junxiandoc.readthedocs.io

5. Flask Request Object — MyDocuments 0.1.0 documentation

5-6 minutes

MyDocuments

5.1. Overview Example¶

```
from flask import request

@app.route('/login', methods=['POST',
'GET'])
def login():
    error = None

    if request.method == 'POST':
        if
valid_login(request.form['username'],

request.form['password']):
            return
log_the_user_in(request.form['username'])
        else:
            error = 'Invalid
```

```
username/password'

    # the code below is executed if the
request method
    # was GET or the credentials were
invalid
    return render_template('login.html',
error=error)
```

Response to Request:

- Plain text string
- response object
- redirect('<ohter-url>')
- abort(<error_code>)

dir(request)

```
"['__class__', '__delattr__', '__dict__',
'__dir__', '__doc__',
'__enter__', '__eq__', '__exit__',
'__format__', '__ge__',
'__getattribute__', '__gt__', '__hash__',
'__init__',
'__le__', '__lt__', '__module__', '__ne__',
'__new__',
'__reduce__', '__reduce_ex__', '__repr__',
'__setattr__',
'__sizeof__', '__str__', '__subclasshook__',
'__weakref__',
'_get_file_stream',
```

```
'_get_stream_for_parsing', '_is_old_module',
'_load_form_data', '_parse_content_type',
'accept_charsets', 'accept_encodings',
'accept_languages', 'accept_mimetypes',
'access_route',
'application', 'args', 'authorization',
'base_url', 'blueprint', 'cache_control',
'charset', 'close', 'content_encoding',
'content_length', 'content_md5',
'content_type',
'cookies', 'data', 'date',
'dict_storage_class',
'disable_data_descriptor',
'encoding_errors', 'endpoint', 'environ',
'files',
'form', 'form_data_parser_class',
'from_values', 'full_path',
'get_data', 'get_json', 'headers', 'host',
'host_url',
'if_match', 'if_modified_since',
'if_none_match', 'if_range',
'if_unmodified_since',
'input_stream', 'is_json',
'is_multiprocess', 'is_multithread',
'is_run_once', 'is_secure', 'is_xhr',
'json', 'list_storage_class',
'make_form_data_parser',
'max_content_length',
'max_form_memory_size', 'max_forwards',
```

```
'method', 'mimetype', 'mimetype_params',
'module', 'on_json_loading_failed',
'parameter_storage_class', 'path', 'pragma',
'query_string', 'range', 'referrer',
'remote_addr', 'remote_user',
'routing_exception', 'scheme',
'script_root',
'shallow', 'stream', 'trusted_hosts', 'url',
'url_charset', 'url_root', 'url_rule',
'user_agent', 'values', 'view_args',
'want_form_data_parsed']"
```

5.2. Get the args in the URL¶

To access parameters submitted in the URL (?key=value):

```
searchword = request.args.get('key', '')
```

5.3. Callback Functions¶

Decorator to define callback function:

@app.before_first_request # Only before the first request

@app.before_request # Before each request

@app.after_request # After each request if no exception

@app.teardown_request # After each request even there is exceptions

Global variable ‘g’ is valid in above functions. For example, we can use g.user to share login information among these functions.

The return value from a view function is automatically converted into a response object for you. If the return value is a string it's converted into a response object with the string as response body, an 200 OK error code and a text/html mimetype.

The logic that Flask applies to converting return values into response objects is as follows:

- If a response object of the correct type is returned it's directly returned from the view.
- If it's a string, a response object is created with that data and the default parameters.
- If a tuple is returned the items in the tuple can provide extra information. Such tuples have to be in the form (response, status, headers) where at least one item has to be in the tuple. The status value will override the status code and headers can be a list or dictionary of additional header values.
- If none of that works, Flask will assume the return value is a valid WSGI application and convert that into a response object.

  If you want to get hold of the resulting response object inside the view, you can use the make_response() function. Imagine you have a view like this:

```
@app.errorhandler(404)
def not_found(error):
    return render_template('error.html'),
404
```

You just need to wrap the return expression with make_response() and get the response object to modify it, then return it:

```
@app.errorhandler(404)
def not_found(error):
    resp =
make_response(render_template('error.html'),
404)
    resp.headers['X-Something'] = 'A value'
    return resp
```