Werkzeug

General Information & Licensing

Code Repository	http:s//github.com/pallets/werkzeug
License Type	BSD 3 Clause
License Description	● With copyright included: Copyright 2007 Pallets ● Commercial Use allowed ● Modification, Distribution and Private use allowed
License Restrictions	● No liability ● No warranty ●
Who worked with this?	James Aquilina, Daniil Khan, Will Marchant, Ryan Yam

.run(*host, *port, *debug) method

Purpose

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

- This tech allows us to run our server on localhost and connects us to a browser to simulate an app working over the internet. It is the entry point into our app and establishes the HTTP protocol connection.
- This function is used in our server.py file, line 27.

Magic ⋆★⋆｡°⋆⋅˚☽˚⁀➷｡˚★彡⋆༄

- This technology opens a port on our local machine (5000) and when entering the localhost hostname in the web browser establishes a connection to our app. This is a development server. From this connection, we can receive different requests based on the type of request that was sent. We can also send information and data back to the client (browser) through this connection.
- After importing the Flask library and creating our app in __init__.py we can call app.run()
- First Link: Run is a method from the base Flask class. This is where Flask objects our created which is the "app" and acts as the central core of the entire app. It takes a few parameters, and in our case we passed it: host= "0.0.0.0", port= 5000, debug=True https://github.com/pallets/flask/blob/a03719b01076a5bfdc2c8f4024eda7b874614bc1/s rc/flask/app.py#L805
- Second Link: Near the bottom of the run function is an import statement: from werkzeug.serving import run_simple The run method calls the run_simple() function and run() passes the same parameters to run_simple() https://github.com/pallets/flask/blob/a03719b01076a5bfdc2c8f4024eda7b874614bc1/s rc/flask/app.py#L920
- Third Link: The run_simple function comes from the werkzeug library and specifically it is from the ForkingWSGIServer class. https://github.com/pallets/werkzeug/blob/c7ae2fea4fb229ffd71187c2b665874c91b962 77/src/werkzeug/serving.py#L917
- Fourth Link: This class is derived from the BaseWSGIServer which is a single process threaded server class. The BaseWEGIServer is derived from the HTTPServer https://github.com/pallets/werkzeug/blob/c7ae2fea4fb229ffd71187c2b665874c91b962 77/src/werkzeug/serving.py#L635
- FIfth Link: HTTPServer is imported from the HTTP library. https://github.com/python/cpython/blob/886b22c4c3c5324bcebb79402c92d18ebc4224 b6/Lib/http/server.py#L130
  The HTTP server and library are where all of the handling of the requests happens

.render_template(template_name_or_list, **context)

Purpose

What this tech does for us in our project is it renders a given template with the context that it is given. This can do things such as setting JS variables in an HTML and then return the rendered template. It is very similar to just using Python .replace method and templeting that way. It is used many times throughout our project. In account.py line 26. In auth.py lines 119, 130, 133, 138, 139, 153, 157, 169, 197, 199, 200, 228. In images.py line 15. In messages.py line 18. In upload.py lines 31, 34, 36. In users.py line 35. In views.py lines 35, 40, 41.

Magic ⋆★⁺₊˚.⋆˚☽˚⁀➴｡˚★彡⁺₊⋆༄

First Link: How Flask’s .render_template() works is that it works in tandem with Jinja2 to replace variables and format HTML templates so they can be sent to the front end. This means that things like {# #}, {{ }},{% %} can be used in your template to denote comments, statements, and expressions. Through templating we can replace things inside these tags and change values. This is used many times throughout our project as each HTML file is templated and before being sent to the front end.
https://jinja.palletsprojects.com/en/3.1.x/templates/

Second Link: The chain of command of render_template is in the file, the python call is made. That python calls Flask and sends the template_name_or_list to Jinja. The sent data is returned to Flask which then returns that to where the python call was made. This template is now ready to be sent to the client.
https://pythonbasics.org/flask-tutorial-templates/

Third Link: Not only will render_template just render the template for front end use but it also can render the template and change given variables. This is done a few times in our code. For example in uploads.py line 34 the context imList = imageList is specified. Then in upload.html {%for i in imList %} has imList replaced with the value contained in imageList. Jinja sees {{ }} tags as what to display and {% %} tags as code. So when the context is sent to Jinja it looks for the variable specified inside of {% %} tags in the template and replaces them with the given value. When rendered by Jinja and returned this is code that is readable by the server.
https://github.com/pallets/flask/blob/2f0c62f5e6e290843f03c1fa70817c7a3c7fd661/tests/test_templating.py (Line 17 shows an example)
https://github.com/Technocat44/Web-App-JWRD/blob/main/webapp/upload.py
https://pythonbasics.org/flask-tutorial-templates/

Blueprint

Purpose

Blueprint was used as a way to organize our view functions. These view functions are called when a request is made to the specified path. Then Blueprint is nothing more than just a way to group together these functions. It is used many times throughout our project to make it cleaner and easier to debug. It is in account.py line 6, auth.py line 17, images.py line 4, messages.py line 5, upload.py line 5, users.py line 6, and views.py line 5.

Magic ⋆★⁺₊˚⋅˚☽˚⁀➷｡˚★彡⁺₊༄

First link: How Blueprint works is that a name is specified for the given Blueprint and __name__ is passed as the second argument. This name is then used when registering the Blueprint. An example of this Blueprint creation is shown below in lines 8-11.
https://github.com/pallets/flask/search?q=Blueprint

Blueprint doesn't really have a long chain of code to run it like other functions. It is basically an enhanced views function. All Blueprint does is specify the name of a Blueprint but where it is important is when it comes to register said Blueprint using the register_blueprint function. This function is gone into detail below. Blueprint is used many times throughout our project to link functions to paths so said function is called when that path is called.

.register_blueprint(blueprint, **options)

Purpose

The tech.register_blueprint is a function that allows us to link our created Blueprint(gone into detail above) to a specific path. This is similar to using a whole slew of ifs in the HWs but a lot nicer. This is used heavily in our code and is very important because without registering a Blueprint it has no use to us. It is in server.py lines 13,14,15,17,18.

Magic ⋆★˚｡⋆˚ ☽˚⁀➷｡˚★彡⋆ ༄

First link: By using .register_blueprint we link the Blueprint specified in the argument blueprint and its functions to our app. So when a path is reached that has a Blueprint connected to it the function linked to said path will be called upon and used in the response by our app. Register_blueprint can take in url_prefix and when the blueprint route is reached the route will be appended with the specified prefix. Otherwise it will just default.
https://flask.palletsprojects.com/en/2.1.x/api/?highlight=register_blueprint#flask.Blueprint.register_blueprint
https://realpython.com/flask-blueprint/#:~:text=Registering%20the%20Blueprint%20in%20Your%20Application,-Recall%20that%20a&text=When%20you%20call%20.,()%20from%20the%20Flask%20Blueprint.

Second link: register_blueprint calls the Blueprints .register() method. This register() method creates a BlueprintSetupState and then proceeds to call .record() with the created BlueprintSetupState
https://github.com/pallets/flask/blob/bd56d19b167822a9a23e2e9e2a07ccccc36baa8d/src/flask/blueprints.py (beginning on line 271)

Third link: This BlueprintSetupState is created by the method make_setup_state(). This function creates an instance of BlueprintSetupState and returns it.
https://github.com/pallets/flask/blob/bd56d19b167822a9a23e2e9e2a07ccccc36baa8d/src/flask/blueprints.py (beginning on line 245)

Fourth link: The method record() is what registers a function to the Blueprint. This record function is called with the returned BlueprintSetupState from the method make_setup_state().
https://github.com/pallets/flask/blob/bd56d19b167822a9a23e2e9e2a07ccccc36baa8d/src/flask/blueprints.py (beginning on line 214)

.request()

Purpose

The .request is a Flask object that was used for many things and has many methods to itself that we used. The request object is used for many things in our project as it takes the incoming request data and then allows us to access traits of the incoming data in a request object. The request object and its related methods are found in account.py on lines 15, 20,22,31,32,33,37,52,58, and auth.py on lines 50,53,55.57,126,149,179,180,203,206,230,241,242,243,244,304,315,316,317,318 and in images.py in lines 13,14 and in messages.py on lines 10,13,14 and in upload.py on lines 14,15,16,19,20 and in users.py on lines 51,62,63,78 and in views.py on line 32.

Magic ⋆★⁺₊˚⋆˚☽˚⁀➷｡˚★彡⁺₊⋆༄

.get_data : this method gets the incoming buffered data as bytes and allows us to manipulate it to what we need it for. This method does the same thing as request.recv that was used in the HWs.

.method : this method gets whether the request is a POST or GET request. It parses the headers to find this and it is the same as just using .split on decoded request data. https://github.com/pallets/flask/blob/ea93a52d7d94ba093bbce4680c622cc4fc9771d8/tests/test_async.py (an example call is on line 27)

.form : is used when form data is sent. By doing .form we can use .get to access specific fields of the form. https://github.com/pallets/flask/blob/afc13b9390ae2e40f4731e815b49edc9ef52ed4b/examples/tutorial/flaskr/auth.py (an example is used on line 54 and 55)

.get_json : is used when a json object is received. If the mime type isn't specified as JSON then on_json_loading_failed(e) is called and raises a BadRequest. https://github.com/pallets/flask/blob/afc13b9390ae2e40f4731e815b49edc9ef52ed4b/tests/test_json.py (an example call is on line 20)

These method calls are all used by the request object is our project and make parsing and accessing request data as easy as a simple method call. https://flask.palletsprojects.com/en/2.1.x/api/#flask.Request.get_json

flask.session *class*

Purpose

This class works almost the same as a regular dictionary. It keeps track of all the data that is passed in for each user session. In our case, we use it for storing some user data (i.e. ids, usernames of users) with the purpose of easy access to it and some checks of what user sent requests to the server. The session class can be found used in *auth.py* on lines 122, 295, 369; in *messages.py* on lines 15, 17; in *users.py* on 64, 79, 80, 81, 82 ,83.; in *views.py* on 39.

Magic ⋆★⁺₊˚ ⋅ ˚☽ ˚⁀➴｡˚★彡⁺₊⋆༄

This technology is nothing but a class that allows to store data of specific users. The easiest way to describe it is by imagining a dictionary where the keys are users and the values are another empty dictionary. In that empty dictionary we can put whatever values we want. Session class just switches users for us based on from what user the request is received, so we don't have to specify the user. Flask implements a session as proxy which is connected to a thread. Everytime we use .session, it uses the SessionMixin class which essentially is just an expanded python basic dictionary (since it expands MutableMapping from python).

First Link: When .session is called, first global.py is referred to as SessionMixin class. https://github.com/pallets/flask/blob/bd56d19b167822a9a23e2e9e2a07ccccc36baa8d/src/flask/globals.py on line 56.

Second Link: actual SessionMixin class that was referred above. https://github.com/pallets/flask/blob/bd56d19b167822a9a23e2e9e2a07ccccc36baa8d/src/flask/sessions.py on line 20.

flask.**redirect**(*location*, *code=302*, *Response=None*)

Purpose

flask.redirect is a method that is called when a redirect is in order. It supports many redirect codes except 300 or 304. This call is made a few times throughout our code in auth.py on lines 63,72,88, 298, 372.

Magic ⋆★⁺₊˚⋆˚☽˚⌒➴｡˚★彡⁺⋆ ༄

There isn’t much *Magic* ⋆★⁺₊˚⋆˚☽˚⌒➴｡˚★彡⁺⋆ ༄ behind the redirect method at all it simply just sends a flask.response to the specified url with the specified code. What this does for us is what we would do by crafting a string response and using sendall on the homework. The werkzurg.wrappers.Response is used to set all the headers and create the response if the Response argument is left empty.

https://github.com/pallets/flask/blob/bd56d19b167822a9a23e2e9e2a07ccccc36baa8d/src/flask/typing.py (this github repo file is is for the werkzurg.wrappers.Response class)
https://werkzeug.palletsprojects.com/en/2.1.x/wrappers/#werkzeug.wrappers.Response
https://flask.palletsprojects.com/en/2.1.x/api/#flask.Request.get_json

flask.flash

Purpose

This tech is essentially the base of the notification system for our project. It can act similar to alert() in JavaScript but does not provide a notification that prompts the user to say OK before it allows them to interact with the application again. Flash can be used and not disrupt the flow of the application to provide a better user experience. Flash is used mainly in auth.py and is used about 31 times to provide the user with the knowledge that they have been logged in, or logged out, their username is being taken, and so on.

Magic ⋆★⁺₊°.°☽°⁀➴｡°★彡⁺₊⋆ ༄

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.

- When the socket is created, this flash function allows for messages to be sent as data that will be rendered onto the web application. It is similar to leaving a {{message}} tag on the html and using render_template to display a message on the web application. This is an alternative to using alert() on javascript which is a pop up message compared to a subtle message on the page which is what flash does.
- **First Link:** The first link is the calls in the auth.py file, in which there is a flash message popped up. https://github.com/Technocat44/Web-App-JWRD/blob/9d2d6d2587e7519d203f570f9330db5459fcad8a/webapp/auth.py#L2
- **Second Link:** This call comes from flask and is documented in the helpers.py file in the flask library. Flash as a function takes a message and a category (type of message). https://github.com/pallets/flask/blob/fac630379d31baaa1667894c2b5613304285a4bb/src/flask/helpers.py#L418
- **Third Link:** The flash function adds the new message with it's category to a list of what the flask library will need to flash. https://github.com/pallets/flask/blob/fac630379d31baaa1667894c2b5613304285a4bb/src/flask/helpers.py#L441
- **Fourth Link:** The flash function will reach into the session's flashes and set that value to the flashes it got from the current session. Session is a separate file in flask that holds information for the current session of the application. It does not have any calls for flashes in it but holds that information for the future. https://github.com/pallets/flask/blob/fac630379d31baaa1667894c2b5613304285a4bb/src/flask/helpers.py#L442
- **Fifth Link:** The last call of the function is to send the flashed message through message_flashed. Which is in the signals.py file, the message_flashed value will signal that a message needs to be flashed. https://github.com/pallets/flask/blob/fac630379d31baaa1667894c2b5613304285a4bb/src/flask/signals.py#L56
- **Sixth Link:** Signal essentially sends signals to the app but will error when it fails. It creates a 'FakeSignal' in order to do this and does not import any other libraries. https://github.com/pallets/flask/blob/fac630379d31baaa1667894c2b5613304285a4bb/src/flask/signals.py#L11

*.url_for(endpoint, **values)*

Purpose

This method is pretty straightforward. There isn’t much magic behind this, all it does is it creates a link to the specific method in a file. It will be dynamically updated if it's changed. The purpose of this method is to use those links for the redirect method described in this document. This method is user in *auth.py* on lines 63, 72, 88, 298, 372.

Magic ⋆★⋄⁺ₒ ˚ ⬩ ˚ ☽ ˚⁀➷ ₒ ˚★彡⋄⁺⋆ ༄

As said in the purpose section, there isn’t much magic behind. It literally straightforwardly creates a url. An endpoint is given but a full path is returned. We use that path for the redirect method described in this document.

First link:
https://github.com/pallets/flask/blob/bd56d19b167822a9a23e2e9e2a07ccccc36baa8d/src/flask/helpers.py on line 192.

make_response(*rv*)

Purpose

We use make_response in order to convert a Flask View into a response. The reason behind converting it is because it is impossible to add additional headers to Flask View, so we convert it into a Flask Response and add necessary headers to it.

The make_response function allowed us to quickly create a response that contained specific headers we needed. In the HW we created responses with headers that had different mimetypes, the status code, or the content-type for example. Flask allows us to do the same but with the help of this function we were able to wrap it around a render_template function call and then use it to set the cookies for an authenticated user. This was the idiomatic way to set cookies in Flask.

Magic ⋆★˚₊° ⋆˚☽°⌒➷｡°★彡⁺★༄

Make_reponse is a pretty small function with not too long of chain of calls as it is sort of an extension of Reponse it just allows up to set headers.

First link: If no args are specified then just the response is sent using current_app.response.
https://github.com/pallets/flask/blob/fac630379d31baaa1667894c2b5613304285a4bb/src/flask/helpers.py (in line 190)

Second link: If some args are specified then a convert is used to a view function into an instance of the Response class.
https://github.com/pallets/flask/blob/fac630379d31baaa1667894c2b5613304285a4bb/src/flask/app.py (in line 1677)

class werkzeug.wrappers.**Response**(response=None, status=None, headers=None, mimetype=None, content_type=None, direct_passthrough=False)

Purpose

The Response class is used by make_response and it allows us to set additional headers in a view. Because the views do not have to return response objects but can return a value that is converted into a response object by Flask itself, it becomes tricky to add headers to it. This function can be called instead of using a return and you will get a response object which you can use to attach headers.

We used make_response on line 119 of *auth.py* in the webapp directory.
https://github.com/Technocat44/Web-App-JWRD/blob/9d2d6d2587e7519d203f570f9330db5459fcad8a/webapp/auth.py#L119

Magic ⋆★⋄₊˚‧⋆˚ ☽ ˚⌒➴｡˚★彡⋄⁺★ ༄

*This section may grow beyond the page for many features.
First link: if no arguments are passed, it creates a new response argument if more than one argument is passed, the arguments are passed as a tuple.
https://github.com/pallets/flask/blob/fac630379d31baaa1667894c2b5613304285a4bb/src/flask/helpers.py#L147

Second link: The response is inherited from wrappers.py file and inside that file is the next link in the chain.
https://github.com/pallets/flask/blob/fac630379d31baaa1667894c2b5613304285a4bb/src/flask/wrappers.py#L15

Third link: The response base class that werkzeug uses to make HTTP responses
https://github.com/pallets/werkzeug/blob/bb21bf90b0b121e3ed45b9950b823e4b43a81fd8/src/werkzeug/wrappers/response.py#L67