CMP408 Mini Project Report

Alex McNaughton 2000207

Table of Contents

1. Introduction

In modern computing, the advent of cloud services have provided an opportunity for developers, businesses and regular users to access the resources of larger organizations and improve their own systems far beyond their initial scope. Cloud computing, in comparison to local systems, is unparalleled in its ability to provide reliable, scalable and cost-effective systems for anybody requiring such resources.

Communicating between devices through text is a function of networks older than computers, and is an efficient solution for communicating compared to sending audio or visual data. Combining the functionality of text message sending with cloud computing could provide several benefits for a potential text system. For example, the remote accessibility of cloud based systems would make it much easier to send and receive messages from anywhere in the world, making it much easier to access. The reliability of cloud systems also makes the prospect of developing an application using cloud technology much more appealing. The Amazon Compute Service Level Agreement, which covers the guarantees for the reliability of Amazon's cloud computing solutions, says that they will provide an uptime of at least 99%, going even higher in some cases (Amazon, 2022). This amount of uptime combined with the fact that no maintenance is required from the user to maintain their cloud system makes the benefits of cloud computing obvious for a project such as communicating between devices over a cloud system.

1.1. Objectives

- Design a messaging interface for the Raspberry Pi, using a SSD1306 based display
- Construct a web server using AWS that can communicate with the Pi's messaging system
- Securely send messages to and from the server and Pi

2. Methodology

2.1. Hardware

Developing the messenger interface required some changes to be made to the proposed hardware described in the project proposal. Due to time constraints and the raspberry pi's non-existent analog signal capabilities, the potentiometer was removed as the letter selector and replaced with a keyboard input based approach instead. Due to this new approach, the functionality of the buttons was transferred to the keyboard. This new interface now consists of a keyboard, display, and message LED.

Fig 1 : Optimized Hardware Layout

This improved layout provides a considerably higher amount of inputs to work with, and reduces the development complexity of the project.

2.2. Pi Interface Development

Development of the raspberry pi's interface was supported by the use of the Adafruit SSD1306 library, which manages the drivers for the display. The interface consists of two main functions: a system to send messages, and a system to read messages sent to the device. The interface also includes a LED which is designed to light up when a new message is sent to the Pi.

Fig 2 : Pi Interface Initial Menu

When the software is run on the Pi, the user is presented with two choices on the display which represent the two different functions of the messaging software. If the user presses the ‘1’ key on the keyboard, the software begins the message typing function.

Fig 3 : Typing screen initial state

From here, the user can use the keyboard to type in any 30 character long string.

Fig 4 : Typing screen after the text ‘testing’ is typed using the keyboard

Once the user is ready to send the message, they can press the ‘Enter’ key and the text typed is sent to the server. Once this is completed, the user is sent back to the main menu seen in Fig 2. Pi Users can check the messages sent to them by pressing ‘2’.

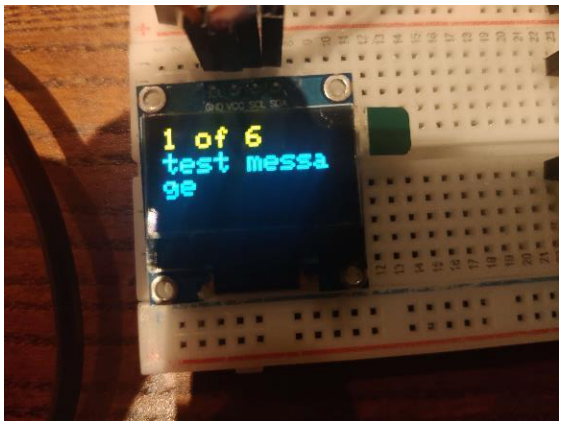

Fig 5: Initial inbox screen

Users can scroll through the messages sent to them using the ‘a’ and ‘d’ keys to scroll through each message, and can exit the screen at any time by pressing escape. Users can then quit the program by pressing escape on the main menu.

2.3. Web Interface Development

The web interface is similar in design to the pi’s interface, as it contains all the same functionality except it is provided within a browser.

Pi Messenger

Welcome to the Pi Messenger!

Inbox
Send a message

Fig 6 : Web interface home screen

Users have the option to access the web interface inbox by selecting the Inbox hyperlink, which gives them all the messages sent to the web server by the Pi.

Fig 7: Inbox.php

Navigating back to the homepage, users can select the ‘Send a message’ hyperlink to send messages to the pi.

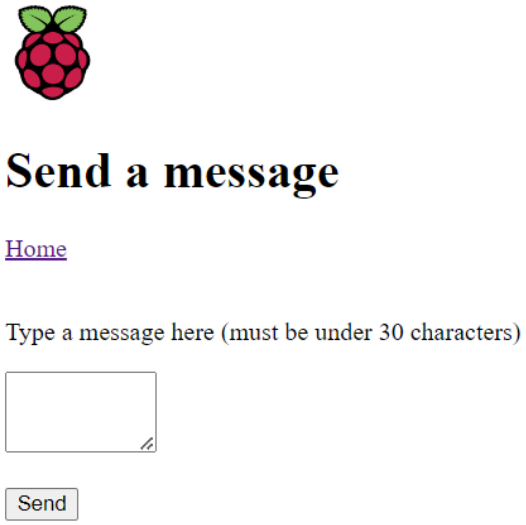

Fig 8 : message.php

2.4. Backend and relevant security

The majority of the messaging service functionality is provided through AWS, more specifically through the use of an Apache webserver running on an EC2 instance which has access to an RDS MySQL database which contains both the messages for the Raspberry Pi interface and the web client.

For the web client, sending messages to the raspberry pi user is done by updating the inbox of the pi user within the database. When the pi user wants to check their inbox, the pi client send a post request to the 'getmessages.php' function, which returns their inbox to them through the request. When a pi user wants to send a message to the web user, the pi uses another post request named 'toweb.php' with the appropriate message to send.

```
def connectdb():
    response = requests.post('http://ec2-54-226-111-140.compute-1.amazonaws.com/getmessages.php')
    raw = response.text[:-1]
    raw = raw.split(',')
    sort = [raw[n:n+2] for n in range(0,len(raw),2)]
    return sort
```

Fig 9: code snippet from Pi client, the messages are requested from the server through the getmessages function.

Using PHP functions in this way means that both clients can only interact with the server in the specific ways that they were designed to do, making it harder to exploit in comparison to directly interfacing with the database. To prevent any kind of SQL injection or cross side scripting, the inputs from both the web user and the pi user are sanitized by the server so as to stop users from inputting special characters.

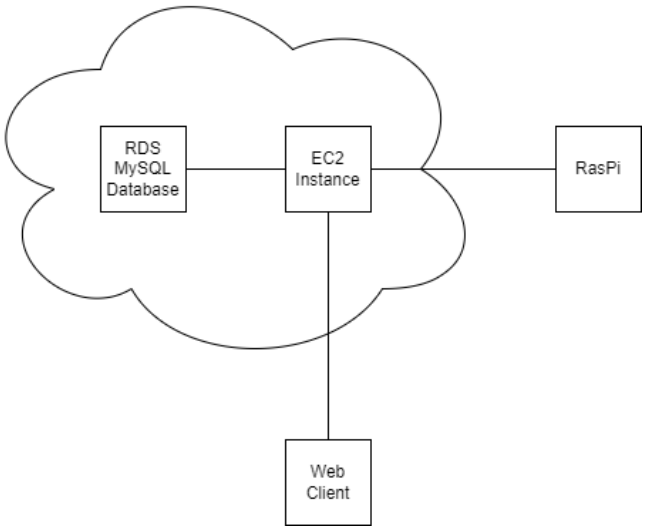

Fig 10: Diagram of the relationships between users and the server

3. Conclusion

Overall the project was able to meet its objectives: a pi client was made, a web server was made to handle messages, and the communication between devices is relatively secure. While the level of security is likely inadequate for sensitive data, there is still enough functionality to allow for easy communication between the two devices.

Reflecting on the functionality of the project, there are some aspects of security and user interaction that could be improved with more resources and time given to the project. One aspect that could have been improved is the pi messaging functionality, which misses some functionality that is common in other message sending software. When typing a message in a contemporary messaging software app such as WhatsApp, users are able to backspace and delete certain parts of the message in order to correct mistakes made while typing. This functionality is not available in the pi's messenger software, and the closest alternative requiring the user to escape from the message typing screen and start again. This interrupts the flow of message creation and would be the first function created if more time were given for the project. Another common feature of message software is the use of a cursor to allow users to write from different parts of the message besides the end. This would allow for quick and easy editing to a message before sending and would be beneficial to the user.

A greater level of security on the pi client's side would have also improved the overall security of the project. In its current form, the way that the Pi client requests its messages could be easily replicated by a third party, making it easy to impersonate the Pi client and view their messages. Implementing some kind of authentication for the getmessages function on the server would fix this issue i.e. requiring a password to be passed with the post request.

The LED notification could have also benefited from a more cloud-based approach. The current implementation of the notification LED means that the user is only notified of new messages when they open their inbox, instead of at any point while using the client. AWS provides the ability to send data using MQTT from servers to devices through its IoT Core which, if implemented, could provide a way to asynchronously update the LED when a message is sent from the web client.

In conclusion, while the project was able to functionally achieve its goals, it lacks in specific areas that could benefit from further development in order to achieve a higher degree of security and software usability, these. However, it is clear that the use of cloud computing greatly improved the functionality and workability of the application, allowing it to be consistently available to developers when working on the project.

4. Bibliography

Adafruit, 2023. *Adafruit CircuitPython SSD1306.* [Online]
Available at: https://github.com/adafruit/Adafruit_CircuitPython_SSD1306

Amazon, 2022. *Amazon Compute Service Level Agreement.* [Online]
Available at: https://aws.amazon.com/compute/sla/
[Accessed December 2023].