Open-Source Technology Use Report

Proof of knowing your stuff in CSE312

[Flask]

General Information & Licensing

Code Repository	https://github.com/pallets/flask
License Type	BSD 3-Clause "New" or "Revised" License
License Description	● A permissive license similar to the BSD 2-Clause License, but with a 3rd clause that prohibits others from using the name of the project or its contributors to promote derived products without written consent. ● Permissions: ○ Commercial use ○ Modification ○ Distribution ○ Private use
License Restrictions	● Liability ● Warranty
Who worked with this?	Eric, Jason, Josh, Devin

Use as many of the sections below as needed, or create more, to explain every function, method, class, or object type you used from this library/framework.

[class Flask]

Purpose

- What does this tech do for you in your project?
    - This tech is the backbone of our project, it handles routing, receiving client data, responding to clients and allows us to implement the backend.
- Where specifically is this tech used in your project? Give us some details like file location and line number, if applicable. If too cumbersome, a general description of where it's used for a given purpose is fine as well.
    - This tech is used in the __init__.py file that is ran when main.py is ran
    - Flask's functionality will be used in almost every python file that makes up our project

Magic

Flask is the main class involved with the creation of the web application. It is defined at
https://github.com/pallets/flask/blob/9486b6cf57bd6a8a261f67091aca8ca78eeec1e3/src/flask/app.py#L97

In our project it is used in the __init__.py file at the root of the repository where the application is initialized on line 16
https://github.com/ericv105/web-app-project/blob/824ab01704bb15113f36e14ba94327b66f3cdf60/website/__init__.py#L16

It is a simple line of code that assigns the Flask object to the variable app with the parameter __name__. According to the source code:

> It is passed the name of the module or package of the application. Once it is created it will act as a central registry for the view functions, the URL rules, template configuration and much more. The name of the package is used to resolve resources from inside the package or the folder the module is contained in depending on if the package parameter resolves to an actual python package (a folder with an :file:`__init__.py` file inside) or a standard module (just a ``.py`` file).

When main.py is run, it imports the create_app() function from __init__.py and assigns its output to a variable called app. From here, app.run() can be called to start the Flask app; however, since the server that Flask is based on, Werkzeug's WSGI, doesn't have built in support for WebSockets, a separate Flask extension, Flask-SocketIO, is required in order to meet the websocket requirement. So instead of using app.run(), we import the SocketIO object from Flask-SocketIO, and assign it to the 'socketio' variable with the 'app' Flask object as a parameter, and THEN socketio.run() is used to start the Flask application and the eventlet server, a requirement for Flask-SocketIO to function properly
https://github.com/ericv105/web-app-project/blob/824ab01704bb15113f36e14ba94327b66f3cdf60/main.py#L57
The run() method is defined at:
https://github.com/miguelgrinberg/Flask-SocketIO/blob/a10ea5cf65007061d7b3fd87b530c382007adebb/src/flask_socketio/__init__.py#L516

This run() method then checks if the eventlet library is installed and set up for Flask-SocketIO and then runs the server with the 'run_server()' method defined at:
https://github.com/miguelgrinberg/Flask-SocketIO/blob/a10ea5cf65007061d7b3fd87b530c382007adebb/src/flask_socketio/__init__.py#L599
In this method, a socket object listening on the host address (0.0.0.0:5000) is assigned to the variable 'eventlet_socket':
https://github.com/miguelgrinberg/Flask-SocketIO/blob/a10ea5cf65007061d7b3fd87b530c382007adebb/src/flask_socketio/__init__.py#L607
This socket contains the eventlet.websocket.WebSocketWSGI module which is defined at
https://github.com/eventlet/eventlet/blob/c6c350eaa9eb819c6bcabe25113464aed75b9cf5/eventlet/websocket.py#L65
This is exactly where the connection get upgraded to websocket connection using headers in the response shown here:
https://github.com/eventlet/eventlet/blob/c6c350eaa9eb819c6bcabe25113464aed75b9cf5/eventlet/websocket.py#L313

This eventlet socket gets passed as a parameter to the eventlet.wsgi.server() method which is what is responsible for actually starting the server:
https://github.com/miguelgrinberg/Flask-SocketIO/blob/a10ea5cf65007061d7b3fd87b530c382007adebb/src/flask_socketio/__init__.py#L624
within that method, the server is created at

https://github.com/eventlet/eventlet/blob/955be1c7227a6df0daa537ebb8aed0cfa174d2e5/eventlet/wsgi.py#L956
with an argument named ‘protocol’ which is of type HttpProtocol; this extends python’s BaseHTTPServer.BaseHTTPRequestHandler which is defined at
https://github.com/python/cpython/blob/8d21aa21f2cbc6d50aab3f420bb23be1d081dac4/Lib/BaseHTTPServer.py#L114
And here is exactly where requests are parsed, handled, and responses are sent back

The Server object is saved to the variable serv. This Server object is defined at https://github.com/eventlet/eventlet/blob/955be1c7227a6df0daa537ebb8aed0cfa174d2e5/eventlet/wsgi.py#L763 which extends the python BaseHTTPServer.HTTPServer class defined at
https://github.com/python/cpython/blob/8d21aa21f2cbc6d50aab3f420bb23be1d081dac4/Lib/BaseHTTPServer.py#L102
It extends socketserver’s TCPServer object, which is exactly where the TCP connections are made.

P.S.
While writing this I realized I have strayed away from the Flask class and explained how the TCP connections are made. However, reaching that explanation partly required the Flask class to be introduced as well as Flask-SocketIO. The next part of this report will cover the variables imported from the Flask class and how they work

[Blueprint]

Purpose

- A blueprint is an object from Flask that is used to hold a collection of routes for the application. It helps with organization and cohesion for certain functions and routes.
- It is used in auth.py, message.py, uploads.py, and views.py and is usually one of the first executed lines of code in each of those files.

Magic ⋆★˚｡⋆୨୧˚ ☽ ˚⁀➴｡˚★ ≋⋆ ༄

In all of the files mentioned before, a Blueprint is used. Let's just use auth.py as an example. In auth.py, a Blueprint object with the parameters 'auth' and __name__ is stored in the variable 'auth'
https://github.com/ericv105/web-app-project/blob/824ab01704bb15113f36e14ba94327b66f3cdf60/website/auth.py#L14

The Blueprint class is defined at
https://github.com/pallets/flask/blob/9486b6cf57bd6a8a261f67091aca8ca78eeec1e3/src/flask/blueprints.py#L112

What this does exactly is allow us to define certain functions and routes using the 'auth' variable without needing to use the Flask app to route. Since these functions and routes are already there prior to the start of the app, it is just imported and registered to the Flask app. This can be seen here:
https://github.com/ericv105/web-app-project/blob/824ab01704bb15113f36e14ba94327b66f3cdf60/website/__init__.py#L26-L34

the register_blueprint method is defined at:
https://github.com/pallets/flask/blob/9486b6cf57bd6a8a261f67091aca8ca78eeec1e3/src/flask/app.py#L1002
which then calls the register() method that is defined at
https://github.com/pallets/flask/blob/9486b6cf57bd6a8a261f67091aca8ca78eeec1e3/src/flask/blueprints.py#L271

Within this method, the Blueprint object is saved to one of Flask app's dictionaries called blueprints with the blueprint's name as the key and the object as the value.
https://github.com/pallets/flask/blob/9486b6cf57bd6a8a261f67091aca8ca78eeec1e3/src/flask/blueprints.py#L324

Since the Blueprint object inherits from the Scaffold object defined at
https://github.com/pallets/flask/blob/9486b6cf57bd6a8a261f67091aca8ca78eeec1e3/src/flask/scaffold.py#L62
We can call the method route() on the Blueprint and use it to decorate a function that gets called when the specified URL rule is requested. In this route() method, another method 'add_url_rule()' is called with the URL rule that was passed to the route() method.
https://github.com/pallets/flask/blob/9486b6cf57bd6a8a261f67091aca8ca78eeec1e3/src/fl

ask/scaffold.py#L439

Finally the function that is being decorated is run, usually returning a render_template() or redirect(), both of which i will cover next

[flask.render_template()]

Purpose

Replace this text with some that answers the following questions for the above tech:

- What does this tech do for you in your project?
- This method responds to the client with an html file to be displayed in the browser.
- Where specifically is this tech used in your project? Give us some details like file location and line number, if applicable. If too cumbersome, a general description of where it's used for a given purpose is fine as well.
- It is used in all application routes where there is not a redirect or a send_from_file being returned

Magic ⋆★｡˚⋆ ☽ ˚｡⋆ ★彡⋆

Continuing from the auth.py example being used previously, if a user were to navigate to localhost:5000/register, they will see the register.html webpage. This happens because the method 'render_template('register.html', user=current_user)' is returned at the end of the function register(). I will explain the 'current_user' object in the report for Flask-Login
https://github.com/ericv105/web-app-project/blob/824ab01704bb15113f36e14ba94327b66f3cdf60/website/auth.py#L76

render_template() is defined at
https://github.com/pallets/flask/blob/9486b6cf57bd6a8a261f67091aca8ca78eeec1e3/src/flask/templating.py#L133
and it creates a Template object from the name of the html file with the method 'get_or_select_template()' using jinja templating engine library.
https://github.com/pallets/jinja/blob/7d72eb7fefb7dce065193967f31f805180508448/src/jinja2/environment.py#L1055

This template object and the user variable is passed to the _render() method that is defined right above
https://github.com/pallets/flask/blob/9486b6cf57bd6a8a261f67091aca8ca78eeec1e3/src/flask/templating.py#L124

Within this method, the template is rendered again with the context, which is the 'user' variable which we passed into the original method.
https://github.com/pallets/flask/blob/9486b6cf57bd6a8a261f67091aca8ca78eeec1e3/src/flask/templating.py#L128
After this method, any and all Jinja syntax is executed and removed from the html and is returned as a string.

Now a response needs to be formed and sent to the client using the finalize_request() method defined at. The make_response(rv) method builds the response with the necessary headers to return back to the client and sends the response via the eventlet's TCP connection that was created and explained previously
https://github.com/pallets/flask/blob/9486b6cf57bd6a8a261f67091aca8ca78eeec1e3/src/flask/app.py#L1520

[flask.redirect()]

Purpose

Replace this text with some that answers the following questions for the above tech:

- What does this tech do for you in your project?
- This method redirects a user to a different webpage
- Where specifically is this tech used in your project? Give us some details like file location and line number, if applicable. If too cumbersome, a general description of where it's used for a given purpose is fine as well.
- This is used in auth.py to redirect a user to login after registering successfully, it is also used to redirect a user to the homepage after logging in successfully. It is used in other parts of the app where redirects are necessary like in uploads.py

Magic ⋆★˚｡⋆☽˚｡⋆★彡⋆

Say that a user registers successfully from the registration page, that means they will get redirected to the login page based on this line from auth.py
https://github.com/ericv105/web-app-project/blob/824ab01704bb15113f36e14ba94327b66f3cdf60/website/auth.py#L75

The redirect() method is import from flask but it is actually part of the Werkzeug library and it is defined at
https://github.com/pallets/werkzeug/blob/347fdbb055c86efe1fd49546bd524cde4b98c103/src/werkzeug/utils.py#L221
Within this method, a response is created using html and returned to a method called full_dispatch_request() which is defined at
https://github.com/pallets/flask/blob/9486b6cf57bd6a8a261f67091aca8ca78eeec1e3/src/flask/app.py#L1504 which then calls the dispatch_request() method defined right above. Once that returns, the finalize_request() is called in the same way it was called for the render_template() method. The response is returned to full_dispatch_request which then returns to Flask's wsgi_app defined at
https://github.com/pallets/flask/blob/9486b6cf57bd6a8a261f67091aca8ca78eeec1e3/src/flask/app.py#L2035
The wsgi_app method returns a response object that takes a 'start_response' method as a parameter.
https://github.com/pallets/flask/blob/9486b6cf57bd6a8a261f67091aca8ca78eeec1e3/src/flask/app.py#L2072
Response object is defined at
https://github.com/pallets/werkzeug/blob/347fdbb055c86efe1fd49546bd524cde4b98c103/src/werkzeug/wrappers/response.py#L66
Since the response object is called as a method, it executes the __call__ method defined at
https://github.com/pallets/werkzeug/blob/347fdbb055c86efe1fd49546bd524cde4b98c103/src/werkzeug/wrappers/response.py#L614
which then calls self.get_wsgi_response() and then calls start_response() which is defined at
https://github.com/eventlet/eventlet/blob/955be1c7227a6df0daa537ebb8aed0cfa174d2e5/eventlet/wsgi.py#L536
which returns back to the write() method defined at
https://github.com/eventlet/eventlet/blob/955be1c7227a6df0daa537ebb8aed0cfa174d2e5/

eventlet/wsgi.py#L483
And since the towrite object has something in it, write() gets called with the contents of towrite in bytes format. In this write() method, the response headers are appended to the towrite object along with the ‘/login’ that we were originally trying to redirect the user to.

the handle method defined at https://github.com/eventlet/eventlet/blob/955be1c7227a6df0daa537ebb8aed0cfa174d2e5/eventlet/wsgi.py#L379 runs an infinite while loop while calling the handle_one_request() method each time which is responsible for taking the user’s request.

Hopefully this explanation covers how requests are handled using the eventlet webserver library.

[flask.flash()]

Purpose

Replace this text with some that answers the following questions for the above tech:

- What does this tech do for you in your project?
- This method is used to give an alert to users based on their use of the application
- Where specifically is this tech used in your project? Give us some details like file location and line number, if applicable. If too cumbersome, a general description of where it's used for a given purpose is fine as well.
- This is used in auth.py to send a message to the user about their registration information or login credentials. It is used in uploads.py line 54, and views.py lines 40 and 43

Magic ⋆★˚｡⋆˚☽˚⁀➷｡˚★彡⋆༄

When the flash() method is used, it takes a message in the form of a string and optionally a category. On lines 61 thru 67, we use flash methods within conditionals to warn the user about their registration credentials not being sufficient enough. flash() is defined at https://github.com/pallets/flask/blob/9486b6cf57bd6a8a261f67091aca8ca78eeec1e3/src/flask/helpers.py#L365

And it uses the global Flask session object/dictionary to store arguments passed into the flash method. From there it stays until get_flashed_messages() method is called. It is defined at https://github.com/pallets/flask/blob/9486b6cf57bd6a8a261f67091aca8ca78eeec1e3/src/flask/helpers.py#L397

This method is used in our base.html template using Jinjia's {% %} delimiters to run python code. https://github.com/ericv105/web-app-project/blob/824ab01704bb15113f36e14ba94327b66f3cdf60/website/templates/base.html#L51-L62

This displays any flashed messages to the end user

[flask.current_app]

Purpose

Replace this text with some that answers the following questions for the above tech:

- What does this tech do for you in your project?
- This variable is used to access the running instance of the Flask app
- Where specifically is this tech used in your project? Give us some details like file location and line number, if applicable. If too cumbersome, a general description of where it's used for a given purpose is fine as well.
- This is used in parts of the app where saving files is necessary such as in views.py for the /profile view and in uploads.py

Magic ⋆★˚｡⋆୨୧˚ ☽ ˚⁀➴｡˚★彡⋆༄

current_app is a global Flask variable that is a proxy to a method named _find_app().
current_app is defined at
https://github.com/pallets/flask/blob/9486b6cf57bd6a8a261f67091aca8ca78eeec1e3/src/flask/globals.py#L54
_find_app() is defined at
https://github.com/pallets/flask/blob/9486b6cf57bd6a8a261f67091aca8ca78eeec1e3/src/flask/globals.py#L44

In this method, _app_ctx_stack.top is called and gets assigned to the variable top.
_app_ctx_stack is a global Flask LocalStack object defined at
https://github.com/pallets/werkzeug/blob/347fdbb055c86efe1fd49546bd524cde4b98c103/src/werkzeug/local.py#L79

the .top() method that gets called is defined at
https://github.com/pallets/werkzeug/blob/347fdbb055c86efe1fd49546bd524cde4b98c103/src/werkzeug/local.py#L142
and returns the last item since LocalStack acts like a stack

According to flask's official website: https://flask.palletsprojects.com/en/2.0.x/appcontext/
Flask automatically *pushes* an application context when handling a request. View functions, error handlers, and other functions that run during a request will have access to **`current_app`**.

So whenever a request is handled, Flask pushes an AppContext object to the top of the stack. AppContext is defined at:
https://github.com/pallets/flask/blob/9486b6cf57bd6a8a261f67091aca8ca78eeec1e3/src/flask/ctx.py#L219

Everytime current_app is used, it calls the _find_app() method which assigns the variabel 'top' the AppContext. From there top.app() is returned and we now have access to Flask's configuration dictionary for use in accessing file storage paths.

[flask.request]

Purpose

Replace this text with some that answers the following questions for the above tech:
- What does this tech do for you in your project?
- This variable is used to access information from the client
- Where specifically is this tech used in your project? Give us some details like file location and line number, if applicable. If too cumbersome, a general description of where it's used for a given purpose is fine as well.
- This is used in parts of the app where user input information needs to be handled such as in auth.py for handling form data, or uploads.py when handling uploaded files, or in views.py when handling uploaded images for avatars.

Magic ⋆★⁺₊˚.⋆˚☽˚⁀➷｡˚★彡⁺★ ༄

Similar to current_app, request is a global Flask variable defined at https://github.com/pallets/flask/blob/9486b6cf57bd6a8a261f67091aca8ca78eeec1e3/src/flask/globals.py#L55

wsgi_app() is called to handle each request and it is defined at https://github.com/pallets/flask/blob/9486b6cf57bd6a8a261f67091aca8ca78eeec1e3/src/flask/app.py#L2035

A request context is created and assigned to the variable 'ctx' at https://github.com/pallets/flask/blob/9486b6cf57bd6a8a261f67091aca8ca78eeec1e3/src/flask/app.py#L2060

The request_context() method is defined at https://github.com/pallets/flask/blob/9486b6cf57bd6a8a261f67091aca8ca78eeec1e3/src/flask/app.py#L1963

That method returns a RequestContext object which is defined at https://github.com/pallets/flask/blob/9486b6cf57bd6a8a261f67091aca8ca78eeec1e3/src/flask/ctx.py#L266

now 'ctx' is a RequestContext object and runs the .push() method defined at https://github.com/pallets/flask/blob/9486b6cf57bd6a8a261f67091aca8ca78eeec1e3/src/flask/ctx.py#L390

This pushes the RequestContext to the top of the _request_ctx_stack global variable https://github.com/pallets/flask/blob/9486b6cf57bd6a8a261f67091aca8ca78eeec1e3/src/flask/ctx.py#L414

Now that it is at the top of the _request_ctx_stack, it can be accessed whenever we use the request variable since it calls _lookup_req_object method defined at https://github.com/pallets/flask/blob/9486b6cf57bd6a8a261f67091aca8ca78eeec1e3/src/flask/globals.py#L30 which returns the RequestContext along with any of the information sent by the client.

[werkzeug.secure_filename]

Purpose

Replace this text with some that answers the following questions for the above tech:

- What does this tech do for you in your project?
- This method is used to secure filenames that will be saved to the server
- Where specifically is this tech used in your project? Give us some details like file location and line number, if applicable. If too cumbersome, a general description of where it's used for a given purpose is fine as well.
- This is used in views.py on line 46.

Magic ⋆★⋆｡°✩ ⋆｡˚☽˚｡⁀➷｡˚★彡⋆༄

secure_filename is called with the filename of the uploaded file
https://github.com/ericv105/web-app-project/blob/824ab01704bb15113f36e14ba94327b66f3cdf60/website/views.py#L46

secure_filename is defined at
https://github.com/pallets/werkzeug/blob/347fdbb055c86efe1fd49546bd524cde4b98c103/src/werkzeug/utils.py#L174

This method replaces dangerous characters within filenames like spaces, periods, forwards and backslashes, with underscores, resulting in a very secure filename that cannot be used to access other files within the server.

[Flask-Login: LoginManager, login_user, logout_user, login_required, current_user]

Purpose

Replace this text with some that answers the following questions for the above tech:

- What does this tech do for you in your project?
    - This tech handles logging in and logging out users as well, authentication on certain web pages, and keeping track of the currently logged in user.
- Where specifically is this tech used in your project? Give us some details like file location and line number, if applicable. If too cumbersome, a general description of where it's used for a given purpose is fine as well.
    - login_user is used on line 30
    - logout_user is used on line 44
    - login_required is used in all routes where authentication is required
    - current_user is used in all routes and gets passed to html

Magic ⋆★˚₊‧⁺˖ ☽ ˚⁀➴｡˚ ★ ⋆ ༄

Dispel the magic of this technology. Replace this text with some that answers the following questions for the above tech:

- How does this technology do what it does for you in the **Purpose** section of this report? Please explain this in detail, starting from after the TCP socket is created. Remember, to be allowed to use a technology in your project, you must be able to know how it works.
- Where is the specific code that does what you use the tech for? You ***must*** provide a link to the specific file in the repository for your tech with a line number or number range.
    - If there is more than one step in the chain of calls *(hint: there will be)*, you must provide links for the entire chain of calls from your code, to the library code that actually accomplishes the task for you.
    - Example: If you use an object of type HttpRequest in your code which contains the headers of the request, you must show exactly how that object parsed the original headers from the TCP socket. This will often involve tracing through multiple libraries and you must show the entire trace through all these libraries with links to all the involved code.

*This section may grow beyond the page for many features.

After the creation of the Flask app, the LoginManager is saved to a variable ‘login_manager’.
https://github.com/ericv105/web-app-project/blob/develop/website/__init__.py#L38

The LoginManager object has an attribute named ‘login_view’ which gets set to the route ‘auth.login’ to redirect users in case they are not authenticated
https://github.com/ericv105/web-app-project/blob/develop/website/__init__.py#L39

After that, it is initialised
https://github.com/ericv105/web-app-project/blob/develop/website/__init__.py#L40
https://github.com/maxcountryman/flask-login/blob/9c43d645672141aba66e652d0d9484784b429072/flask_login/login_manager.py#L104

An important part of the Flask-Login is the user_loader callback. It is used to reload the User object from the user id that's stored in the ‘session’, a global flask object that is also a dictionary. This function is used by methods in flask-login to obtain the user that is currently stored in the session such as ‘logout_user’ and ‘login_required’
https://github.com/ericv105/web-app-project/blob/develop/website/__init__.py#L42

This makes use of a custom User class that I have defined in
https://github.com/ericv105/web-app-project/blob/develop/website/models.py#L20
Which inherits UserMixin class from Flask-Login in order to provide default implementations for methods that are used by LoginManager, namely, is_authenticated. The UserMixIn class is defined at
https://github.com/maxcountryman/flask-login/blob/c760c0ef7ccc95d49b4693200245a4f2b148d41b/flask_login/mixins.py#L9

Now with the LoginManager set up with User inheriting from UserMixIn, we are able to call the method ‘login_user’ which takes a User object as a parameter. An already registered user can go to localhost:5000/login and enter their details in a form to log in. When they submit that form, a POST request is sent to the server containing the form data, the /login route within auth.py handles this request and queries the database for a user with the corresponding username. The query returns the User object of that specific user, which is

then passed to the ‘login_user’ method.
https://github.com/ericv105/web-app-project/blob/develop/website/auth.py#L30

the ‘login_user’ method is defined at
https://github.com/maxcountryman/flask-login/blob/c760c0ef7ccc95d49b4693200245a4f2b148d41b/flask_login/utils.py#L144
What this method does is save the userID, sessionID to the session variable which is of type MutableMapping imported from Flask

Similarly, the ‘logout_user’ method is used to remove the user information from the session variable and it is defined at
https://github.com/maxcountryman/flask-login/blob/c760c0ef7ccc95d49b4693200245a4f2b148d41b/flask_login/utils.py#L194

Since it is just popping the information from the session, it doesn’t require an input. It is used within the logout route’s function within auth.py
https://github.com/ericv105/web-app-project/blob/3d16157d14b566a12eb3ad55a18588da4e32bd81/website/auth.py#L44

login_required is a crucial method that is used for authentication for any page on the web app. It is defined at
https://github.com/maxcountryman/flask-login/blob/c760c0ef7ccc95d49b4693200245a4f2b148d41b/flask_login/utils.py#L233
and it is used in many routes throughout the server such as ‘/’, ‘/profile’, ‘/messages/’, ‘/uploads’
This method works by getting the user object by a variable called current_user (https://github.com/maxcountryman/flask-login/blob/c760c0ef7ccc95d49b4693200245a4f2b148d41b/flask_login/utils.py#L25) which is a LocalProxy object to the user_loader callback function we implemented. If the current user object is returned, it runs the ‘is_authenticated’ method which returns true, and if no user object is returned, ‘is_authenticated’ returns false.

This extension makes the process of authentication, logging in and logging out very simple and intuitive. Since it is built on top of Flask, it makes good use of the session variable to store user login information.

[Flask-SocketIO]

General Information & Licensing

Code Repository	https://github.com/miguelgrinberg/flask-socketio
License Type	MIT
License Description	● A short and simple permissive license with conditions only requiring preservation of copyright and license notices. Licensed works, modifications, and larger works may be distributed under different terms and without source code. ● Permissions ○ Commercial use ○ Modification ○ Distribution ○ Private use
License Restrictions	● Liability ● Warranty
Who worked with this?	Jason, Eric

[@<socketio_object>.on(<event>)]

Purpose

Replace this text with some that answers the following questions for the above tech:

- What does this tech do for you in your project?
    - This allows for listening to specific events incoming to the socketio server
    - It registers a handler for these events so incoming data can be read and manipulated
- Where specifically is this tech used in your project? Give us some details like file location and line number, if applicable. If too cumbersome, a general description of where it's used for a given purpose is fine as well.
    - It's only used in main.py in lines: 13, 17, 21, and 38

Magic ★★

Dispel the magic of this technology. Replace this text with some that answers the following questions for the above tech:

- How does this technology do what it does for you in the **Purpose** section of this report? Please explain this in detail, starting from after the TCP socket is created. Remember, to be allowed to use a technology in your project, you must be able to know how it works.
- Where is the specific code that does what you use the tech for? You ***must*** provide a link to the specific file in the repository for your tech with a line number or number range.
    - If there is more than one step in the chain of calls *(hint: there will be)*, you must provide links for the entire chain of calls from your code, to the library code that actually accomplishes the task for you.
    - Example: If you use an object of type HttpRequest in your code which contains the headers of the request, you must show exactly how that object parsed the original headers from the TCP socket. This will often involve tracing through multiple libraries and you must show the entire trace through all these libraries with links to all the involved code.

*This section may grow beyond the page for many features.

A sample on method call starts at
https://github.com/ericv105/web-app-project/blob/824ab01704bb15113f36e14ba94327b66f3cdf60/main.py#L21

The ‘on’ method is define at
https://github.com/miguelgrinberg/Flask-SocketIO/blob/a10ea5cf65007061d7b3fd87b530c382007adebb/src/flask_socketio/__init__.py#L258

This decorator takes in self, message, and namespace parameters.

If namespace is not given, then namespace will equal ‘/’.
https://github.com/miguelgrinberg/Flask-SocketIO/blob/a10ea5cf65007061d7b3fd87b530c382007adebb/src/flask_socketio/__init__.py#L277

The ‘on’ method will return a decorator which is defined at
https://github.com/miguelgrinberg/Flask-SocketIO/blob/a10ea5cf65007061d7b3fd87b530c382007adebb/src/flask_socketio/__init__.py#L279

The decorator method returns a handler which is the func defined underneath the decorator at
https://github.com/ericv105/web-app-project/blob/824ab01704bb15113f36e14ba94327b66f3cdf60/main.py#L22

Before the handler is returned, _handler is called which is define at
https://github.com/miguelgrinberg/Flask-SocketIO/blob/a10ea5cf65007061d7b3fd87b530c382007adebb/src/flask_socketio/__init__.py#L281

In the _handler method, the _handle_event method is called which is at
https://github.com/miguelgrinberg/Flask-SocketIO/blob/a10ea5cf65007061d7b3fd87b530c382007adebb/src/flask_socketio/__init__.py#L734

The method _handle_event takes in a message which is the event name used, handler which is the function under the decorator, the namespace (where the function is located), and sid.
*We need to remember that the handler variable is ultimately being returned which means that our method underneath the decorator is being called after all this.

In the _handle_event method, the ret value determines the function being called. If the message is 'connect', which is a reserved event, then the ret variable will return the function handler with arg[1] which is the message received from the listener. Otherwise, the function handler is returned with all the arguments incoming from the listener. In this jumble of code, any errors are handled with try except error handling.

Before the handler is returned however, the result of _handler is first sent to socketio library:
https://github.com/miguelgrinberg/python-socketio
and calls the 'on' method at
https://github.com/miguelgrinberg/python-socketio/blob/3bd13578c82e8a94d6b0328180606c2aefd496f1/src/socketio/server.py#L165

(socketio server is already instantiated from
https://github.com/miguelgrinberg/Flask-SocketIO/blob/a10ea5cf65007061d7b3fd87b530c382007adebb/src/flask_socketio/__init__.py#L243)

The 'on' method from socketio registers the event handler given the event and our handler before returning our original handler back to us to be called at
https://github.com/ericv105/web-app-project/blob/824ab01704bb15113f36e14ba94327b66f3cdf60/main.py#L22

Notes:
Following through these series of code, the handler (function below the decorator) is kept in a variable with it's properties like name saved with the @wrap method described here:
https://www.geeksforgeeks.org/python-functools-wraps-function/

The arguments being passed onto the handler is determined by the _handle_event method.

If using a decorator was not the best choice, we could've used an alternative that results in the same thing:
socketio.on_event(<event>, <handler>)

[flask_socketio.send(<message>)]

Purpose

Replace this text with some that answers the following questions for the above tech:

- What does this tech do for you in your project?
  - This allows the event handlers in the previous section to respond to the client(s) when a message/event is received
- Where specifically is this tech used in your project? Give us some details like file location and line number, if applicable. If too cumbersome, a general description of where it's used for a given purpose is fine as well.
  - Just like the previous section, all uses of send are contained within each event_handler in main.py: 19, 36, 53

Magic ⋆★

Dispel the magic of this technology. Replace this text with some that answers the following questions for the above tech:

- How does this technology do what it does for you in the **Purpose** section of this report? Please explain this in detail, starting from after the TCP socket is created. Remember, to be allowed to use a technology in your project, you must be able to know how it works.
- Where is the specific code that does what you use the tech for? You ***must*** provide a link to the specific file in the repository for your tech with a line number or number range.
  - If there is more than one step in the chain of calls *(hint: there will be)*, you must provide links for the entire chain of calls from your code, to the library code that actually accomplishes the task for you.
  - Example: If you use an object of type HttpRequest in your code which contains the headers of the request, you must show exactly how that object parsed the original headers from the TCP socket. This will often involve tracing through multiple libraries and you must show the entire trace through all these libraries with links to all the involved code.

*This section may grow beyond the page for many features.

A sample send method call starts at https://github.com/ericv105/web-app-project/blob/824ab01704bb15113f36e14ba94327b66f3cdf60/main.py#L36

The 'send' method is defined at https://github.com/miguelgrinberg/Flask-SocketIO/blob/a10ea5cf65007061d7b3fd87b530c382007adebb/src/flask_socketio/__init__.py#L843

It takes a message as an argument and a bunch of parameters as optional such as json, to, and broadcast. All the send method does is it takes a message and optionally a bunch of socketio parameters and it creates a socketio object and calls the send method from socketio (not to be mistaken by flask-socketio) An interesting matter is that if broadcast is mentioned, the to parameter will be set to None which will affect who receives the message from the server later on.

This call is defined in another library called socketio https://github.com/miguelgrinberg/python-socketio

The method send is defined by socketio at
https://github.com/miguelgrinberg/python-socketio/blob/3bd13578c82e8a94d6b0328180606c2aefd496f1/src/socketio/server.py#L315

The send method simply returns/calls the function emit at
https://github.com/miguelgrinberg/python-socketio/blob/3bd13578c82e8a94d6b0328180606c2aefd496f1/src/socketio/server.py#L264

The emit function determines who to send the message to. Since earlier in flask-socketio, the to parameter is set to None (if broadcast was True) the line at
https://github.com/miguelgrinberg/python-socketio/blob/3bd13578c82e8a94d6b0328180606c2aefd496f1/src/socketio/server.py#L309
from 309 to 311 sets the room parameter to 'all'.

However, if broadcast was set to False or None, the 'to' parameter would be set to the sid of whoever sent the message to the server. (Therefore the send method would only respond to the client who invoked the server). We used broadcast and room, therefore we should know what is happening behind the scenes.

The emit function returns a call to another emit function in a different directory with all its params at
https://github.com/miguelgrinberg/python-socketio/blob/3bd13578c82e8a94d6b0328180606c2aefd496f1/src/socketio/base_manager.py#L157

This emit function gets all the sids to send the message to through the get_participants method at
https://github.com/miguelgrinberg/python-socketio/blob/3bd13578c82e8a94d6b0328180606c2aefd496f1/src/socketio/base_manager.py#L39

The call in between these squiggly brackets are called for every participant.
{
It then checks for whether each sid is equal to the skip_sid. If it is, then don't send the message. The emit function at base_manager then returns/calls the method _emit_internal in server.py at
https://github.com/miguelgrinberg/python-socketio/blob/3bd13578c82e8a94d6b0328180606c2aefd496f1/src/socketio/server.py#L626
which sends a message to a client through packets defined at
https://github.com/miguelgrinberg/python-socketio/blob/3bd13578c82e8a94d6b0328180606c2aefd496f1/src/socketio/server.py#L639
In here, a new library is introduced called engineio where an object is made engineio.server to call the send method at
https://github.com/miguelgrinberg/python-engineio/blob/e882f5949bdd1618d97b0cade18a7e8af8670b41/src/engineio/server.py#L218
Here, the method send again uses another library, socket, that gets the sid of the user to check whether or not the user is still connected. If it is, the packet is successfully sent to the client using socket.send() This is a method from the socket library which we already worked on this semester in class.
}

Open-Source Technology Use Report

Proof of knowing your stuff in CSE312

Flask - MongoEngine

General Information & Licensing

Code Repository	https://github.com/MongoEngine/flask-mongoengine
License Type	MIT License
License Description	• A short and simple permissive license with conditions only requiring preservation of copyright and license notices. Licensed works, modifications, and larger works may be distributed under different terms and without source code. • Permissions: ○ Commercial Use ○ Modification ○ Distribution ○ Private Use
License Restrictions	• Liability • Warranty
Who worked with this?	Eric, Jason, Devin

Use as many of the sections below as needed, or create more, to explain every function, method, class, or object type you used from this library/framework.

[MongoEngine Extension]

Purpose

Replace this text with some that answers the following questions for the above tech:

- What does this tech do for you in your project?
  - This tech allows us to connect our mongo database to our flask app
- Where specifically is this tech used in your project? Give us some details like file location and line number, if applicable. If too cumbersome, a general description of where it’s used for a given purpose is fine as well.
  - This tech is used in our __init__.py file which is ran when main.py is ran

Magic ⋆★˚｡⋆˚ ☽˚｡⋆ ⋆｡˚★⋆ ༄

This technology is involved with the creation of the database in relation to the flask app. It is created in the __init__.py file where it is passed the flask app as a parameter on line 24,
https://github.com/ericv105/web-app-project/blob/824ab01704bb15113f36e14ba94327b66f3cdf60/website/__init__.py#L24

We then, with flask, can also change the apps settings to connect our mongo database to our docker database. In lines 18-19 we set the connection properly.
https://github.com/ericv105/web-app-project/blob/824ab01704bb15113f36e14ba94327b66f3cdf60/website/__init__.py#L18

[WTForms]

Purpose

Replace this text with some that answers the following questions for the above tech:

- What does this tech do for you in your project?
  - This tech allows us to create objects that can be customized and saved to our database.
- Where specifically is this tech used in your project? Give us some details like file location and line number, if applicable. If too cumbersome, a general description of where it's used for a given purpose is fine as well.
  - This tech is used in the models.py file

Magic ⋆★˚｡⋆୨୧˚ ☽˚⁀➴｡˚★彡⋆

This technology allows us to create classes called “documents” from the MongoEngine library and customize them to allow us to store certain information to the database.

These classes (User, Upload) are created for us in the models.py file Lines 20-24
https://github.com/ericv105/web-app-project/blob/824ab01704bb15113f36e14ba94327b66f3cdf60/website/models.py#L20
The user class is used to store user accounts to the database and lets us save a username, password, online status, and avatar file path

Lines 27-30
https://github.com/ericv105/web-app-project/blob/824ab01704bb15113f36e14ba94327b66f3cdf60/website/models.py#L27
The Upload class is used to store images to the database and lets us save a file name, amount of votes the image received, and the file path.

Once initialized, these objects can be created and saved to the database, as shown in the auth.py file in lines 72-73
https://github.com/ericv105/web-app-project/blob/824ab01704bb15113f36e14ba94327b66f3cdf60/website/auth.py#L72

And once they are saved, they can be accessed to retrieve information. In the file auth.py line 59, the user object is used to check all instances of usernames in the database
https://github.com/ericv105/web-app-project/blob/824ab01704bb15113f36e14ba94327b66f3cdf60/website/auth.py#L59

Open-Source Technology Use Report

Proof of knowing your stuff in CSE312

Jinja

General Information & Licensing

Code Repository	https://github.com/pallets/jinja
License Type	BSD 3-Clause "New" or "Revised" License
License Description	● A permissive license similar to the BSD 2-Clause License, but with a 3rd clause that prohibits others from using the name of the project or its contributors to promote derived products without written consent. ● Permissions: ○ Commercial Use ○ Modification ○ Distribution ○ Private Use
License Restrictions	● Liability ● Warrant
Who worked with this?	Josh

Use as many of the sections below as needed, or create more, to explain every function, method, class, or object type you used from this library/framework.

Purpose

Replace this text with some that answers the following questions for the above tech:

- What does this tech do for you in your project?
  - This tech allows us to create templates of HTML in which we can later extend aspects of it, to other HTML files. These templates contain variables and expressions that get replaced with values, when it is rendered. We wanted a navigation bar across all of our site's pages, so we went with Jinja's templating functionality.

- Where specifically is this tech used in your project? Give us some details like file location and line number, if applicable. If too cumbersome, a general description of where it's used for a given purpose is fine as well.
  - This tech can be found on all of our HTML files. These files can be found at https://github.com/ericv105/web-app-project/tree/main/website/templates

Magic ⋆★˚｡⋆˚☽˚⁀➷｡˚★彡⋆༄

We start off by creating our template. This is an HTML file containing several variables, tags, and expressions.
https://github.com/ericv105/web-app-project/blob/main/website/templates/base.html

To inherit the attributes of this template, we had to extend the file on other htmls. Below is an example of this:
https://github.com/ericv105/web-app-project/blob/824ab01704bb15113f36e14ba94327b66f3cdf60/website/templates/dm.html#L1

At the top of every web page, you will see a title that describes the path you are currently on. To make it change from page to page, we used the block tag which indicates an area of code that will be changed depending on which page you are on. We named this tag 'title'.
https://github.com/ericv105/web-app-project/blob/824ab01704bb15113f36e14ba94327b66f3cdf60/website/templates/base.html#L18

You can see, we have our template initializing it's title to 'Home'. On other pages, in place of home, will be the title of it's page. For example, on the line of code linked below, you can see how we changed the title for our Direct Messaging page.
https://github.com/ericv105/web-app-project/blob/824ab01704bb15113f36e14ba94327b66f3cdf60/website/templates/dm.html#L2

As well, we use a block tag to change the body of our pages. Here we called our block, 'content'. Below is an example of how we used the tag to change the body of our Login Page.(Line 4 is where the block starts / Line29 is where the block ends)

- https://github.com/ericv105/web-app-project/blob/824ab01704bb15113f36e14ba94327b66f3cdf60/website/templates/login.html#L4
- https://github.com/ericv105/web-app-project/blob/824ab01704bb15113f36e14ba94327b66f3cdf60/website/templates/login.html#L29

As stated previously, we wanted a navigation bar to appear on all of the pages of our site. Depending on if the user was logged in or not, their navigation bar will vary. To do this, we needed to use if statements and pass the user object when rendering plates. Lines 36-45 shows how we check if the user is authenticated or not and shows what information we displayed depending on their state.
https://github.com/ericv105/web-app-project/blob/824ab01704bb15113f36e14ba94327b66f3cdf60/website/templates/base.html#L36

Below shows how we passed the user when rendering plates.
https://github.com/ericv105/web-app-project/blob/824ab01704bb15113f36e14ba94327b66f3cdf60/website/auth.py#L37

Lastly, Jinga helps us display any messages sent with the flash() method. Some flashes would indicate to the user that they did something wrong and some would indicate that the user succeeded in completing their task. To do this, we needed to use Jinga's 'with' statement.
https://github.com/ericv105/web-app-project/blob/824ab01704bb15113f36e14ba94327b66f3cdf60/website/templates/base.html#L51

The 'with' statement creates a new inner scope that we use to decide if the user should see an error alert or a success alert. When sending a flash message, users also send the type of flash it is. Below is an example of this how the flash is sent:
https://github.com/ericv105/web-app-project/blob/824ab01704bb15113f36e14ba94327b66f3cdf60/website/auth.py#L45

In the scope we created with the 'with function, we use for loops and if statements to compute the correct type of flash.