEService fallDetect = BLEService("2ce79cb8-0bfd-483f-bbef-d6562ce03304");

BLEUnsignedCharCharacteristic fallAlert = BLEUnsignedCharCharacteristic("2ce79cb9-0bfd-483f-bbef-d6562ce03304", BLERead | BLEWrite | BLENotify);

// ================================================================

// === INTERRUPT DETECTION ROUTINE ===

// ================================================================

volatile bool mpuInterrupt = false; // indicates whether MPU interrupt pin has gone high

void dmpDataReady() {

mpuInterrupt = true;

}

// ================================================================

// === INITIAL SETUP ===

// ================================================================

void setup() {

// join I2C bus (I2Cdev library doesn't do this automatically)

#if I2CDEV\_IMPLEMENTATION == I2CDEV\_ARDUINO\_WIRE

Wire.begin();

Wire.setClock(400000); // 400kHz I2C clock. Comment this line if having compilation difficulties

#elif I2CDEV\_IMPLEMENTATION == I2CDEV\_BUILTIN\_FASTWIRE

Fastwire::setup(400, true);

#endif

// initialize serial communication

// (115200 chosen because it is required for Teapot Demo output, but it's

// really up to you depending on your project)

Serial.begin(115200);

// uncomment these out to utilize BLE

// blePeriph.setDeviceName(localName);

// blePeriph.setLocalName(localName);

// blePeriph.setAdvertisedServiceUuid(fallDetect.uuid());

// blePeriph.addAttribute(fallDetect);

// blePeriph.addAttribute(fallAlert);

// fallAlert.setValue(0);

// blePeriph.begin();

while (!Serial); // wait for Leonardo enumeration, others continue immediately

// initialize device

Serial.println(F("Initializing I2C devices..."));

mpu.initialize();

pinMode(INTERRUPT\_PIN, INPUT);

// verify connection

Serial.println(F("Testing device connections..."));

Serial.println(mpu.testConnection() ? F("MPU6050 connection successful") : F("MPU6050 connection failed"));

// load and configure the DMP

Serial.println(F("Initializing DMP..."));

devStatus = mpu.dmpInitialize();

// supply your own gyro offsets here, scaled for min sensitivity

mpu.setXGyroOffset(220);

mpu.setYGyroOffset(76);

mpu.setZGyroOffset(-85);

mpu.setZAccelOffset(1788); // 1688 factory default for my test chip

// make sure it worked (returns 0 if so)

if (devStatus == 0) {

// turn on the DMP, now that it's ready

Serial.println(F("Enabling DMP..."));

mpu.setDMPEnabled(true);

// enable Arduino interrupt detection

Serial.print(F("Enabling interrupt detection (Arduino external interrupt "));

Serial.print(digitalPinToInterrupt(INTERRUPT\_PIN));

Serial.println(F(")..."));

attachInterrupt(digitalPinToInterrupt(INTERRUPT\_PIN), dmpDataReady, RISING);

mpuIntStatus = mpu.getIntStatus();

// set our DMP Ready flag so the main loop() function knows it's okay to use it

Serial.println(F("DMP ready! Waiting for first interrupt..."));

dmpReady = true;

// get expected DMP packet size for later comparison

packetSize = mpu.dmpGetFIFOPacketSize();

} else {

// ERROR!

// 1 = initial memory load failed

// 2 = DMP configuration updates failed

// (if it's going to break, usually the code will be 1)

Serial.print(F("DMP Initialization failed (code "));

Serial.print(devStatus);

Serial.println(F(")"));

}

// configure LED for output

pinMode(LED\_PIN, OUTPUT);

}

// ================================================================

// === MAIN PROGRAM LOOP ===

// ================================================================

void loop() {

// blePeriph.poll();

// if programming failed, don't try to do anything

if (!dmpReady) return;

// wait for MPU interrupt or extra packet(s) available

while (!mpuInterrupt && fifoCount < packetSize) {

if (mpuInterrupt && fifoCount < packetSize) {

// try to get out of the infinite loop

fifoCount = mpu.getFIFOCount();

}

}

// reset interrupt flag and get INT\_STATUS byte

mpuInterrupt = false;

mpuIntStatus = mpu.getIntStatus();

// get current FIFO count

fifoCount = mpu.getFIFOCount();

// check for overflow (this should never happen unless our code is too inefficient)

if ((mpuIntStatus & \_BV(MPU6050\_INTERRUPT\_FIFO\_OFLOW\_BIT)) || fifoCount >= 1024) {

// reset so we can continue cleanly

mpu.resetFIFO();

fifoCount = mpu.getFIFOCount();

Serial.println(F("FIFO overflow!"));

// otherwise, check for DMP data ready interrupt (this should happen frequently)

} else if (mpuIntStatus & \_BV(MPU6050\_INTERRUPT\_DMP\_INT\_BIT)) {

// wait for correct available data length, should be a VERY short wait

while (fifoCount < packetSize) fifoCount = mpu.getFIFOCount();

// read a packet from FIFO

mpu.getFIFOBytes(fifoBuffer, packetSize);

// track FIFO count here in case there is > 1 packet available

// (this lets us immediately read more without waiting for an interrupt)

fifoCount -= packetSize;

// display initial world-frame acceleration, adjusted to remove gravity

// and rotated based on known orientation from quaternion

mpu.dmpGetQuaternion(&q, fifoBuffer);

mpu.dmpGetAccel(&aa, fifoBuffer);

mpu.dmpGetGravity(&gravity, &q);

mpu.dmpGetLinearAccel(&aaReal, &aa, &gravity);

mpu.dmpGetLinearAccelInWorld(&aaWorld, &aaReal, &q);

//BEGINNING OF FALL DETECTION ALGORITHM

count++;

if (aaWorld.z < minNum) minNum = aaWorld.z;

if (aaWorld.z > maxNum) maxNum = aaWorld.z;

if(count%25 == 0){

maxChange = maxNum-minNum;

Serial.print(maxNum);

Serial.print(" - ");

Serial.print(minNum);

Serial.print(" = ");

Serial.println(maxChange);

minNum = 100000;

maxNum = -100000;

if (maxChange > 10000){

maxCount++;

if (falldetected > 0 || timer > 0)

{

fallCount++;

}

}

else if ((maxCount == 1 || maxCount == 2) && maxChange > 4000 && (timer == 0 || timer == 1)){

falldetected++;

}

else if (falldetected < 5 && fallCount < 2 && maxCount < 3 && timer > 2){

Serial.println("FALL DETECTED");

fallAlert.setValue(1);

falldetected = 0;

maxCount = 0;

fallCount = 0;

timer = 0;

}

else{

if(maxCount > 0){

timer++;

}

if (timer > 5){

maxCount = 0;

fallCount = 0;

falldetected = 0;

timer = 0;

}

}

}

}

// blink LED to indicate activity

blinkState = !blinkState;

digitalWrite(LED\_PIN, blinkState);

}

Test Results:

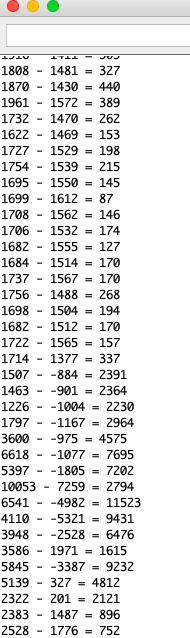
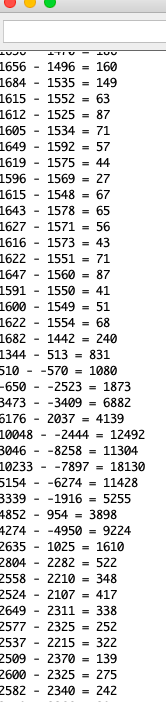
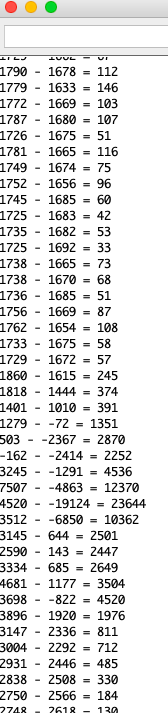
In order to test our project, the fall detection device got connected to the application via a cord. The code was tested over and over using the cord. Some challenges were getting the data to plot on the graph in real time, then taking the graph and getting exact points. So we switched from plotting the data on the graph to having actual points display. Then Bluetooth from the wearable device to a generic bluetooth app (not our fall detection app) on the phone was implemented into the code.

Then once again one of the group members had to wear the device. After wearing the device this member had to fall intentionally time after time and check if the application got notification messages or not for each fall. It only took less than five seconds to get the notification message on the application. A bug that appeared once bluetooth was added was the device sometimes had to be reprogramed randomly due to the bluetooth. Another bug that we discovered was when the user jumped only one time, the device would consider the user falling, but if the user jumped twice or more the device realized that the user is jumping and not falling. The test results from the data are shown below for falling, running and jumping. You are able to tell where the jump, run or fall has changed that data shown. The following data are the raw values displayed via the serial monitor in the Arduino IDE.

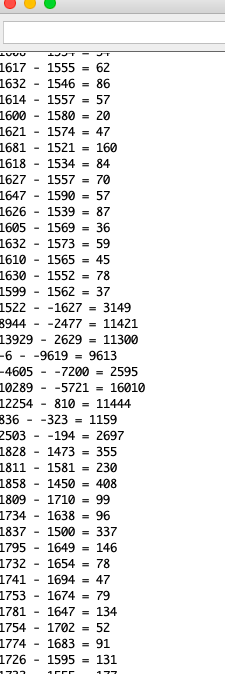
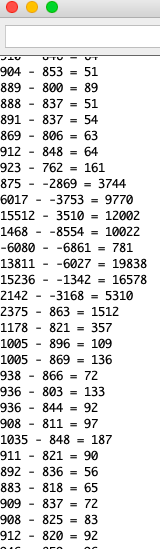
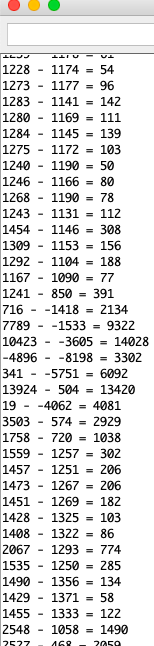
To understand the numbers, each number = ((VerticalAcceleration)\*(9.8m/s^2)/16384).

What is seen from left to right is the max Acceleration minus the minimum Acceleration equals the change in Acceleration. This data was taken every 4th of a second. To determine the type of action (fall, jumping, etc.), patterns in the change of Acceleration (the far right number in each row) were observed and ascribed to each type of action.

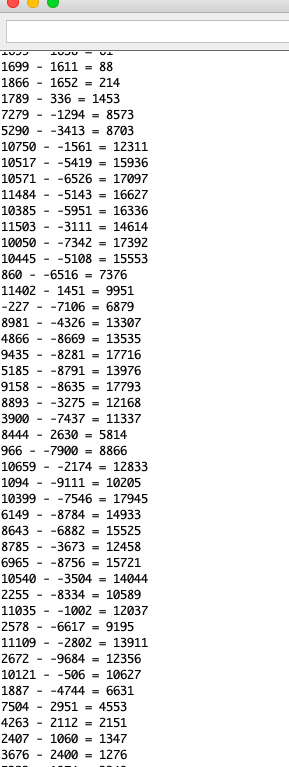
**The Data for 3 Separate Falls**



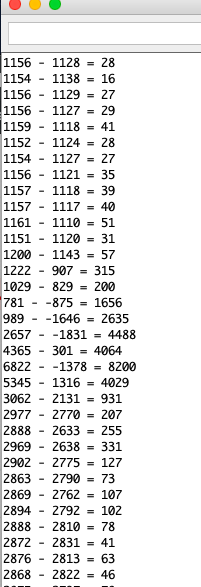
**Data for 3 Separate Singular Jumps**



**Running**



**Sitting Down**



# Safety Precautions:

Some safety precautions that should be taken are make sure that the battier does not get too hot. Also if the plastic housing gets crushed then the plastic pieces might cause bodily harm to the user.

# Reflections:

# 

Problems that the team came across included: the bluetooth capability from device to the app directly was an issue due to Expo not having a BLE library, the device was too large to be worn as a watch, the device can count a single jump as a fall, and we were not be able to use the heart rate monitor. One might be able to use a different program to create the app other than Expo that has a bluetooth feature or if the time is right and Expo has released the BLE API it is possible to code the bluetooth. The device could be rearranged and possibly made smaller so it can be worn on the wrist. If the device is able to be worn on the wrist the heart rate monitor could be implemented into the device. The housing component could have a place where the heart rate monitor is able to make contact with the skin.

# **Appendix 1**

# User Manual:

\*The main challenge for future users is to get the bluetooth working directly between the app and the device within React Native Expo. Expo is currently working on a bluetooth library: <https://expo.canny.io/feature-requests/p/bluetooth-api>

It has not been finished yet. For now, an intermediary computer is needed\*

To run and edit the app, react native expo must be installed in the user’s computer and phone.

Download the Expo app from the app store on your phone.

On the computer, type :

npm install -g expo-cli

within the terminal/command prompt. (Must have npm installed too, refer to google).

Next, navigate to the directory of the project folder in terminal/command prompt (LineLife).

Once within that directory type, “expo start” in the command prompt.

This should then present you with a bar code. Scan that bar code with your phone to open up the app on your phone.

Sign in the app with: [test1@gmail.com](mailto:test1@gmail.com)

Password is “password”

Now the app is ready to go.

Next the device needs to be hooked up.

The device is charged with a microUSB.

Charge the device for 10 min at a time. Too much can heat it up and mess with the gyroscope/accelerometer.

Once charged for 10 min, remove the device. The lights should be on. The device is now ready.

Next step is synching the Bluetooth. (This is for the intermediary Bluetooth. If the future group can figure out how to directly get it installed on Expo, then these next few steps are irrelevant).

Python needs to be installed on the computer as well.

In a new terminal (or IDE), run the python code “FallDetectionData.py” (for testing purposes, this code has it so only [test1@gmail.com](mailto:test1@gmail.com) will be updated when falling. Make sure that is signed on in the app. This can be changed. The way it works is that our app is connected to Firebase. The python code updates Firebase when a fall is detected. The app receives this update and sends the alert. In the future, hopefully

This code will hook up to the fall device’s bluetooth.

If the code successfully runs everything is ready to go.

Wear the device around a belt.

Fall.

The app should respond.

\*For future testing purposes, notice the white wires connected on the device. Take note of where the hook up. They are for the classic Bluetooth. They will need to be removed if a future user plans to modify the program on the device. To do that a microUSB is needed to connect with this device….



…which is included. Also that device needs to be hooked up where the white bluetooth wires originally were. Visit <https://learn.sparkfun.com/tutorials/nrf52832-breakout-board-hookup-guide?_ga=2.153648120.1735792845.1556732629-1515171873.1547500032> for detailed hook up instructions.

**Appendix 2**

# Maintenance:

Update the device location.

If the device has been landed on then check to make sure all chips are not snapped or broken.

Make sure that there is no damage to the device by the stimulating a fall every month to make sure that the device is still working properly.

Recharge the battery every night when the device is taken off when the user goes to bed.