Fall Detection For The Elderly

A device for detection falls and notifying careers and loved ones in emergencies

Tomasz Klebek

BEng (Hons) Computer and Electronic Engineering
Galway-Mayo Institute of Technology

2019/2020

Poster

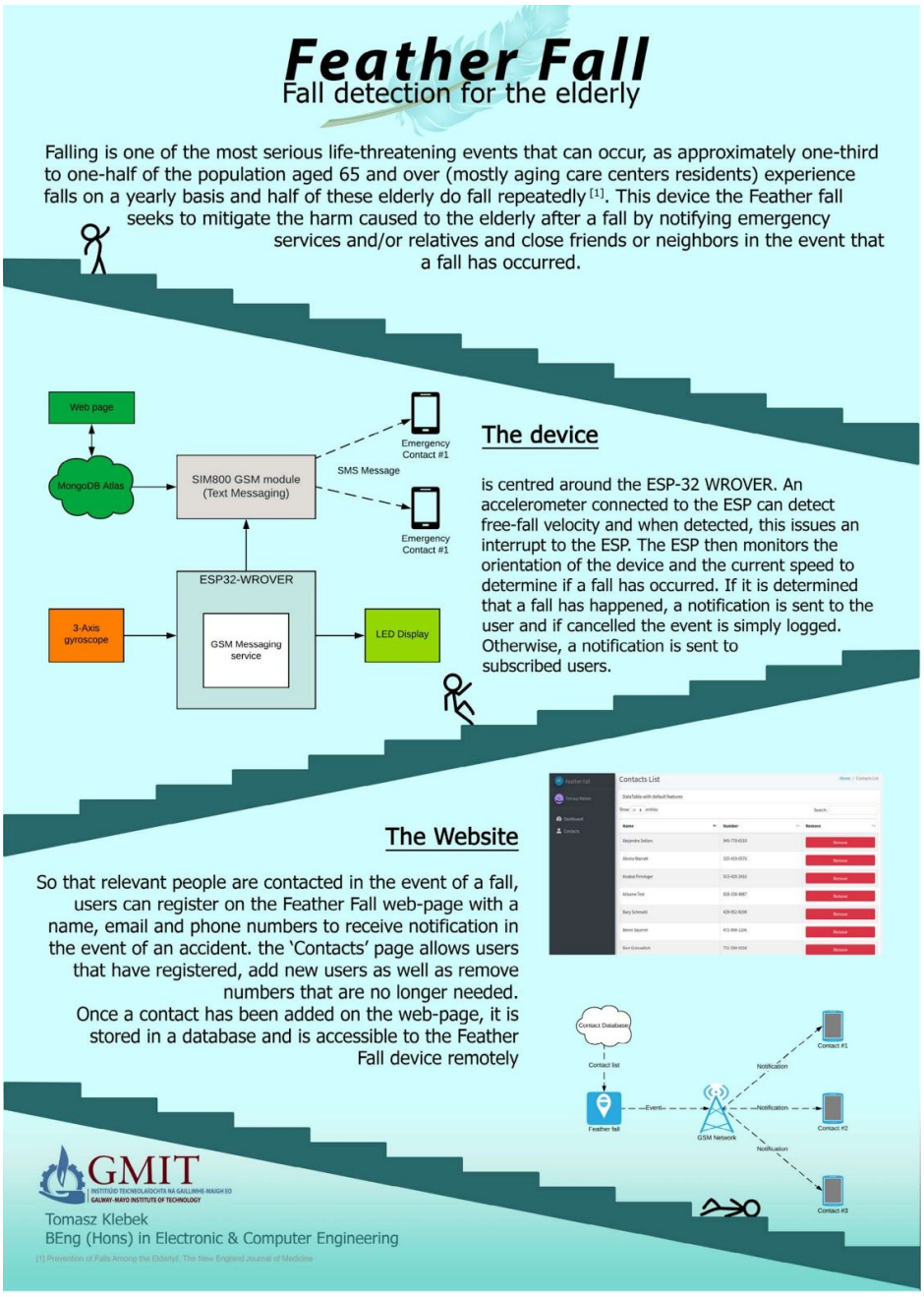

Figure 1 Project Poster

Declaration

This project is presented in partial fulfilment of the requirements for the degree of Bachelor of Engineering in Computer & Electronic Engineering at Galway-Mayo Institute of Technology.

This project is my own work, except where otherwise accredited. Where the work of others has been used or incorporated during this project, this is acknowledged and referenced.

_____

Acknowledgements

Thank you to all of my lectures for their guidance and support. Thank you to my family for their support and understanding.

Table of Contents

1. Summary

The goals of this project are to design and build a system that will automatically contact the assigned person when the device has been triggered to detect a fall. The customer can use a website that is designed to be accessible and easy to use. The customer is able to add as many contacts as they wish, through an easy to use web page, created using HTML, Node.js and MongoDB.

The device I have designed is meant to be both portable and reliable, so I opted to use ESP32, GSM Module and an accelerometer as my main components. The ESP32 is the heart of the project, interpreting inputs from the accelerometer and handling communication to and from the GSM Module.

Many aspects of the project were met, including the GSM communication, allowing multiple users to receive text messages simultaneously. As well as the easy to use web page, for storing contact information. One of the areas where the project fell short was in the accelerometer and I2C Communication. Although the device did not meet all the intended outcomes, I am still happy with the final project. In this project, I have explored different concepts of web development and have discovered many resources that will support me in my future endeavours.

2. Introduction

By 2050, it's estimated that more than one in each group of five people will be aged 65 or over[1]. In this age group, falling is one of the most serious life-threatening events that can occur, as approximately one-third to one-half of the population aged 65 and over (mostly ageing care centres residents) experience falls on a yearly basis and half of these elderly do fall repeatedly.[2] That is one of the main reasons I have decided to develop a fall detection device, my aim is to have a device that is able to detect if a patient has fallen and notified emergency services and/or relatives and close friends or neighbours. Timely detection of falls enables immediate assistance by the caregiver and minimizes the negative consequences of falls [3].

In 2018, I went home to South Africa and during my visit, my 96-year-old Granny suffered a major fall and this went unnoticed for two hours. My Granny couldn't move, was unable to get help and just laid there. If we had a device like this, we could have been notified immediately and the damage to her would have been greatly reduced. Falls take both a physical and mental toll on the elderly and this device will hopefully give others peace of mind and assurance. That is why I want to implement a reliable system that can be used both in the home and outdoors.

In this report, I will cover:

- Considerations made and research done during the initial planning.
- The technologies used to implement GSM communication using ESP 32.
- Web frameworks used in development on the web page.
- Github and Jenkins Version control and deployment.
- Challenges encountered during this project.
- Solutions used to tackle issues.

3. Research and Development

3.1 Overview of fall technologies

By 2050, 1 in 6 people in the world will be over the age of 65, up from 1 in 11 in 2019.[1]

Almost all countries in the world are experiencing an increase in their population of ageing residents[1]. Falls are an increasing cause of deaths amongst the ageing community, and it is estimated in the US 62 deaths in every 100,000 among the elderly can be attributed to a fall [1], [4].

“Older adults’ perceptions of technologies aimed at falls prevention, detection or monitoring: A systematic review”[5] takes a look at how elderly people view fall detection technologies. Fall detection technologies are something that is viewed as an assistant in allowing elderly people to enjoy their independence and allows them a feeling of security, as well as giving loved one’s peace of mind.

Elderly people living in care homes are three times more likely to suffer a fall than elderly people living in their own homes, due to unfamiliar surroundings or taking multiple medications, and can in many cases be more detrimental to their health[6].

The people that would benefit most from a device like this would be the elderly people themselves, that want the peace of mind of knowing that there are people that will be contacted in the case of an emergency, Loved ones and family members that want to make sure their older family members are protected and cared, and care home that hold the safety and wellbeing of their clients to be most important.

3.1.1 Products currently on the market

There are currently many options for fall detection devices on the market, products that are able to tell if someone has had a fall and send an emergency notification to either a monitoring service or to subscribed individuals. Some of these devices are

- Sentry In-Home Medical Alert by BlueStar SeniorTech
- GeatCall Lively
- Fall Detection Pendant - Smartzone

Although some of these devices can be very reliable, they suffer from issues in other areas, notably, the sentry in-home medical alert requires the availability of landline and has only a limited range from the base station[7]. Other devices like the GeatCall Lively do not need to be connected to a landline but do need to be connected to a Bluetooth enabled phone running a proprietary app[8], which lends itself to issues for elderly people that are not particularly tech-savvy. Yet another issue with existing devices is while they provide brilliant fall detection and emergency notification they come with a hefty price tag in the way of expensive monitoring services.

One of the closest devices on the market to what I'm trying to achieve is the Vodafone V-SOS Band. The V-SOS Band is a wrist-mounted device that issues alerts through a proprietary app linked to the device. It uses GSM communication and GPS to send registered users alerts when a fall is detected and has a battery life of approximately 1 month[9].

3.2 Hardware selection

Initial hardware consideration was for a device that would be able to send GSM communication, GPS coordinates, monitor heart rate and connect with Bluetooth

devices. I will cover each of the devices I had originally considered and explain the reasons I decided to not go with each of those options.

3.2.1 ATMEGA324PB-XPRO

The first development board I considered was the ATMEGA324PB-XPRO which is an 8-bit AVR microcontroller. The features most relevant to my project available in this microcontroller are listed below.

- Peripheral Features
  - Peripheral Touch Controller (PTC)
    - Capacitive Touch Buttons, Sliders and Wheels
    - 32 Self-Sap Channels and 256 Mutual Cap Channels
  - Two 8-bit Timer/Counters with Separate Prescaler and Compare Mode
  - Three 16-bit Timer/Counters with Separate Prescaler, Compare Mode, and Capture Mode
  - Real Time Counter with Separate Oscillator
  - Ten PWM Channels
  - 8-Channel 10-Bit ADC
    - Differential Mode with Selectable Gain at 1×, 10× or 200×
  - Three Programmable Serial USARTs
  - Two Master/Slave SPI Serial Interfaces
  - Two Byte-oriented 2-wire Serial Interfaces (Philips I^2C Compatible)
  - Programmable Watchdog Timer with Separate On-chip Oscillator
  - On-chip Analog Comparator
  - Interrupt and Wake-up on Pin Change
- Operating Voltage:
  - 1.8 - 5.5V
- Power Consumption at 1MHz, 1.8V, 25°C
  - Active Mode: 0.24mA
  - Power-Down Mode: 0.2µA
  - Power-Save Mode: 1.3µA (Including 32kHz RTC)

One of the key features I was looking for was a microcontroller that could support multiple UART devices, for example, GSM and GPS. The ATMEGA324PB met this requirement by having three separate hardware UARTS available on board. The 324PB was also very power efficient and with a large operating voltage range from 1.8V - 5.5 it would be easy to power with rechargeable battery or a lithium-ion cell. The low energy consumption of 0.24mA in normal operation and only 0.2uA in power saving mode meant I would be able to run the device for long periods of time without it having to be recharged, exactly what I wanted to do for my fall detection device. The ATMEGA324PB-XPRO development board also comes with an onboard debugger allowing me to run, upload and debug code without having to purchase an expensive programming tool such as the ATATMEL-ICE which comes in at around 130 euro.

Although the 324PB met a lot of the specification that I was looking for there were downsides that, in the end, led me to use another device altogether. Firstly the development environment designed for use with this microcontroller was Atmel Studios, this IDE while being a powerful tool had drawbacks such as it did not support Linux based operating systems and the documentation for this particular microcontroller was seriously lacking. The Advanced Software Framework (ASF) provided by Atmel studio was lacking friendly user interaction especially for some as unfamiliar with the suite as me. After trying to write SPI drivers for the 324PB and failing, as well as trying to use the libraries available through Atmel studios. Although Microchip says they plan to support both Atmel Studio 7 and MPLAB X for the foreseeable future[10] it definitely felt that Atmel studio and older AVR microcontrollers had been left behind. Due to the incompatibility, I chose to use a different device.

3.2.2 PARTICLE Electron

The particle electron was another good possibility for my project, as it had many of the key features I was looking for. Based around an STM32F205RGT6 ARM Cortex M3 microcontroller with an operating voltage of between 3.9V-12VDC, it was a bit more

robust in terms of input voltage and could be easily powered from a 3.7v LiPo battery. Having the necessary UART connection was still important and the particle was capable of connecting three hardware UARTS. However the biggest issue with the particle was availability and at the time I was ordering components I would have had to wait up to 6 months for new ones to arrive at suppliers, so that was not an option. Another issue is that the GSM module that was fitted to the device was not easily accessible to use, I was hoping to use it to send text messages but the device was intended for over the air updates and not text messaging. Rather than causing myself more issues with this, I decided to abandon the particular in favour of the ESP32.

3.2.3 ESP32

ESP32 is a single 2.4 GHz Wi-Fi-and-Bluetooth combo chip. The ESP32 is a considerably more powerful device than the two mentioned before with a single or dual-core 32-bit LX6 microprocessor(s). Performance-wise it would be more than capable of handling what I was looking to use it for. The ESP32 is very popular at the moment for hobbies and enthusiasts which means that there is a lot of support for all aspects of the ESP32. The ESP32 comes in many flavours and there is a wide variety of development boards to choose from, the board I chose to go with was the ESP32-LilyGo-T-Call development board with a SIM800L GSM module. I chose this development board because it would allow me to implement GSM communication without the need to wire up a separate GSM module. The ESP32 has 5 power modes, which are:

- Active
- Modem Sleep
- Light Sleep
- Deep Sleep
- Hibernation

- When inactive mode the ESP32 draws in around 160-250mA and in hibernation, as little as 2.5uA. Power consumption was something that was key during my investigation of different development boards, although the different power modes are not something that I will necessarily insist on implementing by the end of this project. The development board that I choose to work with is the ESP32-LilyGo-T-Call which comes with a SIM800L GSM/GPRS module, that is fairly common and low cost. Again this module is quite popular amongst hobbyists and enthusiasts so there is a lot of support that can be found online for this device.

3.3 Software selection

In this section, I want to briefly cover the software that is used in my project, for example, the web frameworks and the firmware used on the ESP32. There were many web frameworks that I considered using, including React, View and

3.3.1 Website

The website I have developed stores contact information of users and saves it into a database. The website uses Node.Js, Express and a MongoDB database hosted by MongoDB Atlas

Node.js is a cross-platform Javascript runtime environment that allows me to write javascript code that can be run on the server-side and on the client-side. In conjunction with Express, it is used in writing the back end API.

Express is a Web Framework used with Node.js to build web API's

Another technology crucial to the web application I am building is Mongoose, Mongoose is an Object Data Modeling (ODM) library for MongoDB and Node.js. It manages relationships between data, provides schema validation and is used to

translate between objects in code and the representation of those objects in MongoDB.

3.3.2 ESP32 MicroPython

MicroPython is a version of python developed for use with microcontrollers and microprocessors. For the ESP32-LilyGo-T-Call I used the version of MicroPython recommended on their product page Loboris MicroPython. The reason I chose to use MicroPython on the ESP32 was that I wanted to use something that I hadn't used before and involve myself with a new language that I had little experience with. As well as learning something new, the MicroPython firmware allowed me to use REPL. REPL stands for Reading Evaluate Print Loop. It's the name given to the interactive command prompt that can be accessed on the ESP32 using MicroPython firmware. REPL allowed me to run and test code on the ESP32 without the need to compile and upload code on each iteration. This saves time and reduces the chance of adding errors into existing code.

4. How it works

4.1 Architecture

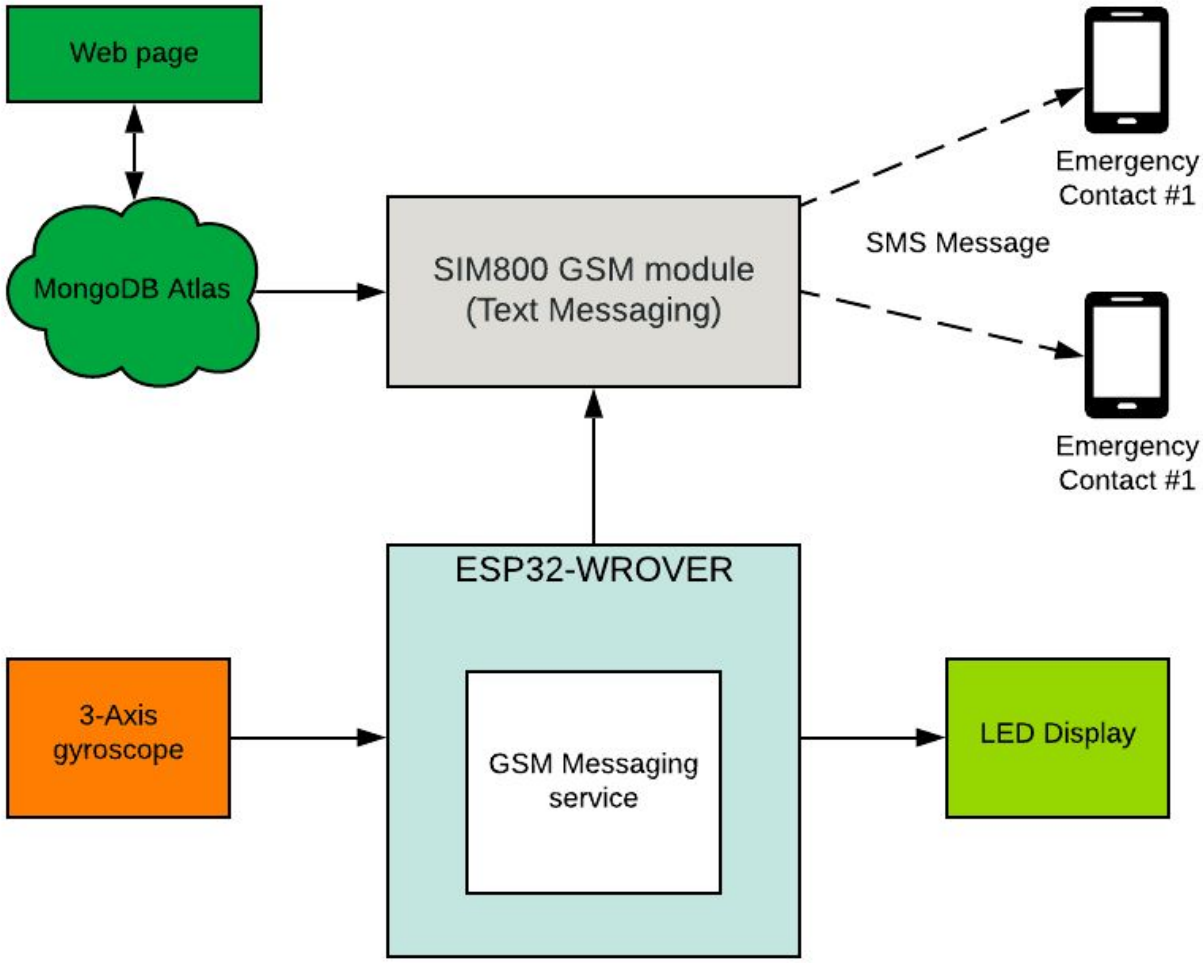

Figure 4.1 Project Architecture

4.2 Web page

The Website backend is built using Node.js and Express. The front end is displayed in HTML using Bootstrap and handlebar templates for displaying JSON data returned from the back end API. The whole site is hosted on DigitalOcean using an Ubuntu 18.04 virtual machine. I will demonstrate the functionality of the site and then explain the technologies behind how it functions. The layout Template used for this site is based on AdminLTE, a free Bootstrap Template.

I have a domain registered at 'wiredzenith.tech' that forwards to the digital ocean IP where the website is hosted.

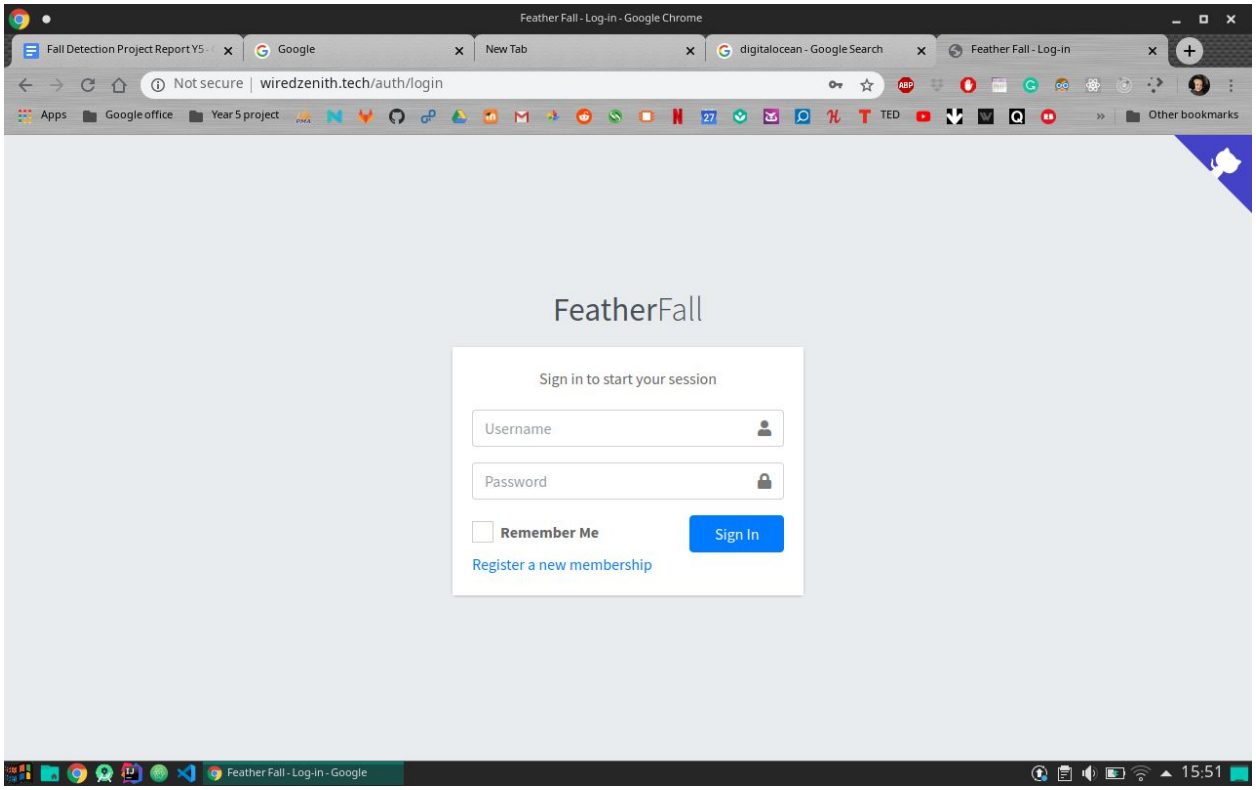

Figure 4.2 wiredzenith.tech login page

The landing page for the website is a login page where registered users can log in. If a user does not have an account they can register for a new account using the "Register a new Membership" on the sign-in page.

Users are then redirected to a registration form where they can register their new account.

The registration form has validation criteria and if any of the fields are left blank and error is returned from the validation function and displayed on the forum.

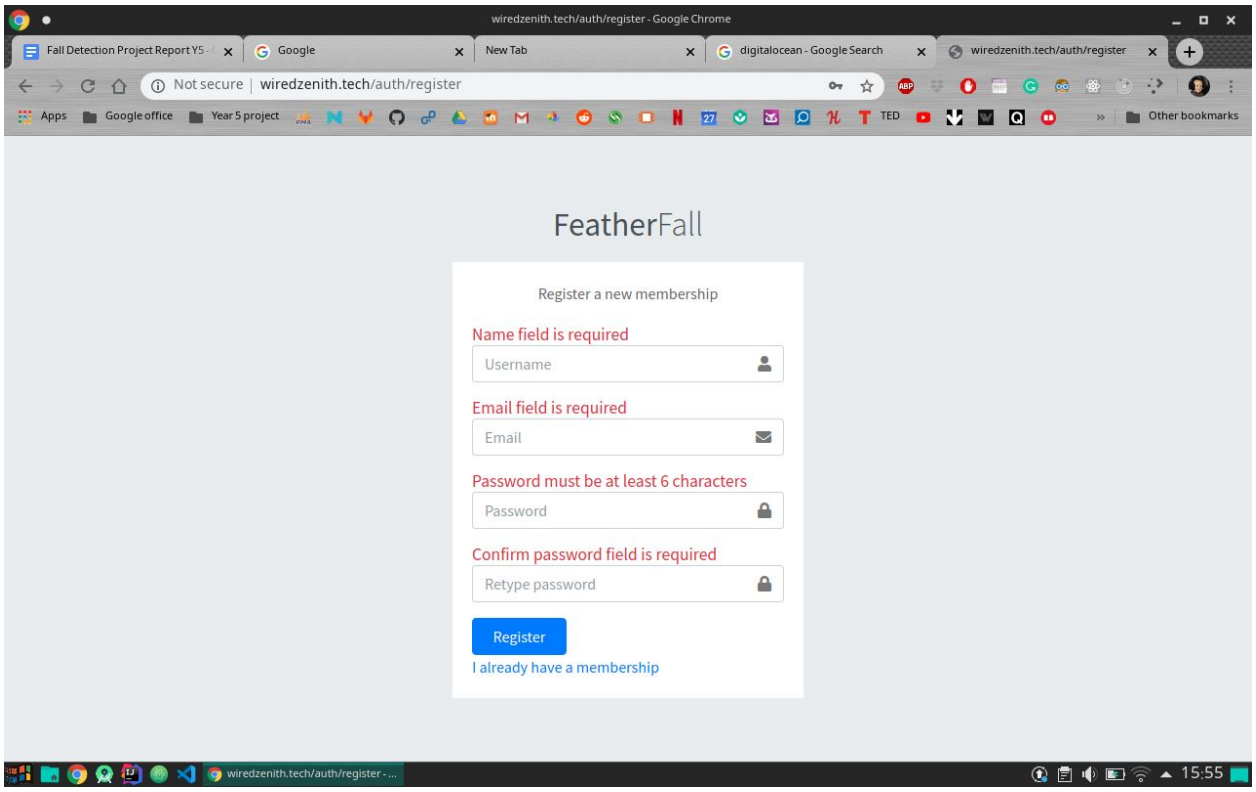

Figure 4.3 wiredzenith.tech user registration page

Side note:

The validation function does not check if the user already exists in the database, adding this would be a good idea to prevent multiple users with the same name and different passwords and errors when the multiple users are called back during login.

```
// check if email was entered
        if (validator.isEmpty(data.email)) {
            errors.email = "Email field is required";
            //check if email is valid
        } else if (!validator.isEmail(data.email)) {
            errors.email = "Email is invalid";
        }
```

The validator checks if any of the fields are empty, and in the case of the email if it is a valid email address. If a field is seen as not valid, a test string containing the error is added to the object. Once all the fields have been validated the function returns either true if the errors object is empty or the errors object itself.

In the case the validation function returns a true result the password is hashed and a new account is created. For example if the user's password was ‘iLoveJavascript’

the resulting hashed password would look something like this:

67ed0745b0a6575971d50b2f594bd55b

```
   if (validation.isValid) {
        bcrypt.genSalt(saltRounds, function (err, salt) {
            bcrypt.hash(req.body.password, salt, function (err, hash) {
                var newAccount = {
                    username: req.body.username,
                    email: req.body.email,
                    password: hash
                }
                const newAccountEntry = new Account(newAccount);
                newAccountEntry.save();
            });
        });
        console.log('account created!');

        res.redirect('/auth/login');
    }
```

In the case that the password didn't match, the email address was not valid or any of the fields were left blank, the object of errors is returned and displayed on the registration form.

When a user logs in, their details are called back from the database by searching the account name. If the user can't be found an error is returned and displayed on the forum. If an account matching the username is found, the password string is checked against the hashed password using a node plugin called 'bcrypt'(also used for hashing the password on registration).

```
if (bcrypt.compareSync(password, user.password)) {
                return done(null, username)
            } else {
                return done(null, false, { message: 'Password incorrect!' })
                  }
```

When the user is logged in, their session needs to be tracked to ensure that only logged-in users are able to access protected parts of the site. A node package called 'passport' is responsible for session tracking. When the user is logged in they are serialised allowing the passport strategy to tell that they are a verified login and allowing them to navigate protected pages. Once logged in users are redirected to the dashboard, this page only serves to show that users have logged in, displaying in the introduction and showing the number of registered contacts and user accounts.

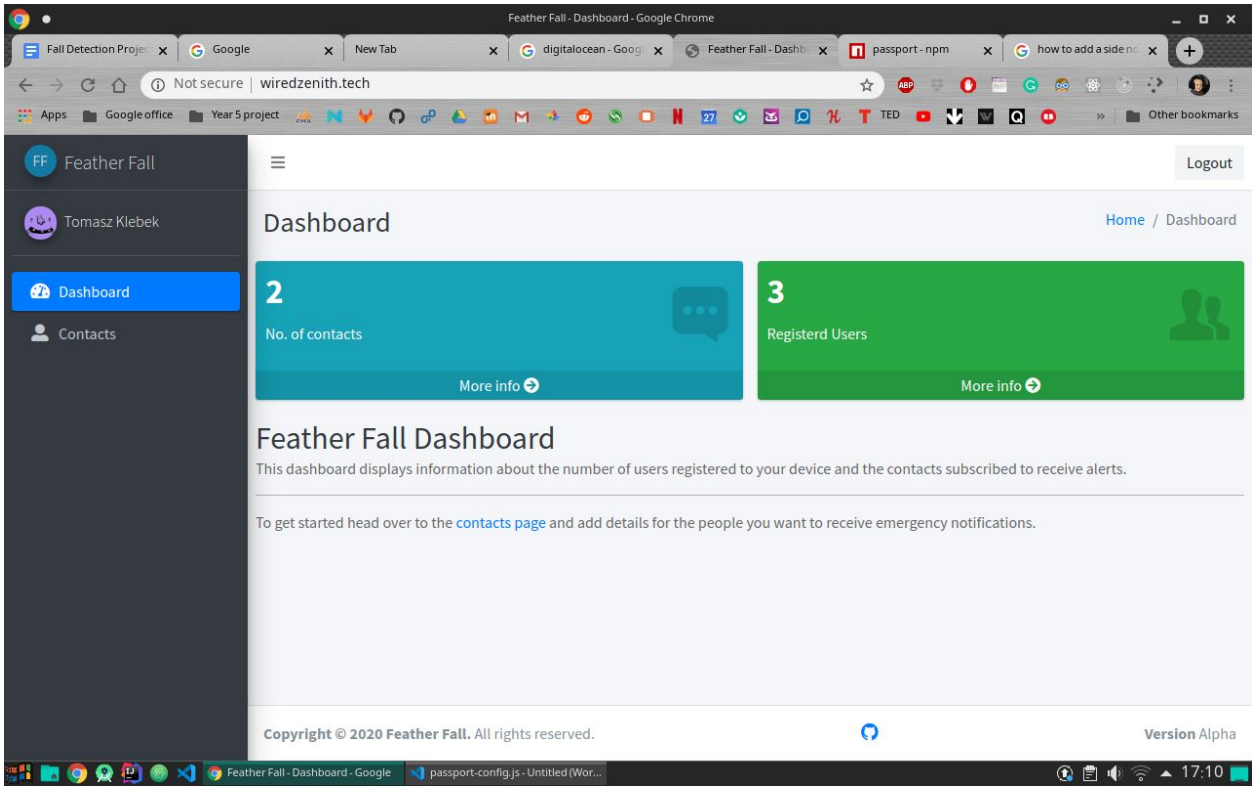

Figure 4.4 wiredzenith.tech dashboard page

The most important page in the site is the contacts page where users can add their contact numbers so that the fall detector can send them an alert in the case of an event being triggered. The logged-in user can remove or edit their contacts at any stage.

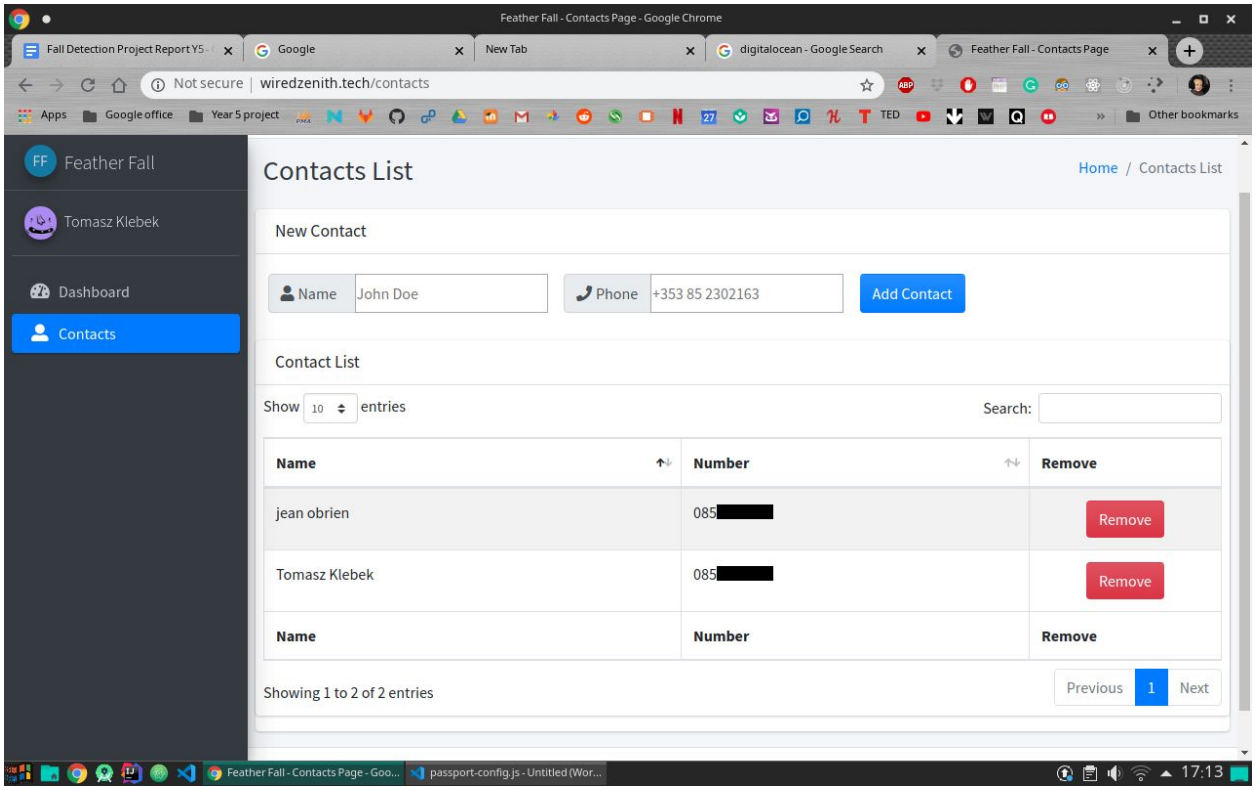

Figure 6.5 wiredzenith.tech contacts page

Here users can enter contact numbers for themselves and others. The name and phone number field are validated to check if the field is empty, and the phone number field is validated to ensure that only valid Irish phone numbers are stored in the database to reduce possible issues when contacting people using the GSM communication on ESP32.

```
if(!validator.isMobilePhone(data.number , "en-IE")){
            errors.number = "Must be an Irish mobile number";
        }
```

New users are created by sending forum data to an API endpoint that validates the form data and creates the new entry in the data using Mongoose schema. And if validation fails reloads the page with the relevant errors displaying on the forum.

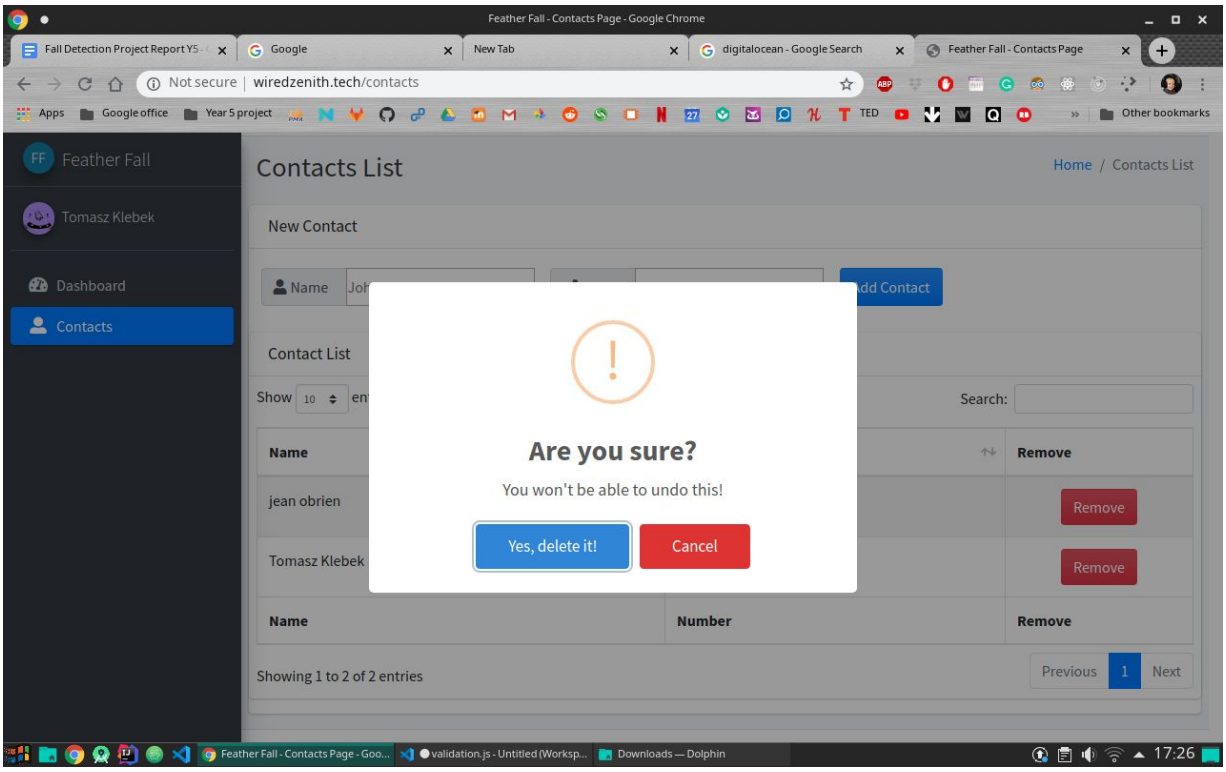

Figure 4.6 Delete user conformation dialog

```
router.delete('/:id', checkAuthenticated, async function (req, res, next) {
   console.log(req.params);

   try {
       var result = await Contacts.deleteOne({ _id: req.params.id }).exec();
       res.send(result);
   } catch (error) {
       res.status(500).send(error);
   }

});
```

When a contact is deleted an HTTP POST request is sent to the removeContacts.js script running on the backend that issues a confirmation dialogue asking the user to confirm that they want to delete the selected contact. After that, the id of the contact entry is sent to the API and the contact is removed from the database.

The confirmation pop-up on deleting contacts is issued using a node plugin called Sweetalerts2. The code for the pop-up window is standard boilerplate code supplied from Sweetalerts website but the handling of the request is passed to the delete API endpoint.

```
fetch("/contacts/" + id, {
  method: "delete"
}).then(async function (res) {
     var response = await res.json();
```

4.3 Web page deployment

To ensure that the web page is always available and easily accessible I used a few different technologies to host, deploy and keep the page up to date. The first part of the deployment is the web hosting itself. The website is served up from an Ubuntu virtual machine running in a digital ocean droplet. I used PM2, PM2 is a daemon process manager that helps manage and keep applications online, it allows me to start, stop and restart my node processes. Another application used to serve up my page is NGINX which is a web server that can also be configured as a reverse proxy. When users connect to wiredzenith.tech that are automatically forwarded to port 80, NGINX acting as a reverse proxy forwards users to the port that my node application is running on, port 3000. Finally, to ensure that my web page is up to date I employed the use of a continuous development application called Jenkins to monitor my code repository on GitHub and to build and deploy the latest code from my master branch to the webserver. To do this Github is set up to issue webhooks to my Jenkins deployment running on port 8080 of the Ubuntu server. When a change is detected the master branch of my repository Jenkins pulls the code from the web page folder in my repository and builds it ensuring that the node applications are operating as intended. Once the application is built successfully Jenkins navigates to a local deployment script in the folder and runs the steps specified below.

```
#!/bin/sh
ssh root@wiredzenith.tech<<EOF
  cd /FallDetection
  git stash push --include-untracked
  git stash drop
  git pull
  cd Webpage/
  npm audit fix
  npm install
  cd ~
  pm2 restart all
  exit
EOF
```

Jenkins is configured with its own ssh key to access the digital ocean machine, and on logging in navigate to the FallDetection folder stashes any changes that may have been made to the local files by me or temporary files created either by PM2 or other processes. These stashed changes are then discarded to prevent any conflicts when merging the latest version of the web page from my repository with what already exists on the Ubuntu virtual machine. New code is pulled from GitHub, navigates into web page folder and runs npm audit fix, to repair any issues that may occur when adding the npm packages, npm install to build the node project nad install any node pages that have been added and finally it uses PM2 to restart all the of the processes, relaunching the node application. The deployment of the web page while not being extremely complex does allow for scaling, using NGINX would allow me to handle many requests to the web page, PM2 ensures that the web page will have a reliable and consistent uptime and The Jenkins CI/CD pipeline means making changes to the site and repairing issues and a bug is quick and easy to resolve.

4.4 ESP32

In order for people to be contacted in the case a user of this device falls a few things need to happen, there needs to be a way to detect a fall, and notifications need to be sent. In this section I want to cover the code I developed for operating the ESP32, enabling to send and receive SMS notifications. The device also needs to be able to detect if a fall has occurred, I intended to do this using an accelerometer, I will explain the method behind the fall detection and how I had planned to implement it as well as explain why I did not manage to add it into the final product. All of the code written is my own, however, I have taken reference from the firmware wiki page[11].

4.4.1 GSM Communication

To send text messages and establish mobile data communication for downloading contact information from the contacts database, I use an ESP32 development board that has a GSM module onboard. When the device boots up it first makes a connection to the mobile network declaring its credentials, the APN(Access Point Name) the username and password for accessing the network, this step if so that I can later connect the device to the GPRS network. Communication with the SM800L GSM module is done using UART communication and sending AT commands using this method.

Setting up the GPRS credentials are pretty straight forward.

```
# APN credentials
GSM_APN = 'data.myeirmobile.ie'  # APN
GSM_USER = ''  # User name
GSM_PASS = ''  # Password

gsm.start(tx=27, rx=26, apn=GSM_APN, user=GSM_USER, password=GSM_PASS)
```

First APN credentials, as can be seen above, no need for username or password on this network. Then I set up the new gsm device by calling start from the gsm class and passing it the transmit(TX) and receive(RX) pins that are connected to the gsm module. The RX and TX pins are found from the schematic provided with the ESP32-LilyGo-T-Call.

Figure 4.7 ESP32-LilyGo-T-Call GSM pinout

The pin names in the code and in the picture are counter-intuitive. we transmit to the receive pin on the GSM module and receive from the transmit pin on the GSM module.

The gsm.start function sends various AT commands to the GSM module to initialise the GPRS communication. One the device is set up it will respond to the ‘AT’ command with 1 I check this with a for loop till a response comes back true otherwise it returns that the modem is not responding

```
sys.stdout.write('Waiting for AT command response...')
for retry in range(20):
    if gsm.atcmd('AT'):
        break
    else:
        sys.stdout.write('.')
        time.sleep_ms(5000)
else:
    raise Exception("Modem not responding!")
print()
```

Now the module is set up and ready to send and receive text messages and communicate using mobile data. One the initial set up is complete, the first thing that I need to do is get the list of contacts from the database

```
def getContacts():
    usersList = []
    if gsm.status() == "98, 'Not started'":
        gsm.connect()
    r = req.get("http://wiredzenith.tech/contacts/list")
    usersList = r.json()
    gsm.disconnect()

    return usersList
```

First I connect to the GPRS network using the gsm.connect(), Then I send a rest request to the API endpoint on my webserver, the reason I send this request to the webserver API rather than just making a call to the database directly is it allows me to format the data on the server-side rather than having to download that data and having the ESP32 sort it and save it this saves time. Once the call is complete and the information is saved as a JSON object the function returns that JSON object. The next step is to save that information to a text file.

Now with the contact information saved to a text file as a JSON object, sending messages to users is quite simple. Open up the file and load it as a JSON object then iterate through the file sending a message to each user

```
def sendMessageToAllContacts():
    with open(contactsPath, 'r') as contacts_json:
        contactsList = json.load(contacts_json)

    for contact in contactsList:
        recipient = contact['name']
        outgoingNumber = contact['number']
        outgoingMsg = "Dear " + recipient + " we regret to inform you that an
emergency alert has been triggered"
        sendSMS(outgoingNumber, outgoingMsg)
```

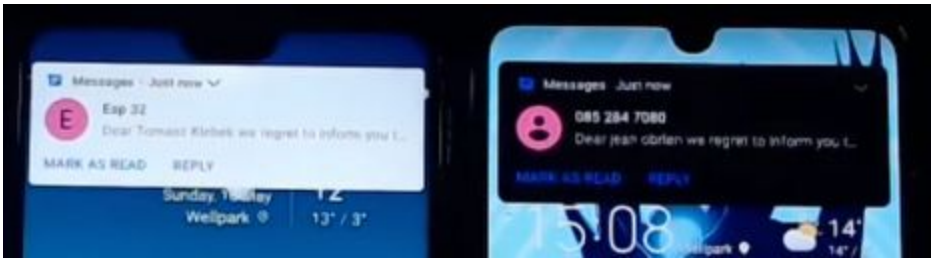

Figure 4.8 Mobile device receiving notification form ESP32 GSM module

4.4.1 I2C Communication

To detect if a user has experienced a fall I used an ADXL345 Accelerometer form analogue devices. In initial planned on using SPI (Serial Peripheral Interface) for communicating between the ADXL345 and the ESP32 but after failed attempts I decided to try using i2c, I did not have much more success with this method but none the less I would like to cover the operation of i2c and how it would be implemented to get reading from the ADXL345.

I2C (Inter-Integrated Circuit) is a synchronous serial communication bus that is able to support up to 1008 slaves on the same bus. I2C only requires two lines for data communication and the hardware setup is straightforward.

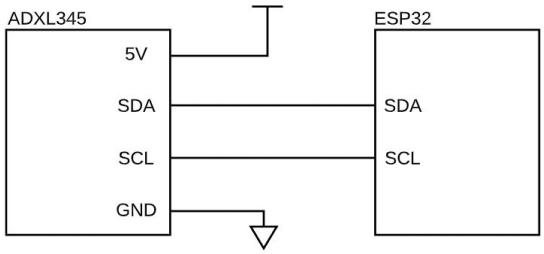

Figure 4.9 I2C wiring diagram

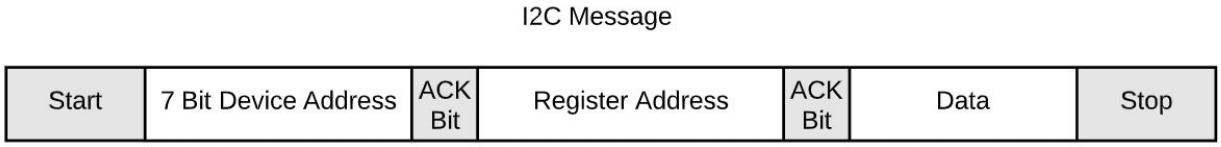

Figure 4.10 I2C Message

The I2C message is as shown above.

Start - The SDA (Serial Data) line switches from a high voltage level to a low voltage level before the SCL (Serial Clock) line switches from high to low.

7 Bit Device Address - The unique address of the slave device the message is intended for.

ACK - Each frame in the message is followed by an acknowledgement bit. If an address frame or data frame was successfully received, an ACK bit is returned to the sender from the receiving device.

Register Address - The address of the register that is to be written to.

Data - The data to be written into the register.

Stop - Final a stop bit denoting the end of the message. The SDA line switches from a low voltage level to a high voltage level after the SCL line switches from low to high.

To initialize the ADXL345 a series of registers needs to be set up first.

```
def ADXLInit():
    i2c.writeto_mem(ADXL342_ADDR, DATA_FORMAT_ADDR, DATA_FORMAT)
    i2c.writeto_mem(ADXL342_ADDR, POWER_CTL_ADDR, POWER_CTL)
    i2c.writeto_mem(ADXL342_ADDR, INT_ENABLE_ADDR, INT_ENABLE)
```

First The data format register that handles the presentation of data in the data output registers needs to be configured to 13 Bit mode. Then the power Control register is set and tells the device to begin measurements, finally, the interrupt enable register is set to 1 to allow the device to issue interrupts when defined data values are reached for example the tap detection or the free fall interrupts. In order to detect falls, there are two registers that need to be set, the THRESH_FF and TIME_FF, the free fall threshold and the free-fall time. The acceleration on all axes is compared with the value in THRESH_FF to determine if a free-fall event occurred. Once a freefall event occurs an interrupt is issued from the ADXL345 and can be sent to the ESP32 and dealt with appropriately, by sending alert messages out to contacts that are saved on the device.

5.Issues and problem-solving

During the development of any project, it is inevitable that problems will occur and that bugs will develop within the code to handle these issues. I used a few different tools, the first was OpenProject. OpenProject is an open-source project management tool. Initially, I used this to plan out my project phases, log bugs and generally keep track of my progress.

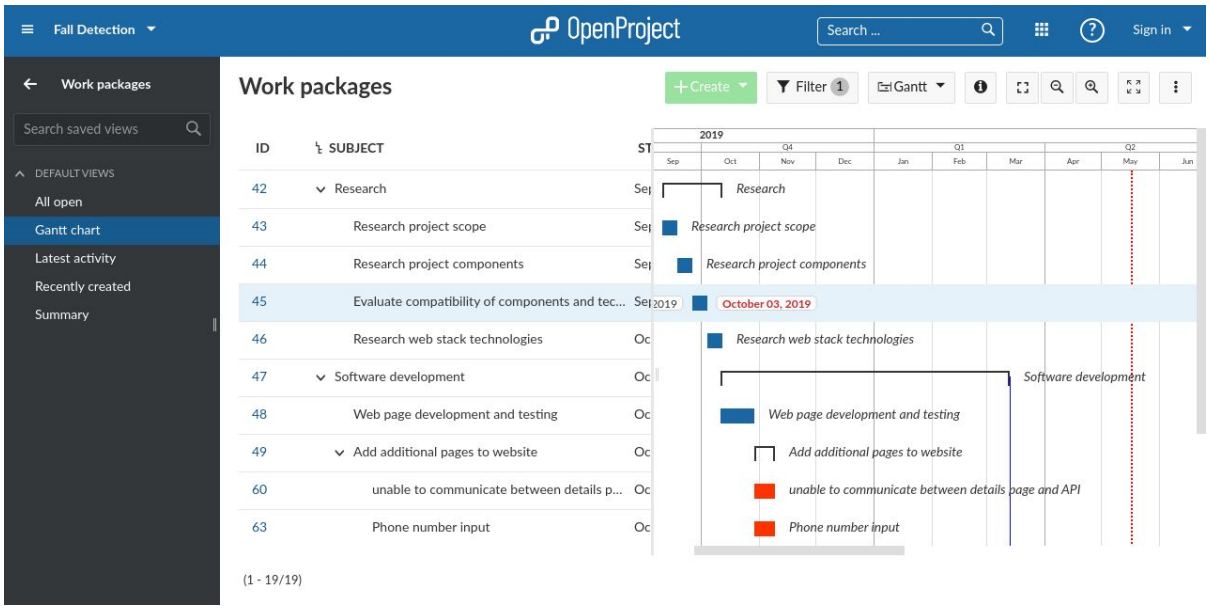

Figure7.1 OpenOffice web interface

Later I discovered the 'issues' functionality on GitHub and decided to move my bug reporting and issue handling here.

5.1 Web page

Developing the web page was enjoyable and challenging at the same time, being unfamiliar with JavaScript meant a lot of trial and error writing code and testing it. For testing my Node.js project I used nodemon, and postmaster for testing the API. nodemon monitors the folder where my application runs and restarts the server whenever a change is detected allowing me to write code, Save it, and see the result.

Postmaster was also very useful for testing mt API's postmaster is an application that allows me to send REST requests to my exposed API's and see their responses.

5.2 I2C

The largest and most problematic issue I faced on during my project was the method of fall detection as mentioned above I use an ADXL345 accelerometer, initially, I tried using SPI to communicate with this device, as it seemed to be the easiest and would allow me to get up and running quicker, allowing me to work on other aspects of the project too. However, this did not happen writing the code for SPI proved more challenging than expected after numerous weeks testing and debugging different versions of SPI code I was still unable to get usable data from the ADXL345. To prevent further delays, I decided to move to I2C communication protocols and use example code I had found online and still I was unable to make progress. using the SPI and I2C methods I was able to establish communication with the device however the data returned was garbage (useless characters) with no structure. I have done a lot of research and concluded in the end that the device must have been damaged during my first round of trying to develop drivers. I can only guess that I must have made aa mistake with the wiring, perhaps mixing the 5V and ground and damaging the device. I have recently found example code for the ADXL345 written in MicroPython and upon running it shows that the device is indeed damaged.

6.Conclusion

At the start of this project, I set out to build a device that would have a simple easy to use web page where users could log in and add contact information. As well as a hardware device that was able to detect if someone had fallen and notify the numbers on the contact site by text message. While I did not manage to achieve all of the outcomes I had intended, namely the fall detection. I still have a device that is able to send notifications to users and a sleek web page and in the process, I have learnt a lot about web development and continuous development as well as developed new skills in JavaScript, Node.js, Express, Python and embedded programming skills.

7.Glossary

ACK	-	Acknowledgement bit used by I2C
API	-	Application Programming Interface
APN	-	Access Point Name
ASF	-	Advanced Software Framework
AT	-	Abbreviation of ATtention. Used for modem communication
AVR	-	AVR is a family of microcontrollers
CI/CD	-	Continuous-integration and Continuous-deployment
GPRS	-	General Packet Radio Service
GPS	-	Global Positioning System
GSM	-	Global System for Mobile Communications
HTML	-	Hypertext Markup Language
I2C	-	Inter-Integrated Circuit
IDE	-	Integrated Development Environment
JSON	-	JavaScript Object Notation
LiPo	-	Lithium polymer
NGINX	-	Open-source, high-performance HTTP server and reverse proxy
ODM	-	Object Document Mapping
REPL	-	Read–eval–print loop
SPI	-	Serial Peripheral Interface
UART	-	Universal asynchronous receiver-transmitter

8.References

[1] United Nations Publications, *World Population Ageing 2019 Highlights*. 2020.

[2] H. Steinmetz and S. Hobson, 'Prevention of Falls Among the Community-Dwelling Elderly':, *Physical & Occupational Therapy In Geriatrics*, vol. 12, no. 4. pp. 13–29, 1995[Online]. Availablehttp://dx.doi.org/10.1300/j148v12n04_02.

[3] F. Bagalà et al., 'Evaluation of accelerometer-based fall detection algorithms on real-world falls', *PLoS One*, vol. 7, no. 5, p. e37062, May 2012.

[4] (2019, September.17), *Deaths from Falls | Home and Recreational Safety | CDC Injury Center*. [Online]. Available: https://www.cdc.gov/homeandrecreationalsafety/falls/fallcost/deaths-from-falls.html. [Accessed: 06 May 2020].

[5] H. Hawley-Hague et al., 'Older adults' perceptions of technologies aimed at falls prevention, detection or monitoring: A systematic review', *International Journal of Medical Informatics*, vol. 83, no. 6. pp. 416–426, 2014[Online]. Availablehttp://dx.doi.org/10.1016/j.ijmedinf.2014.03.002.

[6] *[No title]*. [Online]. Available: https://www.careinspectorate.com/images/documents/2737/2016/Falls-and-fractures-new-resource-low-res.pdf. [Accessed: 06 May 2020].

[7] Britta et al., *Sentry - In-Home Medical Alert | BlueStar SeniorTech | Best Prices*, BlueStar SeniorTech. [Online]. Available: https://bluestarseniortech.com/shop/safety-products/sentry-in-home-medical-alert/. [Accessed: 06 May 2020].

[8] M. Pickavance, 'Best fall detection sensors of 2020', Mar. 2020[Online]. Availablehttps://www.techradar.com/best/best-fall-detection-sensors[Accessed: 6May2020].

[9] *V by Vodafone | V-SOS Band*. [Online]. Available: https://eshop.v.vodafone.com/ie/v-sos-band. [Accessed: 06 May 2020].

[10] B. Benchoff, (2016, October.18), *What's The Deal With Atmel And Microchip?*, Hackaday. [Online]. Available: https://hackaday.com/2016/10/18/whats-the-deal-with-atmel-and-microchip/. [Accessed: 07 May 2020].

[11] loboris, *loboris/MicroPython_ESP32_psRAM_LoBo*, GitHub. [Online]. Available: https://github.com/loboris/MicroPython_ESP32_psRAM_LoBo. [Accessed: 15 May 2020].