bottle-tools Documentation

Release 2019.12.22.rc1

Arjoonn Sharma

Dec 22, 2019

Contents

Bottle is a wonderfully small framework for writing web based interfaces. It's size however means that if you mean to use it for a lot of things, some things become boring quickly. This project aims to provide techniques / code which will make those tedious tasks easier.

If you have a feature in mind, do let me know. Open a request on the Issue Tracker.

CHAPTER 1

Functions

bottle_tools.**add_cors**(*app*, *allow_credentials=True*, *origin=None*)

Automatically adds CORS routes to an app instance. This function must be called after **ALL** routes have been registered to the app. This does not add OPTIONS to those routes which already have an OPTIONS method registered.

```
# add your routes however you want
# in the END, add the next line
app = add_cors(app)
# then continue to run your app or whatever you wanted to do
app.run()
```

bottle_tools.**fill_args**(*function=None*, *, *json_only=False*, *coerce_types=False*)

Use to populate function arguments from json/query string/post data provided in API call.

If supplied, the json_only argument ensures that a POST api call only ever works under the content_type of *application/json*.

Some example usages:

```
@app.post('/user')
@fill_args
def change_user_data(name, age):
    # Do something with name and age

@app.post('/user')
@fill_args(json_only=True)
def change_user_data(name, age):
    # Do something with name and age

@app.get('/search/<folder>')
@fill_args
def search_for_string(folder, query):
    # folder is from the URL and query can be passed as params
    # Do something with query
```

If you provide simple type annotations the decorator will ensure those types. For example

```
@app.post('/calculate')
@fill_args
def fancy_calculation(a: int, b: float):
    return {'result': a + b}
```

This bit of code raises an error if *a* is not supplied as an integer or if *b* is not given as a float. Complex types like those described in the Typing module in Python docs are not yet supported. If you would like them to be added, please open up an issue.

Some arguments which need to be present throughout your application may be provided via the *common_kwargs* dictionary. For example, you might need your ORM's table throughout your routes.

```
import bottle_tools as bt

bt.common_kwargs.update({"UserTable": UserTable})

@app.post('/login')
@bt.fill_args
def login_function(uname: str, pwd: str, UserTable):
    user = UserTable.get_or_none(uname=uname)
    if user is None:
        raise bt.abort(404, 'Not found')
    if not user.check_password(pwd):
        raise bt.abort(404, 'Not found')
    user.new_session()
    ...
```

bottle_tools.**prefix_docs**(*app*)

Automatically prefixes docstrings of functions with the method and route in the decorator. Use this just like *add_cors* but **BEFORE** you register any routes. For example

```
app = bottle.Bottle()
app = prefix_docs(app)

@app.get('/some')
def my_fn():
    "This function returns some"
    return 'some'
```

When this code is used in Sphinx or via *help(my_fn)*, the docstring being processed is the following

```
**GET** */some*

This function returns some
```

This information changes as per the code. So if you register more than one url to the same function, it will reflect that in the code.

```
@app.get('/search')
@app.get('/')
def my_search():
    "Perform some fancy search"
    return 'found it!'
```

The docstring looks like:

```
**GET** */some*
**GET** */*

Perform some fancy search
```

CHAPTER 2

Plugins

class bottle_tools.plugins.**ReqResp**(*pass_request=True*, *pass_response=True*)

Pass Request and Response objects explicitly to the route function as the first two arguments.

```
from bottle_tools.plugins import ReqResp
app = Bottle()
app.install(ReqResp())

@app.get('/')
def function(request, response):
    pass
```

This will make migration easier in the future when Bottle moves to a similar function signature.

api = 2

apply(*callback*, *route*)

name = 'reqresp'

setup(*app*)

CHAPTER 3

CLI Tools

```
usage: bottle_tools [-h] [-t]
```

3.1 Named Arguments

-t, --template Add a standard django like folder/file layout

Default: False

Python Module Index

b

Index