Flask

General Information & Licensing

Code Repository	https://github.com/pallets/flask/
License Type	BSD-3-Clause Source
License Description	● Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following restrictions/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?	Collin, Ray, Jieli, Kelly, Richard

Flask Object (app)

Purpose

- The app object is created on line 9 and sets up our templates as well as our MYSQL database for our web server.
- This tech allows us to easily route HTTP requests for our web server and respond appropriately, by referring to this object. It uses decorators to allow easier readability for our code.
- This technology is used in the entirety of our main.py and you can see examples of the decorators on lines 18, 25, and 35. There are more, but this highlights a few.

Magic ⋆★˚｡⋆୨୧˚ ☽ ˚⁀➷｡˚★ ⋆⭑ ༄

- The app is initialized in line 9 of the code. When the app is created it takes multiple parameters. Starting with a __name__, and then additional parameters that it can take. Those being a template_folder, or a static_folder. In our case we use both a template_folder and a static_folder. The template_folder is a folder that contains templates that are used by the application. These templates can be things such as HTML, CSS, or JavaScript pages that we need to send to the server. In addition, we configure our SQL database here as well. Giving our database a host value, user value password and name. This will be useful later on in our project.
- This code is found in lines 9 through 11 of our code. Without these lines our app object will not be created and our code will not run. Our object will now properly listen to the decorator calls and will respond to the data when necessary.
    - Initializer of Flask Object (Line 97): https://github.com/pallets/flask/blob/main/src/flask/app.py
    - Initializer of Scaffold Object (Line 62): https://github.com/pallets/flask/blob/7620cb70dbcbf71bca651e6f2eef3cbb05999272/src/flask/scaffold.py#L62
        - Allows for the initialization of things such as template and static folders.

flask.render_template()

Purpose

- Renders a template from the given folder with the given context.
- This appears in our event handler for the root path, /auth/register path, /auth/login and our error handler paths.

Magic ⋆★꙳˚⋆˚☽˚⁀➴｡˚★彡꙳★༄

- Using the Flask Web WSGI application, it takes an item found in the template folder we gave it during object initialization and formulates an HTTP response that the browser will be able to understand.
- https://github.com/pallets/flask/blob/2f0c62f5e6e290843f03c1fa70817c7a3c7fd661/src/flask/templating.py (Lines 133-151)
  - Returns the value found by the _render function found here on Lines 124-130. https://github.com/pallets/flask/blob/2f0c62f5e6e290843f03c1fa70817c7a3c7fd661/src/flask/templating.py
  - References a template object found in a different GitHub repo called jinja2. https://github.com/pallets/jinja/blob/main/src/jinja2/environment.py Found on line 1118, this template object allows custom templates to be rendered. In our case, this will allow the HTTP responses to be rendered for given files.

@app.route() / add_url_rule

Purpose

- The following allows us to use decorators to assign the add_url_rule() functio
- This appears wherever in the code you see @app.route(“path”)

Magic ⋆★˚｡⋆˚☽˚｡⋆★⋆

- Whenever the add_url_rule() is called either by using the decorator @app.route or add_url_rule(), it takes in the current flask object, as well as a rule. This rule must be a string which represents the path that the object has to respond to. It also can take in optional information as well that we sometimes use in our project.
- On lines 1036-1093, the following code provides this functionality.https://github.com/pallets/flask/blob/main/src/flask/app.py
  - Inside this function it also calls two helper functions as well. One of them is the _endpoint_from_view_function, which returns the default endpoint for a given function. In most cases, this will be the function name of the handler function inside of our decorator. The function can be found on line 751 here: https://github.com/pallets/flask/blob/7620cb70dbcbf71bca651e6f2eef3cbb05999272/src/flask/scaffold.py#L751
  - The other function is the getattr function which finds out what methods are being called during the add_url_rule. This can be anything from a “GET” or a “POST” request. If not specified it defaults to a “GET” request. The following function can be found here inside of the View class on line 13. https://github.com/pallets/flask/blob/ba6db2e30783f81673bb8a6b5218f7bb8efa8f27/src/flask/views.py
  - The url_map_class referenced by the code keeps track of all the routes currently defined by the Flask object and will call them appropriately. This is defined as a map on line 361.

@app.errorhandler() / handle_exception() / handle_user_exception()

Purpose

- This decorator function allows us to call the handle_exception function in our Flask import. This will allow us to handle HTTP requests errors such as 404 by sending a template in which we have defined back to the appropriate TCP connection.
- This appears in our code whenever we have @app.errorhandler(). Specifically on lines 55 and 60.

Magic ⋆★˚｡⋆୨୧˚ ☽ ˚⁀➴｡˚★ ⋆彡⋆ ༄

- The technology takes our error handler for the specified code (404 and 500) and will send back an HTTP response with the appropriate information in it based on what we provided in our function below the decorator.
- The code for this part is located on lines 1362-1392 in the function called handle_user_exception. https://github.com/pallets/flask/blob/ea93a52d7d94ba093bbce4680c622cc4fc9771d8/src/flask/app.py
    - In addition, if the handle_user_exception is not defined, the following function will be called instead. This function is located on lines 1394-1436 and will by default respond with an error code of 500 (Internal Server Error). https://github.com/pallets/flask/blob/ea93a52d7d94ba093bbce4680c622cc4fc9771d8/src/flask/app.py
    - If it is an HTTP exception, like an error 404, then the handle_http_exception function will be called by our handle_user_exception function. The handle_http_exception function will be located on lines 1292-1325 and can be found here: https://github.com/pallets/flask/blob/ea93a52d7d94ba093bbce4680c622cc4fc9771d8/src/flask/app.py