

DOCUMENTATION FOR FLASK USER PORTAL AND FILE UPLOADING APPLICATION

TABLE OF CONTENTS

1. Overview	1
2. System Components	1
3. File Structure	1
4. Functional Overview	2
5. Prerequisites	2
6. Installations	2
7. Downloading the project file	3
8. Setting up GMAIL SMTP for sending mails in the flask	3
9. Execution and Running of Project	3
10. Running in the web server	4
11. Walkthrough of portal	5
12. Code Explanation	9
13. Sqlite database being used with SQLAlchemy	16
14. Checking up the credentials of registered users	17
15. Conclusion	17

1. Overview:

The following Flask Application provides an interface for new user registration, login, and a dashboard with the user details such as first name, last name, address, Gmail address and the time the user logged in. It also allows user to upload a file from the File Explorer. Once the file is uploaded an email will be sent which was under registration of user regarding successful submission. This system allows the administrator to monitor user activities and secure data management.

2. System Components:

- **Backend:** Python Flask framework for authentication processing and file upload.
- **Frontend:** HTML templates and CSS for User Interface.
- **Database:** SQLite for storing user credentials.
- **Email Integration:** Flask-Mail for sending email notifications.
- **Security:** Flask-Werkzeug for password hashing.
- **Email Service:** Gmail SMTP
- **Libraries:** Flask-Mail, Flask-SQLAlchemy

3. File Structure:

project_portal/

├─ **instance/**

| └─ **users.db:** Stores every individual user's data.

├─ **templates/**

| └─ **login.html**

| └─ **dashboard.html**

| └─ **register.html**

├─ **uploads/:** Stores all the files of user's which are uploaded to the portal.

├─ **portal.py:** The main python script in creation of flask-based portal and database and setting up email server.

├─ **threat.py:** The python script which will create IoC and save them to the csv file.

├─ **check.py:** The python script which authenticates and verify whether the user password is correct or not.

├─ **delete_users.py:** The script will delete if we want to select any existing user.

├─ **registers.py:** The python script for generating random 100000 users.

├─ **users.csv:** The file which got generated by running the above registers.py script. This file contains all the users' details such as username, first name, last name, password, email address, address.

├─ **failed_users.csv:** The file which stores failed registration of users with the cause of failure.

├─ **ioc_combined.csv:** The file which stores all the IoC generated in the proper format.

└─ **userregister.py:** The python script for registering all the users from users.csv.

4. Functional Overview:

- **User Registration:** Captures and securely stores user details including name, email, and address.
- **Amazon Linux 2023 Setup:** Amazon Linux 2023 was used for deploying the Flask app for its reliability and package availability.
- **Automated IOC Aggregation via APScheduler:** IOC data updates every 60 seconds from various threat sources using APScheduler.
- **Authentication System:** Verifies user credentials against the database and implements secure login mechanisms.
- **Google OAuth Login Integration:** Integrated Google login using Flask-Dance and Google Identity. Users can log in with Google accounts.
- **Uploading File:** Allows users to upload files, stores the uploaded files and sends email confirmation upon successful submission to the respected email of the user.
- **Passwords:** Passwords are hashed upon registration of the users and stores securely.

5. Prerequisites:

- Python 3.8 or later
- Active Internet connection
- Web browser
- Amazon Linux 2023

6. Installations:

- a. Open terminal i.e., command prompt or windows power shell to check whether python is installed.
Command: **python --version**
- b. If python is not installed, use the given link, download and install python
Link: <https://www.python.org/downloads/>
- c. After installation to check that whether python is installed or not use the command from step a.
- d. If the command returns: **Python 3.13**, then python is installed.
- e. Now to install pip, stay in the same terminal and execute the following commands.
Commands: **curl https://bootstrap.pypa.io/get-pip.py -o get-pip.py**
python get-pip.py
- f. To verify whether pip is installed or not, Use
Command: **pip --version**.
- g. To install necessary modules, libraries in the python which are required for this project, we need to execute the command given below
Command: **pip install flask flask_sqlalchemy flask_mail werkzeug**
 - Flask is for web application.
 - Flask-SQLAlchemy is for database operations.
 - Flask-Mail is for sending mail
 - Werkzeug for password hashing.

- h. Save the iockit.pem file in the secure folder such as C:/users/harsha/.ssh folder.
- i. Ssh into the terminal where we saved the iockit.pem file. The ssh instance is:
ssh -i "iockit.pem" ec2-user@ec2-52-14-174-93.us-east-2.compute.amazonaws.com
- j. Saving all the files into this instance to run the program.

7. Downloading Project File:

Copy the given folder in to the file folder and save the file folder as “project_portal” in a preferred location. (For example: D:/project_portal)

Make sure all the files such as instances, templates are present in the folder along with html and python files.

Now all the files need to be moved into instance:

```
PS C:\Users\harsh\.ssh> scp -i "C:\Users\harsh\.ssh\iockit.pem" -r "D:\project_portal" ec2-user@ec2-52-14-174-93.us-east-2.compute.amazonaws.com:~/project_p
ortal
check.py 100% 504 10.3KB/s 00:00
documentation.docx 100% 1978KB 2.5MB/s 00:00
documentation.pdf 100% 1372KB 2.2MB/s 00:00
failed_users.csv 100% 1018 6.9KB/s 00:00
users.db 100% 42MB 2.9MB/s 00:14
iocs_combined.csv 100% 4425KB 2.8MB/s 00:01
portal.py 100% 8512 307.9KB/s 00:00
registers.py 100% 1160 37.8KB/s 00:00
change_password.html 100% 654 30.4KB/s 00:00
dashboard.html 100% 7353 194.1KB/s 00:00
login.html 100% 2534 91.7KB/s 00:00
register.html 100% 1109 57.0KB/s 00:00
threat.py 100% 4895 164.8KB/s 00:00
check.py 100% 401 14.0KB/s 00:00
images.jpg 100% 5690 158.8KB/s 00:00
login_page.png 100% 35KB 547.4KB/s 00:00
PAGE 1.pdf 100% 216KB 2.1MB/s 00:00
text&csv_generation(1).py 100% 1647 57.4KB/s 00:00
trying.py 100% 2483 89.8KB/s 00:00
userregister.py 100% 2246 104.5KB/s 00:00
users.csv 100% 11MB 4.0MB/s 00:02
portal.cpython-313.pyc 100% 7071 209.3KB/s 00:00
~$documentation.docx 100% 162 6.3KB/s 00:00
~$MRL2147.tmp 100% 549KB 3.3MB/s 00:00
PS C:\Users\harsh\.ssh> |
```

8. Setting up GMAIL SMTP for sending mails in the flask:

- Open Gmail account in the phone and navigate to “Google Account Settings”.
- In the security option enable 2-step verification (2FA).
- After enabling 2FA navigate to settings and search for app password.
- Google will generate a 16-digit password which we need to use in the project for sending mails.
- Now in the Gmail SMTP configuration code block replace that section with your Gmail and 16-digit password and save the file.

9. Execution and Running of the project:

- Open the terminal in command prompt or windows power shell.

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\harsh> |
```

- Open the directory where you saved your iockit.pem file such as: c:/users/harsh/.ssh

```
PS C:\Users\harsh> cd c:/users/harsh/.ssh
PS C:\users\harsh\.ssh> |
```

- Once you are into this folder ssh instance for this folder for amazon linux 2023:
ssh -i "iockey.pem" ec2-user@ec2-52-14-174-93.us-east-2.compute.amazonaws.com

- Once you are into this instance open the project folder to see the list and change the directory to `project_portal`.


- Start the server and run the following commands to get the access to the portal

10. Running in the web server:


- Access the application via the provided localhost URL in a web browser.
- Here you can see the flask will be running and project runs.
- All the user's data is stored in the users.db file which is inside instance folder.

11. Walkthrough of the portal:

- a) Once you access the localhost, you can see the following Login Interface.
If you are an existing user, then you can login with the details. You can also login through google sign from this portal.



Login

 **Sign in with Google**

Username

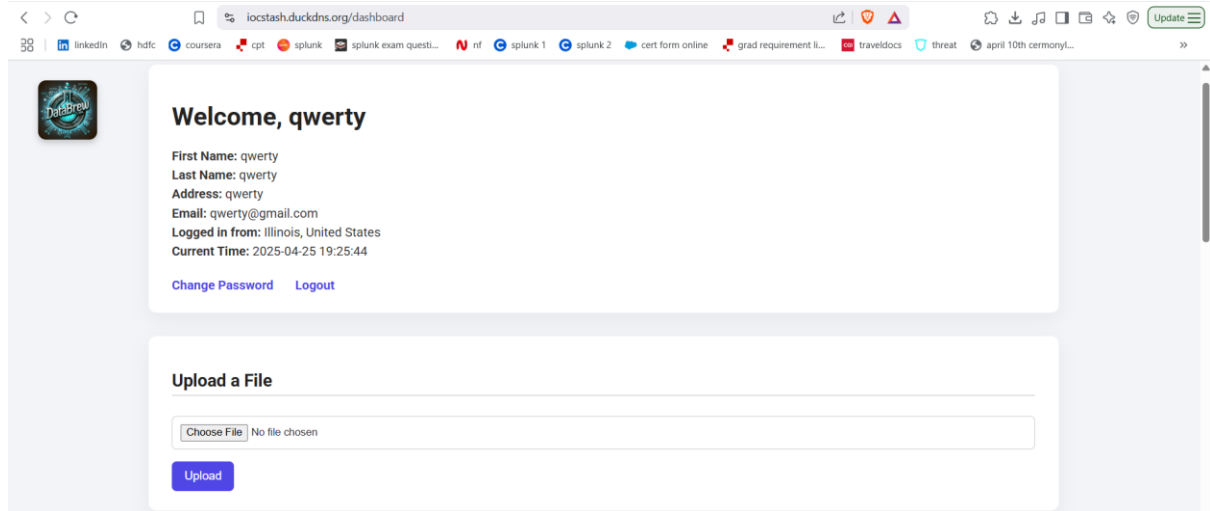
Password

Login

Don't have an account? [Register here](#)

- b) Once you logged in to the portal then you will see your details. And there will be an option to upload the file and also there is an option to logout. Choose file from your file explorer and click upload. Bottom to that you will be seeing IoC list which

generated and refreshed with its types and causes. I have added pagination for this so that the page will not crash. Each page consists 10 IoC and with a search option to search any keyword to find in the list.



IOC Dashboard

Total IOCs: 36128

Search IOCs by Keyword

IOC Type Counts

IOC Type	Count
Hash	1094
IP	4
URL	34373
domain	234
ip:port	185
md5_hash	4
sha256_hash	12
url	222

IOC Source Counts

Source	Count
Feodo Tracker	3
GreyNoise	1
MalwareBazaar	1094
ThreatFox API	657
URLHaus	34373

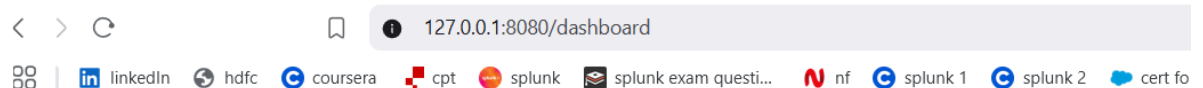
IOC List

Type	Value	Source	Threat Category	Date
url	https://todocarritos.top/www/sss.php	ThreatFox API	payload_delivery	2025-04-25 19:18:22 UTC
url	https://todocarritos.top/www/select.js	ThreatFox API	payload_delivery	2025-04-25 19:18:17 UTC
domain	todocarritos.top	ThreatFox API	payload_delivery	2025-04-25 19:18:14 UTC
url	https://todocarritos.top/www/good.js	ThreatFox API	payload_delivery	2025-04-25 19:18:12 UTC
domain	haidao10.top	ThreatFox API	payload_delivery	2025-04-25 19:10:51 UTC
url	https://www.coligeme.org/cloudflare.msi	ThreatFox API	payload_delivery	2025-04-25 19:10:34 UTC
domain	www.coligeme.org	ThreatFox API	payload_delivery	2025-04-25 19:10:34 UTC
url	https://haidao10.top/www/good.js	ThreatFox API	payload_delivery	2025-04-25 19:10:25 UTC
url	https://haidao10.top/www/index.php?0dRf8bcr	ThreatFox API	payload_delivery	2025-04-25 19:10:25 UTC
url	https://haidao10.top/www/sss.php	ThreatFox API	payload_delivery	2025-04-25 19:10:25 UTC

Page 1 of 3613

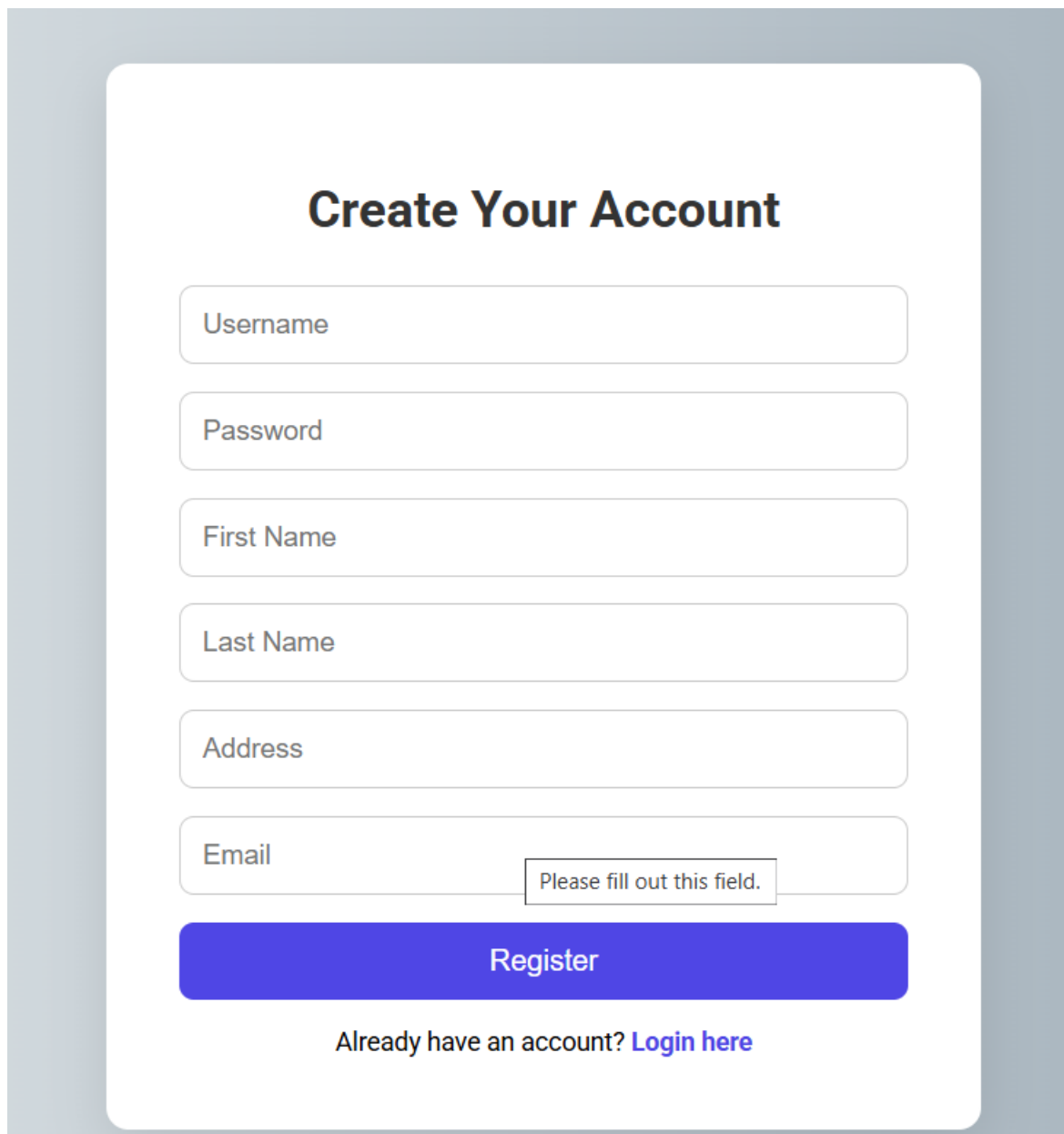
Next

- c) Once the file gets uploaded the user will get the email notification for the successful uploading and a copy of file will be sent to the email. The uploaded file will be stored in the uploads folder.



File uploaded and emailed successfully

- d) If you are a new register, then click on Register here section to create an account for yourself. Enter your details and complete registration.



The image shows a registration form titled "Create Your Account" centered at the top. Below the title are seven input fields stacked vertically: "Username", "Password", "First Name", "Last Name", "Address", "Email", and a "Register" button. The "Email" field has a small error message "Please fill out this field." to its right. Below the "Register" button is a link that says "Already have an account? Login here".

Create Your Account

Please fill out this field.

Already have an account? [Login here](#)

12. Code Explanation:

- a) Importing all the required libraries necessary for the application to run.

```
from flask import Flask, render_template, request, redirect, url_for, session, jsonify
from flask_sqlalchemy import SQLAlchemy
from flask_mail import Mail, Message
from flask import request
from flask_apscheduler import APScheduler
import requests
import pytz
import os
from datetime import datetime
from werkzeug.security import generate_password_hash, check_password_hash
from sqlalchemy.exc import IntegrityError
from werkzeug.middleware.proxy_fix import ProxyFix
import csv
from google_auth_oauthlib.flow import Flow
from google.oauth2 import id_token
from google.auth.transport import requests as grequests
from sqlalchemy import func
from datetime import datetime
import subprocess
from flask_dance.contrib.google import make_google_blueprint, google
from flask_dance.consumer import oauth_authorized
from flask import flash
from sqlalchemy.orm.exc import NoResultFound
```

- b) Initializes the Flask application with secure cookie settings and ProxyFix to properly handle IPs behind a proxy (important for real deployments like AWS EC2).

```
# Initialize the Flask application
app = Flask(__name__)
app.config['SESSION_COOKIE_SECURE'] = True
app.config['SESSION_COOKIE_SAMESITE'] = 'Lax'

app.wsgi_app = ProxyFix(app.wsgi_app, x_for=1, x_host=1)
```

- c) Integrates **Google OAuth** login using Flask-Dance. This allows users to log in using their Google account.

```
# Google OAuth Configuration
app.config["GOOGLE_OAUTH_CLIENT_ID"] = "590471980283-au7aeb65jng31kdvgsq00eramvfou8d.apps.googleusercontent.com"
app.config["GOOGLE_OAUTH_CLIENT_SECRET"] = "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"

google_bp = make_google_blueprint(
    client_id=app.config["GOOGLE_OAUTH_CLIENT_ID"],
    client_secret=app.config["GOOGLE_OAUTH_CLIENT_SECRET"],
    scope=[
        "openid",
        "https://www.googleapis.com/auth/userinfo.email",
        "https://www.googleapis.com/auth/userinfo.profile"
    ],
    redirect_to="google.authorized",
    storage=None
)

app.register_blueprint(google_bp, url_prefix="/login")
```

- d) Configuring database to store user credentials. The details of user are all stored in this user.db file. The database type here I used is SQLite because of its lightweight and efficiency for simple data. The database here is a file instead of server because of smaller data.

SQLALCHEMY_DATABASE_URI: This specifies the database URI (Uniform Resource Identifier) i.e., sqlite:///users.db to use this file from project directory to the flask.

```
# Secret key for session management
app.secret_key = 'your_secret_key'

# Database Configuration
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///users.db'
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False
db = SQLAlchemy(app)
```

- e) Setting up Flask Mail Configuration to send mails to the users from a dedicated mail. MAIL_SERVER uses smtp server for sending emails. The port to use SMTP connection is 587. This number is used because of sending the emails from server to clients using TLS secure connection. It is given TRUE so that it uses TLS encryption for email sending. The username I have given is my personal email which consists of the app password similar to regular password.

```
# Email Configuration
app.config['MAIL_SERVER'] = 'smtp.gmail.com'
app.config['MAIL_PORT'] = 587
app.config['MAIL_USE_TLS'] = True
app.config['MAIL_USERNAME'] = 'somaharsha71@gmail.com'
app.config['MAIL_PASSWORD'] = 'dtgp djyz ukup tmmv'
mail = Mail(app)
```

- f) Sets up automatic background scheduled tasks using APScheduler — used here to run IOC data update every 60 seconds.

```
# Scheduler Configuration
class Config:
    SCHEDULER_API_ENABLED = True

app.config.from_object(Config())
scheduler = APScheduler()
scheduler.init_app(app)

# IOC Scheduler Task: Runs every 60 seconds
@scheduler.task('interval', id='update_ioc_task', seconds=60, misfire_grace_time=30)
def scheduled_ioc_update():
    with app.app_context(): # <-- FIXED: wrap all DB-related work
        print("[DEBUG] Scheduled IOC update triggered")
        print("[Scheduler] Updating IOC...")
        try:
            subprocess.Popen(["python3", "threat.py"], stdout=subprocess.DEVNULL, stderr=subprocess.DEVNULL)
            update_ioc()
            print("[🔴] update_ioc() called at:", datetime.now().strftime("%Y-%m-%d %H:%M:%S"))
        except Exception as e:
            print(f"[Scheduler Error] {e}")
```

- g) This defines how the user details need to be added in the SQLite users.db database. The id is the unique identifier for each person. Username needs to be unique and the password will be stored as hashes password instead of plaintext for security reasons. The other are the given requirements for the user.
- IoC: Manages Indicators of Compromise (IOC) records fetched from threat intelligence sources.
 - User: Manages registered users.

```
# Models
class User(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    username = db.Column(db.String(50), unique=True, nullable=False)
    password = db.Column(db.String(255), nullable=False)
    email = db.Column(db.String(100), unique=True, nullable=False)
    first_name = db.Column(db.String(50), nullable=False)
    last_name = db.Column(db.String(50), nullable=False)
    address = db.Column(db.String(255), nullable=False)

class IoC(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    ioc_type = db.Column(db.String(50), nullable=True)
    value = db.Column(db.String(256), nullable=True)
    source = db.Column(db.String(100), nullable=True)
    threat_category = db.Column(db.String(100), nullable=True)
    date = db.Column(db.String(50), nullable=True)

with app.app_context():
    db.create_all()
```

- h) Reads a CSV file (iocs_combined.csv) and **updates the IoC table** by clearing old data and inserting new records.

```
def update_ioc(csv_filepath='iocs_combined.csv'):
    if not os.path.exists(csv_filepath):
        print(f"CSV file '{csv_filepath}' not found.")
        return

    try:
        with open(csv_filepath, 'r', newline='', encoding='utf-8') as csvfile:
            reader = csv.DictReader(csvfile)
            ioc_rows = list(reader)
            print(f"[DEBUG] Loaded {len(ioc_rows)} IOCs from CSV")
    except Exception as e:
        print(f"Error reading IOC CSV: {e}")
        return

    try:
        num_deleted = db.session.query(IoC).delete() # <-- RAW SQL deletion
        db.session.commit()
        print(f"[✓] Deleted {num_deleted} old IOCs")
    except Exception as e:
        db.session.rollback()
        print(f"[!] Error clearing existing IoC records: {e}")
        print(f"[✓] Inserting {len(ioc_rows)} new IOCs...")
        print(f"[✓] Total IOCs after update: {IoC.query.count()}")

    try:
        for row in ioc_rows:
            new_ioc = IoC(
                ioc_type=row.get("Type"),
                value=row.get("Value"),
                source=row.get("Source"),
                threat_category=row.get("Threat_Category"),
                date=row.get("Date")
            )
            db.session.add(new_ioc)
        db.session.commit()
        print("IOC data updated successfully.")
    except Exception as e:
        db.session.rollback()
        print(f"Error updating IOC data: {e}")
```

- i) **get_geo_info**: Gets region, country, and timezone based on user's IP address using ipapi.co and ipwho.is.
- get_client_ip**: Extracts the user's real client IP (respecting proxies).

```
def get_geo_info(ip):
    try:
        # Try ipapi.co
        response = requests.get(f"https://ipapi.co/{ip}/json/", timeout=5)
        data = response.json()
        print(f"[GeoInfo ipapi.co] {data}")

        region = data.get("region") or "Region Not Available"
        country = data.get("country_name") or "Country Not Available"
        timezone = data.get("timezone") or "UTC"

        if region != "Region Not Available" and country != "Country Not Available":
            return region, country, timezone

        # Fallback to ipwho.is
        response = requests.get(f"https://ipwho.is/{ip}", timeout=5)
        data = response.json()
        print(f"[GeoInfo ipwho.is] {data}")

        region = data.get("region") or "Region Not Available"
        country = data.get("country") or "Country Not Available"
        timezone = data.get("timezone", {}).get("id", "UTC")

        return region, country, timezone

    except Exception as e:
        print(f"[GeoInfo Error] {e}")
        return "Region Not Available", "Country Not Available", "UTC"

def get_client_ip():
    if request.headers.get('X-Forwarded-For'):
        ip = request.headers.get('X-Forwarded-For').split(',')[0].strip()
    else:
        ip = request.remote_addr
    return ip
```

j) **Authentication Routes:**

Login : This section is for handling user login. The POST request will check the credentials such as username and password. If the details are correct, they will be redirected to the dashboard and the username will be stored in the session using cookies. If the details are incorrect then it outputs Invalid Credentials.

```
@app.route('/', methods=['GET', 'POST'])
def login():
    # Redirect to dashboard if already logged in
    if 'username' in session:
        return redirect(url_for('dashboard'))

    if request.method == 'POST':
        username = request.form['username']
        password = request.form['password']
        user = User.query.filter_by(username=username).first()

        if user and check_password_hash(user.password, password):
            session['username'] = username
            subprocess.run(["python", "threat.py"])
            update_ioc()
            return redirect(url_for('dashboard'))
        else:
            return "Invalid Credentials", 401

    return render_template('login.html')
```

Register /: This section is for handling registration. If there is a new user then the login portal asks for register them. The POST requests checks whether the given username is taken or available. Once all the details have been entered, the details will be stored in the database and the password is hashed for security purposes. Once the registration is done then it redirects user to login page.

```
@app.route('/register', methods=['GET', 'POST'])
def register():
    if request.method == 'POST':
        username = request.form['username']
        password = request.form['password']
        email = request.form['email']
        first_name = request.form['first_name']
        last_name = request.form['last_name']
        address = request.form['address']

        if User.query.filter_by(username=username).first():
            return "Username already exists!", 400

        try:
            new_user = User(
                username=username,
                password=generate_password_hash(password),
                email=email,
                first_name=first_name,
                last_name=last_name,
                address=address
            )
            db.session.add(new_user)
            db.session.commit()
            return redirect(url_for('login'))
        except IntegrityError:
            db.session.rollback()
            return "Email or username already exists!", 400

    return render_template('register.html')
```

Logout /:

```
@app.route('/logout')
def logout():
    session.pop('username', None)
    return redirect(url_for('login'))
```

Change Password /:

```
@app.route('/change_password', methods=['GET', 'POST'])
def change_password():
    if 'username' not in session:
        return redirect(url_for('login'))

    user = User.query.filter_by(username=session['username']).first()
    if not user:
        return redirect(url_for('login'))

    if request.method == 'POST':
        current_password = request.form['current_password']
        new_password = request.form['new_password']
        confirm_password = request.form['confirm_password']

        if not check_password_hash(user.password, current_password):
            return "Current password is incorrect", 400
        if new_password != confirm_password:
            return "New passwords do not match", 400

        user.password = generate_password_hash(new_password)
        db.session.commit()
        return "Password changed successfully"

    return render_template('change_password.html')
```

Purpose:

Handles **user login, registration, password change, and logout** operations securely, with session management.

k) Google OAuth User Flow:

Handles post-login operations for Google OAuth users:

If user exists → logs them in

If user is new → redirects to profile completion form.

```
@oauth_authorized.connect_via(google_bp)
def google_logged_in(blueprint, token):
    if not token:
        flash("Failed to log in with Google.", category="error")
        return False

    resp = blueprint.session.get("/oauth2/v2/userinfo")
    if not resp.ok:
        flash("Failed to fetch user info from Google.", category="error")
        return False

    user_info = resp.json()
    email = user_info.get("email")
    if not email:
        flash("Email not available from Google.", category="error")
        return False

    user = User.query.filter_by(email=email).first()
    if user:
        session["username"] = user.username
        subprocess.run(["python", "threat.py"])
        update_ioc()
        return redirect(url_for("dashboard"))

    # NEW USER → Redirect to complete-profile
    session["pending_email"] = email
    session["pending_username"] = email.split("@")[0]
    return redirect(url_for("complete_profile"))
```


l) Complete Profile for new google users:

Allows new users logged in via Google to provide missing information (First name, Last name, Address) and create a complete account.

```
@app.route('/complete-profile', methods=['GET', 'POST'])
def complete_profile():
    if 'pending_email' not in session or 'pending_username' not in session:
        return redirect(url_for('login'))

    if request.method == 'POST':
        new_user = User(
            username=session["pending_username"],
            password=generate_password_hash("oauth_login"),
            email=session["pending_email"],
            first_name=request.form['first_name'],
            last_name=request.form['last_name'],
            address=request.form['address']
        )
        db.session.add(new_user)
        db.session.commit()

        session['username'] = new_user.username
        session.pop("pending_email", None)
        session.pop("pending_username", None)

        subprocess.run(["python", "threat.py"])
        update_ioc()
        return redirect(url_for("dashboard"))

    return render_template("complete.html")
```

m) **Dashboard Route/**: This part of code ensures the user is logged in before using the dashboard. Once after login of user happens it will display the user details with the current time. There will be an option for file uploading in the bottom of user details and this dashboard handles the file uploading and sends email notification to the user email. Logged-in user details. IOC Records with **pagination** and **searching**. Real-time IP and Geolocation. File Upload feature → **uploads file and emails user**

```
@app.route('/dashboard', methods=['GET', 'POST'])
def dashboard():
    if 'username' not in session:
        return redirect(url_for('login'))

    user = User.query.filter_by(username=session['username']).first()
    if not user:
        return redirect(url_for('login'))

    # Get IP & Geo Info
    ip_address = get_client_ip()
    print(f"[DEBUG] IP Address: {ip_address}")
    region, country, user_timezone = get_geo_info(ip_address)
    print(f"[DEBUG] Geo Info: Region={region}, Country={country}, Timezone={user_timezone}")
    print(f"[LOGIN INFO] IP: {ip_address}, Region: {region}, Country: {country}")

    try:
        tz = pytz.timezone(user_timezone)
    except Exception as e:
        print(f"[Timezone Error] {e}")
        tz = pytz.utc # fallback if invalid timezone

    current_time = datetime.now().strftime('%Y-%m-%d %H:%M:%S')

    if request.method == 'POST' and 'file' in request.files:
        file = request.files['file']
        if file.filename == '':
            return "No selected file", 400
        filepath = os.path.join('uploads', file.filename)
        file.save(filepath)
        send_email(user.email, filepath)
        return "File uploaded and emailed successfully"

    ioc_keyword = request.args.get('ioc_keyword', '').strip()
    page = request.args.get('page', 1, type=int)
    per_page = 10

    ioc_query = IoC.query
    if ioc_keyword:
```

- n) This part of code works on sending email to the user after uploading the documents. It creates a message with the file attachment. Sends an email with the uploaded file as an attachment to the logged-in user's email.

```
def send_email(recipient, filepath):
    with app.app_context():
        msg = Message('File Upload Notification', sender=app.config['MAIL_USERNAME'], recipients=[recipient])
        msg.body = 'A file has been uploaded. See the attachment.'
        with open(filepath, 'rb') as f:
            msg.attach(os.path.basename(filepath), 'application/octet-stream', f.read())
        mail.send(msg)
```

o) **Session Management:**

Ensures that users cannot access dashboard, profile pages without logging in.

```
@app.before_request
def require_login():
    allowed_routes = ['login', 'register', 'static', 'google.login', 'google.authorized', 'complete_profile']

    if request.endpoint and (
        request.endpoint.startswith('google.') or
        request.endpoint in allowed_routes
    ):
        return

    if 'username' not in session:
        return redirect(url_for('login'))
```

p) **Main Runner:**

Creates uploads/ directory if it doesn't exist.

Starts the Flask application accessible from **all IPs** on port 8080.

```
if __name__ == '__main__':
    os.makedirs('uploads', exist_ok=True)
    app.run(host='0.0.0.0', port=8080, debug=True)
```

13. Sqlite database being used with SQLAlchemy:

- The database is a light-weight database which can store 100,000+ users efficiently as long as the file is within the few GB's.
- This database supports 100+ concurrent users with only reading permissions without any issues.
- This database supports 1 concurrent write at a time. It allows up to 1-10 concurrent writers but there can be significant delays which are not noticeable.
- Since the project is small scale, using light weight database is optimal.

The below screenshot is the complete folder for the project_portal. The instance folder contains the users.db file. There are three supporting html files for this python script such as for dashboard, login and register. These files are stored under templates folders. The uploads folder contains all the files which are uploaded by the user. The check file is a python file for checking whether the plain password and the hashed password is same or not. The documentation document is the explanation of the code and implementations. The portal file is the main python script where all the logics are written.

__pycache__	25-03-2025 12:59	File folder	
instance	02-04-2025 12:49	File folder	
templates	21-03-2025 11:10	File folder	
uploads	02-04-2025 11:33	File folder	
check	02-04-2025 14:17	PY File	1 KB
documentation	25-03-2025 14:17	Microsoft Word D...	365 KB
documentation	25-03-2025 14:48	Microsoft Edge PD...	353 KB
failed_users	02-04-2025 12:49	Microsoft Excel Co...	1 KB
portal	02-04-2025 13:37	PY File	7 KB
registers	02-04-2025 13:37	PY File	2 KB
userregister	02-04-2025 13:35	PY File	3 KB
users	02-04-2025 11:29	Microsoft Excel Co...	11,008 KB

14. Checking up the credentials of registered users:

```
PS D:\> cd project_portal/instance
PS D:\project_portal\instance> sqlite3 users.db
SQLite version 3.49.1 2025-02-18 13:38:58
Enter ".help" for usage hints.
sqlite> .tables
user
sqlite> SELECT * FROM user;
1|shesh|script:32768:8:1$ftgl9eBIKofIzL3x$f35912429ba0335adaadfc1484bf2a39c6f616b8374e165addb23054bb03a3c873f128162d96c4
73b67b7fea61fa846899911c18271c541f26f7cb7eb9eb31ac|shesh@gmail.com|shesh|shesh|india
2|hash|script:32768:8:1$DZKGIHoTpvJi02H$fdce09311cc0e59102373640a3a7d1c3171f08d16e99e0c2e2ed38a6720f197c0da5ed3b3008ba2
999341450e03e284d69d927d1ecf763907e65daa311d112a9|hash@gmail.com|hash|hash|hash
3|Harshavardhanguptha|script:32768:8:1$AQ5XfwMG1ahDa7V$7a37aaa3a28c233c348e91cf6451bb049110c90e11b611c4a9dead65a6d839bb
5b259d0f1caa24eb283b2e539df9d5667d8c12d3c3894a0d86db4152c65715|hsoma@databrew-llc.com|Harshavardhan|Soma|APT:1507, 501
E32ND ST, CHICAGO ILLINOIS, 606016
4|asd|script:32768:8:1$6K06hnr6ypLAFb$737bc8e6a331c185e65b4270b417a5d731617d1b227bcf48a1482e4348172e4791fb0dbe34da3350
41ef53a808a0a200fbf405e8d376272082b2e51deb6dfd9c|sharshasoma@gmail.com|asd|asd|asd
sqlite> |
```

Run the “sqlite3 users.db” command in the instance folder since this folder has users.db file.

Next execute “. tables” in the sqlite to view what are the tables that are present in the .db file. We need to select from the user using query i.e., “SELECT * FROM user;”.

This query helps us retrieving all the user details in the above format.

The password is hashed since plain text has having higher chances of security concerns.

The format is userid | username | hashed password | Gmail | first name | last name | address.

15. Conclusion:

This Flask application is a basic user login and file upload application with authentication, file upload and email notifications. This application gives an overview of how to manage user data securely, send emails with attachments and handling file uploads in the web application.