# DOCUMENTATION FOR FLASK USER PORTAL AND FILE UPLOADING APPLICATION

## TABLE OF CONTENTS

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.
├── **check.py:** The python script which authenticates and verify whether the user password is correct or not.
├── **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.
├── **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.
- **Authentication System:** Verifies user credentials against the database and implements secure login mechanisms.

- **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

## 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
  - Werlzeug for password hashing.

## 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.

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

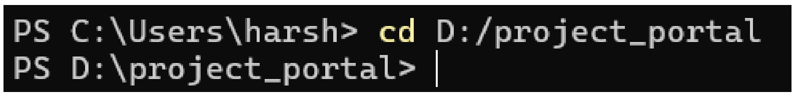
- Open Gmail account in the phone and navigate to “Google Account Settings”.
- In the security option enable 2-step verification (2FA).
- After enabling 2FA navigate to settings and search for app password.

- 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:

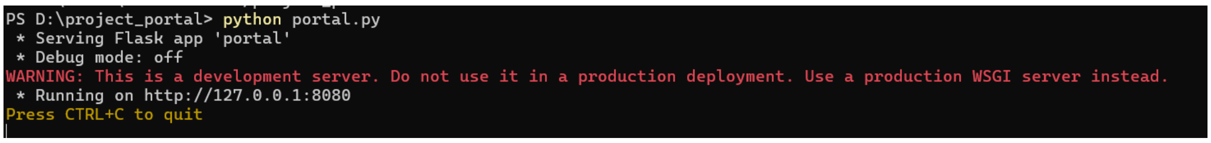
- Open the terminal in command prompt or windows power shell.
- Navigate to your file folder in the file explorer and copy the folder path.
- Now in terminal change your directory using command cd.

Example: My folder is in D folder and the folder name is project_portal.



```
PS C:\Users\harsh> cd D:/project_portal
PS D:\project_portal>
```

- Now run the following command: **python project.py**



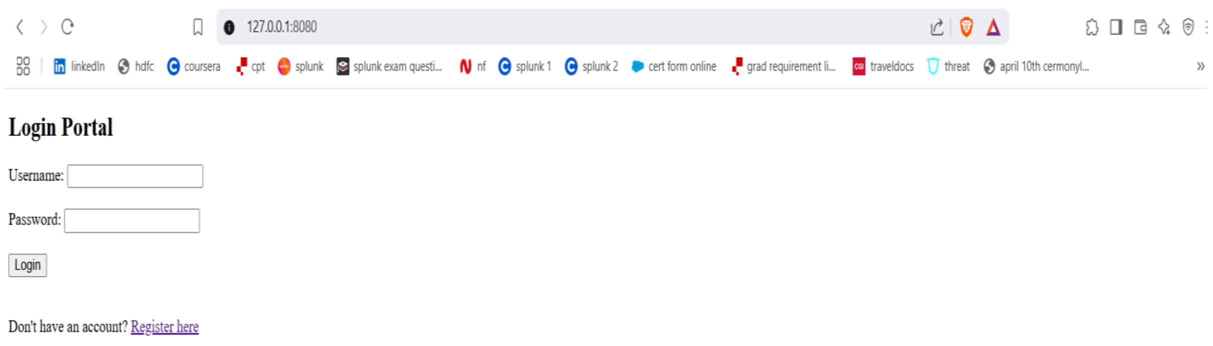
```
PS D:\project_portal> python portal.py
 * Serving Flask app 'portal'
 * Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:8080
Press CTRL+C to quit
```

## 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.



127.0.0.1:8080

**Login Portal**

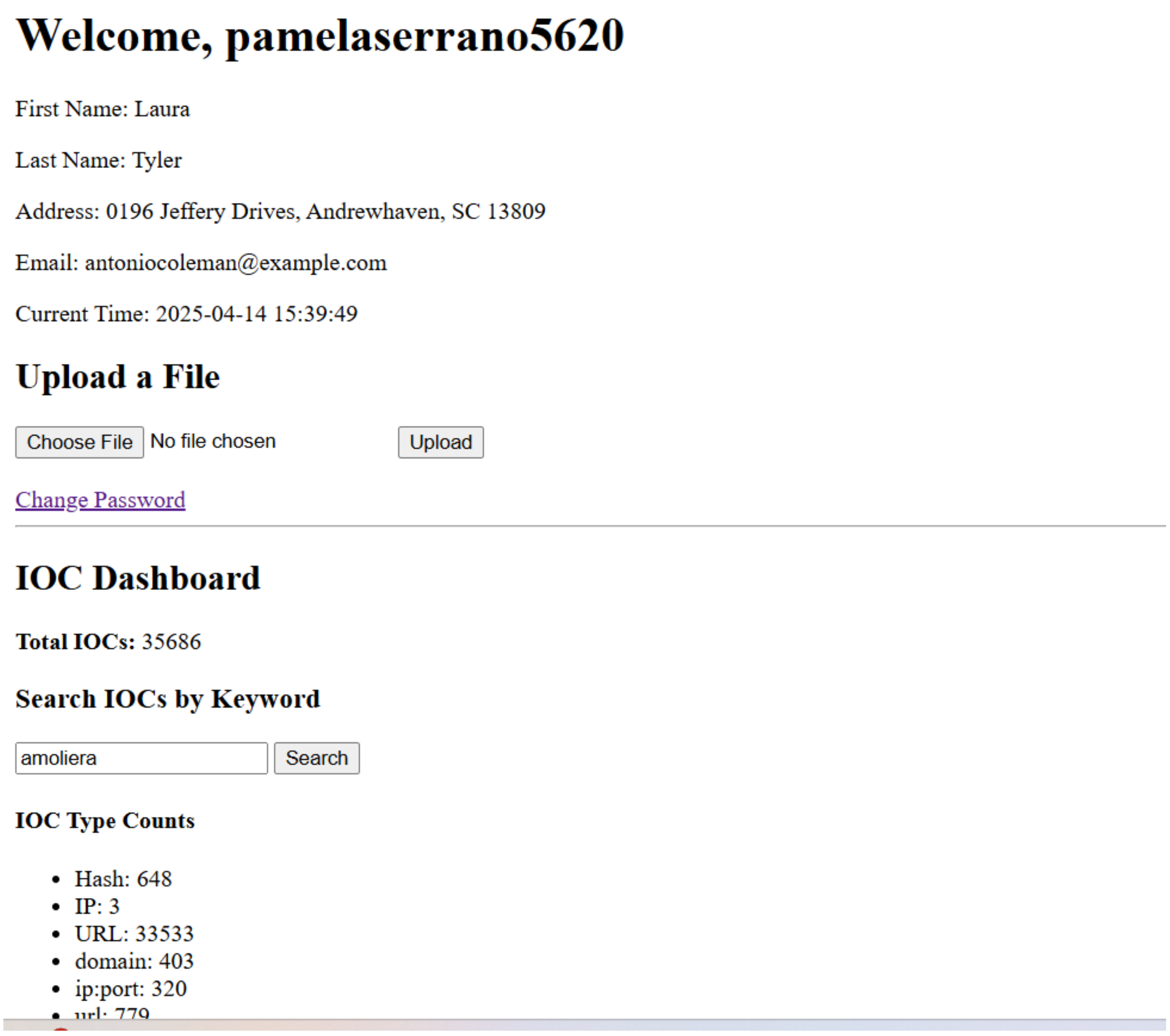
Username:

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. 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.



Welcome, pamelaserrano5620

First Name: Laura

Last Name: Tyler

Address: 0196 Jeffery Drives, Andrewhaven, SC 13809

Email: antoniocoleman@example.com

Current Time: 2025-04-14 15:39:49

Upload a File

IOC Dashboard

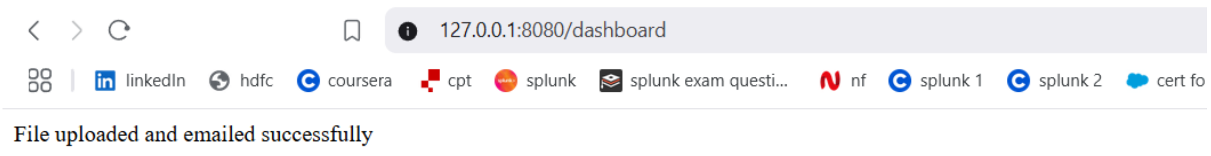
**Total IOCs:** 35686

Search IOCs by Keyword

IOC Type Counts

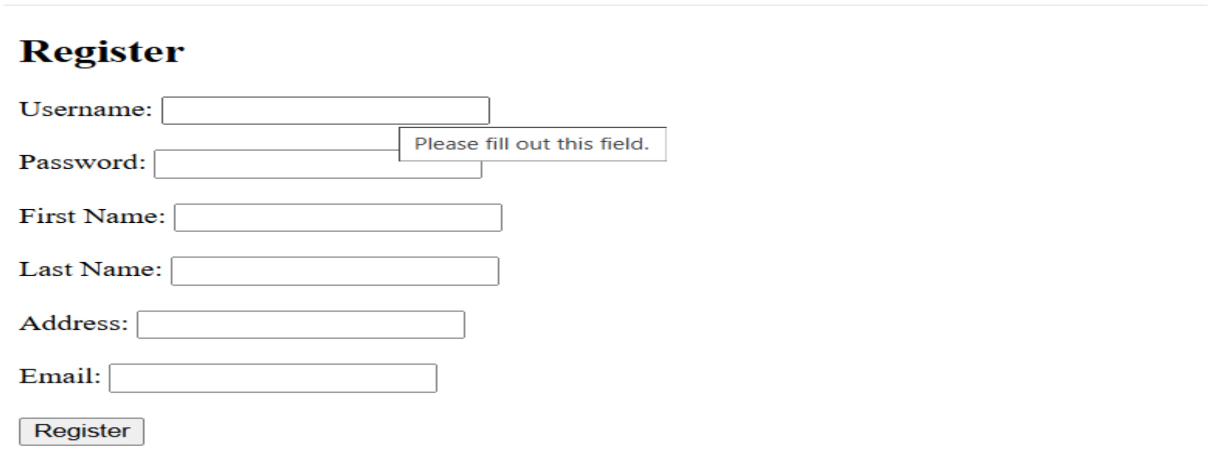
- Hash: 648
- IP: 3
- URL: 33533
- domain: 403
- ip:port: 320
- url: 779

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.



Register

Username:

Password:

Please fill out this field.

First Name:

Last Name:

Address:

Email:

**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
from flask_sqlalchemy import SQLAlchemy
from flask_mail import Mail, Message
import os
from datetime import datetime
from werkzeug.security import generate_password_hash, check_password_hash
```

b) Creating a security key with randomness for secure user and client server interaction or communication. This is used to store session data such as user logins.

```
# Secret key for session management (used for securing user sessions)
app.secret_key = os.urandom(24)
```

c) 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.

```
# Database Configuration (Using SQLite as the database)
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///users.db'  # SQLite database file
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False
db = SQLAlchemy(app)  # Initializing SQLAlchemy for database management
```

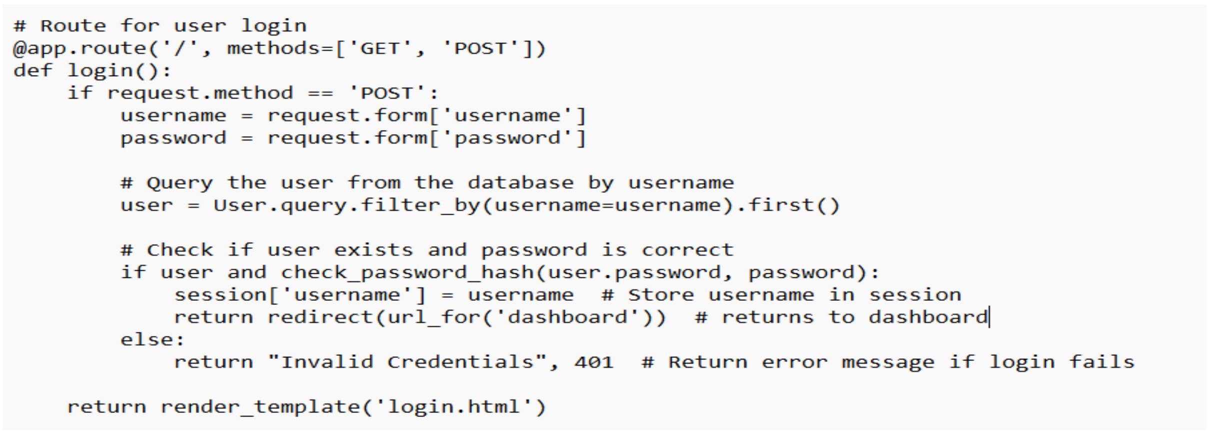
d) Setting up Flask Mail Configuration to send mails to the users from a dedicated mail. MAIL_SERVER uses smtp server for sending emails. The port to use SMTP connection is 587. This number is used because of sending the emails from server to clients using TLS secure connection. It is given TRUE so that 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 (For sending notifications to users after file upload)
app.config['MAIL_SERVER'] = 'smtp.gmail.com'  # SMTP server for sending emails
app.config['MAIL_PORT'] = 587  # SMTP port
app.config['MAIL_USE_TLS'] = True  # Use TLS for secure connection
app.config['MAIL_USERNAME'] = 'somaharsha71@gmail.com'  # Dedicated Sender email
app.config['MAIL_PASSWORD'] = 'dtgp djyz ukup tmmv'  # App-specific password
mail = Mail(app)
```

e) 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.

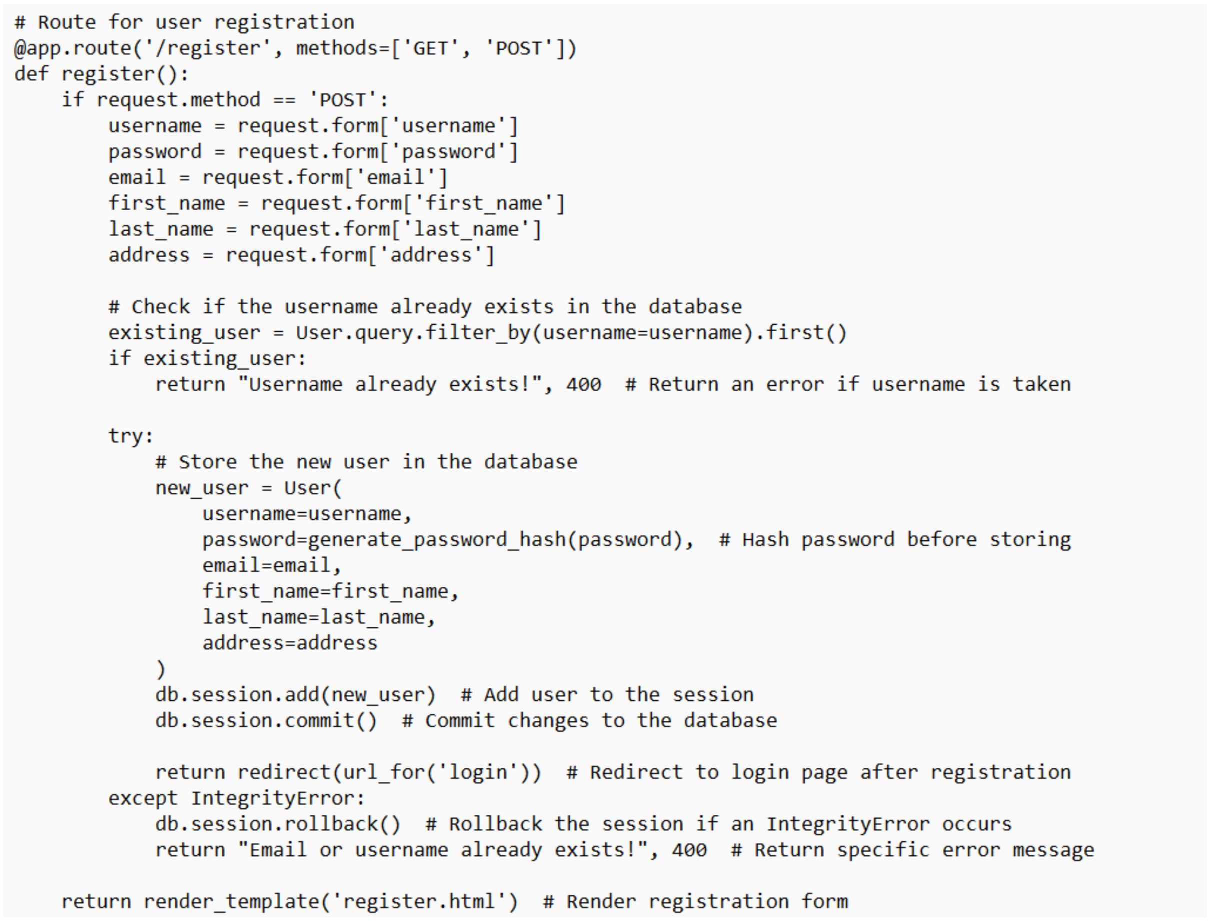
```
# User Model for Database the structure of the user table
class User(db.Model):
    id = db.Column(db.Integer, primary_key=True)  # Unique user ID
    username = db.Column(db.String(50), unique=True, nullable=False)  # Unique username
    password = db.Column(db.String(255), nullable=False)  # Hashed password for security
    email = db.Column(db.String(100), unique=True, nullable=False)  # User's email
    first_name = db.Column(db.String(50), nullable=False)  # User's first name
    last_name = db.Column(db.String(50), nullable=False)  # User's last name
    address = db.Column(db.String(255), nullable=False)  # User's address
```

f) 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.



```
# Route for user login
@app.route('/', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        username = request.form['username']
        password = request.form['password']

        # Query the user from the database by username
        user = User.query.filter_by(username=username).first()

        # Check if user exists and password is correct
        if user and check_password_hash(user.password, password):
            session['username'] = username  # Store username in session
            return redirect(url_for('dashboard'))  # returns to dashboard
        else:
            return "Invalid Credentials", 401  # Return error message if login fails

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

g) 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.



```
# Route for user registration
@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']

        # Check if the username already exists in the database
        existing_user = User.query.filter_by(username=username).first()
        if existing_user:
            return "Username already exists!", 400  # Return an error if username is taken

        try:
            # Store the new user in the database
            new_user = User(
                username=username,
                password=generate_password_hash(password),  # Hash password before storing
                email=email,
                first_name=first_name,
                last_name=last_name,
                address=address
            )
            db.session.add(new_user)  # Add user to the session
            db.session.commit()  # Commit changes to the database

            return redirect(url_for('login'))  # Redirect to login page after registration
        except IntegrityError:
            db.session.rollback()  # Rollback the session if an IntegrityError occurs
            return "Email or username already exists!", 400  # Return specific error message

    return render_template('register.html')  # Render registration form
```

h) 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.

```
# Route for user dashboard (only accessible after login)
@app.route('/dashboard', methods=['GET', 'POST'])
def dashboard():
    if 'username' not in session:  # Check if user is logged in
        return redirect(url_for('login'))  # Redirect to login if not logged in

    # Get user details from the database
    user = User.query.filter_by(username=session['username']).first()
    if not user:
        return redirect(url_for('login'))  # Redirect if user doesn't exist

    # Get current server time
    current_time = datetime.now().strftime('%Y-%m-%d %H:%M:%S')

    # Handle file upload
    if request.method == 'POST':
        if 'file' not in request.files:  # Check if file was uploaded
            return "No file part", 400
        file = request.files['file']
        if file.filename == '':  # Check if filename is empty
            return "No selected file", 400
        filepath = os.path.join('uploads', file.filename)  # Define the file save path
        file.save(filepath)  # Save the uploaded file
        send_email(user.email, filepath)  # Send email notification to user
        return "File uploaded and emailed successfully"
```

i) This part of code works on sending email to the user after uploading the documents. It creates a message with the file attachment.

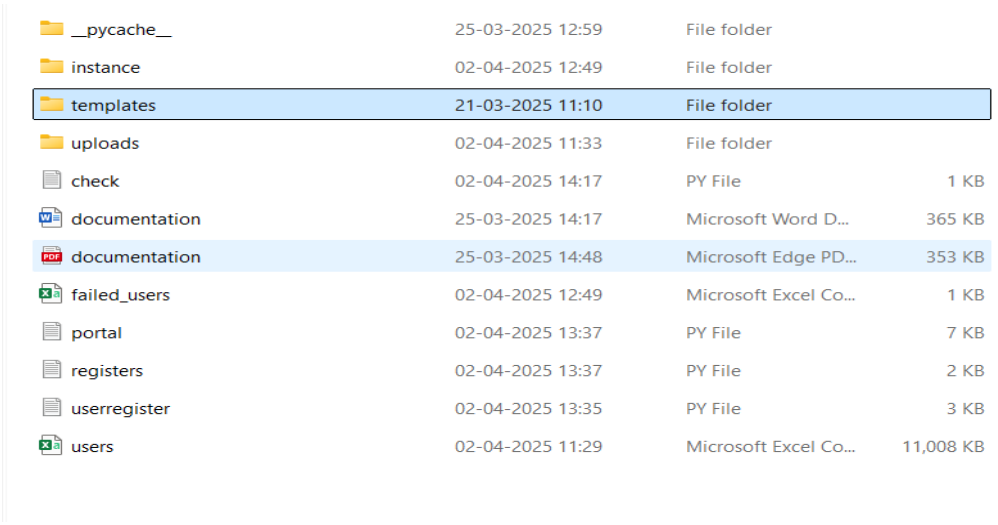
```
# Function to send an email notification after file upload
def send_email(recipient, filepath):
    with app.app_context():
        msg = Message('File Upload Notification', sender=app.config['MAIL_USERNAME'],
recipients=[recipient])
        msg.body = 'A file has been uploaded. See the attachment.'  # Email body

        # Attach the uploaded file to the email
        with open(filepath, 'rb') as f:
            msg.attach(os.path.basename(filepath), 'application/octet-stream', f.read())

        mail.send(msg)  # Send the email
```

## 13. <u>Sqlite database being used with SQLAlchemy:</u>

- 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.

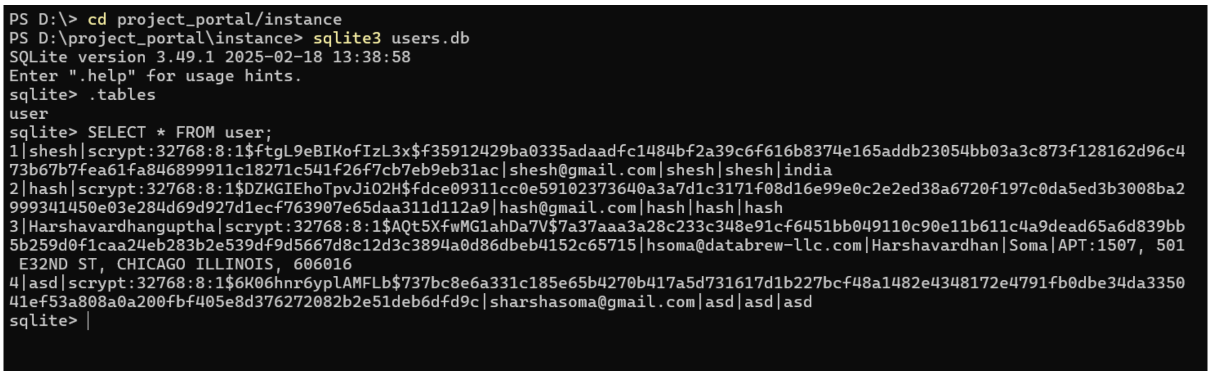
- Since the project is small scale, using light weight database is optimal.

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|scrypt:32768:8:1$ftgL9eBIKofIzL3x$f35912429ba0335adaadfc1484bf2a39c6f616b8374e165addb23054bb03a3c873f128162d96c4
73b67b7fea61fa846899911c18271c541f26f7cb7eb9eb31ac|shesh@gmail.com|shesh|shesh|india
2|hash|scrypt:32768:8:1$DZKGIEhoTpvJiO2H$fdce09311cc0e59102373640a3a7d1c3171f08d16e99e0c2e2ed38a6720f197c0da5ed3b3008ba2
999341450e03e284d69d927d1ecf763907e65daa311d112a9|hash@gmail.com|hash|hash|hash
3|Harshavardhanguptha|scrypt:32768:8:1$AQt5XfwMG1ahDa7V$7a37aaa3a28c233c348e91cf6451bb049110c90e11b611c4a9dead65a6d839bb
5b259d0f1caa24eb283b2e539df9d5667d8c12d3c3894a0d86dbeb4152c65715|hsoma@databrew-llc.com|Harshavardhan|Soma|APT:1507, 501
 E32ND ST, CHICAGO ILLINOIS, 606016
4|asd|scrypt:32768:8:1$6K06hnr6yplAMFLb$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.