Open-Source Report for TCP Connections

[Flask.requests]

General Information & Licensing

Code Repository	https://github.com/pallets/flask
License Type	BSD-3
License Description	• The BSD-3-Clause license applies to all files in the Flask repository and source distribution. This includes Flask's source code, the examples, and tests, as well as the documentation. • Redistribution and use in source and binary forms, with or without modification, are permitted provided that the conditions below 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.

Magic

When the Flask application handles a request, it creates a Request object based on the environment it received from the WSGI server. Because a worker (thread, process, or coroutine depending on the server) handles only one request at a time, the request data can be considered global to that worker during that request. Flask uses the term context local for this.

Flask automatically pushes a request context when handling a request. View functions, error handlers, and other functions that run during a request will have access to the request proxy, which points to the request object for the current request.

HTTP Header parser:

- The Flask requests library creates a Request object in: https://github.com/pallets/flask/blob/066a35dd322f689ec07d7c0e82b19eacadac3c6b/src/flask/globals.py#L78 .
- The Request object is initialized in: https://github.com/pallets/flask/blob/066a35dd322f689ec07d7c0e82b19eacadac3c6b/src/flask/wrappers.py#L15. The request object is used by default in Flask and it remembers the matched endpoint and view arguments.
- The flask Request class is a subclass of the werkzeug.wrappers.Request that can be found in: https://github.com/pallets/werkzeug/blob/3115aa6a6276939f5fd6efa46282e0256ff21f1a/src/werkzeug/wrappers/request.py#L29 .
- The Werkzeug Request class is a subclass of the sansio request class. This class represents an outgoing WSGI HTTP response with body, status, and headers.
- The sansio class can be found in: https://github.com/pallets/werkzeug/blob/3115aa6a6276939f5fd6efa46282e0256ff21f1a/src/werkzeug/sansio/response.py#L63 . This class uses the werkzeug.http file to parse the HTTP header. An example of it being used can be seen in: https://github.com/pallets/werkzeug/blob/3115aa6a6276939f5fd6efa46282e0256ff21f1a/src/werkzeug/sansio/response.py#L318 .

- The Werkeug http file is where the HTTP headers get parsed: https://github.com/pallets/werkzeug/blob/3115aa6a6276939f5fd6efa46282e0256ff21f1a/src/werkzeug/http.py

Route rule building:

- In our project we used the “route” method to define functionality when a certain route is called. Our first use of the method is seen in: https://github.com/rickyjorgensen2000/cse312/blob/48a62fd31c0b2430cfc3261191bcf589b7bc23b0/flaskr/main.py#L22
- This line calls the route function located in: https://github.com/pallets/flask/blob/066a35dd322f689ec07d7c0e82b19eacadac3c6b/src/flask/scaffold.py#L423 . The headers default to a GET request. This function will decorate a view function to register it with the given URL rule and options.
- It then calls a method called “add_url_rule” seen at : https://github.com/pallets/flask/blob/066a35dd322f689ec07d7c0e82b19eacadac3c6b/src/flask/scaffold.py#L455 . This function registers a rule for routing incoming requests and building URLs.