Page No.
Date

Flask App Description

File 1: __init__.py

__init__.py → Creating such a file tells the python compiler that it is a package. This file can also be empty, it is just used to initialize a package

Following libraries have been imported in this file and we'll see what it does line by line

→ os library → provides a way of using operating systems dependent functionality

→ Flask → This module is imported from flask library. This module helps in initializing the web app

→ SQLAlchemy → This module is imported from flask_sqlalchemy module (library). It is an extension of flask. It provides us with a database

→ Bcrypt → This module has been imported from flask_bcrypt library which is an extension of flask. It provides hashing utilities for the application

→ LoginManager → This module has been imported from flask_login, an extension of flask. It handles common tasks of logging in users logging them out as well as remembering the users.

→ Mail → this module has been imported from flask_mail library, an extension of flask. This is used to connect to a mail server.

Line 8: We give our app a name. The name of the webapp is app

Line 9: We need to set a 'secret key' for our application. The secret key will protect against modifying cookies and forgery attacks. We set it as shown. To generate a secret key we use built in 'secrets' module and then use token_hex method.

→ import secrets

→ secrets.token_hex(16) # 16 is no. of bytes

Writing this code we get a 16 byte secret key and use that as shown.

• Line 10: In this line we configure our SQLAlchemy database having name site.db We create SQLite database on our file system. The 3 forward slashes denote a relative path

Line 12: After creating database we create an instance of it as shown.

Line 13: Using plain text passwords can be risky and hence we use hashed passwords Bcrypt is used to hash passwords

Line 15: We create instance of LoginManager so that we can use it for logging in and out users. It is used to handle sessions in

background. This adds functionality to database models [models.py file]

Line 16: In this we set the login route. The login_view that we pass here is function name of login route which is called 'login'

Line 17: It is used to give an info type message telling to login into account incase user tries to access some feature of application which requires him to be logged in.

Line 19, Line 20, Line 21, Line 22, Line 23: These lines are used to connect to a gmail server to send reset password link mail in case user forgets his password.

Line 25: In this line we create a mail extension

Line 27: In this line we import our routes.py file

File 2: routes.py

The routes.py file consists of all the possible routes to which our webapp can be redirected like the home page, about page, etc.

Following are the libraries used in routes.py file:

→ os → Described earlier

→ secrets → Used to generate a secret code, described earlier

→ Image → This module is imported from pillow library denoted as PIL. This library would be used to save (when we) and update profile pic and have to view the pictures

→ render_template → Module imported from flask library. It redirects to the HTML page corresponding that route

→ url_for → Module imported from flask library. It is used for creating a URL to prevent the overhead of having to change URLs throughout an application. Without url_for if there is a change in root URL of app then you have to change it in every page where link is present

→ flash → Used to give a flash message for the user. This module is also imported from flask

→ redirect → Module imported from flask. It is used to redirect user to target location

→ request → Module imported from flask. Data from client's web page is sent to server as

a global request object
→ abort → Module imported from flask. It aborts a request with an HTTP error code
→ app, db, bcrypt, mail → These have been imported from __init__.py file and have been described earlier
→ RegistrationForm, LoginForm, UpdateAccountForm, PostForm, RequestResetForm, ResetPasswordForm → These have been imported from forms.py file and will be described later
→ User, Post → These have been imported from models.py file and would be described later
→ login_user, current_user, logout_user, login_required → These have been imported from flask_login module, an extension of flask and are self-explanatory
→ Message → Module imported from flask_mail an extension of flask. This would send a message for reset password in case user forgets password

Line 12 to Line 17 (both inclusive)
In this we create our home route which redirects to home page of website. In line 12 the forward slash means that if we don't specify any route, website would open at home page. To open home page of website we can specify nothing or /home route (line 13) Line 15 specifies the default page that we get when we first open the website. We get page 1 of website when we open it. In line 16 we order the posts in descending order of

date created and paginate it so that it contains 5 blogs per page. In line 17 we return the html file corresponding to that page.

Line 20 to Line 22 (Both inclusive)
Line 20 redirects to about route. We return the corresponding html file in line 22

Line 25 to line 37 (Both inclusive)
This is for registering a new user so that he can write blogs on the website. Line 27, 28 say if user is already logged in and tries to re-register himself, he would automatically be taken to the home page. Line 29 calls the RegistrationForm class from forms.py file. Line 30 to line 36 says if we enter a valid username, password, email, the password is hashed, user is added to database and we get a flash message saying account has been created. This would redirect to the login page after we commit changes in database. register.html is corresponding html page for registering users

Line 40 to line 53 (Both inclusive)
This route is created for user to login once account has been created. Line 42, 43 say if user is logged in and tries to re login, he'll

be directed to the home page of website. Line 44 calls the login form from forms.py file. Line 45 to line 52 say if user enters a valid email and passwords and clicks on login button the email and password are checked & verified from database and user is redirected to home page if login is successful. In case login is not possible a flash message is displayed saying login unsuccessful. login.html is the corresponding html page

Line 56 to Line 59 (Both inclusive)
this route would log a user out who has logged in and redirect to the home page of website.

Line 62 to Line 73 (Both inclusive)
This function is used to save a picture when we upload a new profile pic. We take picture as an argument. We'll save uploaded image to our filesystem. Here we won't keep the name of file same that they uploaded because it can collide with name of image already in our file system. We'll use secrets module to name images and save them. We'll grab file extension of image using splitext as module line 64 returns file name and extension. We don't need file name and store it as an underscore and store file extension in f_ext. In line 65 we combine our new generated filename with

extension. In line 66 we provide full path of where the image would be stored. We also need to resize our image so that not too much storage is taken. We resize image to 125 x 125 pixels in line 68 i.e store our final image size in output_size variable on line 69 we open image, resize it to output_size and save it at the path mentioned and in line 73 return the image name

Line 76 to Line 94 (Both Inclusive)

Line 77 means that a user must be logged in to access the account route. When a logged in user goes to this account he can view the user name, email, as well update them and also update the profile picture. In line 79, the UpdateAccountForm() is in forms.py file. If a user enters a valid email, valid username and uploads a valid picture i.e a picture with .png or .jpg extension then the picture, username, email would be update in the database and a flash message is sent saying account has been updated and we are redirected to account page. In line 89 to 91, we want that the original username, email of user should be filled and when user goes to account route, he can see the original username & email id. Line 92 says that profile pic of user would be the already set one. Line 93 tells that account.html is the corresponding html

page.

Line 97 to line 108 (Both inclusive)
This route is used to create a new post. Only a user that is logged in would be able to create a new post. In line 100 we use PostForm() create in forms.py file. To create a new post, title and content of post must be filled. Once the user creates a new post, the post would be saved in database and user would get a flash message saying post is created and user would be redirected to home page. create_post.html is the corresponding html file.

Line 111 to line 114 (Both inclusive)
A user can also access the post by typing /post/<post_id> of a particular post. If the user enters a valid post id, he will be redirected to that post otherwise we get error 404. post.html is the corresponding html file.

Line 117 to line 134 (Both inclusive)
A user can update/delete only his or her posts. If a user tries to update a post created by other user, the application throws an error. A user can also try to update a post using this route /post/<post_id>/update but if the

Page No.
Date

post doesn't belong to user a 403 forbidden error will be shown. Line 123 shows PostForm() taken from forms.py file. The title and content of post would be filled with the original post title and content. Once a user makes changes, the post will be update, changes will be made to database and user will get a flash message saying post is updated and would be redirected to post page. Line 130 to 132 tells that the title and content field would be filled with original content and its upto the user to either completely delete the content & title or update the existing content & title. create_post.html is the corresponding html page.

Line 137 to Line 146 (Both inclusive)

This route allows user to delete a post. A user can only delete a post that he has created and he should be logged in to delete the post. If a user tries to delete post of another user by directly going to that route an error will be thrown. Once user clicks on delete button a conformation would be asked if he wants to delete the post. If he clicks on delete, changes would be made in database and user will get a flash message saying post is deleted and would be redirected to home page.

Line 149 to Line 169 (Both inclusive)

* a particular user

A user might want to reset his password
A user can view all the posts written by a particular user by going to this route, /user/<username>. The user would be redirected to first page. In case username is invalid a 404 error would be thrown. The posts would arranged in order of date created with the latest post being on top. Only 5 post would be displayed per page. user_posts.html is the corresponding html page.

Line 159 to Line 169 (Both inclusive)
A user might want to reset his password/ change the password. get_reset_token() is a function in models.py file. Line 161 contains email subject, Line 162 contains the sender's email id and line 163 contains the receivers mail. Lines 164, 165, 166, 167, 168 contain message to be sent and finally the message is sent in line 169.

Line 172 to Line 182 (Both inclusive)
At the login page if user clicks on forgot password, he is redirected to the reset password route. In case user is already logged in, he would be redirected to home page if he tries to access the reset_password route. RequestResetForm() in line 176 is imported from forms.py. If the correct email is entered and email is found in database, the email

containing reset password link would be sent and a flash message would appear saying the email has been sent. The user would be redirected to login page. reset_request.html is html page for the same

Line 185 to line 200 (Both inclusive)
Once the reset password link is sent, the user clicks on the link and it takes user to /reset_password/<token> route. If the user is logged in, he'll be redirected to home page. If the link has expired or is invalid, user would be redirected to reset password page and will get a flash message saying invalid or expired token. Line 193 shows ResetPasswordForm() imported from forms.py file. If the link is valid, user would be asked to re-enter password, confirm password, the password would be hashed and would be update in database. User will get a flash message saying password is updated and he can log in and would be redirected to login page. reset_token.html is html page for the same

File 3: forms.py

In forms.py we create all the forms used in our website.

Following libraries have been used in forms.py file:

→ FlaskForm: Module imported from flask_wtf library, an extension of flask. This is used to create forms in flask

→ FileField: Module imported from flask_wtf.file, an extension of flask. FileField is used when we want to upload a file. In our case we upload the image file

→ FileAllowed: Module imported from flask_wtf.file, an extension of flask. FileAllowed is used when we have to specify extension of file that can be uploaded. In our case, we can upload the image file as .png or .jpg.

→ current_user: Module imported from flask_login library, an extension of flask. current_user defines the currently logged in user

→ StringField, PasswordField, SubmitField, BooleanField, TextAreaField: Modules imported from wtforms which define the type of field in which user would enter data

→ DataRequired, Length, Email, EqualTo, ValidationError: Modules imported from wtforms.validators. These are conditions which need to be fulfilled when entering data in a form.

Line 9 to Line 27 (Both inclusive)

In these lines, we create the registration form for registering new users. We take the username, email, password, confirm_password field and add certain self-explanatory validators. In line 17, we create a submit button to sign up. Line 19 to 22 are used to validate whether the username is correct or not. If the username is taken i.e username is found in database, the user would have to choose a different username. Same is the case for email (line 24 to 27)

Line 30 to Line 35 (Both inclusive)
In these lines we create our login form. The form accepts email, password with certain validators. A user can choose whether to 'Remember Me' or not by clicking the checkbox. In line 35, we create a submit button to login the user.

Line 38 to Line 56 (Both inclusive).
These lines are quite similar to our Registration form. Only some changes have been made. In line 43, the user can update his profile picture. 'FileField' allows to ~~add~~ upload a file and 'FileAllowed' ensures that the uploaded file has a jpg or png extension. In validate_username, validate_email functions, we only validate the email and username if they change

otherwise we don't.

Line 59 to Line 62 (Both inclusive)
These lines allow a user to create a new ~~poem~~ post. It accepts title and content of post with certain validators. User clicks on submit button to add a post.

Line 65 to Line 73 (Both inclusive)
This form allows user to enter his registered email. Once submit button is clicked, the email is validated and then a reset password link is sent

Line 76 to Line 80 (Both inclusive)
Once the password is reset, user would have to enter new password and confirm the same. Then a user clicks on submit button to reset password. The ResetPasswordForm allows this.

Page No.
Date

File 4 : models.py

This file was used to create a database and fill the information of various users. Following libraries have been used in this file :

→ datetime → Module imported from datetime library. Posts have been arranged in according to date posted with latest post being at top. Here we use this library to store date of created post

→ TimedJSONWebSignatureSerializer → This module has been imported from its dangerous library. Usually, the reset link has a time period for expiry. We generate the reset link / token for a fixed period of time using this library to reset password

→ db, login_manager, app → Imported from __init__.py file. Refer there

→ UserMixin → This module is imported from flask_login, an extension of flask. Online we have created a decorator. The extension expects to have certain attributes and methods. There will be 4 methods which are, i) is_authenticated which returns true if provided valid credentials, ii) is_active, iii) is_anonymous, iv) get_ID. We can add all of these ourselves but these are very common methods and extension provides with simple class that we can inherit from that would add all these requirements i.e required attributes and methods. This class is

UserMixin which is imported from flask_login.

Line 7 to Line 9 (Both Inclusive)
This function is used for reloading the user from the user ID stored in the session. This can directly be copied from documentation. We have added a decorator (line 7) which tells the extension that this is the function to get a user by an ID.

Line 12 to Line 34 (Both Inclusive)
From line 13 to 18, we create our database with these columns and the specified constraints that are self explanatory. In line 18, the post & user model will have a relationship, a one to many relationship because 1 user can have multiple posts but vice versa isn't true. We'll set posts attribute (line 18) to a relationship having relationship to Post model, backref = 'author', it is similar to adding another column to the post model. When we have a post, backref allows us to use the author attribute to get the user who created the post. lazy = True defines when SQLAlchemy loads the data from database. lazy = True, putting True here means that SQLAlchemy will load the data as necessary in one go. Using this we can easily get all the posts created by an individual user. Note: This is a

relationship not a column, if we open database we won't see such a column.
Line 20 to 22 gives us the link to reset our password. Here we define the time in which the particular link expires, 30 minutes in this case.
On line 24 to line 31, this function verifies the token created in previous function. We might get an invalid token which could throw an exception. To prevent abrupt errors, we use try - except block. We'll try to get user_id in try block. If we don't get the user_id, then we return None in except block. If we get user_id w/o an exception, we return user with that ID. Clearly this method doesn't do anything with the instance of the user, so we declare it as a static method (line 24)
On line 33, 34 we specify __repr__ function which is a dunder/ magic method. This specifically tells how our object is printed whenever we print it. We'll specify what we want this to look like when we print out a user object

Line 37 to line 45 (Both inclusive)
From line 38 to line 42 we create post database with these columns and specify constraints that are self explanatory. In line 44, 45 we have a __repr__ function which has the same purpose as we

Page No.
Date

write ~~is~~ above i.e line 33, 34.

Q How does the application run?

A. In our directory, we have run.py file and flaskblog folder. To run the webapp we run the run.py file from terminal and import app from flaskblog. Doing this import redirects us to __init__.py file in flaskblog which is used to initialize a package in Python. The __init__.py file does the basic setup for running the app and in the end of __init__.py file we call our routes.py file which consists of all the routes used in our website. The routes.py file imports various classes & functions from forms.py & models.py file and uses those functions wherever required

* Also refer video 5 - Corey Schafer flask Tutorials to see how to create a package structure for our webapp.

* flaskblog folder contains all the files and folders used. site.db is our database, templates folder contain all the HTML files and static folders consists of CSS file & profile pics folder