## Wikipedia Python API Python Wrapper

There were 2 possible options of Wikipidia API wrappers on the web: [Wikipedia API](https://github.com/goldsmith/Wikipedia) в Github and the [Pywikibot](https://www.mediawiki.org/wiki/Manual:Pywikibot). However, neither of them was proper for the functionality this Project required.

The first one was lacking some important functionality (it could send only certain requests, for which the functions in the wrapper were created). Also, it used the cache wrapped function to save all the results and do not send requests multiple times. In our case, this functionality might use up to much memory – as the project would need to send large amounts of requests at a time.

The second option had all the Wikipedia API functionality in it, was mostly created for advanced Wikipedia users – ones, who need to edit pages and was too complicated to use.

Instead of using available modules, a new one was written especially for the requirements of this project. The Wikipedia API has lots of documentation and the requests and responses to it are intuitively understandable and easy to use, so the module should not be complicated and should be able to perform all of its tasks as good as other modules mentioned previously.

To simplify the code and extend its abilities, the Wikipedia API Python module is using the requests module (Which can be installed with pip install requests and then included in the code with import requests)

A short description of requests module functionality:

Main functions include requests.get and requests.post. The first parameter is the the url, where we want to make our request. Optional parameters include params (Object) for uri parameters and headers (Object) for request headers. Another parameter is timeout=number (in seconds) and verify=bool (to check whether request is ready before timeout time runs out). The result of the function execution is an instance of module’s Response object. Specific parts of response could be accessed with the use of response.status\_code, response.headers, response.encoding, response.text, response.json().

So the main request to the Wikipedia API would look like this:

response = requests.get(WIKI\_URL, params={ … }, header={ … }, timeout=REQUEST\_TIMEOUT)

Here the WIKI\_URL is ‘http://en.wikipeda.org/w/api.php’.

The header should contain the a user\_header attribute with the information about the project and possibly some contact information. The Wikipedia API guide gives the following example:

‘MyCoolTool/1.1 (https://example.org/MyCoolTool/; MyCoolTool@example.org) BasedOnSuperLib/1’

(More detailed information on [User-Agent policy](https://meta.wikimedia.org/wiki/User-Agent_policy) page)

The url and header are parameters, that do not get changed too often. So we can save their values as global variables in our module. Then we can define functions in module to modify these – the set\_user\_agent(user\_agent) will just change the user\_agent variable to one specified as its parameter. Another – set\_lang(lang) will change the language in the WIKI\_URL variable.

The params argument is a dict with all the parameters, as defined in the Wikipedia API Functionality Document.

Another important argument – timeout could be set and changed in multiple ways and is not that important for functionality.

Now, that we created the functionality to send requests, the module needs to handle some of the common errors, that can appear during the request and response.

The default Wikipedia reaction to an invalid request (for example server overload or request timeout) is to add the “error” attribute to the “query” object (as described in Wikipedia API functionality document), so we can add an exception for this:

|  |
| --- |
| if 'error' in response:  if response['error']['info'] in ('HTTP request timed out.', 'Pool queue is full'):  raise Exception('Request timed out. Try again later') |

The errors mentioned here are the most common ones. But others might still happen, so we can also add a statement to raise these:

|  |
| --- |
| else:  raise Exception('Unknown error occured') |

So now the function or piece of code, that runs this function will need to take care of the following possible errors:

Exception ('Request timed out. Try again later' or ‘Unknown error occurred)

RequestException, ConnectionError, ReadTimeout … - all the typical requests module Exceptions

In order to stop the user from entering invalid parameters in the request we could also add a statement to check whether parameters are one of the allowed ones and raise an Exception if not:

|  |
| --- |
| for key in possible\_params:  if (key in params) and (params[key] not in possible\_params[key]):  raise ValueError(‘You specified the wrong parameters in the url’) |

It might be useful not to handle this error in the code, but to display it to user, so he could fix the code instead.

Finally, we can add a few more functions to make our further code shorter and more simplified.