DESIGN LAB PROJECT RAPORT

Development of a Smart Locker System Using Raspberry Pi

authors:

Tomasz Bartłomowicz

Jan Hahn

17.01.2025

1. Introduction

This report outlines the design and implementation of a smart locker system utilizing Raspberry Pi. The system allows users to securely store items and access them via a unique code sent via email. The project combines hardware components, such as servos and sensors, with software solutions implemented in Python. Testing confirmed the system's reliability, demonstrating its potential for practical applications in package delivery and secure storage.

2. Material and Construction

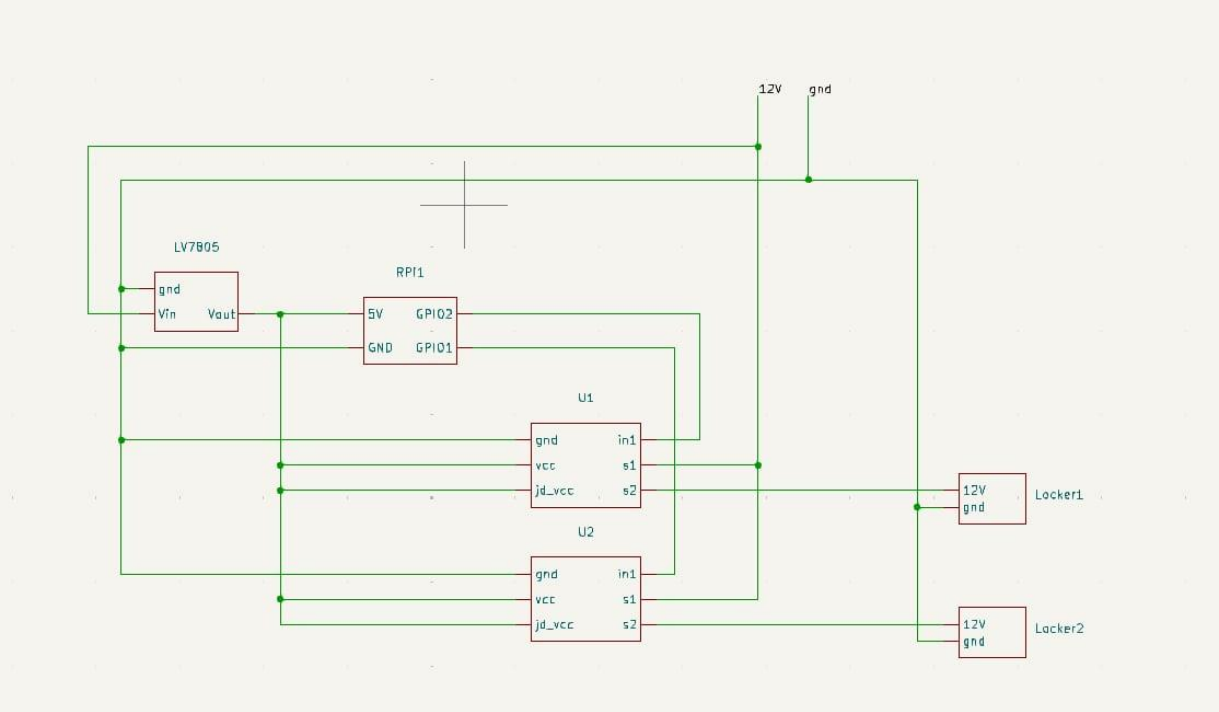
The entire smart locker system was constructed using high-quality wood, ensuring both durability and aesthetic appeal. The lockers are designed to provide secure storage while maintaining a sleek and functional appearance. The dimensions of each locker are as follows: [34 cm] in width, [26 cm] in height, and [30 cm] in depth

3. Hardware description

- Raspberry Pi 4B (4GB)
- Waveshare C Touch screen IPS 7'' 1024x600px
- 12V Electromagnetic lock (2 pcs)
- 12V DC Power supply
- High power relay (2psc)
- L7805ABV 5V Voltage Regulator

The entire system is powered by a 12V power supply, which is connected to a 5V voltage regulator to provide the necessary 5V for the Raspberry Pi. The

GPIO pin on the Raspberry Pi is linked to a relay, which controls the lock by toggling its state between 1 (open) and 0 (closed). This setup ensures stable operation of the Raspberry Pi and lock mechanism, while effectively managing the power requirements.



4. Software description

Software can be divided into frontend and backend. Both implemented in Python

Backend: The backend is responsible for managing the system's logic, data processing, and communication with external services or hardware components. It handles tasks such as database interactions, processing requests, and executing the main program's core functions. In this project, the backend operates on a Raspberry Pi that processes commands and

coordinates the locking mechanism. The backend of our program consists of following files, each containing necessary functions:

- backend.py
- locker.py
- file_handler.py
- mail_sender.py
- codes.txt

Backend (backend.py) initializes hardware settings for controlling lockers using GPIO pins on a Raspberry Pi and manages the main backend loop of the application. It handles requests from a queue, such as checking locker availability, updating codes, and opening lockers, with communication to external components like files and hardware states.

Locker (locker.py) defines a Locker class representing a single locker that can be controlled using GPIO pins on a Raspberry Pi. The class includes methods to open the locker (by controlling the locker pin) and check if the locker is open using a reed switch, which detects whether the locker door is open or closed.

File handler (file_handler.py) defines a FileHandler class responsible for reading and writing locker status information to a file. It includes methods to check a locker's status by reading from the file and update the locker's information by modifying the file when the status changes or new data is added.

Email sender (mail_sender.py) defines a function send_email() that sends an email with a specified subject and body using the SMTP protocol. The function connects to an SMTP server (such as Gmail's server), logs in with the provided credentials, and sends the email to the receiver, including the unlock code in the email body

Frontend: The frontend is the user interface responsible for interacting with the user. It communicates with the backend to send requests (e.g., unlocking commands) and receive feedback. In this project, the frontend consists of a simple interface that allows the user to input a code to open the lock or provide an email address where the unlock code will be sent. The main window displays two buttons: 'STORE' and 'PICK UP'. After selecting one of these options, a prompt to choose a locker appears. The buttons may be disabled depending on whether the locker is occupied (if storing something) or empty (if picking up an item). After selecting a locker, the user is prompted to either provide their email or enter an unlock code. If an email is provided, a confirmation prompt appears. If the unlock code is correct, the locker opens automatically; if the code is incorrect, the user is asked to try again. The frontend of our program consists of following python files, each representing a specific window in the interface.

- main_window.py
- store.py
- pick_up.py
- enter_email.py
- enter_code.py
- email_confirmation.py
- wrong_code.py
- locked_successfully.py
- opened_successfully.py

Each one is a separate Python class implemented using the PyQt5 library

Main Program (main.py) connect frontend and backend together. It sets up a PyQt5-based GUI application and runs the backend logic in a separate process for better performance. It creates two Queue objects to facilitate communication between the main window (frontend) and the backend process. The backend is initialized with the application function, which manages locker interactions, and runs concurrently with the frontend. The MainWindow is displayed to the user, allowing them to interact with the

system while the backend handles requests in parallel. The application continues running until the user closes the window, with the event loop managed by app.exec_()

```python
import sys
from PyQt5.QtWidgets import QApplication
from main_window import MainWindow
from multiprocessing import Process, Queue
from backend import application

def main():

    app = QApplication(sys.argv)
    q = Queue()
    q2 = Queue()
    backend = Process(target=application, args=(q, q2))
    backend.start()
    # Tworzymy i pokazujemy główne menu
    main_menu = MainWindow(q, q2)
    main_menu.show()

    sys.exit(app.exec_())



if __name__ == "__main__":
    main()
```

The rest of the source code is available at the following link to the GitHub repository

GitHub repository

5. Conclusion

The smart locker system successfully integrates both hardware and software components to provide a reliable solution for secure storage. The system performs as expected, with user interactions efficiently handled through a simple interface, and the backend manages locking mechanisms and email communication effectively. While the system shows strong

potential for real-world applications, future work may include enhancing scalability, improving error handling, and exploring alternative authentication methods like QR codes or mobile app integration.

6. Possible future improvements

While the current implementation of the smart locker system provides a functional and secure solution for item storage and retrieval, there are several opportunities for further enhancement. Future improvements could focus on optimizing the system's performance, expanding its features, and making it more adaptable to a wider range of applications. For instance:

- Integration of QR code-based authentication for quicker and contactless
- Addition of a mobile app interface to allow users to interact with the locker system remotely
- Implementation of an admin panel for managing locker statuses and user data.

These upgrades would make the system more scalable and user-friendly, ensuring it meets the demands of diverse use cases.

7. Data sheets and sources

- Raspberry pi
- Relay
- L7805ABV 5V