FLASK

Flask is a web framework that provides us with many tools, libraries and technologies that allow us to build web applications.

What does this technology (library/framework/service) accomplish for you?

We use many functions Flask provides to give us the functionality we need to build our web app.

These include:

- Routing, Requests and Responses
- Templating
- Session Information

How does this technology accomplish what it does?

Routing, Requests and Responses:

Flask accomplishes routing by using the @app.route() decorator

From the TCP socket connection, Flask receives TCP packets from the client. The server (Flask) receives the data from the packet, which contains an HTTP request. Flask will then parse the headers of the request and, based on the path specified, call the correct route() decoration. This route() decoration will specify the possible request types as well as a function to run when a request for the specified path is made. The route() decoration will map the given path to the function given and run the function whenever a request for the path is made. This function will typically render a template and serve the HTTP response to the client. Flask hides many of the complexities of routing from the developer to make developing apps easier. Behind this, Flask is following the same process we did in our homeworks to manually handle HTTP requests.

Flask's implementation of routing can be found in the source code here (scaffold.py:407). Flask wraps Werkzeug and uses it to handle the details of WSGI (Web Server Gateway Interface) while providing more structure and patterns for defining powerful applications. Werkzeug contains all the functionality needed to do routing and Flask utilizes Werkzeug to handle routing and makes it easy for the developer building applications to use.

Werkzeug implementation to support routing that flask utilizes can be found in 3 files. The first of these files is called wsgi.py and can be found here: https://github.com/pallets/werkzeug/blob/master/src/werkzeug/wsgi.py. Requests are sent from the client to the server. Once these requests reach the server wsgi will then forward that request to the server.py web application. The server.py web application will then process that request forwarded from the wsgi and send a response back to the server which is then forwarded to the client. How are the client and server communicating with one another though? The client and

server are communicating by using the http protocol. The second file Werkzeug utilizes is http.py which has an implementation that handles the necessary parsing of http requests by the server needed to allow the server to communicate with the client via http. The source code for http.py can be found here https://github.com/pallets/werkzeug/blob/master/src/werkzeug/http.py. The third file werkzeug utilizes is routing.py which has an implementation that creates a map with rules on how to deal with certain urls. routing.py also receives urls forwarded from wsgi which it will parse to determine if the url matches a rule if it does then the server will do the defined action when that rule is matched. The source code for routing.py can be found here https://github.com/pallets/werkzeug/blob/master/src/werkzeug/routing.py

Werkzeug is licensed under a three clause BSD License found here: Werkzeug License

Templating:

Flask accomplishes templating using the Jinja templating language. All Flask templates must be located within the /templates/ directory of the project to be usable with Flask. To render a template with Flask, render_template (template.py:130) must be called with an appropriate .html template file in the /templates/ directory and any data necessary for templating (e.g. variables to be inserted or iterated). This function is a wrapper function for the Jinja render (environment.py:1106) function. The render function will read the template, parse the template with the Parser (parser.py:36) class and pass it into the CodeGenerator (compiler.py:227) which will run any necessary python code and insert to the template based on Jinja's built in templating language. The returned template from render_template is typically used to construct an HTTP response in the function mapped to a route() decoration (see Routing, Requests and Responses above).

Jinja is licensed under a three clause BSD License found here: Jinja License

Sessions

A Flask session is a modified dictionary data structure used by Flask to store information about the current session. Sessions are only available if the Flask secret key is set. For our purposes in this project, we used sessions to authenticate users after logging in. After logging in, we add the user's username to the session dictionary, much like setting an authentication cookie. When the user returns to the site and clicks the sign in button, the server checks the session to see if the username key is set. If it is, the user is logged in without having to input their login credentials again.

License:
Flask is licensed under a three clause BSD License. It basically means: do whatever you want with it as long as the copyright in Flask sticks around, the conditions are not modified and the disclaimer is present. The full text of the license can be found here: Flask License