Open-Source Technology Use Report

Proof of knowing your stuff in CSE312

Guidelines

Provided below is a template you must use to write your report for each of the technologies you use in your project.

Here are some things to note when working on your report, specifically about the **General Information & Licensing** section for each technology.

- **Code Repository**: Please link the code and not the documentation. If you'd like to refer to the documentation in the **Magic** section, you're more than welcome to, but we'd like to see the code you're referring to as well.
- **License Type**: Three letter acronym is fine.
- **License Description**: No need for the entire license here, just what separates it from the rest.
- **License Restrictions**: What can you *not* do as a result of using this technology in your project? Some licenses prevent you from using the project for commercial use, for example.
- **Who worked with this?**: It's not necessary for the entire team to work with every technology used, but we'd like to know who worked with what.

Also, feel free to extend the cell of any section if you feel you need more room.

If there's anything we can clarify, please don't hesitate to reach out! You can reach us using the methods outlined on the course website or see us during our office hours.

Flask

General Information & Licensing

Code Repository	https://github.com/pallets/flask/
License Type	BSD 3-Clause License
License Description	• Commercial use • Modification • Distribution • Private use
License Restrictions	• Liability • Warranty
Who worked with this?	Zaki, Chi Ho, Tenzin, Gorden

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.

App.route() (Function)

Purpose

- The app object, which is created from the Flask class, acts as a router for our project.
- It determines what HTTP requests are made and executes a function accordingly
- It is used in app.py.
    - On line 1, the Flask class is imported
    - On line 8, the app object is created as an instance of the Flask class
    - On lines 14, 20, 28, 36, 42, and 50, the app object uses its route function to determine the http request path and method. If it is a match, then it will execute the function below as a response.

Magic

- We imported flask's app.route functionality to simplify and shorten code when users travel from path to path within our website. This method directly correlates to flask's function add_url_rule. The method app.route can take two parameters. One is called rule, which is of type string and is the path name. The other parameter is called options, which is of type any and, for our implementation, is a list of strings. Our list of strings can contain the strings 'GET' and 'POST' and defines whether the path takes 'GET' and/or 'POST' requests. If the list is empty, it defaults to the path taking 'GET' requests only.
- Currently in our app.py file, we have several methods that use the app.route (aka add_url_rule) method such as routeLogin which uses the app.route to take the user to the path of /login and allows that path to take both 'GET' and 'POST' requests.

function add_url_rule
https://github.com/pallets/flask/blob/main/src/flask/app.py (lines 1036-1093)
Lines 1038-1043 contains the parameters of add_url_rule

- self: referring to the class
- rule: (string) referring to the path name
- endpoint: (optional type of List of Strings) The endpoint name to associate with the rule and view function. Used when routing and building URLs. Defaults to view_func.__name__.
- view_func: (optional type of List of callable types) The view function to associate with the endpoint name.
- provide_automatic_options: (optional type of List of booleans) Adds to the options method and responds to options requests automatically.
- options: (type of any) Extra options added to the Rule object from werkzeug

Lines 1045-1046 checks our endpoints and if it's not set to anything, it defaults to the name of the view function. An example in our code is @app.route('/logout', methods=['POST']) whose associated view function is the def logout function.

Line 1047-1048 sets the endpoint in options equal to whatever lines 1045-1046 found the view function to be, which could be the default view function name or a specifically set one. Going back to our example, options["endpoint"] is equal to logout (the def logout function)

In summary Lines 1045-1048 attaches the app.route to the appropriate function name so that when the path is called for that route, the function according to the path runs.

Line 1048 checks if any keys containing “methods” were passed to options. If there were any keys containing methods sent, then it’s key value pair is added to a variable called *methods*. Otherwise if there are no keys containing “methods” the variable methods is set to NONE. For our example since we wrote ‘POST’, the variable *methods* contains ‘POST’

Lines 1053-1060 works with *method* (which was the List of ‘GET’ and/or ‘POST’) and checks if it’s empty. If empty, it makes the page accept ‘GET’ requests only by default.

Since app.route doesn’t utilize provide_automatic_options we can ignore lines 1066-1077

Line 1082 sets rule equal to a rule object which contains the path name, the *methods* variable (list of type of requests) and options. This uses werkzeug’s rule class in https://github.com/pallets/werkzeug/blob/main/src/werkzeug/routing.py/

Within the rule class all it initializes itself (line 681 https://github.com/pallets/werkzeug/blob/main/src/werkzeug/routing.py/) by first checking if the url contains a leading / on line 696 and without it, it’s considered invalid. After they validate the string, they then set rule equal to the url string value on line 698 We can skip until line 711 where our *methods* variable is called which once again checks if anything is contained in the methods list and adds ‘HEAD’ in the message if ‘GET’ is within *methods* and ‘HEAD’ isn’t. Since those were the only times our parameters were mentioned we can generally ignore the rest of the code within https://github.com/pallets/werkzeug/blob/main/src/werkzeug/routing.py for the rule class.

Going back to https://github.com/pallets/flask/blob/main/src/flask/app.py we can see on line 1082 we add a map which calls back to https://github.com/pallets/werkzeug/blob/main/src/werkzeug/routing.py on line 1411 which just stores the url.

Then from lines 1086-1093 it utilizes endpoints which essentially sets the endpoint to the view_func although through app.route it is defaulted to that.

Overall what the add_url_rule function does is that it takes the url path name, the type of requests it uses and when those specific path and requests get sent, it executes the function corresponding to it. Since it’s the more descriptive version of what app.route does, we explained it using this function

Request (object)

Purpose

- HTTP request attributes are stored in this object
- It is used in app.py. It is used to check the HTTP request method.
- Through this we can quickly find things within the HTTP request such as methods like ‘GET’ or ‘POST’, cookie values, and if a form was submitted, form values
- It’s used whenever we want to find the cookie values from the http header to determine login, the form data when users submit login, register or post data, the file when users submit some form of media and method when we want to figure out whether a specific request is a “GET” or a “POST” to either send them to the login/register page or to the index page (if successful)

Magic

We use the request object as a way for us to parse through the http requests that we receive and get specific attributes from the http request. The attributes that we use from the request object is request.cookies, request.form, request.method and request.files

https://github.com/pallets/flask/blob/main/src/flask/wrappers.py line 16
https://github.com/pallets/werkzeug/blob/main/src/werkzeug/wrappers/request.py line 29
https://github.com/pallets/werkzeug/blob/main/src/werkzeug/sansio/request.py line 38

From the flask/wrappers.py github we find the request object which uses werkzeugs werkzeug/wrapper/request.py library which uses an object from werkzeug’s werkzeug/sansio/request.py library which uses functions in werkzeug’s werkzeug/http.py and werkzeug/formparser.py libraries

To begin we start in werkzeug’s sansio/request.py library where it parses its header values through the following libraries.

https://github.com/pallets/werkzeug/blob/main/src/werkzeug/http.py

- Line 1163 contains the parse_cookie method which decodes and parses the http request if a cookie is contained and, if it does exist, returning it in a dictionary

This function is used in sansio/request.py library on line 247 where the function def cookies gets the cookie values in a dictionary and uses the parse_cookie method on line 251

For https://github.com/pallets/werkzeug/blob/main/src/werkzeug/wrappers/request.py the form data can be found through

https://github.com/pallets/werkzeug/blob/main/src/werkzeug/formparser.py

- Line 74 contains the parse_form_data method which decodes and parses the http request for the form data and returns it to a class named form_parser (line 156) which sends it to the parse_from_environ function (line 219) which passes the information to the parse function (line 230) which returns it as a tuple containing the input stream, the form data and the file (the data (ie image) in bytes).

And the methods value which contains the type of request can be found from line 46 in the

https://github.com/pallets/werkzeug/blob/main/src/werkzeug/sansio/request.py library. From the werkzeug/sansio/request.py we can find and parse the values of the methods, cookie, form data and the file data through the usage of other functions of other linked libraries. The libraries https://github.com/pallets/flask/blob/main/src/flask/wrappers.py and https://github.com/pallets/werkzeug/blob/main/src/werkzeug/wrappers/request.py build off of the https://github.com/pallets/werkzeug/blob/main/src/werkzeug/sansio/request.py and adds functionality that we did not use.

Render_template (Function)

Purpose

The render_template function's purpose is to render the html template with the variable given. It takes the template and changes the variables given the key value pair.
Used in app.py, lines 34,35,46,75,147. In Line 35 it was used to render and send a http response for the index page that only logged in users can see with their saved settings of username and darkmode. Line 34 simply sends the landing page response to unlogged. in..
Line 46 to send the user the login form on the path /login.
Line 75 renders another user's page to send to the user wanting to see it.
Used in postHandlers.py line 24,61,63, ChatHandlers.py line 10, Authhandlers.py line 50 to render some form of html.

Magic

We used this mainly to render our html templates with our stored user data.

https://github.com/pallets/flask/blob/main/src/flask/templating.py line 133
https://docs.python.org/3/library/typing.html#typing.Union
https://github.com/pallets/flask/blob/9486b6cf57bd6a8a261f67091aca8ca78eeec1e3/src/flask/templating.py#L124

From the flask/templating.py github,the function gets the top value of a stack called LocalStack from werkzeug. Context Locals — Werkzeug Documentation (2.0.x)
This stack is gives us the in flask env, which then calls `update_template_context.`
https://github.com/pallets/flask/blob/9486b6cf57bd6a8a261f67091aca8ca78eeec1e3/src/flask/app.py#L731

That function returns a dictionary containing the variables that we want to change and those the context preprocessor wants to inject.
https://github.com/pallets/flask/blob/main/src/flask/templating.py#L124

After that, render is called with the 3 params. The first parameter `ctx.app.jinja_env.get_or_select_template(template_name_or_list),` is called through jinja. If it is a list it calls find_template to iterate through it to find one that works and render it. Else it calls get_template and returns it as a template.
https://jinja.palletsprojects.com/en/3.0.x/api/#jinja2.Environment.get_or_select_template

`rv = template.render(context)`, template.render is from jinja, which replaces the template with the values desired in the context dictionary and returns the finished template as a string.
https://github.com/pallets/jinja/blob/main/src/jinja2/environment.py#L1257

Rv is the now the rendered template in string.

Redirect / url_for (function)

Purpose

- The purpose of the redirect function is to redirect users into the new url. It occurs several times in our code for example when there are successful logins and we write a 302 response to the web page to redirect the user to their main profile page.
- The redirect function works with the url_for function where the url_for function takes a string as a parameter and returns the url for the function that matches that string.
- These functions were used in our app in
    - Line 54 to redirect to the url thats with the renderhome function
    - Line 63 to redirect to the url thats with the routeLogin function
    - Line 81 to redirect to the url thats with the renderhome function
    - Line 83 to redirect to the url thats with the routeLogin function
    - Line 99 to redirect to the url thats with the renderhome function
    - Line 118 to redirect to the url thats with the routeLogin function
    - Line 183 to redirect to the url thats with the routeUser function with the user id sent as a variable to send to the routeUser Function

Magic

To start we can find the url_for function on line 192 to line 340 https://github.com/pallets/flask/blob/main/src/flask/helpers.py where it creates two objects of class Local Stack from https://github.com/pallets/flask/blob/main/src/flask/globals.py which can be referenced on line 5 where its from and line 52 and 53. https://github.com/pallets/werkzeug/blob/main/src/werkzeug/local.py

- Local stack works like a normal stack with the same functionality for pop, top and remove.

https://github.com/pallets/flask/blob/main/src/flask/helpers.py

- Since we only hand a single parameter, we can see that due to the values** being equal to none in line 306-309 the values for anchor, method and scheme are set to none. Then we can ignore lines 314-319 and 329-339 since it's only when those values are set to something other than none. Finally we see that the return value is a string equal to an adapter.build

Overall the url_for function just converts a string into a function, which then finds the url attached to that function and if there are variables put in url_for, it carries it over and puts it to the endpoint (string with the name of the function) and puts it into its parameters.

https://github.com/pallets/flask/blob/main/src/flask/__init__.py. Line 4
In the init.py file flask uses werkzeug's implementation of redirect fully https://github.com/pallets/werkzeug/blob/main/src/werkzeug/utils.py line 221
Within the def redirect function on line 221, the function takes three parameters which are the url string (which we get from url_for), the response code which defaults to 302 and a class of Response. It also escapes HTML. It returns a Response object class with location set to the url location received in the string.

The Response object class is from the library https://github.com/pallets/werkzeug/blob/main/src/werkzeug/wrappers/response.py

- This class builds a Http response and returning it sends it to the website. On lines 102-123

Flash()

Purpose

- Flash allows us to display a message on the page to let users know their actions were unsuccessful either because they don't have access or they made an error somewhere.
- Flash is used in app.py and authHandlers.py when registering, logging in, a cookie is edited or removed when accessing your own profile, and accessing another user's profile. When registering, a flash message is displayed when the username is taken or the password doesn't meet the proper requirements. When logging in, a flash message is displayed when either the username or password doesn't match any accounts stored in the database. When a cookie is edited or removed that will mean the person shouldn't have access to the profile anymore since they aren't

logged in, resulting in a flash message saying you aren't logged in. When accessing another user's profile, you will be denied and a flash message will appear since only the user themselves can access their profile.

Magic ★★ ° ˙ ˚ ☽ ˚ ⌒ ˚ ★ ≡ ✧ ⌯

- In app.py and authHandlers.py, we used flash to display messages for certain conditions. Flash takes in two parameters, a string for message and another string for category type of the message. In our code, for example, when the user registers a username that is already taken, we return a flash message of "Username taken!" and the second parameter is left blank since it will default to "info" for information. The html forms will receive the flash message with the get_flashed_messages() function. This function will contain a list of all the flash messages and we will loop through in the html using jinja to add the messages into the html to have it be able to display.
- https://github.com/pallets/flask/blob/main/src/flask/helpers.py
    - Line 365 describes the flash function that allows us to send the message to the html form and have it be "flashed" on the page.
    - Line 397 describes the get_flashed_messages function that allows us to get the message and with jinja we can add it to the html and it will display on the page when loaded.