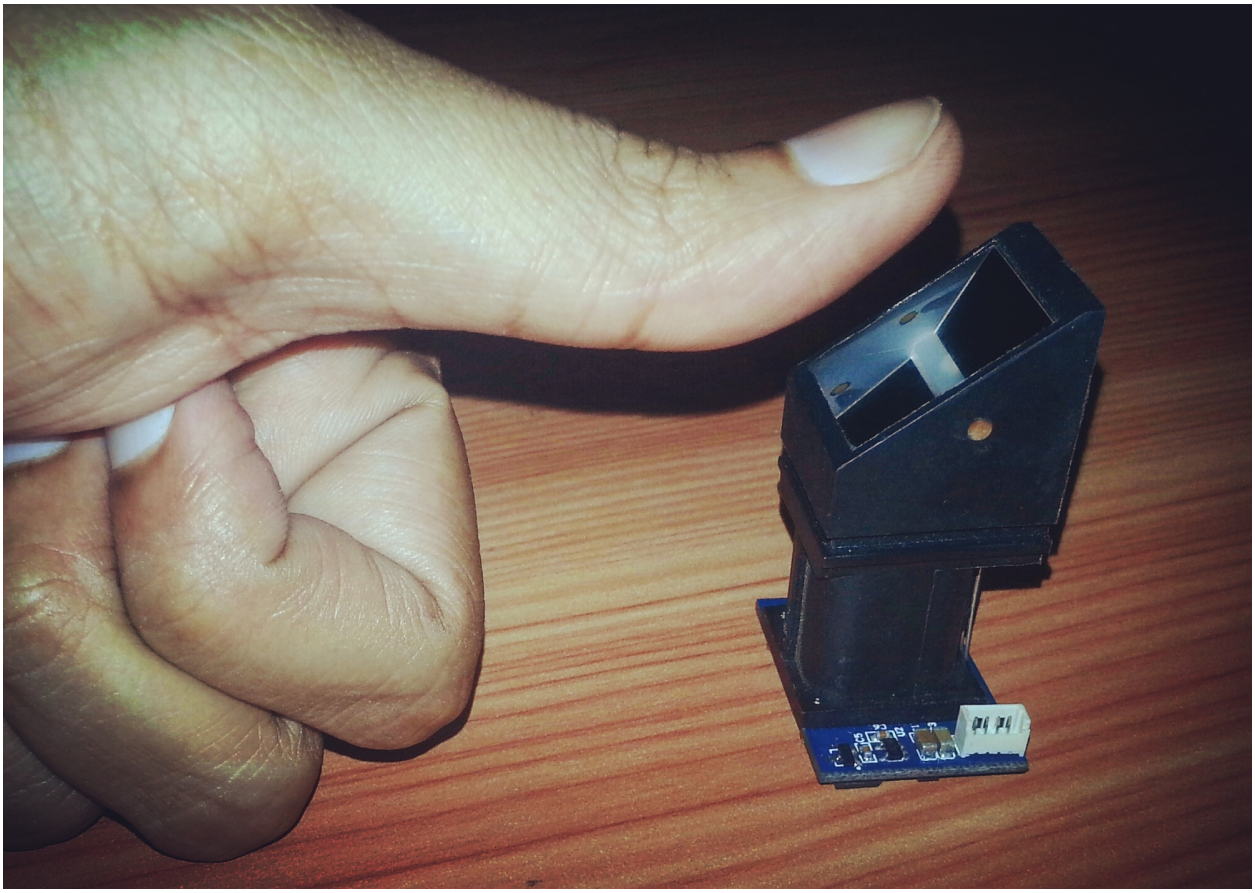
finger_sphinx Documentation

Release v0.0.1

Vinay Chandra

Sep 10, 2017

Contents:



CHAPTER 1

WHATS NEW !

Fingers enrolled using the R305 module and stored in a database along with the person's name. Next just by typing the persons name, his/her finger template can be stored or deleted in the R305 module ready for authentication.

Limitations

1. Storage space: Datasheet says storage capacity is 256.
2. Physical presence: Physical presence of person is required everytime we delete and store somebody's finger template in the R305 module.
3. Time: Storing large number of finger templates is time consuming and tedious.
4. Backup: Damge or loss of the R305 module results in entire data loss.

Limitations overcome with database

1. Storage space: Storage capacity is limited by your hard disc space. You can store almost any number of finger templates.
2. Physical presence: Once the finger template of a person is stored in databse, we can delete and store the person's finger template in the R305 module without his/her physical presence.
3. Time: Finger print templates in database can be stored in the R305 module in a blink of an eye.
4. Backup: If R305 module is damaged or lost, data can be retrived from databse and stored on a new R305 module.

CHAPTER 2

R305 FINGERPRINT MODULE BASICS

Click to Download datasheet

Operation principle:

Fingerprint processing includes two parts: fingerprint enrollment and fingerprint matching (the matching can be 1:1 or 1:N).When enrolling, user needs to enter the finger two times. The system will process the two time finger images, generate a template of the finger based on processing results and store the template. When matching, user enters the finger through optical sensor and system will generate a template of the finger and compare it with templates of the finger library.

Hardware connection:

Via serial interface, the Module may communicate with MCU of 3.3V or 5V power: TD (pin 3 of P1) connects with RXD (receiving pin of MCU), RD (pin 4 of P1) connects with TXD (transferring pin of MCU).

Communication Protocol:

We can communicate with the module using a packet of hex codes in a specific format. The data package format for communication is shown below.

Header	Adder	Package identifier	Package length	Package content (instuction/data/Parameter)	Checksum

Thus a data package transferred to or from the module will include a Header, Address, Package Identifier, Package Length, Package Content and Checksum.

The example shown below asks the module to collect the finger image.

eg.EF 01 FF FF FF FF 01 00 03 01 00 05

Header: EF 01

Address: FF FF FF FF

Package Identifier: 01

Package Length: 00 03

Package Content: 01

Checksum: 00 05

For more details on the data package contents see the table below:

Name	Symbol	Length	Description	
Header	Start	2 bytes	Fixed value of 0xEF01; High byte transferred first.	
Adder	ADDER	4 bytes	Default value is 0xFFFFFFFF, which can be modified by command. High byte transferred first and at wrong adder value, module will reject to transfer.	
Package identifier	PID	1 byte	01H	Command packet;
			02H	Data packet; Data packet shall not appear alone in executing processs, must follow command packet or acknowledge packet.
			07H	Acknowledge packet;
			08H	End of Data packet.
Package length	LENGTH	2 bytes	Refers to the length of package content (command packets and data packets) plus the length of Checksum(2 bytes). Unit is byte. Max length is 256 bytes. And high byte is transferred first.	
Package contents	DATA	—	It can be commands, data, command's parameters, acknowledge result, etc. (fingerprint character value, template are all deemed as data);	
Checksum	SUM	2 bytes	The arithmetic sum of package identifier, package length and all package contens. Overflowing bits are omitted. high byte is transferred first.	

CHAPTER 3

PYTHON CODE

Note: Use Python2.7 version

Note: Linux users install LAMP. Windows user install WAMP.

Note: Create a database. I have created a database named 'fingerdb'. Create a table. I have created a table named 'fingertb' with fields 'slno','name' and 'finger'

Upload to database

Code below uploads finger template to database:

```
import serial, time, datetime
import struct            #Convert between strings and binary data
import sys
import os
import binascii
import mysql.connector
cnx=mysql.connector.connect(user='root',password='',host='localhost',database=
↪'fingerdb')       # connect to MySql database
cur=cnx.cursor()

#ser = serial.Serial('/dev/ttyUSB0',57600)      # serial communication in Linux
ser = serial.Serial("COM6", baudrate=9600, timeout=1)   #serial communication in␣
↪Windows

pack = [0xef01, 0xffffffff, 0x1]        # Header, Address and Package Identifier
```

```
def readPacket():       # Function to read the Acknowledge packet
        time.sleep(1)
        w = ser.inWaiting()
        ret = []
        if w >= 9:
                s = ser.read(9)         # Partial read to get length
                ret.extend(struct.unpack('!HIBH', s))
                ln = ret[-1]

                time.sleep(1)
                w = ser.inWaiting()
                if w >= ln:
                        s = ser.read(ln)
                        form = '!' + 'B' * (ln - 2) + 'H'       # Specifying byte size
                        ret.extend(struct.unpack(form, s))
        return ret

def readPacket1():      # Function to read the Acknowledge packet
        time.sleep(1)
        w = ser.inWaiting()
        ret = []
        form = 'B' * 700
        s = ser.read(700)
        t=binascii.hexlify(s)   # convert to hex
        u=t[24:]
        cur.execute("insert into fingertb(name,finger) values('%s','%s')" %(name,u) )␣
↪    # upadate database
        cnx.commit()
        v=binascii.unhexlify(u)
        form1='B'*688
        ret1=[]
        ret1.extend(struct.unpack(form1, v))
        ret.extend(struct.unpack(form, s))

def writePacket(data):          # Function to write the Command Packet
        pack2 = pack + [(len(data) + 2)]
        a = sum(pack2[-2:] + data)
        pack_str = '!HIBH' + 'B' * len(data) + 'H'
        l = pack2 + data + [a]
        s = struct.pack(pack_str, *l)
        ser.write(s)


def verifyFinger():     # Verify Module?s handshaking password
        data = [0x13, 0x0, 0, 0, 0]
        writePacket(data)
        s = readPacket()
        return s[4]

def genImg():   # Detecting finger and store the detected finger image in ImageBuffer
        data = [0x1]
        writePacket(data)
        s = readPacket()
        return s[4]

def img2Tz(buf):        # Generate character file from the original finger image in␣
↪ImageBuffer and store the file in CharBuffer1 or CharBuffer2.
        data = [0x2, buf]
```

```
        writePacket(data)
        s = readPacket()
        return s[4]

def regModel():         # Combine information of character files from CharBuffer1 and
↪CharBuffer2 and generate a template which is stroed back in both CharBuffer1 and
↪CharBuffer2.
        data = [0x5]
        writePacket(data)
        s = readPacket()
        return s[4]

def UpChar(buf):        # Upload the character file or template of CharBuffer1/
↪CharBuffer2 to upper computer
        data = [0x8,buf]
        writePacket(data)
        s = readPacket1()

print ("Type done to exit")
name=raw_input("Enter name : ")
while (name!='done'):

        if verifyFinger():
                print 'Verification Error'
                sys.exit(0)

        print 'Put finger',
        sys.stdout.flush()

        time.sleep(1)
        while genImg():
                time.sleep(0.1)
                print '.',
                sys.stdout.flush()

        print ''
        sys.stdout.flush()

        if img2Tz(1):
                print 'Conversion Error'
                sys.exit(0)

        print 'Put finger again',
        sys.stdout.flush()

        time.sleep(1)
        while genImg():
                time.sleep(0.1)
                print '.',
                sys.stdout.flush()

        print ''
        sys.stdout.flush()

        if img2Tz(2):
                print 'Conversion Error'
                sys.exit(0)
```

```
        if regModel():
                print 'Template Error'
                sys.exit(0)

        if UpChar(2):
                print 'Template Error'
                sys.exit(0)

        name=raw_input("Enter name : ")
```

Download to R305 fingerprint module

Note: After entering 'name', enter integer values 1,2,3.... for 'store id'.

Warning: Entering the same 'store id' for differnet names will overwrite the finger templete stored.

Code below downloads the finger template from database to R305 fingerprint module:

```
import serial, time, datetime, struct
import sys
import os
import mysql.connector
import binascii
cnx=mysql.connector.connect(user='root',password='',host='localhost',database=
↪'fingerdb')
cur=cnx.cursor()
ser = serial.Serial("COM6", baudrate=9600, timeout=1)
pack = [0xef01, 0xffffffff, 0x1]

def readPacket():
        time.sleep(1)
        w = ser.inWaiting()
        ret = []
        if w >= 9:
                s = ser.read(9) #partial read to get length
                ret.extend(struct.unpack('!HIBH', s))
                ln = ret[-1]

                time.sleep(1)
                w = ser.inWaiting()
                if w >= ln:
                        s = ser.read(ln)
                        form = '!' + 'B' * (ln - 2) + 'H'
                        ret.extend(struct.unpack(form, s))

        return ret

def readPacket1():

        time.sleep(1)
        w = ser.inWaiting()
```

```
        time.sleep(1)
        pack_str='B'* 688

        cur.execute("select finger from fingertb where name='%s'"%name)
        row=cur.fetchone()
        srow = str(row[0])
        v=binascii.unhexlify(srow)
        form1='B'*688
        ret1=[]
        ret1.extend(struct.unpack(form1, v))
        x=ret1
        s = struct.pack(pack_str, *x)
        ser.write(s)
        if store(idno):
                print 'store error'
                sys.exit(0)
                print "Enrolled successfully at id %d"%j

def writePacket(data):
        pack2 = pack + [(len(data) + 2)]
        a = sum(pack2[-2:] + data)
        pack_str = '!HIBH' + 'B' * len(data) + 'H'
        l = pack2 + data + [a]
        s = struct.pack(pack_str, *l)
        ser.write(s)


def verifyFinger():
        data = [0x13, 0x0, 0, 0, 0]
        writePacket(data)
        s = readPacket()
        return s[4]

def DownChar(buf):      # download character file or template from upper computer to␣
↪the specified buffer of Module
        data = [0x9,buf]
        writePacket(data)
        s = readPacket1()

def store(id):          # store the template of specified buffer (Buffer1/Buffer2) at␣
↪the designated location of Flash library
        data = [0x6, 0x1, 0x0, id]
        writePacket(data)
        s = readPacket()
        return s[4]

name=raw_input('enter the name please')
idno=int(raw_input('enter the store id'))
if verifyFinger():              # Verify Password
        print 'Verification Error'
        sys.exit(0)

if DownChar(1):
        print 'Template Error'
        sys.exit(0)
```

Search and authenticate

Code below scans the finger and authenticates:

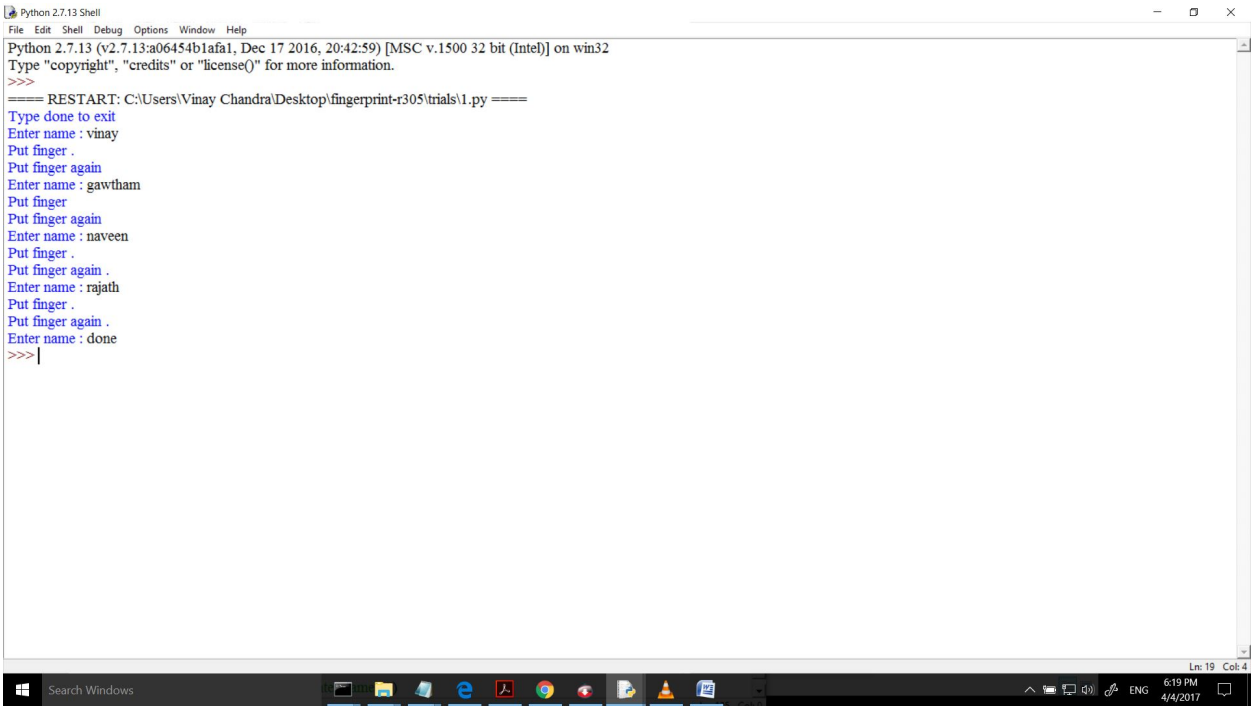
```
import serial, time, datetime, struct
import sys

ser = serial.Serial("COM6", baudrate=9600, timeout=1)

pack = [0xef01, 0xffffffff, 0x1]

def readPacket():
        time.sleep(1)
        w = ser.inWaiting()
        ret = []
        if w >= 9:
                s = ser.read(9)          #partial read to get length
                ret.extend(struct.unpack('!HIBH', s))
                ln = ret[-1]

                time.sleep(1)
                w = ser.inWaiting()
                if w >= ln:
                        s = ser.read(ln)
                        form = '!' + 'B' * (ln - 2) + 'H'
                        ret.extend(struct.unpack(form, s))
        return ret


def writePacket(data):
        pack2 = pack + [(len(data) + 2)]
        a = sum(pack2[-2:] + data)
        pack_str = '!HIBH' + 'B' * len(data) + 'H'
        l = pack2 + data + [a]
        s = struct.pack(pack_str, *l)
        ser.write(s)


def verifyFinger():
        data = [0x13, 0x0, 0, 0, 0]
        writePacket(data)
        s = readPacket()
        return s[4]

def genImg():
        data = [0x1]
        writePacket(data)
        s = readPacket()
        return s[4]

def img2Tz(buf):
        data = [0x2, buf]
        writePacket(data)
        s = readPacket()
        return s[4]

def search():   # search the whole finger library for the template that matches the␣
↪one in CharBuffer1 or CharBuffer2
```

```
        data = [0x4, 0x1, 0x0, 0x0, 0x0, 0x5]
        writePacket(data)
        s = readPacket()
        return s[4:-1]


if verifyFinger():
        print 'Verification Error'
        sys.exit(-1)

print 'Put finger',
sys.stdout.flush()

time.sleep(1)
for _ in range(5):
        g = genImg()
        if g == 0:
                break

        print '.',
        sys.stdout.flush()

print ''
sys.stdout.flush()
if g != 0:
        sys.exit(-1)

if img2Tz(1):
        print 'Conversion Error'
        sys.exit(-1)

r = search()
print 'Search result', r
if r[0] == 0 :
        print 'Authentication Successful'
        sys.exit(0)

else:
        print 'Authentication fail'
sys.exit(1)
```

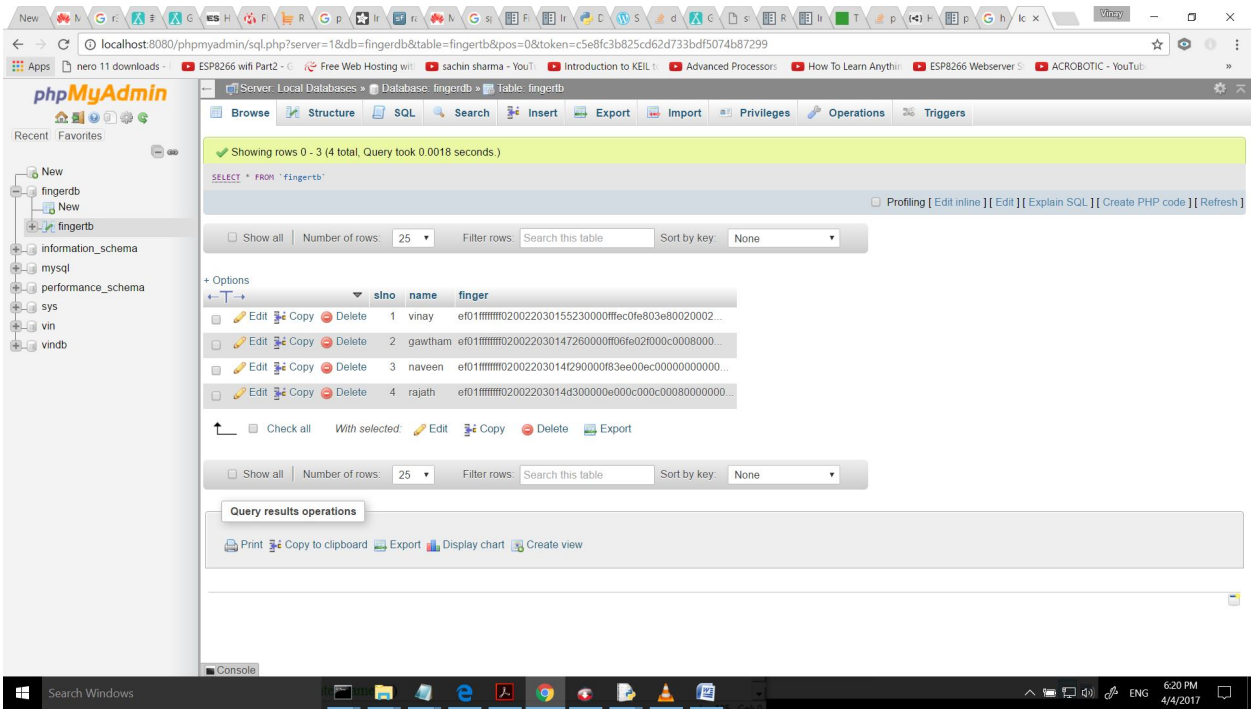
CHAPTER 4

RESULTS

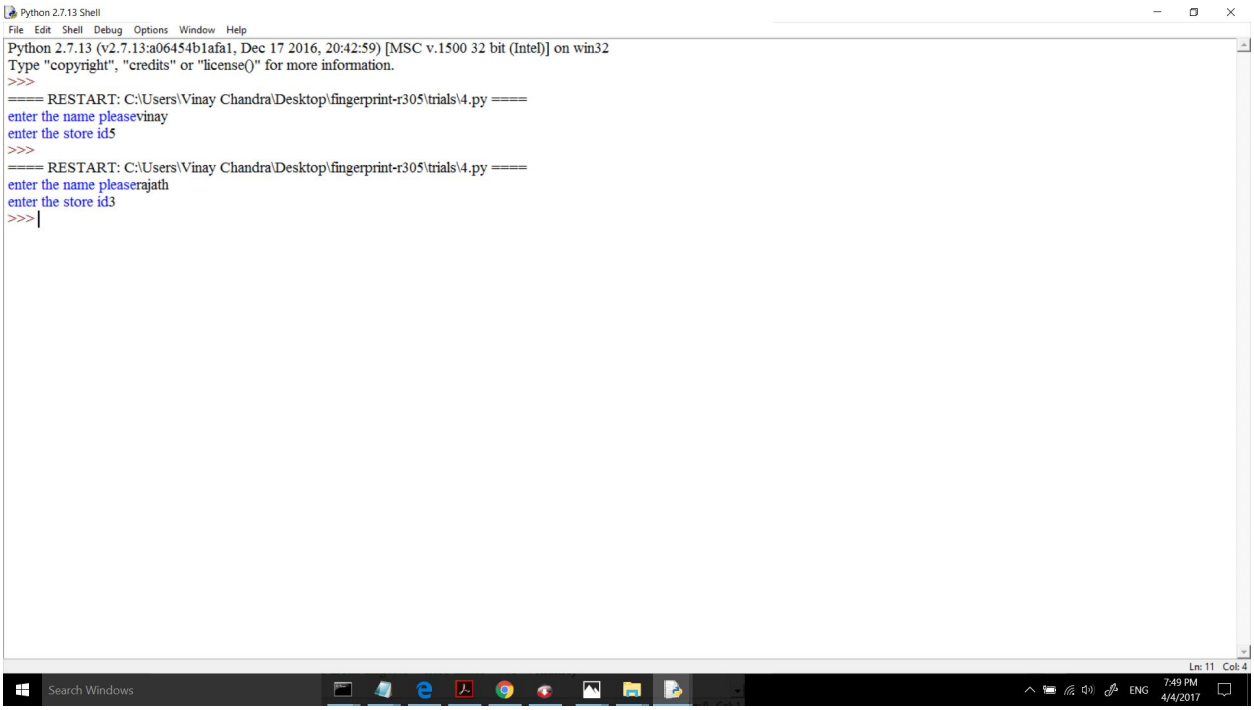
Uploading finger templates to databse



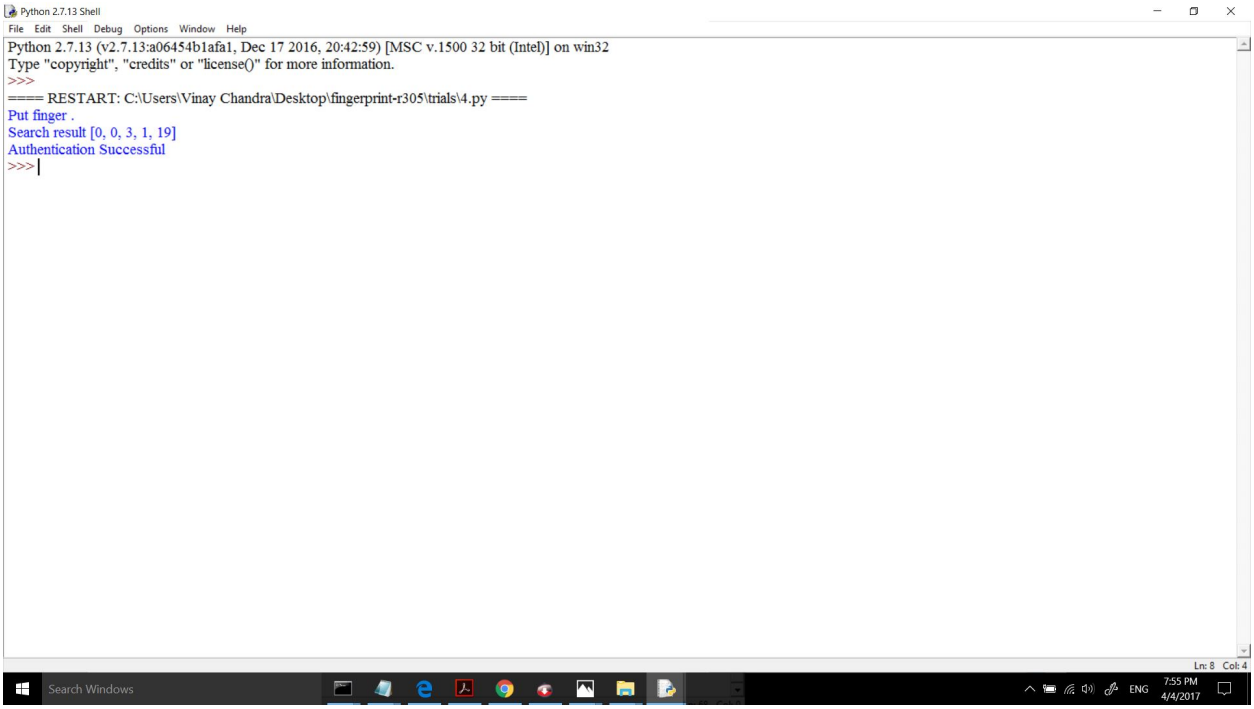
Databse Updated



Downlading finger templates from databse to R305 module



Authenticating finger



CHAPTER 5

Indices and tables

- genindex
- modindex
- search