

Baciu H. Alexandru,



Corduneanu Vlad,



Haralamb Marian



Mail:

alexandru.baciu2@student.tuiasi.ro,
marian.haralamb@student.tuiasi.ro, vlad-florin.corduneanu@student.tuiasi.ro

Proiect:

Smart Safe - Remote Version

Hackster: <https://www.hackster.io/the-team/smart-safe-box-sars-cov-2-version-fb3ba5>

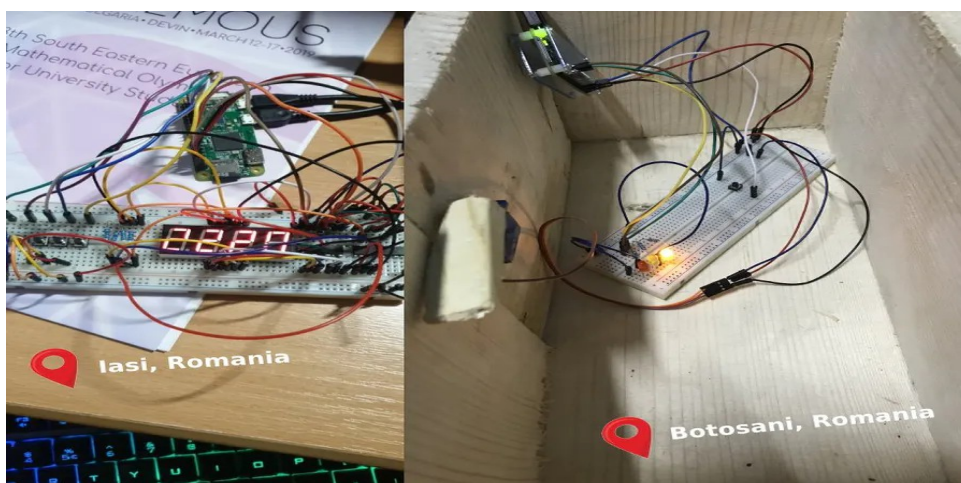
Video: https://www.youtube.com/watch?v=7BznDw_dvoU&feature=youtu.be

Echipa nr. 2: Smart Safe - Remote Version

Elevator Pitch

This project respects the norms of social distancing. Thus, the owner can communicate with the safe from any distance. It is great for this period!

Cover Image



Story

We all know that there are very few people who declare all their income to the state. We always run into the expression "keep your money under the mattress". Technology is evolving, and the money is coming.

This safe comes as a solution to this problem. The safe has 3 states, given by the 3 LEDs. The closed state in which the yellow LED is continuously lit, the wrong password state given by the red LED and the open state given by the green LED. When the safe is open, the password can be changed by introducing a new password.

The controller of the safe has five buttons and a display. Four buttons are used to control the digits value (each button for a digit, when pressed the value is increased by one and from 9 goes to 0) and one is used to validate the password, which is used to




























send the password to the safe module via kafka on a public IP.

Another important part of this project is the kafka server. It must be installed on a cloud server because a public IP is required. This solution was chosen for its lowest difficulty of being used remotely. The VPS comes with debian 10 installed. The next step is to connect to it with putty (SSH2 connection). After that you can follow this tutorial to install and run kafka on your machine:

<https://www.digitalocean.com/community/tutorials/how-to-install-apache-kafka-on-debian-10>

So, how does Kafka work? Kafka stores key-value messages that come from arbitrarily many processes called producers. The data can be partitioned into different "partitions" within different "topics". Within a partition, messages are strictly ordered by their offsets (the position of a message within a partition), and indexed and stored

Hardware components





	Raspberry Pi Zero Wireless	x 2	 
	Digilent Stepper Motor	x 1	
	SparkFun 7-Segment Serial Display - Red	x 1	
	SparkFun Pushbutton switch 12mm	x 6	 
	5 mm LED: Red	x 1	
	5 mm LED: Yellow	x 1	
	3 mm LED: Green	x 1	
	Through Hole Resistor, 150 ohm	x 7	
	Breadboard (generic)	x 2	 
	74HC595 Shift Register (DIP-16)	x 2	
	Male/Male Jumper Wires	x 42	 
	Male/Female Jumper Wires	x 20	 

to

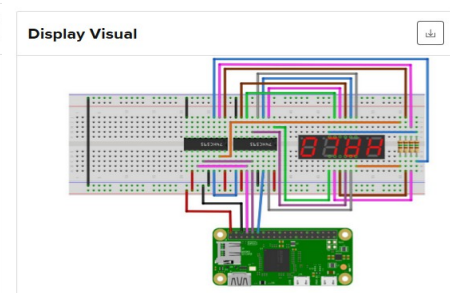
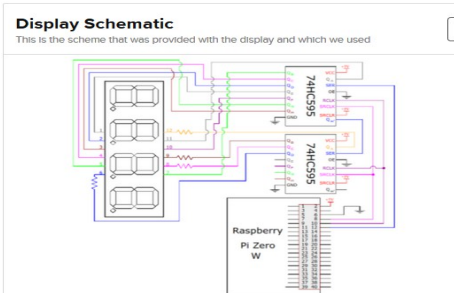
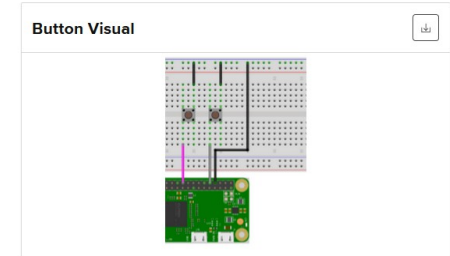
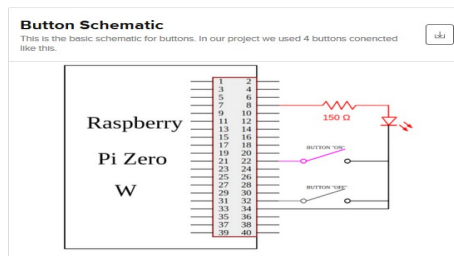
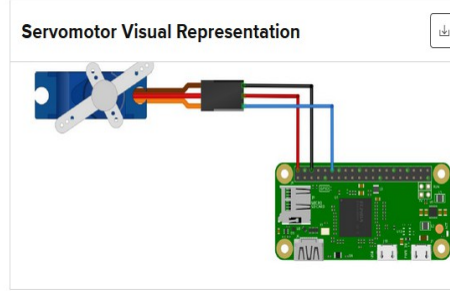
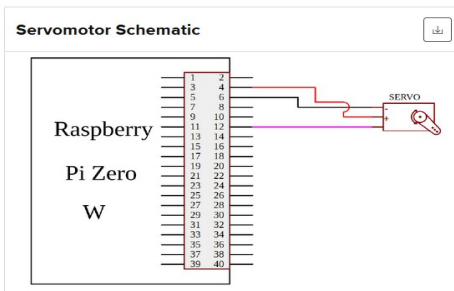
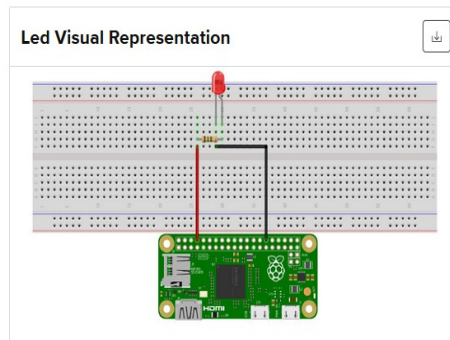
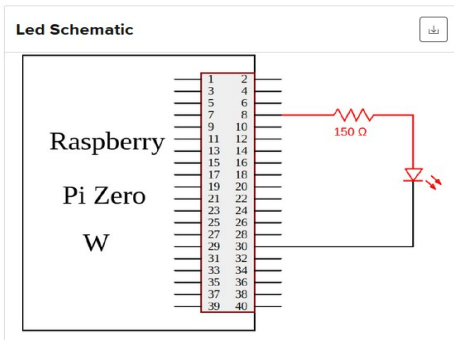
gether with a

timestamp. Other processes called "consumers" can read messages from partitions

Software apps and online services

	Raspberry Pi Raspbian	
	Kafka - Message Broker	
	Cloud VPS	

Schematics



Code

Safe.py

```
#import the libraries
import pigpio
import time
import RPi.GPIO as GPIO
import threading
import ctypes
from kafka import KafkaConsumer
```

```
#define kafka password topic
topic = "password_topic"

#creating consumer for getting messages from password topic
result_consumer = KafkaConsumer(topic, bootstrap_servers='PUT_HERE_PUBLIC_SERVER_IP')

GPIO.setmode(GPIO.BOARD)

pi = pigpio.pi()

#store the pin used by the servo
servo = 18

#set the pin as OUTPUT
pi.set_mode(servo, pigpio.OUTPUT)

#position for closed safe
closed = 500

#position for open safe
opened = 1500

#password for safe
password = "0000"

#define the pin used for button close safe
buttonClose = 7

#define the pin used for red led
redLed = 37

#define the pin used for green led
greenLed = 35

#define the pin used for yellow led
yellowLed = 33

#set the pin for the led as OUTPUT
GPIO.setup(redLed, GPIO.OUT)
GPIO.setup(greenLed, GPIO.OUT)
GPIO.setup(yellowLed, GPIO.OUT)
```

```

#define flag for password set
flag_password = 0

#define password from user
user_password = None

#global variable for reading password thread
shouldClose = False

#set the pins for the buttons as INPUT, and we will
#set the initial value to 0n, or we can say that will be pulled up
GPIO.setup(buttonClose, GPIO.IN, pull_up_down = GPIO.PUD_UP)

#function for opening the safe
def open_safe():
    #put the servo in the position open position
    pi.set_servo_pulsewidth(servo, opened)
    time.sleep(1)

#function for closing the safe
def close_safe():
    #put the servo in the closed position
    pi.set_servo_pulsewidth(servo, closed)
    #sleep 1 second
    time.sleep(1)

#function for #setiing the password
def get_password():
    global user_password
    global should_close
    global flag_password
    while shouldClose == False:
        message = next(result_consumer)
        user_password = message.value.decode("utf-8")
        print(user_password)
        flag_password = 1

#function that raise exception to close the thread
def raise_exception(thread):

```

```

thread_id = thread.ident
res
ctypes.pythonapi.PyThreadState_SetAsyncExc(thread_id, ctypes.py_object(SystemExit))
if res > 1:
    ctypes.pythonapi.PyThreadState_SetAsyncExc(thread_id, 0)
    print('Exception raise failure')

#creating and starting a new thread for password listening
thread = threading.Thread(target = get_password)
thread.start()

#turn on the yellow Led
GPIO.output(yellowLed, GPIO.HIGH)
#turn off the green Led
GPIO.output(greenLed, GPIO.LOW)
#turn off the red Led
GPIO.output(redLed, GPIO.LOW)

try:
    while True:
        #getting user password
        if flag_password == 1:
            flag_password = 0
            #check password availability
            if user_password == password:
                #turn on the green Led
                GPIO.output(greenLed, GPIO.HIGH)
                #turn off the red Led
                GPIO.output(redLed, GPIO.LOW)
                #turn off the yellow Led
                GPIO.output(yellowLed, GPIO.LOW)
                #check if safe it is already opened
                if pi.get_servo_pulsewidth(servo) != opened:
                    open_safe()
            else:
                if pi.get_servo_pulsewidth(servo) == opened:
                    password = user_password
                else:
                    #turn on the red Led
                    GPIO.output(redLed, GPIO.HIGH)

```



```

        #turn off the green Led
        GPIO.output(greenLed, GPIO.LOW)
        #turn off the yellow Led
        GPIO.output(yellowLed, GPIO.LOW)

    #check close button pressed
    if GPIO.input(buttonClose) == GPIO.LOW:
        #turn on the yellow Led
        GPIO.output(yellowLed, GPIO.HIGH)
        #turn off the green Led
        GPIO.output(redLed, GPIO.LOW)
        #turn off the red Led
        GPIO.output(greenLed, GPIO.LOW)
        #check if safe it is already closed
        if pi.get_servo_pulsewidth(servo) != closed:
except KeyboardInterrupt:
    #stop reading thread
    shouldClose = True
    raise_exception(thread)
    #wait for thread to finish work
    thread.join()
    #stop the servo pulses
    pi.set_servo_pulsewidth(servo, 0)
    #stop the connection with the daemon
    pi.stop()
    #clean all the used ports
    GPIO.cleanup()
    #close consumer
    result_consumer.close()

```

Owner.py

```

#import the libraries used
import time
import RPi.GPIO as GPIO
import threading
import sys
from kafka import KafkaProducer

```

```
#kafka variables
topic = "password_topic"
password_producer = KafkaProducer(bootstrap_servers=['PUT_HERE_PUBLIC_SERVER_IP'])

#we will set the pin numbering to the GPIO.BOARD numbering
GPIO.setmode(GPIO.BOARD)

#define the pins used
#Buttons
button1 = 35
button2 = 33
button3 = 31
button4 = 29
buttonVALIDATE = 37

#LCD
dataPin = 12
latchPin = 10
clockPin = 8

cifra1 = 0
cifra2 = 0
cifra3 = 0
cifra4 = 0

p1 = False
p2 = False
p3 = False
p4 = False
pValidate = False

#set the pins for the buttons as INPUT, and we will
#set the initial value to On, or we can say that will be pulled up
#Buttons
GPIO.setup(button1, GPIO.IN, pull_up_down = GPIO.PUD_UP)
GPIO.setup(button2, GPIO.IN, pull_up_down = GPIO.PUD_UP)
GPIO.setup(button3, GPIO.IN, pull_up_down = GPIO.PUD_UP)
GPIO.setup(button4, GPIO.IN, pull_up_down = GPIO.PUD_UP)
GPIO.setup(buttonVALIDATE, GPIO.IN, pull_up_down = GPIO.PUD_UP)

#LCD
GPIO.setup(dataPin, GPIO.OUT)
GPIO.setup(latchPin, GPIO.OUT)
```

```

GPIO.setup(clockPin, GPIO.OUT)
#set the initial state for the pins as LOW
GPIO.output(dataPin, GPIO.LOW)
GPIO.output(latchPin, GPIO.LOW)
GPIO.output(clockPin, GPIO.LOW)

#create byte variables that have only one segment set to HIGH
a = 0b00000001
b = 0b00000010
c = 0b00000100
d = 0b00001000
e = 0b00010000
f = 0b00100000
g = 0b01000000
dot = 0b10000000

#declare variables that will be send to the first shift
#register and turn on the LED segments and show a digit
#from 0 to 9, with or without the dot
zero = 191      #0b10111111
zero_no_dot = 63 #0b00111111
one = 134      #0b10000110
one_no_dot = 6 #0b00000110
two = 219     #0b11011011
two_no_dot = 91 #0b01011011
three = 207   #0b11001111
three_no_dot = 79 #0b01001111
four = 230    #0b11100110
four_no_dot = 102 #0b01100110
five = 237    #0b11101101
five_no_dot = 109 #0b01101101
six = 253     #0b11111101
six_no_dot = 125 #0b01111101
seven = 135   #0b10000111
seven_no_dot = 7 #0b00000111
eight = 255  #0b11111111
eight_no_dot = 127 #0b01111111
nine = 239   #0b11101111
nine_no_dot = 111 #0b01101111

numbers=[zero_no_dot,one_no_dot,two_no_dot,three_no_dot,
        four_no_dot,five_no_dot,six_no_dot,
        seven_no_dot,eight_no_dot,nine_no_dot,
        dot]

```

```
#declare a variable that will store the current digits active
digit = 0
```

```
def Digit(x):
    global digit
    if x == 1:
        digit = 14 #0b00001110
    elif x == 2:
        digit = 13 #0b00001101
    elif x == 3:
        digit = 11 #0b00001011
    elif x == 4:
        digit = 7 #0b00000111
    elif x == 5:
        digit = 0 #0b00000000
```

```
#function to send the values to the shift registers
```

```
def shift(buffer):
    #make the global variable available
    global digit
```

```
#send the bits to the second shift register
```

```
for i in range(0,8):
    GPIO.output(dataPin, (128 & (digit << i)))
    GPIO.output(clockPin, GPIO.HIGH)
    time.sleep(0.00005)
    GPIO.output(clockPin, GPIO.LOW)
```

```
#send the bits to the first shift register
```

```
for i in range(0,8):
    GPIO.output(dataPin, (128 & (buffer << i)))
    GPIO.output(clockPin, GPIO.HIGH)
    time.sleep(0.00005)
    GPIO.output(clockPin, GPIO.LOW)
```

```
#shift the bits
```

```
GPIO.output(latchPin, GPIO.HIGH)
time.sleep(0.00005)
```

```
GPIO.output(latchPin, GPIO.LOW)
```

```
def refresh():
```

```
    global cifra1
```

```
    global cifra2
```

```
    global cifra3
```

```
    global cifra4
```

```
    global numbers
```

```
    Digit(1)
```

```
    shift(numbers[cifra1])
```

```
    Digit(2)
```

```
    shift(numbers[cifra2])
```

```
    Digit(3)
```

```
    shift(numbers[cifra3])
```

```
    Digit(4)
```

```
    shift(numbers[cifra4])
```

```
def checkButtons():
```

```
    global cifra1
```

```
    global cifra2
```

```
    global cifra3
```

```
    global cifra4
```

```
    global p1
```

```
    global p2
```

```
    global p3
```

```
    global p4
```

```
    global pValidate
```

```
while shouldClose == False:
```

```
    #check if the 1 button was pressed
```

```
    if GPIO.input(button1) == GPIO.LOW:
```

```
        #turn on the Led
```

```
        if p1 == False:
```

```
            cifra1 = (cifra1+1)%10
```

```
            p1 = True
```

```
        else:
```

```
            p1 = False
```

```

#check if the 2 button was pressed
if GPIO.input(button2) == GPIO.LOW:
    #turn off the Led
    if p2 == False:
        cifra2 = (cifra2+1)%10
        p2 = True
else:
    p2 = False

#check if the 3 button was pressed
if GPIO.input(button3) == GPIO.LOW:
    #turn off the Led
    if p3 == False:
        cifra3 = (cifra3+1)%10
        p3 = True
else:
    p3 = False

#check if the 4 button was pressed
if GPIO.input(button4) == GPIO.LOW:
    #turn off the Led
    if p4 == False:
        cifra4 = (cifra4+1)%10
        p4 = True
else:
    p4 = False

#check if the VALIDATE button was pressed
if GPIO.input(buttonVALIDATE) == GPIO.LOW:
    #turn off the Led
    if pValidate == False:
        pValidate = True
        #make password as string
        password = cifra1.__str__() + cifra2.__str__() + cifra3.__str__() +
cifra4.__str__()

        #send password via kafka to the door
        password_message = bytearray(password, encoding="utf-8")

```

```

        password_headers = []
        password_producer.send(topic=topic, value=password_message,
headers=password_headers)
        password_producer.flush()

        #reset digits
        cifra1 = 0
        cifra2 = 0
        cifra3 = 0
        cifra4 = 0
    else:
        pValidate = False

    time.sleep(0.1)

#Thread stuff
shouldClose = False
thread = threading.Thread(target = checkButtons)

try:
    #call the "Digit" function in order to update
    #the value of the "digit" variable
    Digit(5)

    #send two byte values to the shift registers
    #after this, all the A LEDs will turn on, because
    #all the digits are turned on (they are set to LOW)
    shift(dot)
    time.sleep(1)

    thread.start()

while True:
    refresh()

```

```
except KeyboardInterrupt:
```

```
    pass
```

```
    shouldClose = True
```

```
    thread.join()
```

```
digit = 0
```

```
shift(0)
```

```
#clean all the used ports
```

```
GPIO.cleanup()
```

Tasks Details

Baciu H. Alexandru	Corduneanu Vlad	Haralamb Marian
Implementing circuit for stepper and closing button	Implementing circuit for the owner (keyboard with display)	Safe exterior build: walls, floor and door
Building the safe lock and adding logic for safe lock	Adding logic for the owner input and sending data to the server	Adding led's to project
Logic for getting password from server and validate it	Logic for showing display to owner	Logic for the state of the safe