**SOURCE CODE**

from flask import \*

import cryptography as randompseudocode

import json

from werkzeug.utils import secure\_filename

from werkzeug.datastructures import FileStorage

import os

import re

from IPFS server.storage.blob import \*

import webbrowser

import sqlite3

config={"IPFS server\_storage\_connectionstring":"DefaultEndpointsProtocol=https;AccountName=mdae;AccountKey=SOkVTdcWFLj3WdqvAxoBRLZAzYc02hbfR4K/Zu7pwWt9Wn0hSh778PCK48LG1GUVr4rkKY2zuEYs+AStRY46OQ==;EndpointSuffix=core.windows.net",\

"videos\_container\_name":"pdfstore","source\_folder":"static/upload"}

basedir = os.path.abspath(os.path.dirname(\_\_file\_\_))

app=Flask(\_\_name\_\_)

app.secret\_key = 'any random string'

UPLOAD\_FOLDER='static/upload'

app.config['UPLOAD\_FOLDER'] = UPLOAD\_FOLDER

from web3 import Web3

from solcx import compile\_standard, install\_solcx

def soliditycontract(e):

import json

install\_solc("0.6.0")

with open("./SimpleStorage.sol", "r") as file:

simple\_storage\_file = file.read()

compiled\_sol = compile\_standard(

{

"language": "Solidity",

"sources": {"SimpleStorage.sol": {"content": simple\_storage\_file}},

"settings": {

"outputSelection": {

"\*": {

"\*": ["abi", "metadata", "evm.bytecode", "evm.bytecode.sourceMap"]

}

}

},

},

solc\_version="0.6.0",

)

with open("compiled\_code.json", "w") as file:

json.dump(compiled\_sol, file)

bytecode = compiled\_sol["contracts"]["SimpleStorage.sol"]["SimpleStorage"]["evm"][

"bytecode"

]["object"]

# get abi

abi = json.loads(

compiled\_sol["contracts"]["SimpleStorage.sol"]["SimpleStorage"]["metadata"]

)["output"]["abi"]

w3 = Web3(Web3.HTTPProvider('HTTP://127.0.0.1:8545'))

chain\_id = 1337

print(w3.is\_connected())

my\_address = e[0]

private\_key =e[1]

# initialize contract

SimpleStorage = w3.eth.contract(abi=abi, bytecode=bytecode)

nonce = w3.eth.get\_transaction\_count(my\_address)

# set up transaction from constructor which executes when firstly

transaction = SimpleStorage.constructor().build\_transaction(

{"chainId": chain\_id, "from": my\_address, "nonce": nonce}

)

signed\_tx = w3.eth.account.sign\_transaction(transaction, private\_key=private\_key)

tx\_hash = w3.eth.send\_raw\_transaction(signed\_tx.rawTransaction)

tx\_receipt = w3.eth.wait\_for\_transaction\_receipt(tx\_hash)

tx\_receipt = "".join(["{:02X}".format(b) for b in tx\_receipt["transactionHash"]])

return tx\_receipt

def createkey():

import random

def generate\_random\_permutation(string):

# Convert the string to a list of characters

chars = list(string)

# Traverse the list from the last element to the second element

for i in range(len(chars), 1, -1):

# Generate a random index between 0 and i (exclusive)

random\_index = random.randint(0, i - 1)

# Swap the current character with the randomly selected character

chars[i-1], chars[random\_index] = chars[random\_index], chars[i-1]

# Join the characters back into a string

randomized\_string = ''.join(chars)

# Return the generated random permutation

return randomized\_string

# Generate a random string of 100 characters

characters = 'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789'

random\_string = ''.join(random.choices(characters, k=44))

# Generate a random permutation of the characters

random\_permutation = generate\_random\_permutation(random\_string)

return random\_permutation

@app.route('/')

def cloudloger():

return render\_template("home.html")

@app.route('/login')

def cloudlogin():

return render\_template("login.html")

@app.route('/register')

def cloudregister():

return render\_template("register.html")

@app.route('/logout')

def cloudlogout():

dir = 'static/download'

for f in os.listdir(dir):

os.remove(os.path.join(dir, f))

return redirect('/')

def cloudinsertdata(user,pas,email,mobile,name,role,add,keyadd):

from cryptography.fernet import Fernet

import sqlite3

mydb=sqlite3.connect("next.db")

#mydb.execute("drop table compilerlogin")

#mydb.execute("create table compilerlogin(id int,name varchar(50),username varchar(50),password varchar(50),email varchar(10),mobile varchar(50),key text,role varchar(100),address varchar(100),keyadd varchar(100),approve int)")

e=mydb.execute('select count(\*) from compilerlogin')

x=e.fetchall()[0][0]

reg=(x+1,name,user,pas,email,mobile,Fernet.generate\_key(),role,add,keyadd,0)

re='insert into compiler login (id,name,username,password,email,mobile,key,role,address,keyadd,approve)values(?,?,?,?,?,?,?,?,?,?,?)'

mydb.execute(re,reg)

mydb.commit()

r=mydb.execute("select \* from compilerlogin").fetchall()

print(r)

mydb.close()

@app.route('/csignup' ,methods=["post"])

def cloudcregister():

user=request.form["user"]

pas=request.form["pass"]

email=request.form["email"]

mobile=request.form["mobile"]

name=request.form["name"]

role=request.form["role"]

address=request.form["add"]

keyadd=request.form["keyadd"]

cloudinsertdata(user,pas,email,mobile,name,role,address,keyadd)

return redirect("/")

def cloudcmylogin(user,pas):

import sqlite3

mydb=sqlite3.connect("next.db")

if(user=="admin" and pas=="admin"):

session["name"]="admin"

session["role"]="admin"

else:

reg=(user,pas)

x=mydb.execute("select \* from compilerlogin").fetchall()

print(x)

try:

x='select role,approve from compilerlogin where username=? and password=?'

r=mydb.execute(x,reg)

myresult = r.fetchone()

if(myresult[1]!=0):

session["name"]=user

session["role"]=myresult[0]

else:

session["role"]="wrong"

mydb.close()

except:

session["role"]="wrong"

@app.route('/comlogin',methods=["post"])

def cloudcomlogin():

user=request.form["user"]

pas=request.form["password"]

session["user"]=user

r=cloudcmylogin(user,pas)

print(session["role"])

if (session["role"]=="wrong"):

return redirect("/")

elif(session["role"]=="admin"):

return redirect("/admin")

elif (session["role"]=="user"):

return redirect("/loger")

elif (session["role"]=="hospital"):

return redirect("/loger")

elif (session["role"]=="doctor"):

return redirect("/loger")

else:

return redirect("/")

@app.route('/loger')

def cloudcompiler():

v=session["user"].upper()

return render\_template("index.html",rx=v)

@app.route('/admin')

def admin():

v=session["user"].upper()

mydb=sqlite3.connect("next.db")

x=mydb.execute("select \* from compilerlogin").fetchall()

print(x)

return render\_template("admin.html",rx=x)

@app.route('/approve')

def approve():

username = request.args['i']

mydb=sqlite3.connect("next.db")

mydb.execute("update compilerlogin set approve=? where address=?",('1',username))

mydb.commit()

x=mydb.execute("select address,keyadd from compilerlogin where address=?",([username])).fetchone()

soliditycontract(x)

return redirect("/admin")

def cloudcsven(val,key):

from cryptography.fernet import Fernet

fernet = Fernet(key)

# opening the original file to encrypt

with open(val, 'rb') as file:

original = file.read()

# encrypting the file

encrypted = fernet.encrypt(original)

# opening the file in write mode and

# writing the encrypted data

with open(val, 'wb') as encrypted\_file:

encrypted\_file.write(encrypted)

def cloudcsvendata(val,key,n,x):

print(val)

from cryptography.fernet import Fernet

fernet = Fernet(key)

# opening the original file to encrypt

with open(val, 'rb') as file:

original = file.read()

# encrypting the file

encrypted = fernet.encrypt(original)

# opening the file in write mode and

# writing the encrypted data

v="static/encrypt/shared\_"+n+"\_"+x

print(v)

with open(v, 'wb') as encrypted\_file:

encrypted\_file.write(encrypted)

def cloudcsvde(val,key):

from cryptography.fernet import Fernet

fernet = Fernet(key)

# opening the encrypted file

with open(val, 'rb') as enc\_file:

encrypted = enc\_file.read()

# decrypting the file

decrypted = fernet.decrypt(encrypted)

# opening the file in write mode and

# writing the decrypted data

with open(val, 'wb') as dec\_file:

dec\_file.write(decrypted)

def generate\_initial\_state(key, size):

state = []

# Use a hash function (e.g., SHA-256) to generate a pseudorandom sequence based on the key

hash\_value = hash(key)

# Convert the hash value to binary representation

binary\_value = bin(hash\_value)[2:].zfill(size)

# Create the initial state as a list of binary digits

state = [int(bit) for bit in binary\_value[1:]]

return state

def apply\_rules(state):

# Define your cellular automaton rules here

# This is a simple example using the XOR operation with the neighboring cells

new\_state = []

length = len(state)

for i in range(length):

left = state[(i - 1) % length]

center = state[i]

right = state[(i + 1) % length]

# Apply XOR operation

new\_bit = left ^ (center ^ right)

new\_state.append(new\_bit)

return new\_state

def encrypt\_file(filename, key):

print(filename)

# Read the file contents in binary mode

with open("static/upload/"+filename, 'rb') as file:

file\_data = file.read()

# Convert the file contents to binary representation

binary\_data = ''.join(format(byte, '08b') for byte in file\_data)

# Generate the initial state based on the key

size = len(binary\_data)

state = generate\_initial\_state(key, size)

# Apply the cellular automaton rules to encrypt the binary data

encrypted\_data = ''

for bit in binary\_data:

# Apply XOR operation between the file bit and the state bit

encrypted\_bit = int(bit) ^ state.pop(0)

encrypted\_data += str(encrypted\_bit)

state = apply\_rules(state) # Update the state using the cellular automaton rules

# Convert the encrypted binary data back to bytes

encrypted\_bytes = bytes(int(encrypted\_data[i:i + 8], 2) for i in range(0, len(encrypted\_data), 8))

# Write the encrypted data to a new file

encrypted\_filename = filename

with open(encrypted\_filename, 'wb') as file:

file.write(encrypted\_bytes)

print('File encrypted successfully:', encrypted\_filename)

dir = app.config['UPLOAD\_FOLDER']

for f in os.listdir(dir):

os.remove(os.path.join(dir, f))

def decrypt\_file(filename, key):

# Read the encrypted file contents in binary mode

with open(filename, 'rb') as file:

encrypted\_data = file.read()

# Convert the encrypted data to binary representation

binary\_data = ''.join(format(byte, '08b') for byte in encrypted\_data)

# Generate the initial state based on the key

size = len(binary\_data)

state = generate\_initial\_state(key, size)

# Apply the reverse cellular automaton rules to decrypt the binary data

decrypted\_data = ''

for bit in binary\_data:

# Apply XOR operation between the encrypted bit and the state bit

decrypted\_bit = int(bit) ^ state.pop(0)

decrypted\_data += str(decrypted\_bit)

state = apply\_rules(state) # Update the state using the cellular automaton rules

# Convert the decrypted binary data back to bytes

decrypted\_bytes = bytes(int(decrypted\_data[i:i + 8], 2) for i in range(0, len(decrypted\_data), 8))

# Write the decrypted data to a new file

decrypted\_filename = filename.replace('.encrypted', '.decrypted')

with open(decrypted\_filename, 'wb') as file:

file.write(decrypted\_bytes)

print('File decrypted successfully:', decrypted\_filename)

def cloudgetmyfile(user,e):

connect\_str=config["IPFS server\_storage\_connectionstring"]

blob\_service\_client = BlobServiceClient.from\_connection\_string(connect\_str)

container\_name =config["videos\_container\_name"]

container\_client = blob\_service\_client.get\_container\_client(container\_name)

blob\_list = container\_client.list\_blobs()

va=[]

for blob in blob\_list:

if user in blob.name:

va.append(str(blob.name).split("\_\_")[1])

return json.dumps(va)

@app.route('/getdetails',methods=["Post"])

def cloudgetdetails():

user=request.form["rid"]

x=cloudgetmyfile(user,'de')

return x

def clouddownloadfileuser(k):

connect\_str=config["IPFS server\_storage\_connectionstring"]

blob\_service\_client = BlobServiceClient.from\_connection\_string(connect\_str)

container\_name =config["videos\_container\_name"]

container\_client = blob\_service\_client.get\_container\_client(container\_name)

blob\_list = container\_client.list\_blobs()

local\_path = "static/download"

for blob in blob\_list:

if k in blob.name:

local\_file\_name = blob.name

blob\_client = blob\_service\_client.get\_blob\_client(container=container\_name, blob=local\_file\_name)

download\_file\_path = os.path.join(local\_path, local\_file\_name)

with open(download\_file\_path, "wb") as download\_file:

download\_file.write(blob\_client.download\_blob().readall())

import sqlite3

mydb=sqlite3.connect("next.db")

key=mydb.execute("select key from compilerlogin where username='%s'"%(session["user"])).fetchone()[0]

cloudcsvde(download\_file\_path,key)

# print("http://edusphere.co.in/getPlotpng?va="+local\_file\_name)

#webbrowser.open\_new\_tab("http://127.0.0.1:5000/getPlotpng?va="+local\_file\_name)

@app.route('/getPlotpng',methods=["get"]) # this is a job for GET, not POST

def datalakeplot\_csv():

user = request.args.get('va')

va=user.split("\_\_")[1]

print(va)

return send\_file('static/download/'+user,

attachment\_filename=va,

as\_attachment=True)

@app.route('/downloaduserall',methods=["Post"])

def clouddownloaduserall():

user=request.form["rid"]

print(user)

clouddownloadfileuser(user)

return 's'

def cloudgetmyfile(user,e):

connect\_str=config["IPFS server\_storage\_connectionstring"]

blob\_service\_client = BlobServiceClient.from\_connection\_string(connect\_str)

container\_name =config["videos\_container\_name"]

container\_client = blob\_service\_client.get\_container\_client(container\_name)

blob\_list = container\_client.list\_blobs()

va=[]

for blob in blob\_list:

if user in blob.name:

va.append(str(blob.name).split("\_\_")[1])

return json.dumps(va)

@app.route('/getformat',methods=["post"])

def getformat():

user=request.form["rid"]

x=cloudgetmyfile(user,'de')

r=json.loads(x)

de=[]

for k in r:

de.append(k[k.find('.')+1:])

de=list(set(de))

x=json.dumps(de)

return x

@app.route('/download',methods=["Post"])

def clouddownload():

user=request.form["rid"]

da=request.form["da"]

file=user+'\_\_'+da

print(file)

connect\_str=config["IPFS server\_storage\_connectionstring"]

blob\_service\_client = BlobServiceClient.from\_connection\_string(connect\_str)

container\_name =config["videos\_container\_name"]

container\_client = blob\_service\_client.get\_container\_client(container\_name)

blob\_list = container\_client.list\_blobs()

local\_path = "static/download"

for blob in blob\_list:

if file== blob.name:

local\_file\_name = blob.name

blob\_client = blob\_service\_client.get\_blob\_client(container=container\_name, blob=local\_file\_name)

download\_file\_path = os.path.join(local\_path, local\_file\_name)

with open(download\_file\_path, "wb") as download\_file:

download\_file.write(blob\_client.download\_blob().readall())

import sqlite3

mydb=sqlite3.connect("next.db")

key=mydb.execute("select key from compilerlogin where username='%s'"%(session["user"])).fetchone()[0]

cloudcsvde(download\_file\_path,key)

# print("http://edusphere.co.in/getPlotpng?va="+local\_file\_name)

#webbrowser.open\_new\_tab("http://127.0.0.1:5000/?va="+local\_file\_name)

return 's'

@app.route('/delete',methods=["Post"])

def clouddelete():

print(request.form)

user=request.form["rid"]

da=request.form["da"]

file=user+'\_\_'+da

connection\_string=config["IPFS server\_storage\_connectionstring"]

container\_name=config["videos\_container\_name"]

blob\_service\_client = BlobServiceClient.from\_connection\_string(conn\_str=connection\_string)

container\_client = blob\_service\_client.get\_container\_client(container\_name)

container\_client.delete\_blob(file)

return 's'

@app.route('/downfile')

def clouddownfile():

y=os.listdir("static/download")

return render\_template("downfile.html",r=y)

def cloudget\_files(dir):

with os.scandir(dir)as entries:

for entry in entries:

if entry.is\_file() and not entry.name.startswith("."):

yield entry

def cloudupload(files,connection\_string,container\_name):

container\_client=ContainerClient.from\_connection\_string(connection\_string,container\_name)

print("uploading")

for file in files:

blob\_client=container\_client.get\_blob\_client(file.name)

with open(file.path,"rb")as data:

blob\_client.upload\_blob(data)

print(f'{file.name}is uploaded')

@app.route('/uploaderfiler', methods=['POST'])

def cloudupload\_file():

if request.method == 'POST':

if 'files[]' not in request.files:

flash('No file part')

return redirect(request.url)

files = request.files.getlist('files[]')

dir = app.config['UPLOAD\_FOLDER']

for f in os.listdir(dir):

os.remove(os.path.join(dir, f))

import sqlite3

for file in files:

filename = secure\_filename(session["user"].upper()+"\_\_"+file.filename)

file.save(os.path.join(app.config['UPLOAD\_FOLDER'], filename))

import sqlite3

mydb=sqlite3.connect("next.db")

key=mydb.execute("select key from compilerlogin where username='%s'"%(session["user"])).fetchone()[0]

cloudcsven(os.path.join(app.config['UPLOAD\_FOLDER'], filename),key)

flash('File(s) successfully uploaded')

videos=cloudget\_files(config["source\_folder"])

cloudupload(videos,config["IPFS server\_storage\_connectionstring"],config["videos\_container\_name"])

return redirect('/loger')

def cloudgetmyfile(user,e):

connect\_str=config["IPFS server\_storage\_connectionstring"]

blob\_service\_client = BlobServiceClient.from\_connection\_string(connect\_str)

container\_name =config["videos\_container\_name"]

container\_client = blob\_service\_client.get\_container\_client(container\_name)

blob\_list = container\_client.list\_blobs()

va=[]

for blob in blob\_list:

if user in blob.name:

va.append(str(blob.name).split("\_\_")[1])

return json.dumps(va)

@app.route('/deleteuserall',methods=["Post"])

def deleteuserall():

user=request.form["rid"]

va=cloudgetmyfile(user,'e')

va=json.loads(va)

for e in va:

file=(user+"\_\_"+e)

print(file)

connection\_string=config["IPFS server\_storage\_connectionstring"]

container\_name=config["videos\_container\_name"]

blob\_service\_client = BlobServiceClient.from\_connection\_string(conn\_str=connection\_string)

container\_client = blob\_service\_client.get\_container\_client(container\_name)

container\_client.delete\_blob(file)

return 's'

@app.route('/uploadshare',methods=["Get"])

def uploadshare():

user = request.args.get('d')

import sqlite3

mydb=sqlite3.connect("next.db")

e=mydb.execute('select username from compilerlogin where username!="%s"'%(session["user"]))

x=[n[0] for n in e.fetchall()]

return render\_template("uploadshare.html",x=x,value=user)

@app.route('/sharedfile',methods=["Post","Get"])

def sharedfile():

import sqlite3

mydb=sqlite3.connect("next.db")

e=mydb.execute('select \* from sharedfile where sharename="%s"'%(session["user"]))

x=e.fetchall()

print(x)

return render\_template("sharedfile.html",x=x)

@app.route('/sharedocument',methods=["Post","Get"])

def sharedocument():

x=request.form['x']

r=request.form['r']

print(x,r)

import sqlite3

mydb=sqlite3.connect("next.db")

mydb.execute("drop table sharedfile")

mydb.execute("create table sharedfile (sid integer primary key autoincrement,sharename varchar(1000),user varchar(1000),filename varchar(1000),key text)")

import sqlite3

mydb=sqlite3.connect("next.db")

from cryptography.fernet import Fernet

file=session["user"]+'\_\_'+x

print(file)

connect\_str=config["IPFS server\_storage\_connectionstring"]

blob\_service\_client = BlobServiceClient.from\_connection\_string(connect\_str)

container\_name =config["videos\_container\_name"]

container\_client = blob\_service\_client.get\_container\_client(container\_name)

blob\_list = container\_client.list\_blobs()

local\_path = "static/download"

for blob in blob\_list:

if file== blob.name:

local\_file\_name = blob.name

blob\_client = blob\_service\_client.get\_blob\_client(container=container\_name, blob=local\_file\_name)

download\_file\_path = os.path.join(local\_path, local\_file\_name)

with open(download\_file\_path, "wb") as download\_file:

download\_file.write(blob\_client.download\_blob().readall())

import sqlite3

mydb=sqlite3.connect("next.db")

key=mydb.execute("select key from compilerlogin where username='%s'"%(session["user"])).fetchone()[0]

cloudcsvde(download\_file\_path,key)

key=Fernet.generate\_key()

filename="shared\_"+r+"\_"+x

mydb.execute("insert into sharedfile(sharename,user,filename,key) values(?,?,?,?)",(r,session["user"],filename,key))

file="static/download/"+file

print(file,key,r,x)

cloudcsvendata(file,key,r,x)

videos=cloudget\_files("static/encrypt/")

print(videos)

cloudupload(videos,config["IPFS server\_storage\_connectionstring"],config["videos\_container\_name"])

mydb.commit()

e=mydb.execute('select \* from sharedfile')

x=e.fetchall()

print(x)

dir = 'static/encrypt/'

for f in os.listdir(dir):

os.remove(os.path.join(dir, f))

return 's'

@app.route('/downloadshare',methods=["Post"])

def clouddownloadshare():

file=request.form["da"]

connect\_str=config["IPFS server\_storage\_connectionstring"]

blob\_service\_client = BlobServiceClient.from\_connection\_string(connect\_str)

container\_name =config["videos\_container\_name"]

container\_client = blob\_service\_client.get\_container\_client(container\_name)

blob\_list = container\_client.list\_blobs()

local\_path = "static/download"

for blob in blob\_list:

if file== blob.name:

local\_file\_name = blob.name

blob\_client = blob\_service\_client.get\_blob\_client(container=container\_name, blob=local\_file\_name)

download\_file\_path = os.path.join(local\_path, local\_file\_name)

with open(download\_file\_path, "wb") as download\_file:

download\_file.write(blob\_client.download\_blob().readall())

import sqlite3

mydb=sqlite3.connect("next.db")

key=mydb.execute("select key from sharedfile where filename='%s'"%(file)).fetchone()[0]

cloudcsvde(download\_file\_path,key)

return 'success'

if \_\_name\_\_ == '\_\_main\_\_':

app.run(debug = True)