[Flask]

General Information & Licensing

Code Repository	https://github.com/pallets/flask
License Type	BSD-3-Clause License
License Description	• The BSD-3-Clause license applies to all files in the Flask repository and source distribution. This includes Flask's source code, the examples, and tests, as well as the documentation. • Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:
License Restrictions	• Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer. • Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution. • Neither the name of the copyright holder nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.
Who worked with this?	Hyukjoo, Changho, Brian, Timothy

[app = flask.Flask(__name__)]

Purpose

The Flask class passes the parameter __name__ which allows the class to find resources on the file system. Flask uses folders such as templates and statics to store and reference html and css files. Therefore, passing in __name__ allows the class to find files in the directory in reference to the file where the class is initiated.

This class is initiated in the 4th line of the server.py file as the route function in the class will be used to route directories for the website.

Magic ★★ ° ˙ ˚ ☽ ˚ ⌒ ˚ ★ ≡ ★ ≡

The Flask class can be found at https://github.com/pallets/flask/blob/main/src/flask/app.py from line 98 to the end of the file. The class can take parameters but are in the form of the Scaffold class; which can be found at https://github.com/pallets/flask/blob/ea93a52d7d94ba093bbce4680c622cc4fc9771d8/src/flask/scaffold.py#L751 from lines 62 to the end of the file. When only 1 parameter is passed, __name__, this class uses the passed file directory to find the other folder; such as templates and statics for uses stated above.

Beyond this the class itself does not have any specific functionality until they are called.

Chain: https://github.com/pallets/flask/blob/main/src/flask/app.py -> https://github.com/pallets/flask/blob/ea93a52d7d94ba093bbce4680c622cc4fc9771d8/src/flask/scaffold.py#L751

[@app.route()]

Purpose

The route function is called from the Flask class as a decorator and is used to route users to specific directories. Then the decorated function will be called.

This function is called several times in server.py for each directory that is created. When a user goes to a directory that does not exist a route function is linked to all non existing directories. Here a not found message will be displayed.

Magic ★★ ° ˙ ˚ ☽ ˚ ⌒ ˚ ★ ≡ ★ ≡

The .route() function is called from a Flask class instance and is always in the form of a decorator. The code can be found at https://github.com/pallets/flask/blob/main/src/flask/app.py from lines 1037 to 1094. This function takes the input string, which indicates the path, and optional POST/GET methods - if not stated it will default to get. In line 1047 endpoint is called which was a function call that is passed as a parameter that can be found at https://github.com/pallets/flask/blob/ea93a52d7d94ba093bbce4680c622cc4fc9771d8/src/flask/wrappers.py#L62 from line 62 to 74. In lines that contain the function getattr() - which determines if the attribute is contained - of https://github.com/pallets/flask/blob/main/src/flask/app.py, are parsing the data. In line 1054 the code is parsing the GET method and the elements contained in the GET method are uppercased. In line 1063 the code is finding the required_methods in view_func, and in line 1068 it is finding the data in provide_automatic_options. Then in line 1082 the function url_rule_class is used to build the HttpRequest and once the building is complete then the program adds this response to the url map. Then the endpoint is found. This file imports https://github.com/pallets/werkzeug/blob/main/src/werkzeug/serving.py which uses the socket library and returns the HttpRequest(more details on the werkzeug import in the app.run() function Magic). The endpoint function parses the returned HttpRequest which will then be used to check with the .route() parameter to determine if the appropriate function should be run.

Chain: https://github.com/pallets/flask/blob/main/src/flask/app.py -> https://github.com/pallets/flask/blob/ea93a52d7d94ba093bbce4680c622cc4fc9771d8/src/flask/wrappers.py#L62 -> https://github.com/pallets/flask/blob/main/src/flask/app.py -> https://github.com/pallets/werkzeug/blob/main/src/werkzeug/serving.py

[flask.render_template()]

Purpose

The render_template function is called from the Flask class and is used to render an html template. This function looks for the html file in the templates folder. This is possible because the class was initiated with the directory of the called file.

This function is called every time it is necessary to render a pre-made html.

Magic ★★ ° ˙ ˚ ☽ ˚ ⌒ ˚ ★ ≡ ✧ ⌯

The function can be found at https://github.com/pallets/flask/blob/ea93a52d7d94ba093bbce4680c622cc4fc9771d8/src/flask/templating.py from lines 133 to 151.

The render_template function takes the name of an html file and looks for it in the templates folder. This can be done thanks to the pathing as explained in the Flask(__name__) class. This function calls the _render() function which is in line 124 of the same file. The input for render_template is passed into the get_or_select_template function which can be found at https://github.com/pallets/jinja/blob/7d72eb7fefb7dce065193967f31f805180508448/src/jinja2/environment.py on lines 1055 to 1072. This function passes the input into the select_template function on line 1001 on the same file. Going back to the _render function; this function takes the parameters and passes them through the render function that is then returned. THe render function can be found in the previously mentioned jinja library at https://github.com/pallets/jinja/blob/7d72eb7fefb7dce065193967f31f805180508448/src/jinja2/environment.py#L941 on line 1257. This function takes in input and returns a rendered template as a string. The function does this by calling the concat function from https://github.com/pallets/jinja/blob/7d72eb7fefb7dce065193967f31f805180508448/src/jinja2/utils.py on line 26. The input for this concat is root_render_func which is on line 1144. This function is a function that is called from the built-in typing library. With the resulting output the _render function calls the send() function at https://github.com/pallets/flask/blob/ea93a52d7d94ba093bbce4680c622cc4fc9771d8/src/flask/signals.py#L25 from lines 25 to 37. This function takes the rendered html and sends it to the server with the proper formatting.

Chain:
https://github.com/pallets/flask/blob/ea93a52d7d94ba093bbce4680c622cc4fc9771d8/src/flask/templating.py ->
https://github.com/pallets/flask/blob/ea93a52d7d94ba093bbce4680c622cc4fc9771d8/src/flask/templating.py ->
https://github.com/pallets/jinja/blob/7d72eb7fefb7dce065193967f31f805180508448/src/jinja2/environment.py->
https://github.com/pallets/jinja/blob/7d72eb7fefb7dce065193967f31f805180508448/src/jinja2/environment.py#L941->
https://github.com/pallets/jinja/blob/7d72eb7fefb7dce065193967f31f805180508448/src/jinja2/utils.py ->
https://github.com/pallets/flask/blob/ea93a52d7d94ba093bbce4680c622cc4fc9771d8/src/flask/signals.py#L25

[flask.session()]

Purpose

flask.session is an extension for Flask that supports server-side sessions to your application. The session is the time between the clients logging in to the server and logging out of the server. The data that is required to be saved in the session is stored in a temporary directory on the server. Some uses of flask.session are: remember user when they log in, store user specific website settings (theme), store E-Commerce site user items in cart

flask.session is used everywhere throughout the code. The purpose of flask.session is to access the saved data. So whenever the data is needed in the code is where flask.session is used.

Magic

We worked with Flask to route the directories. flask.session is used when accessing data is required. flask.session has many different builtin configuration values. Some of which are: SESSION_COOKIE_NAME, SESSION_TYPE, SESSION_PERMANENT, and many more... After the TCP socket is created and after the user is directed to the homepage the code uses flask.session to access the data stored whenever clients log in and out of the server. Session allows programmers to store information specific to a user from one request to the next. This is implemented on top of cookies and signs the cookies cryptographically. In other words, the user could look at the contents of the programmers cookies but they are unable to modify the code, unless the user knows the secret key - which are in bytes - used for signing. The cookies are used to save the data for the user to return to. Accessing these key and value pairs work like a dictionary. You would use the same functionalities as a dictionary to access the key and values.

Citation: https://flask-session.readthedocs.io/en/latest/

The code to open the session is shown here https://github.com/pallets/flask/blob/main/src/flask/sessions.py in lines 362 - 376. In line 365 the code finds a serializer, which is defined between lines 347 - 360 https://github.com/pallets/flask/blob/9486b6cf57bd6a8a261f67091aca8ca78eeec1e3/src/flask/sessions.py#L347 . Here the code checks the correctness of the secret key, if false return None and if true assign a dictionary with the keys as the key derivation, and the value as the digested method. And the return of this function returns the secret key, the salted key, the serializer, and the dictionary. Then the code checks if the returned statement is None or not. If false return None, if true find the cookie name as defined in line 178 https://github.com/pallets/flask/blob/9486b6cf57bd6a8a261f67091aca8ca78eeec1e3/src/flask/sessions.py#L178 . Then does another check to see if the cookie name is true or not. If true then continue the program, if false then return the session https://github.com/pallets/flask/blob/9486b6cf57bd6a8a261f67091aca8ca78eeec1e3/src/flask/

sessions.py#L178 defined on line 345. Then the time is considered here. If the time is equal to the max time then return the session with the data. Every Time at the end of each request the code saves the session. When saving the cookie if the session is not the session and if modified then the cookie is deleted. If the session is accessed then the code adds “Cookie” to the response. Once all of these checks have passed then the response is set a new cookie with the cookie name, a new dictionary, expire time, a new cookie, the cookie domain, a path, a boolean which determines if the cookie is secure, and the value of the data with “SESSION_COOKIE_SAMESITE”.

Chain: https://github.com/pallets/flask/blob/main/src/flask/sessions.py -> https://github.com/pallets/flask/blob/9486b6cf57bd6a8a261f67091aca8ca78eeec1e3/src/flask/sessions.py#L178

Link: https://github.com/jaykwonproject/CSE312/blob/registration_fuctionality/server.py
flask.session is used on lines 35 and 36. In both of these lines flask.session is used to assign a user who has just logged in to the boolean True and to assign a new key and value pair, which are ‘user’->username, into the data

[flask.session.get()]

Purpose

flask.session.get() retrieves data the programmer is requesting for in the data stored in flask.session. The function flask.session.get() iterates through the data to find the requested data. If the requested data isn’t found in the data then flask.session.get() returns NULL.

flask.session.get() is used on lines 11, 14, 23, 26, 50, and 53.

Magic

As explained before, flask.session is where data is stored by signing cookies cryptographically. flask.session.get() is used to retrieve data that is requested by the programmer https://github.com/fengsp/flask-session/blob/9f591b5c7ecf25c44678c7d18d94a0bf91132e2d/examples/hello.py#L30. In this implementation the function get() iterates through the data to find the key in the parameter. Once the key is found then the function returns the object from the data. If the key isn't found then the function will return NULL.

Citation: https://flask-session.readthedocs.io/en/latest/ -> https://github.com/fengsp/flask-session/blob/9f591b5c7ecf25c44678c7d18d94a0bf91132e2d/examples/hello.py#L30

Link: https://github.com/jaykwonproject/CSE312/blob/registration_fuctionality/server.py flask.session.get() is used on lines 8, 11, and 14. In all of these lines flask.session.get() are used to retrieve data from flask.session. In this specific context the code is determining whether the user is logged in or not. If the user is not logged in then the code will require the user to sign in or make an account. If the user is logged in then the homepage will stay as it is.

[flask.redirect()]

Purpose

flask.redirect() returns a response object and redirects the user to another target location with a specified status code.

flask.redirect() is used when the user logs out or if the user is creating a new account. Once the user enters a username the code will redirect to another route to check if the username is available, if the user is already logged in, or the code will redirect the user to the homepage after the user logs out.

Magic

The redirect function is called from the werkzeug library at https://github.com/pallets/werkzeug/blob/347fdbb055c86efe1fd49546bd524cde4b98c103/src/werkzeug/utils.py on lines 221. This function takes a url as a parameter and creates a response object(line 255) from the class that is defined at https://github.com/pallets/werkzeug/blob/347fdbb055c86efe1fd49546bd524cde4b98c103/src/werkzeug/wrappers/response.py#L66 on line 66.

Chain:
https://github.com/pallets/werkzeug/blob/347fdbb055c86efe1fd49546bd524cde4b98c103/src/werkzeug/utils.py ->
https://github.com/pallets/werkzeug/blob/347fdbb055c86efe1fd49546bd524cde4b98c103/src/werkzeug/wrappers/response.py#L66

[flask.url_for()]

Purpose

flask.url_for() is accompanied with flask.redirect(). flask.url_for() determines where the route should be sent next. The function creates a url preventing the overhead of changing URLs through the application.

flask.url_for() is used wherever flask.redirect() is used. flask.redirect() needs to use flask.url_for() to send the user to the appropriate webpage.

Magic

The function url_for() can be found at https://github.com/pallets/flask/blob/7620cb70dbcbf71bca651e6f2eef3cbb05999272/src/flask/helpers.py#L1 from lines 192 to 340. This function takes a string as an input and passes it on as a variable on the build function that is called on the url_adapter instance on line 323. Url_adapter is a _request_ctx_stack instance which can be found at https://github.com/pallets/flask/blob/7620cb70dbcbf71bca651e6f2eef3cbb05999272/src/flask/globals.py#L52 on line 52. This is a value set to the LocalStack class that is imported from the werkzeug library at https://github.com/pallets/werkzeug/blob/347fdbb055c86efe1fd49546bd524cde4b98c103/src/werkzeug/local.py on lines 79 to 149. This class is similar to a stack data structure. In essence the top function is referred to when using this data structure which gets the top most value. This value is then used to build a url with the input endpoint to create a readable url.

Citation: https://flask.palletsprojects.com/en/2.0.x/api/

Chain:
https://github.com/pallets/flask/blob/7620cb70dbcbf71bca651e6f2eef3cbb05999272/src/flask/helpers.py#L1 ->
https://github.com/pallets/flask/blob/7620cb70dbcbf71bca651e6f2eef3cbb05999272/src/flask/globals.py#L52 ->
https://github.com/pallets/werkzeug/blob/347fdbb055c86efe1fd49546bd524cde4b98c103/src/werkzeug/local.py

[flask.request.form()]

Purpose

This function is used to get the data from the html form that contains user input. For example in the login feature. The user will input their username and password and the function will get the corresponding data.

In our code this is called depending on the functionality but is almost always used under functions that need to accept user input. Primary examples are under the check credentials function where the username and password are requested when they are imputed into the form. The username and password are also requested under the check availability function.

Magic ★★ ° ˙ ˚ ☽ ˚ ⌒ ˚ ★ ≡ ★ ≡

The request function can be found from https://github.com/pallets/flask/blob/main/src/flask/wrappers.py where it calls the form function from https://github.com/pallets/werkzeug/blob/main/src/werkzeug/wrappers/request.py at lines 413 to 429. In the form function the load_form_data() function is called which is in the same file lines 251 to 277. This function parses the form data from lines 264 to 271. In line 265 the function calls the make_form_data_parser function which is in line 236. This function creates an instance of the form_data_parser_class class which can be found in https://github.com/pallets/werkzeug/blob/347fdbb055c86efe1fd49546bd524cde4b98c103/src/werkzeug/formparser.py#L259 line 156. From this class instance the parse function is called which can be found on line 230. This function calls get_parse_func which is on line 209. This function returns the result of the get_parse_func which can be found on 314. This function takes the input and adds the form data to a dictionary with appropriate key values. This function calls 3 different functions _parse_multipart, _parse_urlencoded, and _parse_urlencoded. These functions are passed as the appropriate values. All of these functions do similar things as they individually parse the imputed form data. As an example the _parse_multipart function which can be found in line 271 creates a MultiPartParser object which has a parse function on line 397 which takes the form data in bytes and decodes them into ascii. This parsed data is then used by load_form_data() which is then used by request.form().

This chain only tracks new directories or libraries. A lot of the chaining happens within the last link which makes the chain look shorter than it is.
Chain: https://github.com/pallets/flask/blob/main/src/flask/wrappers.py -> https://github.com/pallets/werkzeug/blob/main/src/werkzeug/wrappers/request.py -> https://github.com/pallets/werkzeug/blob/347fdbb055c86efe1fd49546bd524cde4b98c103/src/werkzeug/formparser.py#L259

[app.run()]

Purpose

App is an instance of the Flask class that was initiated at the start of the program. This function allows the application to be run in a local environment. This function deals with creating the TCP socket connection.

The function is called under the __name__ == __main__ statement at the end of the program. This allows the application to be run and, therefore, allows all the other functionalities to be accessed.

Magic

The code for the function can be found at https://github.com/pallets/flask/blob/main/src/flask/app.py from lines 805 to 924.

For the purpose of running the application lines 917 to 925 are used. In line 917 run_simple is imported from werkzeug.serving and it is run in line 920; passing the host and port as parameters. Werkzeug is a WSGI web application library that allows web servers to forward requests to python web applications. The code for run_simple can be found at https://github.com/pallets/werkzeug/blob/main/src/werkzeug/serving.py from lines 788 to 970. The function calls the make_server function that can be found on line 742 to 776. This function calls 3 different possible classes depending on the input but they all result in calling the BaseWSGIServer class. This class calls fromfd on line 655 from the socket library. This can be found at https://github.com/python/cpython/blob/main/Lib/socket.py from lines 539 to 546. This function creates a socket object that will be used to create a connection. This can be seen from line 942 where it calls the server_forever on the server that was made from lines 929 to 939.

Chain: https://github.com/pallets/flask/blob/main/src/flask/app.py -> https://github.com/pallets/werkzeug/blob/main/src/werkzeug/serving.py -> https://github.com/python/cpython/blob/main/Lib/socket.py

[resp.set_cookie()]

Purpose

Resp is an instance of flask.redirect that takes flask.url_for('home') as a parameter(both these functions were explained above). Set_cookie takes 2 parameters which are used to set cookies.

This function is used several times in server.py when it is necessary to set cookies.

Magic

The instance of the function can be found at https://github.com/pallets/flask/blob/6b0c8cdac1fb9bbec88de3a514907c19e372c72a/src/flask/sessions.py on line 407. Here the function is called from the werkzeug library. This can be found at https://github.com/pallets/werkzeug/blob/d72a713efde2587e9d69fc52e9a5e299ced7ea3b/src/werkzeug/sansio/response.py from lines 196 to 247. This function takes a key and a value(with other optional parameters but those were not used in our implementation) calles the add function on a header object. The add function can be found at https://github.com/pallets/werkzeug/blob/d72a713efde2587e9d69fc52e9a5e299ced7ea3b/src/werkzeug/datastructures.py under the header class from lines 1106 to 1127. This function takes the dump_cookie function which has the set_cookie parameters passed in it. The add function adds parameters to the referenced header object. The set_cookie function can be found at https://github.com/pallets/werkzeug/blob/d72a713efde2587e9d69fc52e9a5e299ced7ea3b/src/werkzeug/http.py from line 1235. The dump_cookie function takes the key and value, converts them into bytes and formats them into the proper response; decodes them and returns it. This will be set and sended when it is called by the original response object in the flask library.

Chain:
https://github.com/pallets/flask/blob/6b0c8cdac1fb9bbec88de3a514907c19e372c72a/src/flask/sessions.py ->
https://github.com/pallets/werkzeug/blob/d72a713efde2587e9d69fc52e9a5e299ced7ea3b/src/werkzeug/sansio/response.py ->
https://github.com/pallets/werkzeug/blob/d72a713efde2587e9d69fc52e9a5e299ced7ea3b/src/werkzeug/datastructures.py ->
https://github.com/pallets/werkzeug/blob/d72a713efde2587e9d69fc52e9a5e299ced7ea3b/src/werkzeug/http.py

[app.secret_key()]

Purpose

app.secret_key is used to securely sign a session cookie, or for any other security reasons. app.secret_key should be a long random bytes or string. A secret key is needed to start the session. The purpose of app.secret_key is for protecting the chat app against attackers, so the attackers don't tamper and alter the cookie data, so it is very crucial for the programmer to never reveal the secret key.

Magic ★★ ° ˙ ˚ ☽ ˚ ⌒ ˚ ★ ˚ ★

When creating a secret key the programmer should use a random generated number that is either in string or bytes so attackers don't alter the cookie's data. To create a secret key the programmer would have to import random to create a random sequence of numbers. Then those numbers would have to be turned into bytes. Once that is complete then the random sequence of bytes would have to be hashed. Only the programmer knows this combination to protect his/her cookies. Once the cookie is received by the browser the content is decoded by using sha1 and hexdigest with the original cookie. The result of this is compared and if the results match each other then the cookie has not been tampered with. However, the code from https://github.com/MireIaI/flask_heroku_example/blob/7adf2a11fbd2ffbfeba64a4969a01b586ccdb4d5/app.py doesn't create a random long combonation of strings or bytes but it should be done for full safety. In the code in line 7 the secret key is created. And the key is checked (line 26) and if it passes then the key is configured (lines 31 and 32) and then app.logger.info() is called to log the information of the configured key and the configured value (line 35). Finally render_template() is called to display the secret key, configured key, and the configured value.

Chain:
https://github.com/MireIaI/flask_heroku_example/blob/7adf2a11fbd2ffbfeba64a4969a01b586ccdb4d5/app.py

Line: app.secret_key() is used on line 158

[flask.flash]

Purpose

flask.flash gives feedback to a user. Flashing makes it possible to record a message at the end of a request and access it in the next request, but only in the next request - it cannot go any further. In our code we used flash.flask to talk to the user whenever the user is registering for an account. For example it determines if a username is taken already or if a password is typed incorrectly, and if the user makes an account it displays the message "Now you are our member!".

Magic

flask.flash is found being used on line 156 https://github.com/pallets/flask/blob/2f0c62f5e6e290843f03c1fa70817c7a3c7fd661/tests/test_signals.py to test if flash will send a message to the user. In line 156 flask.flash is sending “This is a flash message” to the user to determine if the user can receive the flash message. flask.flash is imported from helpers.py https://github.com/pallets/flask/blob/7620cb70dbcbf71bca651e6f2eef3cbb05999272/src/flask/helpers.py. flask.flash is defined from lines 365 - 394. The programmer needs to find “_flashes” in the session and append the parameters, which contain message and the category which are both strings. Then add what is stored into the session with the key value as “_flashes” and the value as what is stored. Once these are complete then the programmer has to send the message to the user.

Chain:
https://github.com/pallets/flask/blob/2f0c62f5e6e290843f03c1fa70817c7a3c7fd661/tests/test_signals.py ->
https://github.com/pallets/flask/blob/7620cb70dbcbf71bca651e6f2eef3cbb05999272/src/flask/helpers.py

Line: flask.flash are used on lines 136, 140, and 147