pwd - to see the current location

mkdir /path - to create a file in the path mentioned

cd /path – to change the directory

ls – to see current files in it

clear – to clear the screen

pyhton -m venv virt - To create virtual environment

source virt/Scripts/activate - To activate the environment

deactivate – to deactivate the environment

pip freeze – shows all the python things installed in there

pip install flask – install flask and other things

touch hello.py - touch creates hello.py named file

Open sublime text editor

Go to project

Select add folder to project

Add flasker in it

Go to hello.py and start the work

flask run - starts the server

Now that we have created our flask app, we need to run our web server so as for our website to appear in a web browser

So while the server is running and we make changes in it, then for those changes to happen, we need to restart the server every time

export FLASK\_ENV=development | It’s is a development env

export FLASK\_APP=hello.py | The file is in which our app is

flask run | Runs the server on our local machine host 5000

We create a templates folder

And creates a file named index.html

And now this index.html file can be rendered via render\_template(“index.html”) method

We create another file named user.html and render it

While rendering it we also pass on user\_name as a new variable which can be accessed on user.html page

{{ variable | filter}}

Objects, lists , dictionaries anything can be passed

{{ user\_name | upper}}

{{ }} Mostly Variables, data , list items

{% %} Logic, Loops

Now we will create pages to handle errors

Thereby creating files 404.html file& 500.html file in templates folder

------- Version Control / Git Repo ------------------------

1.If virtual environment is active , deactivate it

deactivate

2. cd ~/

We will create an ssh key

For that first we need to go in the default directory that comes up when we open up git bash

3.Now we will create a directory to hold our ssh key

and it also needs to be hidden via using dot .

mkdir .ssh

4.Now we will move in that directory to create our key

via command

ssh-keygen.exe

5. The key is created and we can check this via

ls

6. Now via catalog comd we can see our key i.e public key via

cat id\_rsa.pub | cat key.pub

7. Now we will paste this key in git hub ssh key in settings ssh-key folder

8. Now we will push entire thing on git hub but we don’t need to push virtual environment for version control purposes. We do that by creating a git ignore file via comd

touch .gitignore

We will write /virt in gitignore file

9. Now we will activate the virtual environment

AND then put 5 git commands from codemy.com/git

$ git config --global user.name "Your Name"

$ git config --global user.email "you@youraddress.com"

$ git config --global push.default matching

$ git config --global alias.co checkout

$ git init

Now we have created a git repository | (master)

10. Now to turn it on we use

git add .

11. To save the changes we commit

git commit -am ‘initial commit’ | git commit -am ‘msg’

12. Now we will push it to the git hub and for that, first we need to setup a repository on git hub by creating a repository in settings repository.

13. Now we will push an existing repository from the command line

git remote add origin https://github.com/PhantomXNaveen/flasker.git

git branch -M main

git push -u origin main

Now when we make changes in our code and want to push it to git so as to save code in git hub then we use

git add .

git commit -am ‘msg’

git push

Now we will use bootstrap 5 in base files as in order to use that anywhere we want

Now we will create a base.html template file

Which can be used in other templates file via

{% extends ‘base.html’ %}

We will create another file named navbar.html

And we will use it in base.html via

{% include ‘navbar,html’ %}

href = “{{ url\_for(‘index’) }}” | href = “{{ url\_for(‘function’) }}”

href = “{{ url\_for(‘user’, name=’john’) }}”

Now we will create forms or called as wtforms | Flask WTF

For this we first install web form in our project via

pip install flask-wtf

Now we can import this via

from flask\_wtf import FlaskForm

others things required in wtforms are

from wtforms import StringField, SubmitField

from wtforms.validators import DataRequired

In the forms we have csrf (cross site request forgery) tokens

which creates a secret key on the form that then will sync behind the scenes with another secret key to make sure that hacker hasn’t hijacked your form via

app.config[‘SECRET\_KEY’] = “ URKEY ‘’

We will create a form class so we can use it anywhere and be sure that it is not defined under some @app.route()

Try to use Flask WTF for forms

<form method="POST">

{{ form.hidden\_tag() }}

{{ form.name.label }}

{{ form.name() }}

{{ form.submit() }}

</form>

For messages to be shown we import flash and then use

flash(‘msg’)

and use in other templates every time via

{% for msg in get\_flashed\_messages() %}

{{ msg }}

{% endfor %}

Now we will create a folder with exact name static

Where to use css file via link tag

<link href=" {{url\_for('static', filename='css/style.css') }} " rel="stylesheet">

new.js file

document.getElementById('demo')

where we use javascript via script tag

<script src="{{ url\_for('static', filename='js/new.js') }} "></script>

To use database –

1. We will install it in our directory via

pip install flask-sqlalchemy

and check via pip freeze

2. We will now import SQLAlchemy via

from flask\_sqlalchemy import SQLAlchemy

and from datetime import datetime

as well so as to know when the data was added

3. Now we will add database in our app project via

app.config[‘SQLALCHEMY\_DATABASE\_URI’] = ‘sqlite:///users.db’ | users name of db u want to create

4. Now we will initialize the database via

db=SQLAlchemy(app)

5. Whenever we use database we need to define model i.e what we want to save to database and we do it via class i.e

class Users(db.Model):

id=db.Column(db.Integer, primary\_key=True)

name=db.Column(db.String(200),nullable=False)

email=db.Column(db.String(100),unique=True) dat=db.Column(db.DateTime,default=datetime.utcnow)

We will also create a string to sort of designate sth

def\_\_repr\_\_(self):

return ‘<Name %r>’ % self.name

Now we have to create the database and set this all up from our terminal via

Go to python shell –

winpty python

To create database –

from hello import db

db.create\_all()

Second way -

In hello.py file

@app.shell\_context\_processor

def make\_shell\_context():

return {'db': db, 'Users': Users}

create a init\_db.py file

from hello import app,db

with app.app\_context():

db.create\_all()

print("Database tables created")

If u got py file msg via terminal

python init\_db.py

U can verify it via seeing if u see users file in instance or flasker folder

In here we have created a user form and then used it in our @app.route def method

U can use bootstrap for more modifications

----------------------------------------------------------------------------

To use MYSQL Database

First we create a MYSQL DB via

app.config[‘SQLALCHEMY\_DATABASE\_URI’]=’mysql://username:password@localhost/db\_name’

Now we will create a create\_db.py file to create the database

In order to create a database we need a connector

$ pip install mysql-connector

$ pip install mysql-connector-python

$ pip install mysql-connector-python-rf

One of these will work

Now in shell we run file create\_db.py

create\_db.py file

import mysql.connector

mydb = mysql.connector.connect(

host="localhost",

user="root",

password="root")

my\_cursor = mydb.cursor()

my\_cursor.execute("CREATE DATABASE our\_users")

my\_cursor.execute("SHOW DATABASES")

for db in my\_cursor:

print(db)

Run the comd

python create\_db.py

It will show our\_users file in comd if db created

Now then to connect mysql we need to install other things unlike in the case of sqlite

pip install pymysql

pip install cryptography

Since we are using pymysql as connector we need to make some changes in app.config db creation

app.config['SQLALCHEMY\_DATABASE\_URI']='mysql+pymysql://root:root@localhost/our\_users'

After that we run the comd

winpty python

from hello import db

db.create\_all()

Again it shows error like in case of sqlite

So we run the init\_db.py file via

python init\_db.py

from hello import app,db

with app.app\_context():

db.create\_all()

print("Database tables created")

And the msg database created confirm database is connected

---------------------------------------------------------------------------------

Update a Record in Database

For this we create a update route with id for updation and methods GET and POST since this is form

We create a update.html file with form in action in which the form shown is already filled

We also add link to our\_user.name which points to update defined in hello.py file

---------------------------------------------------------------------------------

To Migrate Database | Deal with databases | Migration

Usually a two-step process – Create a Migration

Push the Migration

Create a Migration means defining the data

Now see that we have make changes in the databases i.e in the code, we have added a new column favorite\_color but we haven’t migrated i.e changed anything internally in the database

Which is done by in the virt i.e virt env must be turned on

pip install Flask-Migrate

Now we need to make some changes in hello.py file

from flask\_migrate import Migrate

and where the database was initialized

migrate = Migrate(app,db)

Now we need to sort of turn it on via

flask db init

This has created a directory that hold all the migrations stuff

flask db shows how to use db commands

Now to make our initial migration

flask db migrate -m ‘Initial Migration’

i.e flask db migrate -m ‘message’

Now to push the migration

flask db upgrade

Now next time we make any changes use the above two command flask db migrate -m ‘message’

flask db upgrade

---------------------------------------------------------------------------------

Delete Databases Records

We create a app route with with id for deletion and methods GET and POST since this is a form

We def delete()

With database command db.session.delete(user\_to\_del)

We can also use bootstrap such as for to add red color button properties to href link of delete

---------------------------------------------------------------------------------

Hashing Passwords –

Creates the password into a long hash key

And often used so as one can’t login into another’s account

Via pip freeze we can see Werkzeug in there

Now we will import it via

from werkzeug.security import generate\_password\_hash, check\_password\_hash

new db column | password\_hash=db.Column(db.String(128))

@property

def password(self):

raise AttributeError ('password is not a readable attribute')

@password.setter

def password(self,password):

self.password\_hash=generate\_password\_hash(password)

def verify\_password(self,password):

return check\_password\_hash(self.password\_hash,password)

Naveen Yadav@LAPTOP-9GTOD1UM MINGW64 /c/flasker (main)

$ flask shell

Python 3.12.3 (tags/v3.12.3:f6650f9, Apr 9 2024, 14:05:25) [MSC v.1938 64 bit (AMD64)] on win32

App: hello

Instance: C:\flasker\instance

>>> from hello import Users

>>> u = Users()

>>> u.password = 'cat'

>>> u.password

Traceback (most recent call last):

File "<console>", line 1, in <module>

File "C:\flasker\hello.py", line 38, in password

raise AttributeError ('password is not a readable attribute')

AttributeError: password is not a readable attribute

>>> u.password\_hash

'scrypt:32768:8:1$DPT8FmDtkXSmfS4h$b8af3fb1aff7b22ac6a4fe923ffba3d0ab48ad12db669de16dd0b67eb1dec6794e8bc285d15ca6272090814872019df8e171d1579b04a0fa6c5d537ac56b7b66'

>>> u.verify\_password('cat')

True

>>> u.verify\_password('dog')

False

>>> u2 = Users()

>>> u2.password = 'cat'

>>> u2.password\_hash

'scrypt:32768:8:1$cGKnqjey8JFDAHHY$ac2df43f4f4588e64d15d3eea3a109e2176135b9b2585822ce050f086e1740ba98c43e1a6abd9e5e66556c9a623356702ccdd5d43c0e57ebf55497065bb1c640'

>>> u2.verify\_password('cat')

True

>>>

---------------------------------------------------------------------------------

Using Hashed Passwords for Registration

Now to implement the changes done we done it via

flask db migrate -m ’password added msg’

flask db upgrade

Now we will add password\_hash db column in add\_user.html

Import

PasswordField, BooleanField, ValidationError

EqualTo, Length

hashed\_pw = generate\_password\_hash(form.password\_hash.data, method="pbkdf2:sha256")

We have edited the add\_user function and add\_user html file

And used the hashed\_pw to pass on to the form

--------------------------------------------------------------------------------

Comparing Hashed Password To Plaintext Passwords

Here we have created a

@app.route(‘/test\_pw’, methods=[ ‘GET’ , ‘POST’ ]

And defined def test\_pw()

And rendered test\_pw.html page

---------------------------------------------------------------------------------

To return JSON With Flask for an API

If we return a pyhton dictionary , flask will jsonify it for you

We import date for this one , any dictionary will do

@app.route('/date')

def get\_current\_date():

return {"Date": date.today()}

--------------------------------------------------------------------------------------------------------------------------

Create a Blog Post Model

We will create a Posts Model

content = db.Column(db.Text) | db.Text is used and not db.String

It is so because content needs a whole text field and not some strings.

And in the PostForm of it we will require

from wtforms.widgets import TextArea| widget=TextArea

and in form add attribute rows = “x”

slug = db.Column(db.String(255))

slug is used where we see the url of page so it looks kinda informative or to designate it

Now we will add it to database via

$ flask db migrate -m 'Add Posts Model'

$ flask db upgrade

Now we will create a PostForm class

Then we will create a PostPage

We will also create a add\_post.html file

---------------------------------------------------------------------------------

Show Blog Posts Page

We will create a route @app.route(‘/posts’)

And define def posts():

Which renders posts.html file created for posts

---------------------------------------------------------------------------------

Individual Blog Post

We will create a route @app.route(‘/post/<int:id>’)

And define def post()

Which renders post.html file created for specific post

---------------------------------------------------------------------------------

Edit Blog Page

We wiil create a

@app.route(‘/posts/edit/<int:id>’,methods=[‘GET’,’POST’])

methods is used whenever there is a form

and def edit\_posts(id)

and then to redirect we import redirect to post, url\_for

and then render edit\_post.html

---------------------------------------------------------------------------------

Delete Blog Page

We just create a route @app.route('/posts/delete/<int:id>')

And then define method delete\_post(id)

Which we can call and also to commit changes in database

---------------------------------------------------------------------------------

Login with Flask

We will go in terminal and install login via

pip install flask\_login

pip freeze |to see all the stuff i.e if login is there or not

It does not do account registration

You need to have some sort of model setup for this which we did earlier hashed password

you can check out flask\_login documentation for more info

Now we will import

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

Now we will pass

UserMixin as parameter in Users Model | i.e class Users

We have added a new column username in Users Model so now to add it to database we use

flask db migrate -m ‘msg’

flask db upgrade

If a valueError occours, go in migration version and in

batch\_op.add\_column and batch\_op.drop\_constraint change none to users\_username

Now we will create a login route with login.html file page

And also a dashboard route with login.html file page

We will also make a LoginForm i.e

class LoginForm(FlaskForm):

We will also create a FLASK\_LOGIN Stuff i.e the main stuff

login\_manager = LoginManager()

login\_manger.init\_app(db)

login\_manager.login\_view = ‘login’

@login\_manager.user\_loader

def load\_user(user\_id):

return Users.query.get(int(user\_id))

--------------------------

@login\_required used below dashboard which prevents the access to directly access the page and we need to login

-------------------------------------------------------------------------

Create a UserDashboard

{% if current\_user.is\_authenticated %} is used in navbar link so as to change the login to logout and vice-versa

---- add user changed to register ----

And we can use current\_user to access name,username,etc

---------------------------------------------------------------------------------

Lock Down Your App -

Two ways to do this

In first one we use @login\_required below the route we want to lock in

In second one we use if else statement in the specific template file

---------------------------------------------------------------------------------

Update User Profile From Dashboard

In form action make it to route ‘/dashboard’

And same as update user, copy the code and in case of id use current\_user.id and if current\_user.is\_authenticated

----------------------------------------------------------------------------

Update Correct User

Use @login\_required so as to lock only the authenticated one can update the page

& use id and current\_user.id as a criteria to update the page

-------------------------------------------------------------------------------

Setting up of foreign key with one to many relationship

Create one to many Databases

In Blog Post Model

We will create a new column in Post which is foreign\_key id

And refers to the primary key of the user

To refer we use users.id and Not Users.id cause users is the database and Users is the class

poster\_id=db.Column(db.Integer, db.ForeignKey('users.id'))

IN User Model

We define a posts column | Posts is the class

posts = db.relationship('Posts',backref='poster')

In flask db upgrade a problem may occur so go in migrations versions and change none to ‘sth’

Make changes everywhere where there was author

---------------------------------------------------------------------------------

Only allow Correct user to delete posts

Just add @login\_required below the route

And with the use of current\_user.id = id , delete the posts

Only that are authorized

---------------------------------------------------------------------------------

Allow Correct User to Delete the Post

Allow Correct User to Edit the Post

---------------------------------------------------------------------------------

Search Blog Posts from Navbar

Let’s go in navbar page find search and in the form add method post and action or points it to ‘search’ which we will create now and to prevent form from hijacking by using csrd token i.e {{ form.hidden\_tag() }}

We will create SearchForm in webforms.py

And def a search function in hello.py

Till now we can easily pass stuff to the pages but to pass stuff to the navbar we do it via

@app.context\_processor

def base():

form = SearchForm()

return dict(form = form)

Via this form is passed on to the base file which is then passed to navbar filer

---------------------------------------------------------------------------------

How to Add a Rich text Editor

For this we need to install sth called ckedioter via

$ pip install flask-ckeditor

For further steps visit flask-ckeditor documentation

We will create it’s instance via after importing it

from flask\_ckeditor import CKEditor

ckeditor = CKEditor(app)

Now to the page where we want to use CKEditor we load it in the body of the file via --|

Don’t need it since using it with wtf form, it is implemented in a different way

{{ ckeditor.load() }}

Since we earlier worked with wtforms we will use that one

Since our form is in webforms.py and we need to import to use CKEditorField via

from flask\_ckeditor import CKEditorField

And in the specific form where we want to use this one we will do it via

body = CKEditorField(‘Body’)

Now we need to call the ckeditor.load() on the page itself where the form is |

don’t use above ckeditor.load() since with the wtform it is a different way

we load it below the form tag via

{{ ckeditor.losd() }}

{{ ckeditor.config(name=’Body’) }}

We also need to use safe filter so it allows to make changes in html via

{{ post.content | safe }}

---------------------------------------------------------------------------------

Edit Blog Posts with Rich Text Editor

Same steps as done above.

But since form used is same in both i.e PostForm

Thus, we just need to load the ckeditor for Rich Text Editor via

{{ ckeditor.load() }} | {{ ckeditor.config(name=’body’)}}

---------------------------------------------------------------------------------

Basic Admin Page

Flask doesn’t have admin area like Django (superuser stuff)

We need to create a one

So create a admin page via def @app.route(‘/admin)

And then def admin which return admin.html and redirect to dashboard or anything else if not admin

Use @login\_required to restrict access

And current\_user.id as criteria for admin login or etc.

---------------------------------------------------------------------------------

Format Search Results

Just making some changes in search.html file to make it look nicer

Using of if else statement and some code of posts.html

---------------------------------------------------------------------------------

Set Default Profile Pic

Drag the pic to images folder in static folder of flasker

Add that to the dashboard via

img src={{ url\_for(‘static’,filename=’images/file.img’) }}

Now to set the default pic at right place we need to use some css or bootstrap via using div container row and columns

Below one creates a div container od size 8 column which is done via

<div class=”container”>

<div class=”row”>

<div class=”col -8”> </div>

<div class=”col-4”> </div>

---------------------------------------------------------------------------------

About Author Section

We will Import TextAreaField from wtforms

Add about\_author column in User class Model

And then also make about\_author field in the UserForm

Migrate and Push it to database

Make changes everywhere where you want to see about\_author

---------------------------------------------------------------------------------

Upload Profle Picture -

Add a profile pic field in form field

We need FileField for this so we need to import it to use it

From flask\_wtf.file import FileField

Add profile\_pic in form in dashboard as well

Add column in hello.py file in class Users Model as well

We use String here since we don’t save pic but name of pic

Make migration and upgrade it

Add profile\_pic in dashboard function as well

We need to import

from werkzeug.utils import secure\_filename

import uuid as uuid

import os

since pic name may cause sql injection

we use

UPLOAD\_FOLDER = 'static/images/'

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

The the place to save the pics in the database

----------------------------------------------

Done in dashboard to save pic | files is used

name\_to\_update.profile\_pic = request.files['profile\_pic']

# Grab Image Name

pic\_filename=secure\_filename(name\_to\_update.profile\_pic.filename)

# Set UUID

pic\_name = str(uuid.uuid1()) + "\_" + pic\_filename

# Save that Image

saver = request.files['profile\_pic']

# Change it to a String to save to database

name\_to\_update.profile\_pic = pic\_name

saver.save(os.path.join(app.config['UPLOAD\_FOLDER']), pic\_name)

It may not work so no need to worry but u can see that in database some change is occouring

And in case we don’t need to save it to database, we will save it online.

---------------------------------------------------------------------------------

Deploy Flask App with database on Heroku

Since a problem may occur while deploying it, so lets rename hello.py to app.py

Download the Heroku installer, Install it

Add the path if not added in environment variables

Earlier we use flask run but now we need server via

$ pip install gunicorn

For Heroku we need postgres sql database so we install it

$ pip install psycopg2

pip freeze | Shows all things we are using

And for Heroku to be made aware of it, we do it by making file requirements.txt

pip freeze > requirements.txt

Next we need a proc file which tells Heroku what kind of app we are running via

$ echo web: gunicorn app:app > Procfile

For Reference echo web : server app:app\_name > file\_name

Now let’s go and login to Heroku via

heroku login

Ctrl + c to break Now create app on heroku via

heroku create app\_name

heroku addons:create heroku-postgresql :hobby-dev --app app\_name

heroku config –app app\_name | Gives database url, copy it and use it in

app.config[‘ SQLALCHEMY\_DATABASE\_URI’] = ‘postgres://copiedURL’

Now save changes to GIT via

git add .

git commit -am ‘tweaked app for heroku’

git push

Now to push it to heroku we use

git push heroku main

Now to add database to heroku we do it via

heroku run python

>>> from app import db

>>> db.create\_all()

---------------------------------------------------------------------------------------------------------

Only Delete Correct User

Use @login\_required below the route

And with the use of current\_user.id create if condition to delete and otherwise redirect them to dashboard

---------------------------------------------------------------------------------

Fix And Show Profile Picture

Just change this one line code in add profile\_pic in dashboard

saver.save(os.path.join(app.config['UPLOAD\_FOLDER'],pic\_name))

And use this to show pic

{% if current\_user.profile\_pic %}

<img src="{{ url\_for('static',filename='images/' + current\_user.profile\_pic) }}" width="200" align="right">

{% else %}

<img src="{{ url\_for('static',filename='images/icons8-profile-48.png') }}" width="200" align="right">

{% endif %}

---------------------------------------------------------------------------------

Null Profile pic

Issue not solved

---------------------------------------------

Add Profile pic

Blog post was edited using div containers and img src url\_for pointing to static images files

-------------------------------------------------

Designate Admin To Delete

Just used any id and used in with or condition to bypass the if loop and thus access to various privileges

-----------------------------------------------------------------------