this is the \_\_init\_\_.py: #Flask Dependancies

from flask import Flask, render\_template, request, redirect, url\_for, flash, abort

from flask\_login import LoginManager, UserMixin, login\_user, login\_required, logout\_user, current\_user

from flask\_bcrypt import Bcrypt

from datetime import datetime

from flask\_limiter import Limiter

from flask\_limiter.util import get\_remote\_address

#App dependancies

import os

from flask\_limiter import Limiter

from flask\_limiter.util import get\_remote\_address

from flask import Flask, request, render\_template, redirect, url\_for, send\_from\_directory, flash

import os

from hashlib import sha256

import mimetypes

from PyPDF2 import PdfReader, PdfWriter

from reportlab.pdfgen import canvas

from reportlab.lib.pagesizes import letter

import io

import json

from config import Config

import requests

#QR

import qrcode, base64, pyotp

from io import BytesIO

#databse Connection

from database import engine, Base, dbSession

from DBcreateTables import User, Twofa, Doctor, PatientAssignment, delete\_tables, create\_tables

#WTF

from flask\_wtf.csrf import CSRFProtect

from Forms.appForms import LoginForm, RegistrationForm, TwoFactorForm

#Wrappers

from functools import wraps

# Monitoring

from prometheus\_client import Counter, Gauge, Info, Histogram, Summary, generate\_latest

from prometheus\_flask\_exporter import PrometheusMetrics

from log\_config import logger

#initialising app libraries

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

app.config.from\_object(Config)

SECRET\_KEY = app.config['SECRET\_KEY']

APP\_NAME = app.config['APP\_NAME']

#WTF CSRF Tokens

csrf = CSRFProtect(app)

#Security params

bcrypt = Bcrypt(app)

#Recaptcha tools

RECAPTCHA\_PUBLIC\_KEY = app.config["RECAPTCHA\_PUBLIC\_KEY"]

RECAPTCHA\_PROJECT\_ID = app.config["GOOGLE\_RC\_PROJECT\_ID"]

RECAPTCHA\_PRIVATE\_KEY = app.config["RECAPTCHA\_PRIVATE\_KEY"]

RECAPTCHA\_API\_URL = "https://www.google.com/recaptcha/api/siteverify"

#RBAC validation

def roles\_required(allowed\_roles):

def decorator(func):

@wraps(func)

def wrapped\_function(\*args, \*\*kwargs):

if current\_user.role not in allowed\_roles:

abort(401)

return func(\*args, \*\*kwargs)

return wrapped\_function

return decorator

#Login

login\_manager = LoginManager()

login\_manager.init\_app(app)

login\_manager.login\_view = 'login'

#

#Rate Limiter

limiter = Limiter(

get\_remote\_address,

app = app,

storage\_uri = "memory://"

)

# Prometheus Counters

consumer\_login\_attempts = Counter('consumer\_login\_attempts\_total', 'Total login attempts', ['status'])

consumer\_logout\_attempts = Counter('consumer\_logout\_attempts\_total', 'Total logout attempts', ['status'])

consumer\_register\_attempts = Counter('consumer\_register\_attempts\_total', 'Total register attempts', ['status'])

consumer\_views = Counter('consumer\_page\_views\_total', 'Page views', ['page'])

consumer\_errors = Counter('consumer\_errors\_total', 'Errors', ['error'])

@login\_manager.user\_loader

def load\_user(id):

return dbSession.query(User).filter(User.id == id).first()

@app.route('/')

def home():

consumer\_views.labels(page='home').inc()

return render\_template('home.html')

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

# @limiter.limit("5 per minute")

def login():

form = LoginForm(request.form)

if request.method == 'POST' and form.validate():

grequest = request.form['g-recaptcha-response']

verify\_response = requests.post(url=f'{RECAPTCHA\_API\_URL}?secret={RECAPTCHA\_PRIVATE\_KEY}&response={grequest}').json

print(f'reCATCHA response: {verify\_response()}')

try:

user = dbSession.query(User).filter(User.id == form.id.data).first()

if isinstance(user, User) and bcrypt.check\_password\_hash(user.password, form.password.data):

twofaCheck = dbSession.query(Twofa).filter(Twofa.id == user.get\_id()).first()

remember = form.remember.data

login\_user(user, remember=remember)

if twofaCheck.twofa\_enabled:

return redirect(url\_for('verify2FA'))

elif not twofaCheck.twofa\_enabled:

return redirect(url\_for('setup2FA'))

else:

abort(500)

else:

# login failure log

logger.info("Login failed", extra={

"ip": request.remote\_addr,

"user": user.id,

})

flash("Wrong Username/Password.\n Please try again", 'danger')

except Exception as e:

flash(f"An error {e} occured. Please try again.", "Warning")

finally:

dbSession.close()

consumer\_views.labels(page='login').inc()

return render\_template('login.html', form=form)

@app.route('/logout')

@login\_required

def logout():

logout\_user()

flash("Successfully Logged Out", "success")

consumer\_views.labels(page='logout').inc()

return redirect(url\_for('home'))

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

# @limiter.limit("5 per minute")

def register():

form = RegistrationForm(request.form)

if request.method == 'POST' and form.validate():

grequest = request.form['g-recaptcha-response']

verify\_response = requests.post(url=f'{RECAPTCHA\_API\_URL}?secret={RECAPTCHA\_PRIVATE\_KEY}&response={grequest}').json

print(verify\_response())

if verify\_response()['success'] == False:

flash("Invalid reCAPTCHA")

return render\_template('login.html', form=form)

user = dbSession.query(User).filter(User.id == form.id.data).first()

if not user:

hashed\_password = bcrypt.generate\_password\_hash(form.password.data).decode('utf-8')

new\_user = User(id = form.id.data, username = form.name.data, password = hashed\_password)

if form.email.data:

new\_user.email = form.email.data

if form.phoneNumber.data:

new\_user.phoneNumber = form.phoneNumber.data

if isinstance(new\_user, User):

populateTwoFA = Twofa(id = new\_user.id)

dbSession.add(new\_user)

dbSession.add(populateTwoFA)

dbSession.commit()

flash('Registration Successful!', "success")

return redirect(url\_for('login'))

else:

flash("Already registered please login instead" , 'success')

return redirect(url\_for('login'))

dbSession.close()

consumer\_views.labels(page='register').inc()

return render\_template('register.html', form=form)

@app.route('/patient\_profile', methods=['GET', 'POST'])

@roles\_required("Patient")

def patient\_profile():

consumer\_views.labels(page='patient\_profile').inc()

return render\_template('patient\_profile.html')

@app.route('/createUsers')

def createDoctor():

deleteTables()

createTables()

new\_user = User(

id='T0110907Z',

username='Lucian',

password=bcrypt.generate\_password\_hash("P@ssw0rd").decode('utf-8')

)

userTwoFA = Twofa(id = new\_user.id)

doctorUser = User(

id='S1234567A',

username='Amy',

password=bcrypt.generate\_password\_hash("P@ssw0rd").decode('utf-8')

)

doctorTwoFA = Twofa(id = doctorUser.id)

doctor = doctorUser.add\_doctor(

license\_number='M04637Z',

specialisation='Family Medicine',

facility='Manadr BoonLay'

)

dbSession.add(new\_user)

dbSession.add(doctorUser)

dbSession.add(doctor)

dbSession.add(userTwoFA)

dbSession.add(doctorTwoFA)

dbSession.commit()

dbSession.close()

flash("Doctor Added to DB", "success")

return redirect(url\_for('home'))

@app.route('/setup2FA')

@login\_required

def setup2FA():

try:

twofaCheck = dbSession.query(Twofa).filter(Twofa.id == current\_user.get\_id()).first()

except Exception as e:

dbSession.rollback()

secret = twofaCheck.user\_secret

uri = f'otpauth://totp/{current\_user.get\_id()}?secret={secret}&issuer={APP\_NAME}'

qr = qrcode.QRCode(version=1, box\_size=10, border=5)

qr.add\_data(uri)

qr.make(fit=True)

img = qr.make\_image(fill\_color='black', back\_color='white')

buffered = BytesIO()

img.save(buffered)

base64\_qr\_image = base64.b64encode(buffered.getvalue()).decode("utf-8")

return render\_template("setup2FA.html", secret=secret, qr\_image=base64\_qr\_image)

@app.route('/verify2FA', methods=['GET', 'POST'])

@login\_required

def verify2FA():

form = TwoFactorForm(request.form)

try:

twofaCheck = dbSession.query(Twofa).filter(Twofa.id == current\_user.get\_id()).first()

except Exception as e:

dbSession.rollback()

if form.validate\_on\_submit():

totp = pyotp.parse\_uri(f'otpauth://totp/{current\_user.get\_id()}?secret={twofaCheck.user\_secret}&issuer={APP\_NAME}')

if totp.verify(form.data['otp']):

if twofaCheck.twofa\_enabled:

flash("2FA verification successful. You are logged in!", "success")

logger.info("Login success", extra={

"ip": request.remote\_addr,

"user": current\_user.id,

})

return redirect(url\_for('home'))

else:

try:

twofaCheck.twofa\_enabled = True

dbSession.commit()

flash("2FA setup successful. You are logged in!", "success")

return redirect(url\_for('home'))

except Exception:

dbSession.rollback()

flash("2FA setup failed. Please try again.", "danger")

return redirect(url\_for('verify2FA'))

else:

flash("Invalid OTP. Please try again.", "danger")

return redirect(url\_for('verify2FA'))

else:

if not twofaCheck.twofa\_enabled:

flash(

"You have not enabled 2-Factor Authentication. Please enable it first.", "info")

return render\_template("verify2FA.html", form=form)

@app.route('/doctor\_assignment', methods=['GET', 'POST'])

@roles\_required('Doctor')

def doctor\_assignment():

NRIC = request.args.get('NRIC')

user = dbSession.query(User).filter(User.id == NRIC).first()

if user:

print(user.username)

return render\_template('doctor\_assignment.html', user=user)

else:

return render\_template('doctor\_assignment.html', user=None)

@app.route('/confirm\_patient/<NRIC>', methods=['GET', 'POST'])

@roles\_required('Doctor')

def confirm\_patient(NRIC):

doctor = dbSession.query(Doctor).filter(Doctor.id == current\_user.id).first()

patientAssignment = PatientAssignment(doctor\_id = doctor.license\_number, patient\_id = NRIC)

if isinstance(patientAssignment, PatientAssignment):

dbSession.add(patientAssignment)

dbSession.commit()

dbSession.close()

flash("Successfully assigned patient", "success")

return redirect(url\_for('home'))

@app.route('/metrics')

def metrics():

return generate\_latest()

# error handlers

@app.errorhandler(401)

def not\_authorised(e):

consumer\_errors.labels(error='401').inc()

return render\_template('error.html', error\_code = 401, message = "Please log in to view this page")

@app.errorhandler(403)

def forbidden(e):

consumer\_errors.labels(error='403').inc()

return render\_template('error.html', error\_code = 403, message = "You don't have the required permissions to access this page")

@app.errorhandler(404)

def not\_found(e):

consumer\_errors.labels(error='404').inc()

return render\_template('error.html', error\_code = 404, message = "Page not found. Sorry for the inconvenience caused")

@app.errorhandler(429)

def too\_many\_requests(e):

logout\_user()

consumer\_errors.labels(error='429').inc()

return render\_template('error.html', error\_code = 429, message = "Too many requests. Please try again later")

@app.errorhandler(500)

def internal\_error(e):

consumer\_errors.labels(error='500').inc()

return render\_template('error.html', error\_code = 500, message = "Internal server error")

@app.errorhandler(502)

def bad\_gateway(e):

consumer\_errors.labels(error='502').inc()

return render\_template('error.html', error\_code = 502, message = "Bad gateway")

@app.errorhandler(503)

def service\_unavailable(e):

consumer\_errors.labels(error='503').inc()

return render\_template('error.html', error\_code = 503, message = "Service unavailable")

@app.errorhandler(504)

def gateway\_timeout(e):

consumer\_errors.labels(error='504').inc()

return render\_template('error.html', error\_code = 504, message = "Gateway timeout")

@app.route('/doctor\_upload', methods=['GET', 'POST'])

@login\_required

def staff\_upload():

if current\_user.id not in users or users[current\_user.id]['role'] != 'staff':

flash('Access denied. Staff only.', 'error')

return redirect(url\_for('home.html'))

if request.method == 'POST':

patient\_id = request.form['patient\_id']

document\_type = request.form['document\_type']

document = request.files['document']

notes = request.form['notes']

if document:

filename = f"{patient\_id}\_{document\_type}\_{datetime.now().strftime('%Y%m%d%H%M%S')}.pdf"

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

flash('Document uploaded successfully.', 'success')

else:

flash('No document uploaded.', 'error')

return render\_template('staff\_upload.html')

@app.route('/patient\_records')

@login\_required

def patient\_records():

if current\_user.id not in users or users[current\_user.id]['role'] not in ['staff', 'patient']:

flash('Access denied.', 'error')

return redirect(url\_for('home'))

# Placeholder: Fetch patient records from database

records = [

{'date': '2023-05-01', 'document\_type': 'Medical Report', 'uploaded\_by': 'Dr. Smith', 'view\_url': '#', 'download\_url': '#'},

{'date': '2023-04-15', 'document\_type': 'X-Ray Scan', 'uploaded\_by': 'Dr. Johnson', 'view\_url': '#', 'download\_url': '#'},

]

return render\_template('patient\_records.html', records=records)

@app.route('/admin\_dashboard')

@login\_required

def admin\_dashboard():

if current\_user.id not in users or users[current\_user.id]['role'] != 'admin':

flash('Access denied. Admin only.', 'error')

return redirect(url\_for('home'))

# Placeholder: Fetch user accounts and SIEM events from database

user\_accounts = [

{'username': 'staff1', 'role': 'staff', 'last\_login': '2023-05-01 10:30:00'},

{'username': 'patient1', 'role': 'patient', 'last\_login': '2023-04-30 15:45:00'},

]

siem\_events = [

{'timestamp': '2023-05-01 11:00:00', 'type': 'login', 'user': 'staff1', 'details': 'Successful login'},

{'timestamp': '2023-05-01 11:05:00', 'type': 'upload', 'user': 'staff1', 'details': 'Document uploaded for patient1'},

]

return render\_template('admin\_dashboard.html', users=user\_accounts, siem\_events=siem\_events)

import os

import json

from flask import Flask, request, render\_template, redirect, url\_for, flash, send\_from\_directory

from sqlalchemy.orm import sessionmaker

from database import engine, Base, dbSession # Ensure dbSession and engine are imported

from sqlalchemy.exc import SQLAlchemyError

from hashlib import sha256

from PyPDF2 import PdfReader, PdfWriter

from io import BytesIO

from reportlab.lib.pagesizes import letter

from reportlab.pdfgen import canvas

from DBcreateTables import File

import mimetypes

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

app.config['META\_FOLDER'] = 'meta'

os.makedirs(app.config['UPLOAD\_FOLDER'], exist\_ok=True)

os.makedirs(app.config['META\_FOLDER'], exist\_ok=True)

# Routes

@app.route('/upload')

def upload\_page():

return render\_template('upload.html')

def get\_file\_model():

from DBcreateTables import File # Import only when needed

return File

File = get\_file\_model()

@app.route('/Landing')

def Landing\_page():

return render\_template('Landing.html')

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

def upload\_file():

if 'file' not in request.files:

flash('No file part')

return redirect(request.url)

file = request.files['file']

if file.filename == '':

flash('No selected file')

return redirect(request.url)

if file:

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

file.save(filepath)

# Process metadata

metadata = {key: request.form[key] for key in request.form}

metadata\_filepath = os.path.join(app.config['META\_FOLDER'], f"{file.filename}\_metadata.json")

with open(metadata\_filepath, 'w') as metadata\_file:

json.dump(metadata, metadata\_file)

# Add watermark if the file is PDF

watermark\_hash = None

if file.filename.endswith('.pdf'):

watermark\_text = "Medsync"

add\_watermark(filepath, filepath, watermark\_text)

watermark\_hash = compute\_hash\_from\_text(watermark\_text)

# Compute hash of the file

file\_hash = compute\_hash(filepath)

# Store file and metadata in database

new\_file = File(

filename=file.filename,

file\_path=filepath,

name=metadata.get('name'),

license\_no=metadata.get('license\_no'),

date=metadata.get('date'),

time=metadata.get('time'),

facility=metadata.get('facility'),

patient\_nric=metadata.get('patient\_nric'),

type=metadata.get('type')

)

try:

dbSession.add(new\_file)

dbSession.commit()

except SQLAlchemyError as e:

flash(f"Error saving file to database: {e}")

dbSession.rollback()

return render\_template('upload\_success.html', file\_hash=file\_hash, watermark\_hash=watermark\_hash, metadata=metadata, filename=file.filename)

@app.route('/files', methods=['GET'])

def list\_files():

all\_files = dbSession.query(File).all() # Fetch all file entries from the DB

files\_data = [

{"filename": file.filename, "metadata": json.load(open(os.path.join(app.config['META\_FOLDER'], f"{file.filename}\_metadata.json"))) if os.path.exists(os.path.join(app.config['META\_FOLDER'], f"{file.filename}\_metadata.json")) else {}, "hash": compute\_hash(file.file\_path)}

for file in all\_files

]

return render\_template('download.html', files=files\_data)

@app.route('/download/<filename>')

def download\_file(filename):

file = dbSession.query(File).filter\_by(filename=filename).first()

if file:

return send\_from\_directory(app.config['UPLOAD\_FOLDER'], filename, as\_attachment=True)

flash("File not found!")

return redirect(url\_for('list\_files'))

@app.route('/delete/<filename>', methods=['DELETE'])

def delete\_file(filename):

file = dbSession.query(File).filter\_by(filename=filename).first()

if file:

try:

os.remove(file.file\_path) # Delete the file from the filesystem

dbSession.delete(file) # Delete the file record from the database

dbSession.commit()

return {"success": True}, 200

except Exception as e:

return {"success": False, "error": str(e)}, 500

return {"success": False, "error": "File not found!"}, 404

# Helper Functions

def compute\_hash(filepath):

hasher = sha256()

with open(filepath, 'rb') as f:

while chunk := f.read(8192):

hasher.update(chunk)

return hasher.hexdigest()

def compute\_hash\_from\_text(text):

return sha256(text.encode('utf-8')).hexdigest()

def add\_watermark(input\_pdf\_path, output\_pdf\_path, watermark\_text):

packet = BytesIO()

c = canvas.Canvas(packet, pagesize=letter)

width, height = letter

c.setFont("Helvetica", 60)

text\_width = c.stringWidth(watermark\_text, "Helvetica", 60)

c.setFillAlpha(0.3)

c.setFillColorRGB(0.5, 0.5, 0.5)

c.drawString((width - text\_width) / 2, height / 2, watermark\_text)

c.showPage()

c.save()

packet.seek(0)

new\_pdf = PdfReader(packet)

existing\_pdf = PdfReader(input\_pdf\_path)

output\_pdf = PdfWriter()

for page in existing\_pdf.pages:

page.merge\_page(new\_pdf.pages[0])

output\_pdf.add\_page(page)

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

output\_pdf.write(output\_file)

# Ensure tables exist at runtime

Base.metadata.create\_all(engine)

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

app.run(debug=True)

This is the log\_config.py: import logging, os

from pythonjsonlogger import jsonlogger

from datetime import datetime

from flask import request

base\_dir = os.path.dirname(os.path.abspath(\_\_file\_\_))

log\_dir = os.path.join(base\_dir, "logs")

log\_file = os.path.join(log\_dir, "app.log")

# Ensure the logs directory exists

if not os.path.exists(log\_dir):

os.makedirs(log\_dir)

# log handler

logger = logging.getLogger()

logHandler = logging.FileHandler(log\_file)

formatter = jsonlogger.JsonFormatter("%(asctime)s %(levelname)s %(message)s")

logHandler.setFormatter(formatter)

logger.addHandler(logHandler)

logger.setLevel(logging.INFO)

Error log sample:

{"asctime": "2025-02-09 14:30:15,020", "levelname": "INFO", "message": "User logged in successfully", "ip": "192.168.1.10", "user": "T0110907Z"}

{"asctime": "2025-02-09 14:35:42,543", "levelname": "INFO", "message": "User logged out", "ip": "192.168.1.10", "user": "T0110907Z"}

{"asctime": "2025-02-09 14:40:10,302", "levelname": "INFO", "message": "127.0.0.1 - - [09/Feb/2025 14:40:10] \"GET /dashboard HTTP/1.1\" 200 -"}

{"asctime": "2025-02-09 14:42:18,789", "levelname": "WARNING", "message": "Rate limit exceeded for login attempts", "ip": "192.168.1.11", "user": "S1234567A", "limit": "5 per minute"}

{"asctime": "2025-02-09 14:45:27,112", "levelname": "WARNING", "message": "Unauthorized access attempt to admin dashboard", "ip": "192.168.1.12", "user": "P9988776X", "page": "/admin\_dashboard", "required\_role": "admin"}

{"asctime": "2025-02-09 14:50:14,652", "levelname": "ERROR", "message": "Login failed - incorrect credentials", "ip": "192.168.1.13", "user": "T0110907Z"}

{"asctime": "2025-02-09 14:55:30,987", "levelname": "ERROR", "message": "Database connection error", "error": "SQLAlchemy OperationalError: could not connect to server", "db\_uri": "postgresql://user:password@localhost/medsync"}

{"asctime": "2025-02-09 15:05:42,213", "levelname": "ERROR", "message": "File upload failed - missing metadata", "ip": "192.168.1.14", "filename": "medical\_report.pdf", "reason": "Missing patient NRIC"}

{"asctime": "2025-02-09 15:10:11,345", "levelname": "CRITICAL", "message": "Unexpected application crash", "error": "TypeError: unsupported operand type(s) for +: 'NoneType' and 'str'", "endpoint": "/verify2FA", "stack\_trace": "Traceback (most recent call last):\n File 'app.py', line 210, in verify2FA\n uri = f'otpauth://totp/{current\_user.get\_id()}?secret={twofaCheck.user\_secret}&issuer={APP\_NAME}'\nTypeError: unsupported operand type(s) for +: 'NoneType' and 'str'"}

{"asctime": "2025-02-09 15:20:37,801", "levelname": "CRITICAL", "message": "SQL Injection attempt detected", "ip": "192.168.1.15", "user\_input": "' OR '1'='1'; --", "endpoint": "/login"}

{"asctime": "2025-02-09 15:25:50,978", "levelname": "CRITICAL", "message": "Multiple failed login attempts detected", "ip": "192.168.1.16", "user": "Unknown", "attempts": 10, "lockout\_enabled": true}

Questions to ask the chatbot:

Identify errors by error level:

identify only Critical level data

identify only data with higher error level than WARNING level data: shows ERROR and CRITICAL

Suggest some solutions to SQL INJECTION attempted

Suggest some solutions to Unexpected application crash

Can you detect any unusual activity in the logs?

What is causing database connection failures?

What are some recommended actions to handle SQL INJECTION attempted

What are some recommended actions to handle the latest critical error log

