Technical University of Denmark

Final Project - Smart Security Box

Telecommunication Programming Projects with Arduino - 34338

Authors
Sadaf Ayub, s224027
Frederik Danielsen, s214718
Joachim Touveneau Petersen, s194296
Melek Büsra Øzel, s221706

January 25, 2025

Contents

Introduction

The goal of this project is to design and build a locker system, which we have chosen to name Smart Security Box. The system features a password lock system that combines analog and digital technologies. This provides a safe system for keeping valuable items while still being easy to use and practical.

To ensure that the product is practical and user-friendly, several functionalities are included. These include physical mechanisms for entering passwords, dynamic responses based on the box's current state, and cloud integration for storing and managing passwords.

The physical box is constructed using plastic and cardboard, with a servo motor controlling the lock mechanism. The password is entered through a combination of turning a potentiometer and pressing a button. As will be described in more detail later, each component of the box interacts and responds according to a system state.

Additionally, a cloud-connected Password Manager is implemented to manage passwords and box IDs, allowing for storage and easy updates of passwords in the cloud.

Employed technologies

The following includes a description of the technologies that were employed to implement the devices in this project.

Hardware

Firstly, we will give a description of the hardware components used in the project.

Potentiometer & button

The Smart Security Box makes use of a potentiometer in order to let the user choose an intended digit for input. The potentiometer can add a varying resistance to a circuit giving it the possibility to read of an analog value from it.

The microcontroller on the ESP8266 reads this varying voltage using an analog-to-digital converter, which converts the analog voltage into a digital value between 0 and 1023. This value is then mapped to a digit from 0 to 9 in the `readAnalogValue()` function present in `potentiometer.cpp`. From this reading a function called `chooseNumber()` which through its many given arguments interfaces with the main.ino loop for handling potentiometer input.

Initially it was thought to have a dial representation as seen on figure 1 below.

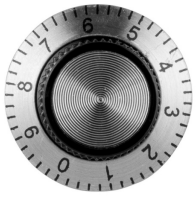


Figure 1: Approximate visualization of how digits should initially be displayed

It was then later decided to show the current digit on to the LCD screen attached on the security box. See figure 2b. This is done by parsing the selectedNumber variable into the `chooseNumber()` function.

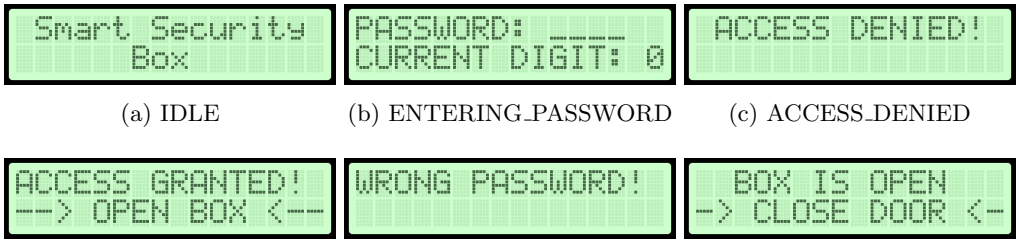
The button is used whenever the user wants to insert the current digit into the code this is also handled in the `chooseNumber()` function. Whenever the state of the security box is in the open state, the press of the button will close the lock.

LCD screens

The project makes use of two LCD modules. Each LCD module is equipped with a PCF8574T inter-integrated circuit (I^2C) that allows for easy interfacing with the Arduino, using only four pins (GND, 5V, SDA, SCL). One LCD is attached to the Smart Security Box and the other to the Password Manager. The LCDs provide real-time information about the state of the devices and guide the user on how to interact with them. The LCDs are linked to instances of the `LCDScreen` class defined in `LCD_screen.cpp`. The class has the methods `clear()`, `printLine()`, and `printLines()` that allow us to update the displayed text easily.

The Smart Security Box: To manage the Smart Security Box's LCD, the `LCD_screen.cpp` file includes the function `updateBoxLCD()` responsible for updating the LCD screen, based on the global state of the Smart Security Box, the current digit chosen using the potentiometer, and the digits already entered. This function is constantly called from the `loop()` function of the `main.ino` file. Figure 2 shows the messages linked to each of the box states.

The Password Manager: The Password Manager does not have a specific update function, but still makes use of the `LCDScreen` class to control the LCD screen. Thus, in the code controlling the Password Manager, the `LCDSCreen` object is used a lot more loosely.



(a) IDLE (b) ENTERING_PASSWORD (c) ACCESS_DENIED

(d) CORRECT_PASSWORD (e) WRONG_PASSWORD (f) BOX_OPEN

Figure 2: LCD screen states for the Smart Security Box

Passive buzzer

The Smart Security Box utilizes a passive buzzer to generate audible signals to the user. This includes beeps when choosing digits as well as sounds when getting the password right or wrong. The buzzer is interacted with through the `Buzzer` class defined in `Buzzer.cpp` which has the methods `toneOn()`, `toneOff()`, `numberSelected()`, `wrongPassword()`, `correctPassword()`, `boxClose()`, and `accessDenied()`. Table 1 shows an overview of the states and the corresponding sounds.

LED lights

The Smart Security Box utilizes three LED lights. One green, one yellow, and one red. These are used to indicate the state of the box. Table 2 shows how the LEDs behave. The LEDs are controlled using the `LEDs` class found in `LEDs.cpp`. This class has the method `update()` which

Entering digit	Short high-frequency beep
Wrong password	Short low-frequency beep
Correct password	C-major scale (C4, E4, G4, C5)
Three wrong passwords (alarm)	Alternating high- and low-frequency tones
Box close	Two high-frequency beeps

Table 1: Buzzer sound overview

takes the state of the system as input, and turns on the correct LEDs corresponding to the state.

State	Behavior
Idle	When the Smart Security Box is in its idle mode, only the yellow LED is turned on.
Entering password	When the user enters the password on the Smart Security Box, only the yellow LED is turned on.
Access granted	When the user enters the correct password, only the green LED is turned on.
Wrong password	When the user enters an incorrect password on the Smart Security Box, only the red LED is on.
Box open	When the box is open, only the green LED is on.
Access denied	When the user enters three incorrect passwords in a row, the Smart Security Box goes into access denied mode, and only the red LED is turned on.

Table 2: State overview of the LEDs.

Keypad

The Password Manager makes use of a membrane switch module (keypad) to read inputs from the user. The keypad allows the user to enter any integer between 0 and 9 as well as the symbols 'A', 'B', 'C', 'D', '*', and '#'. The Keypad does not have its own class or even `.cpp` file as we found this to be unnecessary, considering that the code for the Password Manager is significantly smaller than the code for the Smart Security Box. The 'A' key is used when adding a new Smart Security Box to the system, and the '#' key is used when editing the password of an existing Smart Security Box.

Servo motor

The Smart Security Box utilizes a servo motor to control the locking mechanism. The servo motor is controlled using the functions `setupMotor` and `setLock` included in the files `motor_module.cpp` and `motor_module.h`. These functions set up the motor for use and locks/unlocks the box (turns the attached arm 90 degrees), respectively.

Web-server communication with Firebase

For this project, we need a server that is able to store and access passwords, as well as document logs. For this, Firebase was chosen, which is a cloud-based platform by Google, and was used in the project as the primary backend, but also gives administrative capabilities with its web console.

Firebase's has a NoSQL database, though without timestamps unlike ThingSpeak, which is a shame, since the feature would be nice to have for logs. However, there are certain **Core Features**, that made us choose this over other platforms such as ThingSpeak.

- A secure HTTP-based API
- Nested and hierarchical JSON data storage, unlike traditional SQL databases or Thingspeak channels which eliminates the need for predefined fields. This makes it easy to create a folder for every boxID.
- Scales efficiently with large volumes of data and users, making it ideal for our Smart Security Box, which is intended to be in larger quantities, similar to locker systems. There is also not a limits on the number of folders, whereas Thingspeak channels have field count and storage size limitations
- Firebase offers real-time synchronization, which means you can watch password entries and logs being made in real-time

The advantage of the nested structure can be seen below, with the Firebase webserver user interface:

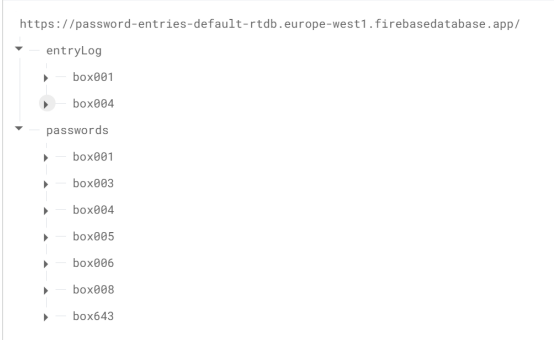


Figure 3: Firebase results

The FirebaseHandler has it's own `.cpp`. The `FirebaseHandler` class encapsulates the interaction with Firebase. Its methods include:

- `logEntry`: Logs an entry status (e.g., success or failure) under a specific boxID.
- `savePassword`: Stores or updates the password for a given boxID.
- `getPassword`: Fetches the most recent password for a specific boxID.

Software

To achieve the goal of this project, we have made use of the Arduino IDE, which has allowed us to program the microcontrollers using C/C++ code. We have sought to make the code that controls the Smart Security Box and the Password Manager as object-oriented as possible. To accomplish this, we have separated the code into a `main.ino` file, and pairs of `.cpp` and `.h` files. The `.cpp` and `.h` files include classes and functions that are used in the `main.ino` file. This is done to make the `main.ino` files as readable as possible. However, along the way, we realized that some parts of the code for the Smart Security Box were difficult to *modulize*, and thus we were forced to include some functions (other than the `setup()` and `loop()` functions) in the corresponding `main.ino` file.

The Smart Security Box

Table 3 shows an overview of the files used to program the Smart Security Box.

File names	Purpose
main.ino	This is the main code file that includes the `setup()` and `loop()` functions.
motor_module.cpp, motor_module.h	These files contain code use to setup and control the servo motor that locks the Smart Security Box.
LCD_screen.cpp, LCD_screen.h	These files include a class with methods used to control the LCD screen of the Smart Security Box.
constants.h	This file contains an `enum` named `State` which is used the main code, to set that state of the system.
Buzzer.cpp, Buzzer.h	Tese files contain code used to control the sound produced by the passive buzzer.
FirebaseHandler.cpp, FirebaseHandler.h	These files include the class `FirebaseHandler` which allows for easy interfacing with the Firebase server.
WiFiManager.cpp, WiFiManager.h	These files include the class `WiFiManager` which is used to manage the ESP8266's connection to the internet.
LEDs.cpp, LEDs.h	These files include the class LEDs, which is used to control the LED lights depending on the state of the system.
potentiometer.cpp, potentiometer.h	These files include logic that handles the reading of values from the potentiometer as well as choosing digits when entering the password.

Table 3: Overview over the files used to control the Smart Security Box

Since the system is setup to have multiple safe boxes, a box identifier is provided for each boxes. Below is an overview of how each component work depending on the state of the program:

- IDLE:
  - LCD screen shows "Smart Security Box".
  - The lock motor is in its closed state.
  - The yellow LED is on.
  - The potentiometer & button awaits action.
- ENTERING_PASSWORD:
  - LCD screen shows "Password: [code], Current Digit: [digit]".
  - The yellow LED is on.
  - The potentiometer saves digit when user pushes button.
  - The buzzer makes a sound whenever button is pushed.
  - The Wifi-module is called, downloading the current password
- WRONG_PASSWORD:
  - LCD screen shows "WRONG PASSWORD!".
  - The red LED is on.

  - The inserted code gets reset
  - The buzzer plays an error sound for a wrong password.
- CORRECT_PASSWORD:
  - LCD screen shows "ACCESS GRANTED!, $\rightarrow OPENBOX\leftarrow$ "
  - The green LED is on.
  - The buzzer plays a melody indicating a correct password.
  - The lock motor is put in a opened state.
  - after 3 seconds, the program state is changed to "BOX_OPEN".
- BOX_OPEN:
  - LCD screen shows "BOX IS OPEN, $\rightarrow CLOSEDOOR\leftarrow$ ".
  - The button awaits action in order to reset the system and go "IDLE" state.
  - The green LED is on.
- ACCESS_DENIED:
  - LCD screen shows "ACCESS DENIED!".
  - The red LED is on.
  - The buzzer plays a continuous siren type alarm sound.
  - The code needs a reset in order to get out this state.

Since almost all functionality for the components are depending on the state of program, the main.ino loop can be rather short. This is by using mostly the chooseNumber() function to control the flow of the program. This is because the potentiometer needs to handle a lot of the usecases.

Figure 4 shows a flow chart of the Smart Security Box system.

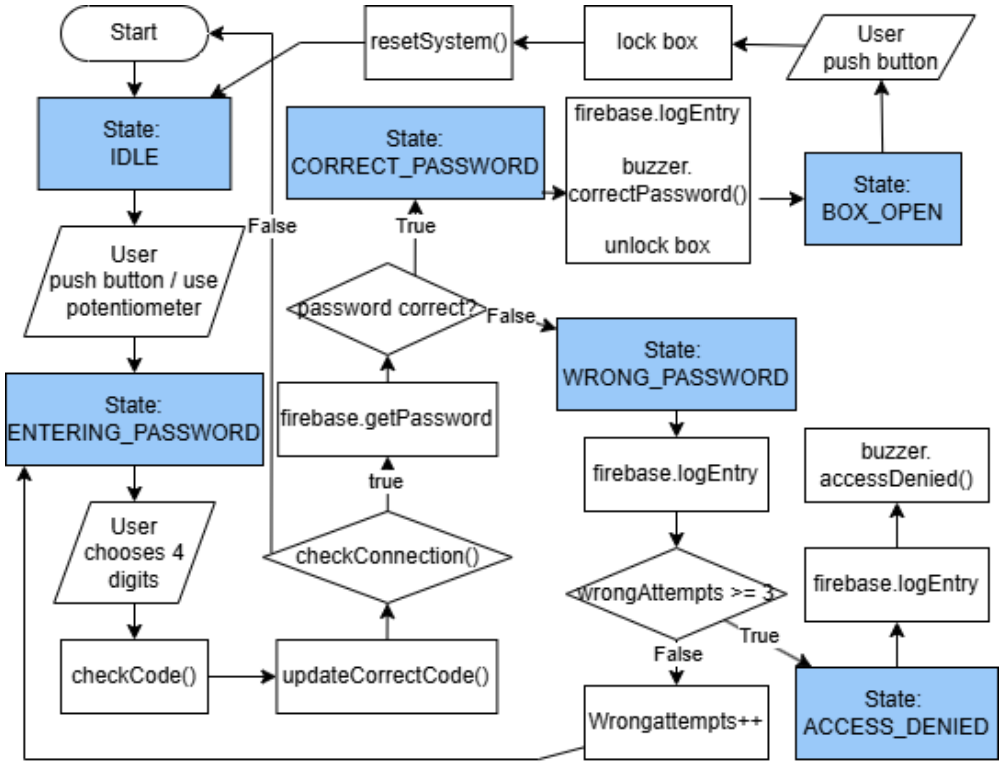


Figure 4: Flow chart of Smart Security Box System

The Password Manager

In a larger scaled system with multiple boxes instead of one, there needs to be some kind of administration of different box ID's and their passwords. In order to do this, we followed a file and function structure similar to the Smart Security Box. The biggest difference between the two that will be described in this section is the content of data sent to the Firebase database. For the Password manager, this is handled by having the cases S_ID - ie. a new ID is entered - and S_PASS- an existing ID is entered and the password needs to be changed. Figure 1 shows the if-statements then executing the desired action based on the entered key.

```
case S_ID:
    if (k == '*') {
        // Clear Box ID
        boxID = "";
        lcd->clear();
        lcd->printLine("Enter Box ID:", 1);
    }
    else if (k == '#') {
        // Validate ID
        if (boxExists(boxID)) {
            password = "";
            state = S_PASS;
            lcd->printLine("Enter Password:", 1);
        } else {
            // Show error briefly, then reset
            lcd->clear();
            lcd->printLine("Invalid Box ID!", 1);
            delay(2000);
            boxID = "";
            lcd->clear();
            lcd->printLine("Enter Box ID:", 1);
        }
    } else if (k == 'A') {
        // etc
    }
    break;
```

Listing 1: Code snippet of the state transition for `S_ID`

Table 4 shows an overview of the files used to program the Password Manager and Figure **??** shows the flow chart for the Password Manager code with all aspects.

File names	Purpose
main.ino	This is the main code file that includes the `setup()` and `loop()` functions.
LCD_screen.cpp, LCD_screen.h	As for the Smart Security Bos, these files include a class with methods used to control the LCD screen of the Password Manager.
FirebaseHandler.cpp, FirebaseHandler.h	These files include the class `FirebaseHandler` which allows for easy interfacing with the Firebase server.
WiFiManager.cpp, WiFiManager.h	These files include the class `WiFiManager` which is used to manage the ESP8266's connection to the internet.

Table 4: Overview over the files used to control the Password Manager

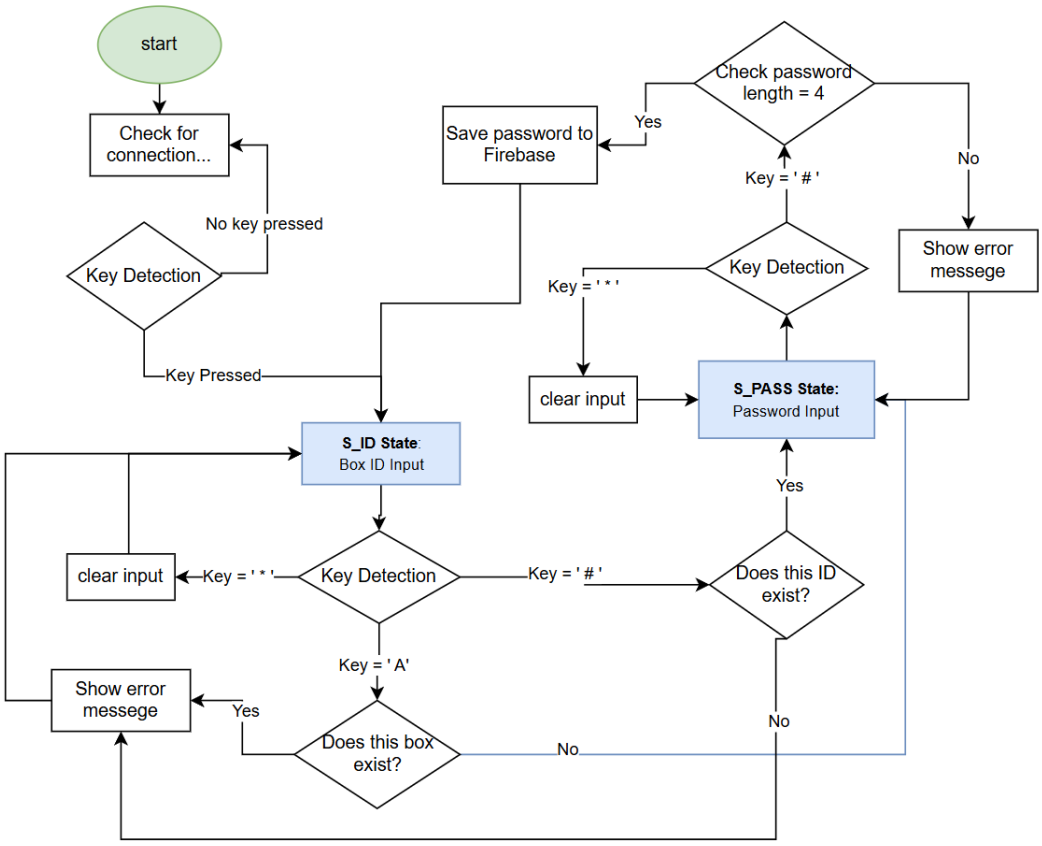


Figure 5: Flow chart of Password Manager

Both the Smart Security Box code and the Password Manager code checks for internet connection multiple times as the connection is required for the boxes to work. This is also shown in Listing 2.

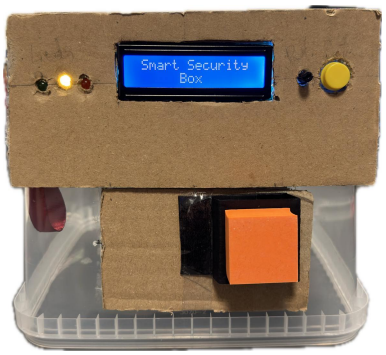
```
/**
 * @brief Checks WiFi connection and attempts to reconnect if lost.
 */
void checkConnection() {
    if (WiFi.status() != WL_CONNECTED) {
        wifiManager->connect();
        if (WiFi.status() != WL_CONNECTED) {
            Serial.println("WiFi lost; resetting...");
            ESP.restart();
        }
    }
}
```

Listing 2: Function to check the WiFi connection status and reconnect if disconnected, with a reset as a fallback.

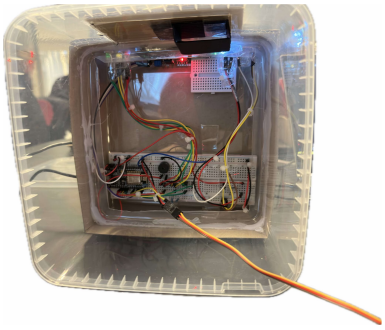
Device setup and structure

The Smart Security Box

The Smart Security Box consists of its electric circuit and components which is built into a shell made from plastic and cardboard. Figure 6a shows the final prototype, and figure 7 shows the electric circuit of the Smart Security Box.



(a) Appearance of the Smart Security Box prototype.



(b) Smart Security Box after removing the plate that separates the electronics from the storage area.

Figure 6: Overview of the Smart Security Box prototype and its internal structure.

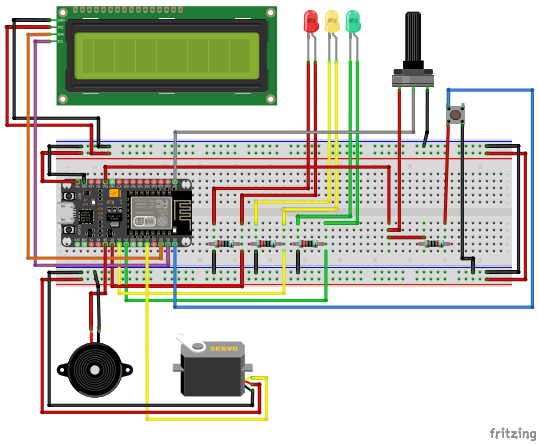


Figure 7: Fritzing diagram of the Smart Security Box circuit

The Password Manager

The password Manager has a simpler circuitry combining the ESP with a keypad and an LCD screen separate from the Smart Security Box and can be seen in Figure 8. As the Password Manager is thought to be an admin panel kept in a secure area, the steps needed to make changes are simple and require no authentication. The deletion of an existing box ID is done directly on Firebase.

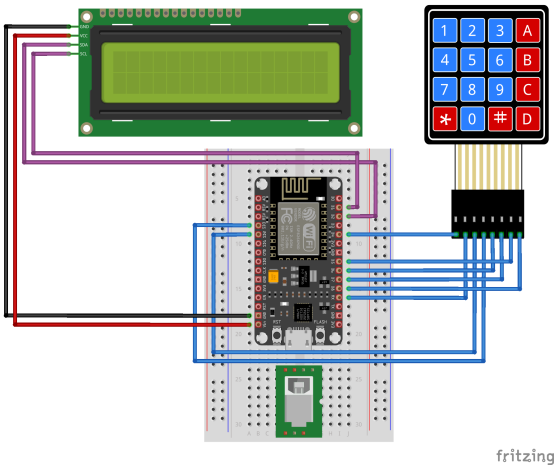


Figure 8: Fritzing diagram of the Password Manager circuit

The main functionalities of this is shown in a flow-chart in Figure 10 designed to be a user overview.

Challenges and comments

ThingSpeak vs Firebase

An issue that was encountered early on in the process was getting the desired functionalities to work on ThingSpeak as this was the database system that was in the plan. As it did not allow dynamic editing of box ID's - because a field would have to be created before a new ID from the Password Manager - and pulling other values than the newest one, it was not optimal for the project. Thus, a lot of time went into researching alternatives, where Firebase proved useful for the goal of this project.

Potentiometer bounds

Another issue encountered was mapping the potentiometerøs input values to values of 0-9 but avoiding "flicker zones" in which the code was unsure of the value as it was close to the middle of two values. This could habe been solved by creating a lower- and upperbound for each value and mapping in these with the extra spaces in between being buffer zones. This issue would have been entirely avoidable by using a digital element instead, such as a keypad as with the Password Manager.

Discussion

An available-offline version of the system would be useful as the current system depends on an active internet connection. Whenever the connection is unstable or lost, it freezes everything and the box or admin cannot be accessed. Additionally, the current system does not need to stash old passwords/logs, however this can become necessary in the future as it grows. Deleting everything immediately can have security risks. Stashing and later on accessing old data is possible with the current Firebase system, however this functionality was not specifically added.

Lastly, as mentioned in the Challenges section, using a digital method for entering the password instead of an analog method can improve the user-friendliness and practicality of the system. The current flickering and risk of entering an unwanted number because of being in the buffer zone can be improved by using a keypad or a touch screen.

Workflow

Figure 9 in the appendix shows how we have used git to branch the project, make separate changes, and merged the changes to the main project. We initially all delegated the individual functionalities among the group members into each of our own branches. The branch was named according to the functionality. Changes or added functionality was put into the title of the git commits. When a finished feature was ready to be merged it was put into the main branch. This could cause some issues but the methods were later adjusted. The "MainLogic" branch was made to ensure a final complete merge was ready to be put into the "main" and final branch.

Conclusion

The Smart Security Box and Password Manager is a useful solution to store valuables safely that combines digital and analog technologies in a practical way. Combining physical elements such as a potentiometer, LCD screens, LED lights with a cloud-based password managing system makes the experience of owning a box and managing passwords for it user-friendly and dynamic.The system reacts according to different states, which makes both the code and the physical box more coherent and systematic. The states included are IDLE state, correct/incorrect password, entering password, access denied and box open. These ensure correct continuous feedback throughout the usage and a smooth experience. The addition of a Password Manager makes a user of the Smart Security Box able to manage their own passwords and ID's as well as creates an admin section. Overall, the Smart Security Box and Password Manager solves a real-life problem by combining multiple aspects of technology in a seemless and user-friendly way.

Doxygen documentation and code

All the code written for this project as well as Doxygen documentation can be found on GitHub via this link: https://github.com/sadafayubb/smart_security_box.

Appendix

Contribution Table

The table bellow summarizes the contributions of each team member.

Team Member	**Contributions**
Joachim	• **Worked on the following code:** – potentiometer.cpp, potentiometer.h – main.ino • **Worked on the following hardware and design:** – Potentiometer – button • **Contributed to the report with the following:** – Flowchart for main.ino – Description for Security Box code – Potentiometer and button descriptions – Workflow – Doyxgen
Sadaf	• **Worked on the following code:** – FirebaseHandler.cpp, FirebaseHandler.h – motor_module.cpp, motor_module.h – WifiManager.cpp, WifiManager.h • **Worked on the following hardware and design:** – Password Manager circuit – Setting up the Firebase webserver • **Contributed to the report with the following:** – Employed Technologies/Web-server communication with Firebase – Device Setup and Structure/Password Manager/ Fritzing diagram & Flowchart

Frederik	• **Worked on the following code:** – LCD_screen.cpp, LCD_screen.h – Buzzer.cpp, Buzzer.h – constants.h – main.ino – password_manager.ino • **Worked on the following hardware and design:** – Smart Security Box plastic/cardboard design – Smart Security Box circuit • **Contributed to the report with the following:** – Employed Technologies/Hardware/LCD screens, Passive Buzzer, LEDs, Keypad, Servo Motor – Employed Technologies/Software/[initial description] – Employed Technologies/Software/[Overview over the files used to control the Smart Security Box and Password Manager (tables)] – Device Setup and Structure/The Smart Security Box (Picture of Smart Security Box, Fritzing diagram of the Smart Security Bos circuit) – Workflow/[Git log tree figure]
Melek	• **Worked on the following code:** – LEDs.cpp, LEDs.h • **Worked on the following hardware and design:** – Password Manager physical design – Password Manager circuit • **Contributed to the report with the following:** – Introduction – Discussion – Description of challenges throughout the project – Conclusion – Code and general description of the Password Manager

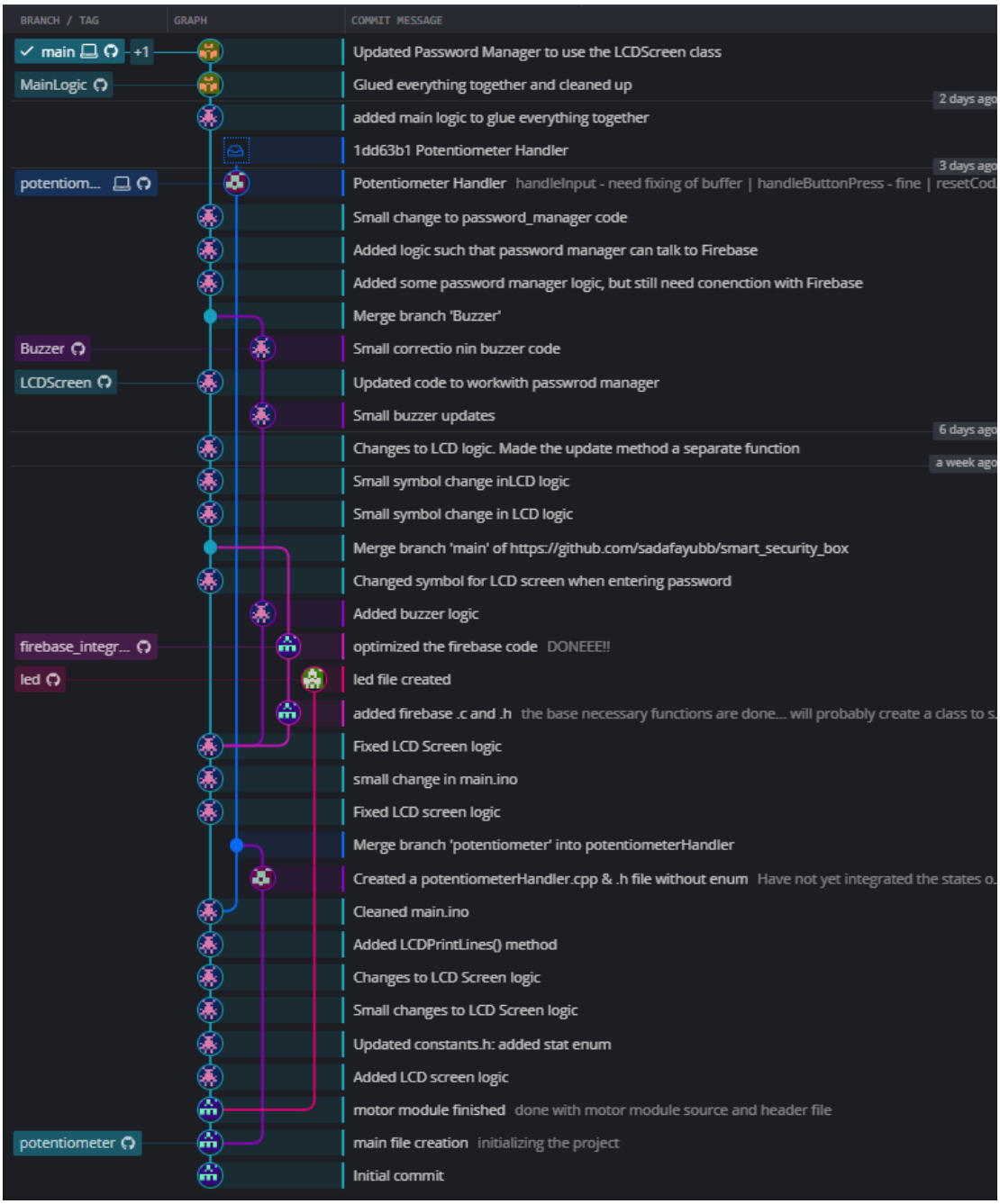


Figure 9: Git log tree showing the branching and merging that has happened during the development.

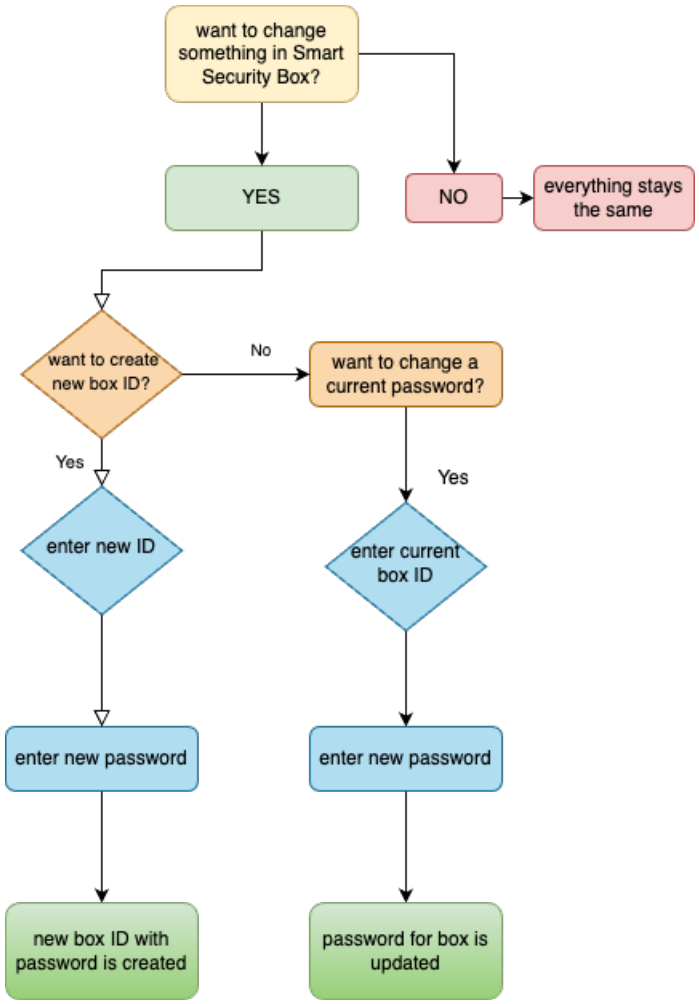


Figure 10: Flow chart of Password Manager