# PROJECT REPORT FILE

## BCE – P663

**SUBMITTED BY:**
Chirag Patel
Reg. No. - 196301029
B.TECH, CSE, VI SEM

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING FACULTY OF ENGINEERING AND TECHNOLOGY
GURUKUL KANGRI UNIVERSITY
2022-2023

# INDEX

# Abstract


- Associated with **Unique Identification Authority of India (UIDAI).**

- A **Real world application** named as **Authenication - Reimagined** developed using Unique Identification Authority of India (UIDAI) **Authentication services.**

- **Authentication of individual without disclosing AADHAR NUMBER.**

- Consist of **Resident(user's device)** and **Verifier(Authority)** application with **smooth UI experience.**

- Tech Stack used **-: Python (Back-end) , Tkinter (Front-end).**

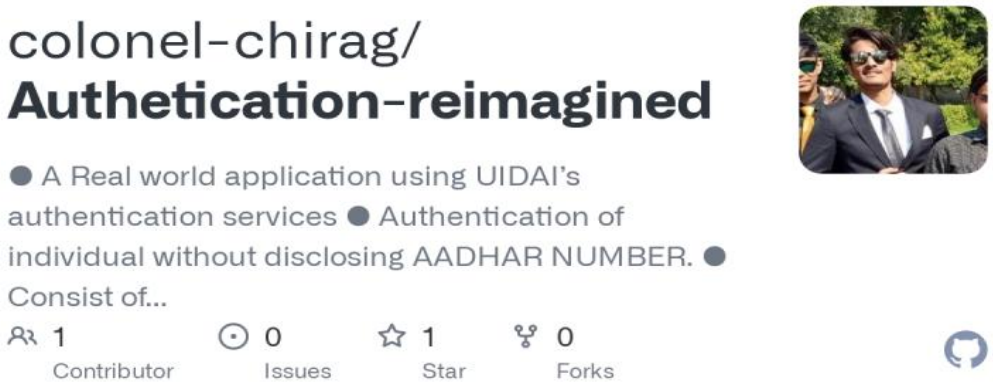
- Aadhar authentication **API (Application Programming Interface).**

- **Demonstration-Link-:**
  https://www.youtube.com/watch?v=XPzVYwW4Edg

- **GitHub-Repo-:**
  https://github.com/colonel-chirag/Authetication-reimagined

# Detailed Description of Project

## Authentication - Reimagined




- Authentication - Reimagined a real world application is developed to counter the situation faced by general public regarding sharing their Aadhar Card number with the particular authority in order to authenticate oneself to avail their services.

- The main motive of this application is to give user a smooth Aadhar Service Experience for their authentication without revealing their Aadhar Number to particular Authority.

- Application is developed using Python Language for Back - end and Tkinter for Front - end.

- Entire Application based on the use of UIDAI authentication services.

- Entire project is consist of two application -:
  (1) Resident Application
  (2) Verifier Application

- Resident Application is installed on the user's device. It is basically a offline KYC (Know Your Confirmation) Pitcher, which takes users data and convert it into a encrypted package.

- This encrypted package is fully secured by encryption code and user can share this file with authority to verify themselves.

- Verifier Application is installed on the Authority's device which works without internet and can only be accessed by authority to verify the users and it cannot access UIDAI servers. Verifier application overchecks the digital signature with the KYC inserted by the users in resident application. If both the digital signature and KYC matches user is verified.

# Tools & Technologies

- This Application is based on **Python** Language.

- **Python** is a high-level, interpreted, general-purpose programming language. Its design philosophy emphasizes code readability with the use of significant indentation.

- Python also comes with an Integrated development environment (IDE) called IDLE, which is more beginner-oriented.

- Other shells, including IDLE and IPython, add further abilities such as improved auto-completion, session state retention and syntax highlighting.

- **Tkinter** is used for **Front - end** .

- Tkinter is the standard GUI library for Python. Python when combined with Tkinter provides a fast and easy way to create GUI applications. Tkinter provides a powerful object-oriented interface to the Tk GUI toolkit.

- The tkinter package (“Tk interface”) is the standard Python interface to the Tcl/Tk GUI toolkit. Both Tk and tkinter are available on most Unix platforms, including macOS, as well as on Windows systems.

# Dependencies

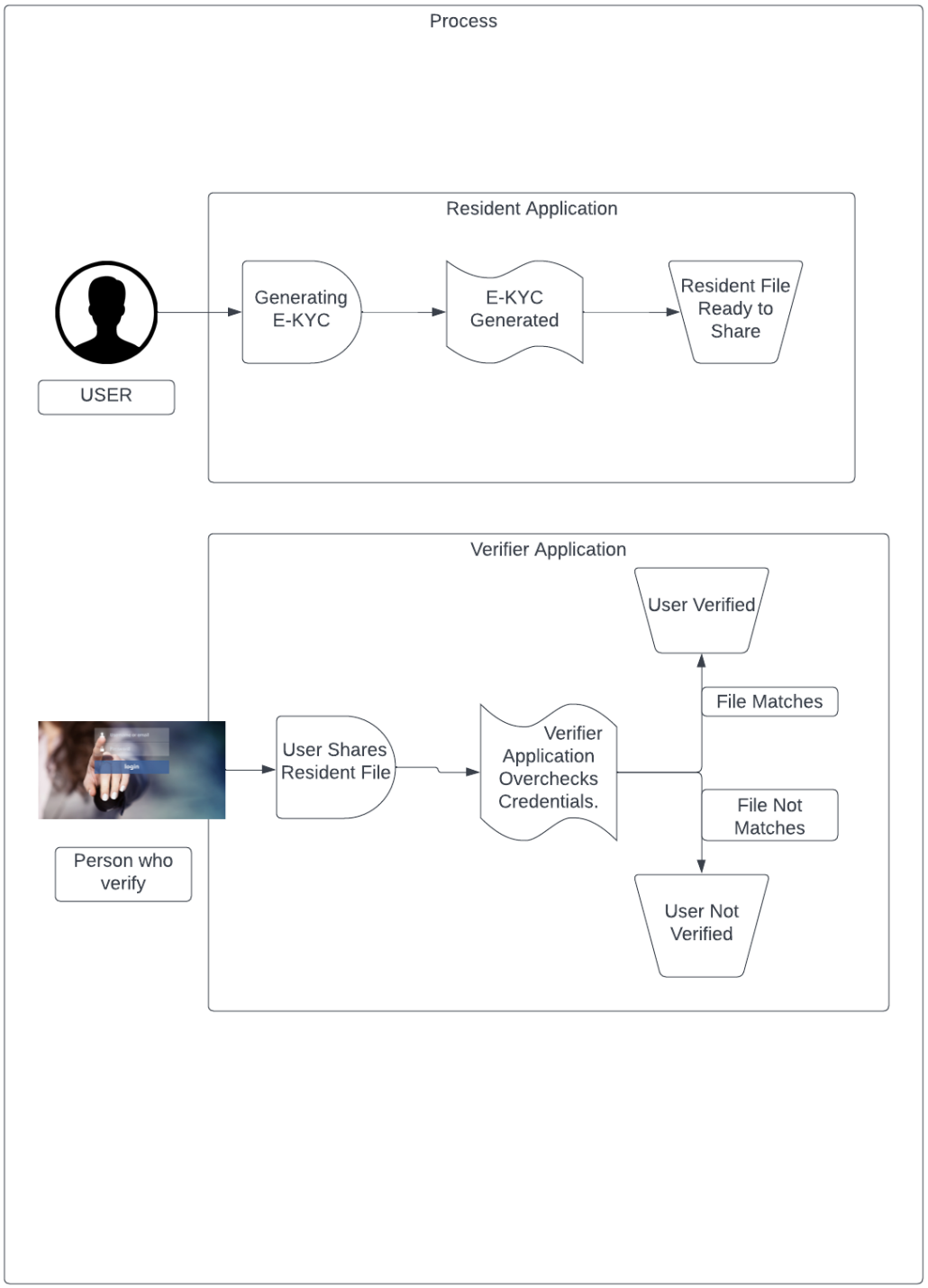
**Following are the factors upon which this application depends -:**

- **Internet Connectivity** only for verifier application which is installed on users end.
- **Valid Aadhar Card Number** for verifier application.
- **OTP (One Time Password)** feature at users end.
- **Person** who access the verfier application , communicate with user and authenticate the user.

# Use Cases

- **Booking Flight / Train Tickets,** Validating a person while authenticating.
- **Banking KYC,** e - Kyc can be done using these services.
- **Check-in / Check-out at Hotels,** Validating a person while authenticating.

# Process Flow



# Code Snippets

## Resident.py

233 lines (194 sloc) | 5.43 KB

```python
from tkinter import *
from tkinter import simpledialog
import requests
import json
import uuid
import base64
import tempfile
import hashlib
from PIL import Image,ImageTk
from cryptography.fernet import Fernet

def genCap(canvas,status,capt):
        global ret
        global cap

        url = 'https://stage1.uidai.gov.in/unifiedAppAuthService/api/v2/get/captcha'
        headers = {
                'Content-Type': 'application/json'
        }
        data = {
                'langCode':'en',
                'captchaLength':'3',
                'captchaType':'2'
        }
        response = requests.post(url=url,data=json.dumps(data),headers=headers)
        jsonData = json.loads(response.text)

        if jsonData and jsonData['status'] == 'Success':
                temp = tempfile.TemporaryFile()
                temp.write(base64.b64decode(jsonData['captchaBase64String']))
                img = Image.open(temp)
                img = img.resize((180,50))
                cap = ImageTk.PhotoImage(img)
                canvas.itemconfigure(capt,image = cap)

                ret = [True,jsonData['captchaTxnId']]

        else:
                ret = [False]
                canvas.itemconfigure(status,text = "Captcha Generation Error",fill="#d32828")

def fetchKyc(canvas,otp,scode,uid,fn,ln,m):
        url = 'https://stage1.uidai.gov.in/eAadhaarService/api/downloadOfflineEkyc'
        if ret[0]==True:
                txNum = ret[1]
        else:
                txNum = ""

```

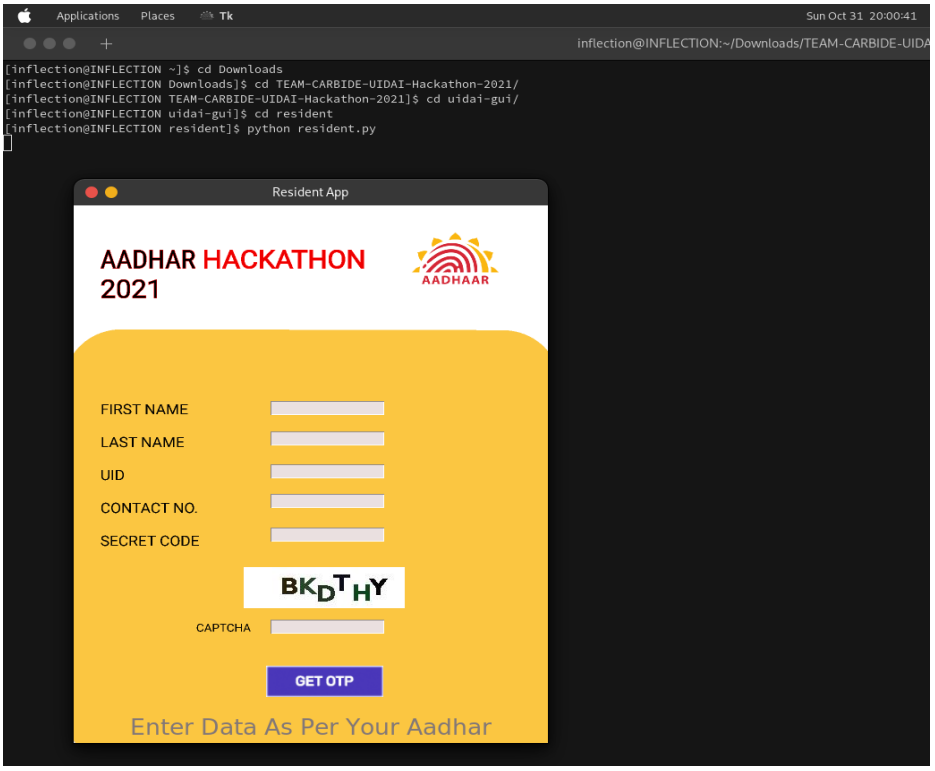
# Verifier.py

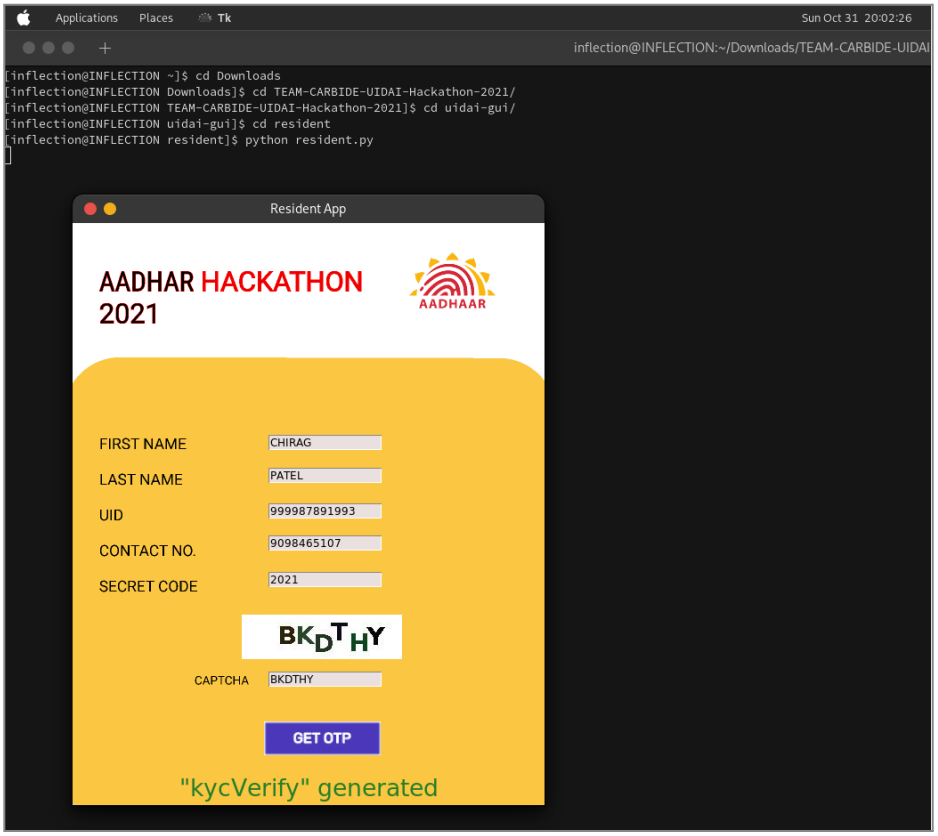
225 lines (187 sloc) | 5.23 KB

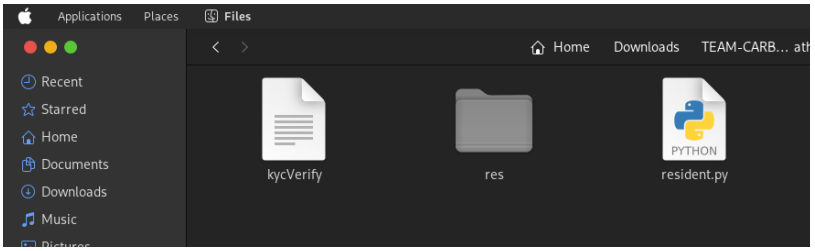
```
from tkinter import *
from tkinter import filedialog
import hashlib
import base64
import json
import tempfile
import zipfile
from xml.dom import minidom
from cryptography.fernet import Fernet
from signxml import XMLVerifier
from PIL import Image, ImageTk

def browse(canvas,filep):
        global kycPath
        kycPath = filedialog.askopenfilename()
        if len(kycPath) < 40:
                canvas.itemconfigure(filep,text = kycPath)
        else:
                canvas.itemconfigure(filep,text = kycPath[-39:])

def delentry(name,dob,gender):
        name.config(state = "normal")
        name.delete(0,END)
        name.config(state = "disabled")

        gender.config(state = "normal")
        gender.delete(0,END)
        gender.config(state = "disabled")

        dob.config(state = "normal")
        dob.delete(0,END)
        dob.config(state = "disabled")


def verify(canvas,name,status,dob,gender,face,code):

        delentry(name,dob,gender)

        global kycPath
        global face_img

        try:
                key = hashlib.md5(code.encode()).hexdigest()
                key = base64.urlsafe_b64encode(key.encode())
                decKyc = Fernet(key).decrypt(open(kycPath,'rb').read())
        except Exception as e:
                canvas.itemconfigure(status,text = "WRONG CODE",fill="#d32828")

```

# Working Snippets

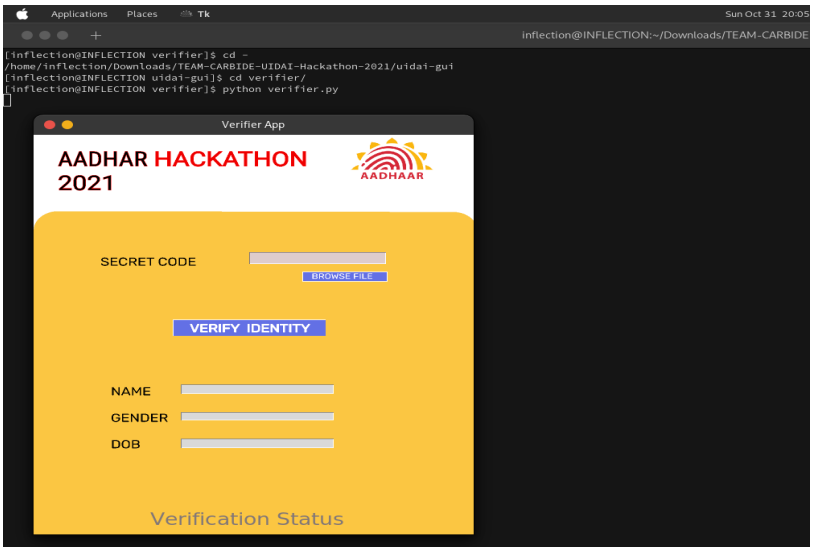
## Resident Application

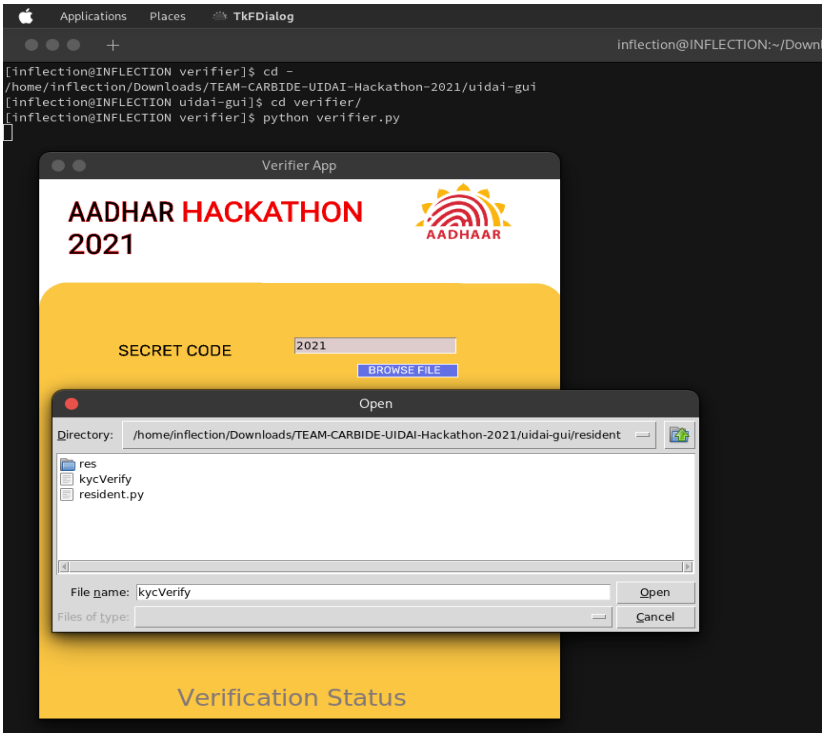


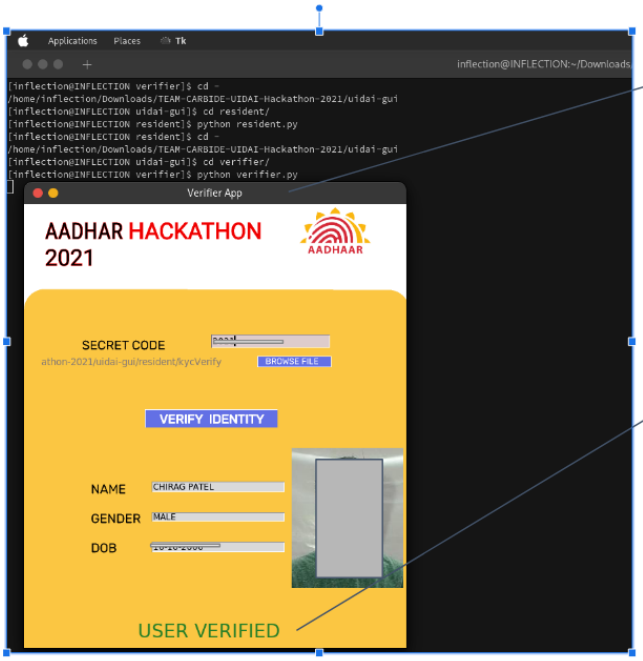






# Verifier Application






**VERIFIER APP**

**USER VERIFIED**

Indicates that user's KYC matches with the data input by user without using internet and without disclosing aadhar number .

# Developer Profile

https://github.com/colonel-chirag

https://chiragpatel.carrd.co

@imchiragpatel7