HÖHERE FACHSCHULE SCHAFFHAUSEN

Projekt mit dem Raspberry Pi 4

Inhaltsverzeichnis

1 Ziele

1.1 Beschreibung

Mit einem Raspberry Pi 4 soll eine Türautomatik erstellt werden. Wird eine RFID-Karte am Leser erkannt soll sich diese öffnen lassen. Es soll ebenfalls möglich sein weitere RFID-Karten hinzuzufügen. Weitere Ziele sind, dass die Türe überwacht werden kann und ein Alarmsignal ausgegeben wird, sollte diese unautorisiert geöffnet werden. Des Weiteren soll ein GPS-Modul verwendet werden, um einen Alarm auszulösen sobald eine bestimmte Position verlassen wird. Das Ziel ist es den Raspberry Pi anschliessend auf einem Boot zu Installieren und somit eine Ankerwache zu realisieren.

1.2 Muss Ziele

RFID-Karten einlesen und verarbeiten

Ausgänge ansteuern, wenn eine bekannte Karte erkannt wird

Eingänge einlesen

RFID-Karten hinzufügen

1.3 Wunsch Ziele

GPS-Modul überwacht die Position des Boots

GPS-Überwachung hat 2 Modis (Ankerwache = Radius, Bootplatz wache = Rechteck und Richtungskontrolle N W S O)

Mit einem RFID-Modul kann die Alarmanlage Ein-/und ausgeschaltet werden

RFID-Code ist einem User zugeordnet = Logfile

Raspberry Pi soll eine Alarmfunktion haben

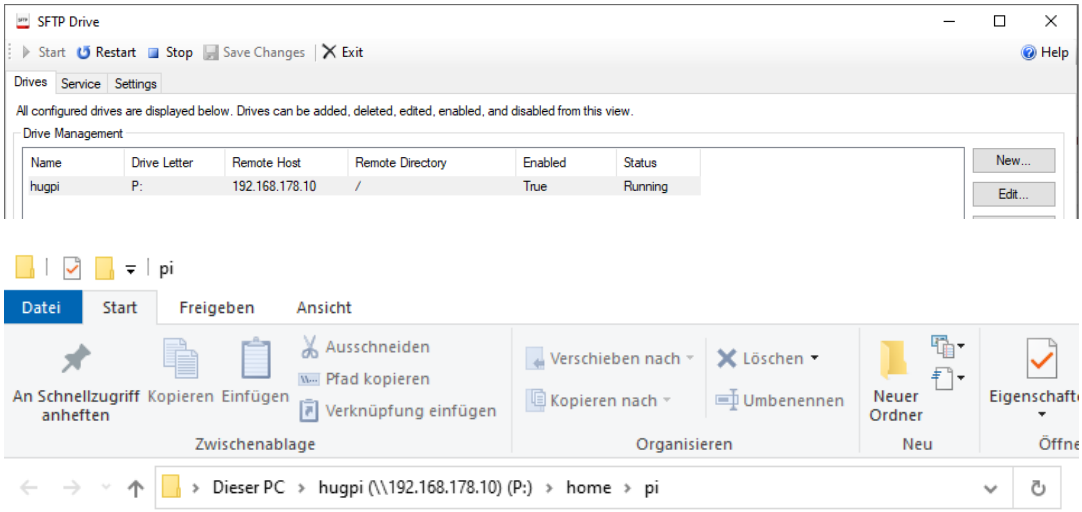
Raspberry Pi soll Emails oder SMS versenden, wenn ein Alarm ausgelöst wird

2 Raspberry Pi Aufsetzen

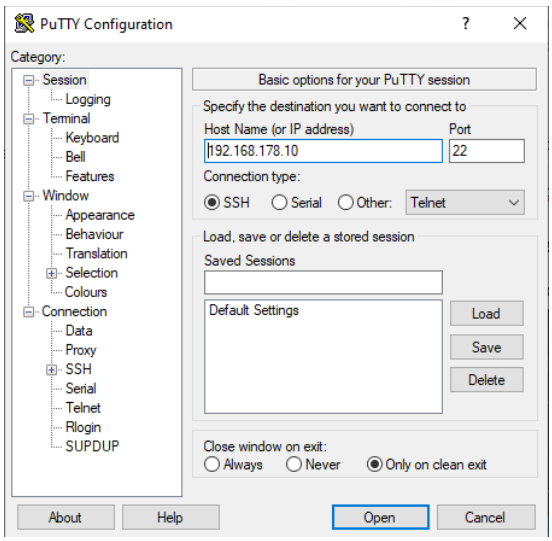
Mit dem Raspberry Pi Imager kann eine SD-Karte mit dem Betriebssystem konfiguriert werden. Wenn man nicht auf dem Raspberry Pi direkt programmieren will, ist es vorteilhaft unter den Einstellungen (Zahnrad) SSH zu aktivieren, wie auch die Netzwerkinformationen und Benutzer bereits einzugeben. Somit ist der Raspberry Pi beim ersten aufstarrten direkt mit dem Netzwerk verbunden und es kann auf ihn zugegriffen werden.

3 SFTP-Drive

Mit dem Tool SFTP Drive kann der Raspberry Pi als Netzwerklaufwerk unter Windows Integriert werden. Dies erleichtert den Datentransfer erheblich. Es sind lediglich die IP-Adresse und Benutzerinformationen nötig, um die Verbindung einzurichten. Teilweise ist es zu Problemen gekommen, wenn auf ein Verzeichnis doppelt zugegriffen wurde. Einerseits via Terminal und andererseits via SFTP Drive. Die Auswirkungen davon waren, dass die Dateien teilweise nicht mehr kopiert oder geöffnet wurden, oder diese nach dem Kopieren und Aktualisieren des Ordners nicht mehr auffindbar waren. Dies kann allerdings durch einen Neustart des SFTP-Drives behoben werden. Zudem solle das Tool ebenfalls neu gestartet werden, wenn der Raspberry Pi selbst neu gestartet wird.



4 Putty



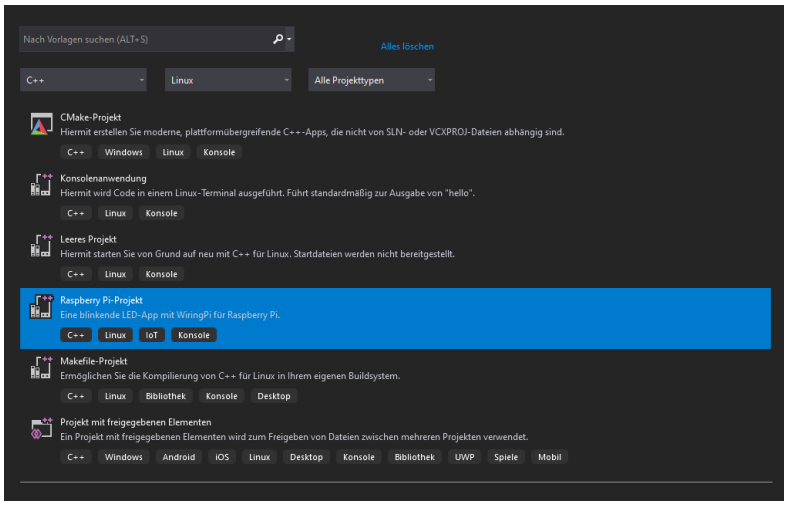
Putty ist ein Tool, mit welchem via SSH simpel auf einen Raspberry Pi zugegriffen werden kann. Es erleichtert die Arbeit vor allem, wenn eine Verbindung oft geöffnet und geschlossen wird. Die Verbindungsinformationen bleiben gespeichert.

5 Visual Studio

Visual Studio hat eine Projektvorlage, mit welcher man den Code in Visual Studio erstellt und anschliessend auf dem Raspberry Pi kompiliert und ausführt. Dies funktioniert ebenfalls über eine SSH-Verbindung. Ist das Visual Studio erstmal entsprechend konfiguriert, bringt dies enorme Vorteile. Man kann nun wie gewohnt, mit einem Klick das Programm ausführen und zudem werden alle Fehler direkt im Code angezeigt. Die Konfiguration benötigt Zeit, jedoch ist diese Methode meiner Meinung nach lohnenswert.

5.1 Projekt eröffnen

In Visual Studio muss lediglich ein Raspberry Pi Projekt eröffnet werden. Anschliessend startet ein Tutorial welches simpel erklärt, wie die SSH-Verbindung zum Raspberry Pi aufgebaut wird. Danach starten leider die Komplikationen. Beim Demo Projekt ist die Bibliothek WiringPi.h verwendet. Diese ist nicht standartmässig auf dem Raspberry Pi installiert. Auf der offiziellen Website ist diese zudem abgekündigt. Das heisst entweder findet man auf GitHub eine Passende Vorlage, oder man wechselt auf eine andere Bibliothek. Um jedoch die Verbindung etc. zu testen, kann einfach ein kleines Demo Programm mit Standardbibliotheken erstellt werden. Ein Beispiel davon wäre Textausgabe in der Konsole o.ä. Dies sollte umgehend funktionieren.

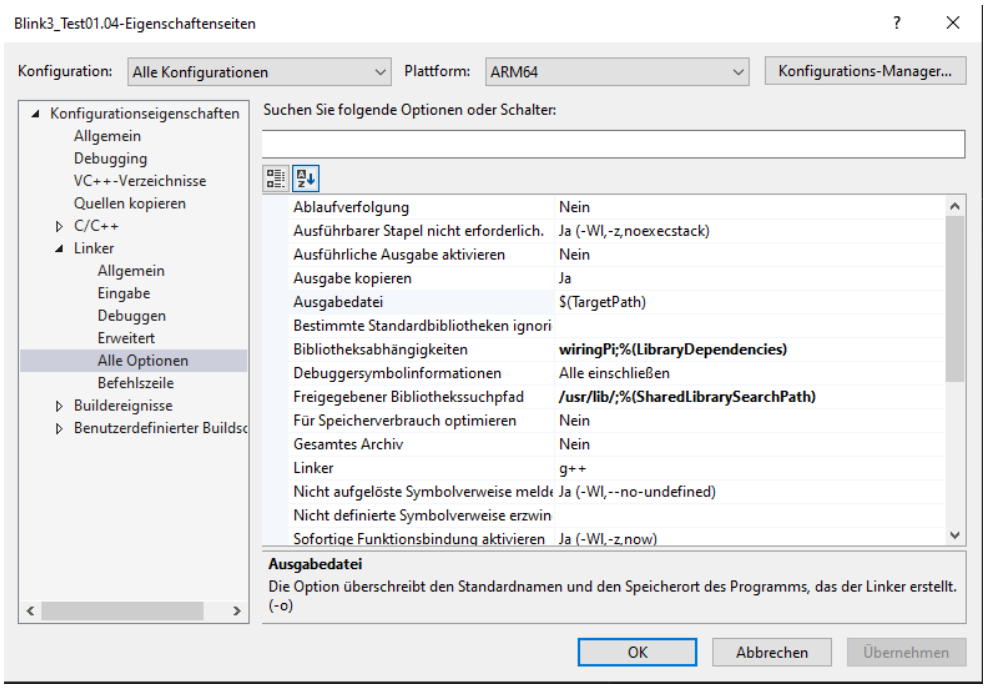


5.2 Einfügen von Bibliotheken in das Projekt

Sobald eine Bibliothek auf dem Raspberry Pi verwendet wird, welche nicht im Standard ist (z.B. WiringPi) muss dies im Visual Studio verlinken. Prüft als erstes, ob die Bibliothek richtig auf dem Raspberry Pi installiert ist. Dafür könnt ihr ein Testprogramm, welches die Bibliothek inkludiert, in der Kommandozeile Kompilieren und ausführen. Die Verlinkung muss einmal für das Kompilieren gemacht werden und andererseits im VC++ Verzeichnis. Das letztere ist dafür, dass im Visual Studio keine roten Markierungen mehr angezeigt werden. Ansonsten wird die Bibliothek und alle dazugehörigen Befehle rot unterstrichen.

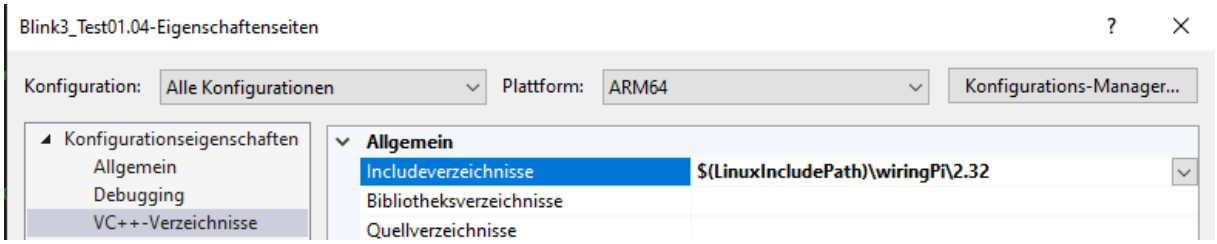
5.2.1 Compiler

1. Reiter Projekt
2. Eigenschaften
3. Im Dropdownfeld Konfiguration «Alle Konfigurationen» Auswählen
4. Reiter Linker / Alle Optionen
5. Bibliotheksabhängigkeiten: wiringPi;%(LibraryDependencies)
6. Freigegebener Bibliothekssuchpfad: /usr/lib/;%(SharedLibrarySearchPath)
7. Mit OK Bestätigen.



5.2.2 VC++

1. Reiter Projekt
2. Eigenschaften
3. Im Dropdownfeld Konfiguration «Alle Konfigurationen» Auswählen
4. Reiter VC++-Verzeichnisse
5. Includeverzeichnisse: $(LinuxIncludePath)\wiringPi\2.32



6 Kompilierer C++

Der C++ Kompilierer muss ebenfalls auf dem Raspberry Pi installiert werden. Wenn man ohne Visual Studio ein Programm auf dem Raspberry Pi ausführen will.

6.1 Installation via Kommandozeile:

sudo apt-get update

sudo apt-get install g++

6.2 Kompilieren

Ein C++ File kann anschliessend in einem Verzeichnis auf dem Raspberry Pi angelegt werden. Z.b Mit SFTP-Drive kann das File einfach im Verzeichnis abgespeichert werden

Mit dem Befehl «cd» kann in das entsprechend Verzeichnis gewechselt werden.

Mit Folgendem Befehl kann das Programm kompiliert werden:

g++ meinprogramm.cpp -lwiringPi -o meinprogramm

Anschliessend wir eine Kompilierte Datei mit dem Namen meinprogramm in diesem Verzeichnis erstellt.

-lwiringPi ist die Bibliothekabhängigkeit. Sobald ihr eine nicht Standartmässige Bibliothek verwendet, müsst ihr diese beim Kompilieren angeben.

6.3 Ausführen

Ist man im Korrekten Verzeichnis kann das Programm mit folgendem Befehl ausgeführt werden

./meinprogramm

7 Kompilierer Python

Der Python Kompilierer muss ebenfalls zuerst installiert werden.

7.1 Installation via Kommandozeile:

sudo apt-get update

sudo apt-get install -y python3-pip

7.2 Kompilieren

Mit Python ist das Kompilieren und Ausführen einfacher. Das Programmfile kann direkt kompiliert und ausgeführt werden. Es ist kein Zwischenschritt notwendig und es wird nicht ein zusätzliches File erstellt.

sudo python3 meinprogramm.py

8 GitHub auf dem Raspberry Pi

Mit der Installation von GitHub können Repository direkt heruntergeladen werden.

8.1 Installation

sudo apt install git

8.2 Repository herunterzuladen

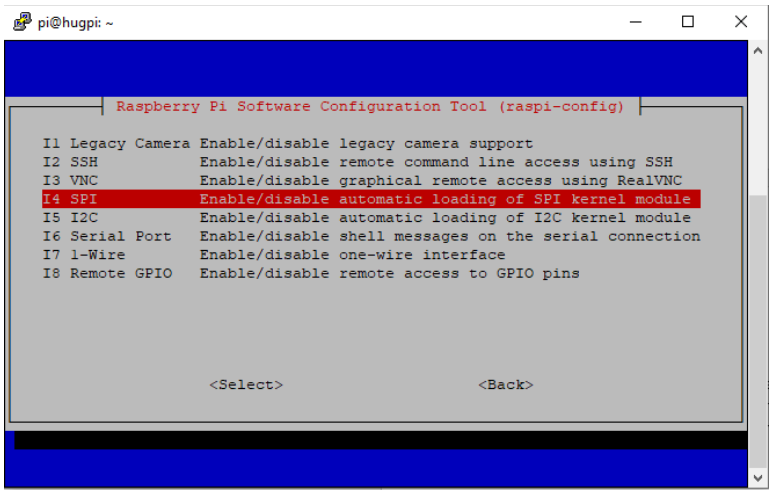
git clone <repository-url>

9 RFID-Modul

Um ein RFID-Modul mit dem Raspberry Pi zu verwenden sind einige Schritte notwendig. Das Auslesen des Chips habe ich mit einem Python Programm realisiert.

9.1 SPI-Schnittstelle konfigurieren

Die SPI-Schnittstelle ist Standartmässig deaktiviert. Folgende Schritte müssen durchgeführt werden um Sie zu Aktivieren



1. sudo raspi-config
2. Das raspi-config-Menü wird angezeigt. Verwende die Pfeiltasten, um zu "Interfacing Options" zu navigieren, und drücke die Eingabetaste.
3. Wähle "SPI" aus den angezeigten Optionen und aktiviere die Schnittstelle

9.2 Verbinden

RFID RC522 Modul	Raspberry Pi
SDA	Pin 24 / GPIO8 (CE0)
SCK	Pin 23 / GPIO11 (SCKL)
MOSI	Pin 19 / GPIO10 (MOSI)
MISO	Pin 21 / GPIO9 (MISO)
IRQ	-
GND	Pin6 (GND)
RST	Pin22 / GPIO25
3.3V	Pin 1 (3V3)

9.3 SPI-Verbindung Testen

Mit folgendem Befehl kann eine Liste der Aktiven SPI-Teilnehmer angezeigt werden: lsmod |grep spi

Wird anschliessend folgendes angezeigt dann funktioniert die Verbindung.

```
pi@hugpi:~ $ lsmod|grep spi
spidev                 20480  0
spi_bcm2835            20480  0
```

9.4 Bibliothek

Um das RFID-RC522 Board zu verwenden, muss die passende Bibliothek installiert werden

1. Spidev wird verwendet, um die SPI-Schnittstelle auszulesen: sudo pip3 install spidev
2. Mfrc522 wird verwendet, um das Modul anzusteuern: sudo pip3 install mfrc522

9.5 Programm

Das Python Programm, welches in diesem Projekt verwendet wird, wurde von Miguelbaloba kopiert und angepasst. Unter folgendem Link kann es heruntergeladen werden:
https://github.com/miguelbalboa/rfid.git

Um die eingelesene UID der RFID-Karte in das Hauptprogramm zu übergeben, wurde das Programm erweitert. Es wird ein .txt File erstellt. Dieses wird anschliessend vom C++ Programm eingelesen und gelöscht. Um Doppelzugriffe etc. zu vermeiden wurde der Befehl so gestaltet, dass nur ein Dokument erstellt wird solange keines mit diesem Namen existiert. Dieser Fehler führte ursprünglich zum Abbruch des Programms. Dies wurde abgefangen und mit einer Ausgabe in der Konsole angezeigt. Dadurch wird sichergestellt das immer nur 1 Programm auf die .txt Datei zugreift und das Programm keinen Haltepunkt hat und ordentlich durchläuft. Folgende Befehl wurde dafür verwendet

```
    try:
        with open('UID.txt', 'x') as f:
            f.write(str(uid[0])+str(uid[1])+str(uid[2])+str(uid[3]))
    except FileExistsError:
        print("File bereits erstellt")
```

Zudem lief das Programm in einer Endlosschlaufe. Diese wurde aufgehoben sodass sobald 1 RFID-tag Erkannt wurde das Programm beendet wird.

10 Programm C++

10.1 Ein und Ausgänge

Die Pinbelegung von Wiring Pi ist nicht die gleiche wie die Hardwaremässige Pin Belegung auf dem Raspberry Pi.

Raspberry Pi Pin	WiringPi Pin	Funktion
3	8	GPIO 2
5	9	GPIO 3
7	7	GPIO 4
8	15	GPIO 14
10	16	GPIO 15
11	0	GPIO 17
12	1	GPIO 18
13	2	GPIO 27
15	3	GPIO 22
16	4	GPIO 23
18	5	GPIO 24
19	12	GPIO 10
21	13	GPIO 9
22	6	GPIO 25
23	14	GPIO 11
24	10	GPIO 8
26	11	GPIO 7
27	30	GPIO 0
28	31	GPIO 1
29	21	GPIO 5
31	22	GPIO 6
32	26	GPIO 12
33	23	GPIO 13
35	24	GPIO 19
36	27	GPIO 16
37	25	GPIO 26
38	28	GPIO 20
40	29	GPIO 21

10.2 Ausführen von Python Skripten

Mit folgendem Befehl kann ein Python Programm in einem C++ Programm aufgerufen werden:

system("python Meinprogramm.py"); // Es wird auf das Ende des Programmes gewartet.

Mit folgendem Befehl kann ein Python Skript im Hintergrund ausgeführt werden. Es ist dabei jedoch zu beachten, dass es zu Komplikationen kommen kann, wenn beide Programme auf die Ein und Ausgangspins zugreifen. Zudem kann das Programm mehrmals gestartet werden.

system("python Meinprogramm.py &");// Programm wird im Hintergrund ausgeführt

10.3 Strings Einlesen

Um die UID aus dem .txt File des Python Programmes einzulesen wurde eine Funktion erstellt, welche als Rückgabewert die UID im String Format hat.

```
string get_UID_Input() {
    system("python Read.py"); // Aufruf eines externen Python-Skripts
    ifstream newfile("UID.txt", ios::in); // Öffnen einer Datei zum Lesen
    string UID_Input ="noInput";

    if (newfile.is_open()) {
        while (getline(newfile, UID_Input)) {
            cout << UID_Input << "\n"; // Ausgabe der Daten
        }
        newfile.close(); // Schließen der Datei
        remove("UID.txt"); // Löschen der Datei

    }

    return UID_Input; // Rückgabe der gelesenen Zeichenkette

}
```

10.4 String Storage

Der folgende Code definiert eine Klasse namens StringStorage, die dazu dient, Zeichenketten (Strings) in einer Datei zu speichern und zu laden. Hier ist eine ausführliche Erklärung des Codes:

1. Die Klasse StringStorage hat zwei private Member-Variablen:

- strings: ein Vektor (Array) von Zeichenketten, in dem die Strings gespeichert werden.
- filename: ein String, der den Dateinamen enthält, in dem die Zeichenketten gespeichert werden sollen.

2. Der öffentliche Teil der Klasse besteht aus einem Konstruktor und mehreren Methoden:

- Der Konstruktor StringStorage(const string& file) nimmt einen Dateinamen als Argument und speichert diesen in der Member-Variable filename. Er ruft auch die Methode loadStrings() auf, um die Zeichenketten aus der Datei zu laden, wenn ein Objekt der Klasse erstellt wird.
- Die Methode addString(const string& str) nimmt eine Zeichenkette als Argument und fügt sie dem Vektor strings hinzu. Anschließend ruft sie die Methode saveStrings() auf, um die Zeichenketten in die Datei zu speichern.
- Die Methode compareWithString(const string& str) nimmt eine Zeichenkette als Argument und durchläuft den Vektor strings, um zu überprüfen, ob die gegebene Zeichenkette bereits im Vektor enthalten ist. Wenn eine Übereinstimmung gefunden wird, gibt die Methode true zurück. Andernfalls gibt sie false zurück.
- Die Methode loadStrings() öffnet die Datei zum lesen, die durch den Dateinamen in filename angegeben ist. Wenn die Datei erfolgreich geöffnet wird, liest die Methode Zeichenketten aus der Datei und fügt sie dem Vektor strings hinzu. Am Ende schließt sie die Datei.
- Die Methode saveStrings() öffnet die Datei zum Schreiben, die durch den Dateinamen in filename angegeben ist. Wenn die Datei erfolgreich geöffnet wird, schreibt die Methode die Zeichenketten aus dem Vektor strings in die Datei. Am Ende schließt sie die Datei.

```cpp
class StringStorage {
private:
    vector<string> strings; // Vektor zur Speicherung von Zeichenketten
    string filename; // Dateiname zur Speicherung der Zeichenketten

public:
    StringStorage(const string& file) : filename(file) {
        loadStrings(); // Zeichenketten aus der Datei laden, wenn das Objekt erstellt wird
    }

    void addString(const string& str) {
        strings.push_back(str); // Eine Zeichenkette zum Vektor hinzufügen
        saveStrings(); // Zeichenketten in die Datei speichern
    }

    bool compareWithString(const string& str) {
        for (const string& storedStr : strings) {
            if (storedStr == str) {
                return true; // Überprüfen, ob eine Zeichenkette im Vektor enthalten ist
            }
        }
        return false;
    }

    void loadStrings() {
        ifstream file(filename); // Datei zum Lesen öffnen
        if (file.is_open()) {
            string str;
            while (getline(file, str)) {
                strings.push_back(str); // Zeichenketten aus der Datei in den Vektor laden
            }
            file.close(); // Datei schließen
        }
    }

    void saveStrings() {
        ofstream file(filename); // Datei zum Schreiben öffnen
        if (file.is_open()) {
            for (const string& str : strings) {
                file << str << endl; // Zeichenketten aus dem Vektor in die Datei schreiben
            }
            file.close(); // Datei schließen
        }
    }
};
```

10.5 Autostart Programm

Um das Programm automatisch nach dem Boot Vorgang zu starten habe ich einen Systemd-Service erstellt. Dafür habe ich die Konsole verwendet.

1. Finale CPP Datei in das gewünschte Verzeichnis laden und mit der Kommandozeile Kompilieren.
2. Erstellen der Datei: sudo nano /etc/systemd/system/meinprogramm.service
3. Einfügen folgender Infos

```
  GNU nano 5.4        /etc/systemd/system/meinprogramm.service
[Unit]
Description=main
After=network.target

[Service]
ExecStart=/home/pi/projectfinal/main
WorkingDirectory=/home/pi/projectfinal/
User=pi
Restart=always

[Install]
WantedBy=multi-user.target
```

4. Speichern und Schliessen
5. Service Aktivieren:
    - sudo systemctl enable meinprogramm
    - sudo systemctl start meinprogramm
6. Kontrolliren, ob der Service aktiv ist: sudo systemctl status meinprogramm

11 Rückblick

Bei diesem Projekt sind einige Stolpersteine aufgetaucht. Im Grossen und Ganzen ist es relativ umständlich, um auf einem Raspberry Pi Remote Compiling mit C++ zu machen. Ich hatte es mir schlichtweg einfacher vorgestellt. Das grösste Problem war bei mir, dass keine passende Bibliothek mit C++ für das RFID-Modul und anfänglich für WiringPi vorhanden war. Somit musste ich mich mit der für mich neuen Programmiersprache Python auseinandersetzen. Sobald dies geschah, wollte ich eigentlich nicht zurückwechseln auf C++. Zum jetzigen Zeitpunkt finde ich es schade, dass es kein Teil des Lehrplanes ist. Mit der Hilfe von Diversen Tools, welche ich in der Doku beschrieben habe, konnte ich mich Stück für Stück an die Lösung heranarbeiten. Auch im Visual Studio bin ich eine Zeit lang fast verzweifelt. Bis ich herausgefunden habe welcher Pfade und welche Verweise am richtigen Ort eingefügt werden müssen, ist viel Zeit verstrichen. Ein Grund dafür war auch eine nicht korrekt installierte Bibliothek auf dem Raspberry Pi. Dies fiel mir erst auf, als ich versucht habe ein Testprogramm direkt auf dem Raspberry Pi via Kommandozeile auszuführen. Ich habe anschliessend versucht das Python Programm ständig im Hintergrund laufen zu lassen. Sobald ich dies tat, sind aber diverse Fehler aufgetreten. In der Finalen Fassung wartet das C++ Programm bis das Python Programm beendet wird. Dies macht es schwierig z.b einen Alarm auszulösen. Ich versuchte auch im C++ Programm das Python Skript zu beenden, dies funktionierte bei mir jedoch nicht. Ich habe leider keines der Wunschziele erreicht, jedoch konnte ich die Mindestanforderungen erfüllen. Dies war vor allem den oben Beschriebenen Problemen geschuldet. Aber ich habe gelernt, dass man den Aufwand von zusätzlicher Hardware nie unterschätzen darf und diese Erkenntnis nehme ich für meinen weiteren Berufsweg mit.