BookStack

Get / Post / Delete options

works roughly like this:

```
# Import all markdown files into a book
find . -name '*.md' -print0 | while IFS= read -r -d '' file; do
    python bookstack-api.py markdown_inport "$file"
done

# Trash all pages containing a search query:
python bookstack-api.py search_delete "encryption_cipher_text"
```

Note on import: It does not handle any images, tags. The page title is the filename

```
#!/usr/bin/env python3

import os
import sys
import requests

# This is where BookStack API details can be hard-coded if you prefer
# to write them in this script instead of using environment variables.
default_bookstack_options = {
    "url": 'https://brain.vandragt.com',
    "token_id": 'n0N3L...rDvED',
    "token_secret": 'jshjIdU...TeWqCtZX',
    "book_id": 17,
}



# Gather the BookStack API options either from the hard-coded details above otherwise
# it defaults back to environment variables.
def gather_api_options() -> dict:
    return {
```

```
        "url": default_bookstack_options["url"] or os.getenv("BS_URL"),
        "token_id": default_bookstack_options["token_id"] or os.getenv("BS_TOKEN_ID"),
        "token_secret": default_bookstack_options["token_secret"] or
os.getenv("BS_TOKEN_SECRET"),
        "book_id": default_bookstack_options["book_id"] or os.getenv("BS_BOOK_ID"),
    }


# Send a multipart post request to BookStack, at the given endpoint with the given data.
def bookstack_delete(endpoint: str) -> dict:
    # Fetch the API-specific options
    bs_api_opts = gather_api_options()

    # Format the request URL and the authorization header, so we can access the API
    request_url = bs_api_opts["url"].rstrip("/") + "/api/" + endpoint.lstrip("/")
    request_headers = {
        "Authorization": "Token {}:{}".format(bs_api_opts["token_id"],
bs_api_opts["token_secret"])
    }

    # Make the request to bookstack with the gathered details
    response = requests.delete(request_url, headers=request_headers)

    # Throw an error if the request was not successful
    response.raise_for_status()

    # Return the response data decoded from it's JSON format
    return

# Send a get request to BookStack, at the given endpoint with the given data.
def bookstack_get(endpoint: str, data: dict) -> dict:
    # Fetch the API-specific options
    bs_api_opts = gather_api_options()

    # Format the request URL and the authorization header, so we can access the API
    request_url = bs_api_opts["url"].rstrip("/") + "/api/" + endpoint.lstrip("/")
    request_headers = {
        "Authorization": "Token {}:{}".format(bs_api_opts["token_id"],
bs_api_opts["token_secret"])
    }
```

```
    # Make the request to bookstack with the gathered details
    response = requests.get(request_url, headers=request_headers, params=data)

    # Throw an error if the request was not successful
    response.raise_for_status()

    # Return the response data decoded from it's JSON format
    return response.json()

# Send a post request to BookStack, at the given endpoint with the given data.
def bookstack_post(endpoint: str, data: dict) -> dict:
    # Fetch the API-specific options
    bs_api_opts = gather_api_options()

    # Format the request URL and the authorization header, so we can access the API
    request_url = bs_api_opts["url"].rstrip("/") + "/api/" + endpoint.lstrip("/")
    request_headers = {
        "Authorization": "Token {}:{}".format(bs_api_opts["token_id"],
bs_api_opts["token_secret"])
    }

    # Make the request to bookstack with the gathered details
    response = requests.post(request_url, headers=request_headers, data=data)

    # Throw an error if the request was not successful
    response.raise_for_status()

    # Return the response data decoded from it's JSON format
    return response.json()



# Error out and exit the app
def error_out(message: str):
    print(message)
    exit(1)



def search_delete():
    if len(sys.argv) < 3:
        error_out("search_delete <query> arguments need to be provided")
```

```
    get_data = {
        "query": sys.argv[2],
        "count": 100
    }

    # Send the upload request and get back the attachment data
    try:
        json = bookstack_get("/search", get_data)
    except requests.HTTPError as e:
        error_out("Upload failed with status {} and data: {}".format(e.response.status_code,
e.response.text))

    for page in json['data']:
        try:
            bookstack_delete("/pages/{}".format(page['id']))
        except requests.HTTPError as e:
            error_out("Upload failed with status {} and data:
{}".format(e.response.status_code, e.response.text))

        print("deleted: {}".format(page['name']))


def import_markdown():
    if len(sys.argv) < 3:
        error_out("import_markdown <file_path> arguments need to be provided")

    # Gather details from the command line arguments and create a file name
    # from the file path
    file_path = sys.argv[2]
    file_name = os.path.basename(file_path)
    book_id = bs_api_opts["book_id"]

    # Ensure the file exists
    if not os.path.isfile(file_path):
        error_out("Could not find provided file: {}".format(file_path))

    with open(file_path, 'r') as file:
        markdown_content = file.read()

    # Gather the data we'll be sending to BookStack.
    # The format matches that what the "requests" library expects
```

```
    # to be provided for its "files" parameter.
    post_data = {
        "name": file_name,
        "markdown": markdown_content,
        "book_id": book_id
    }

    # Send the upload request and get back the attachment data
    try:
        page = bookstack_post("/pages", post_data)
    except requests.HTTPError as e:
        error_out("Upload failed with status {} and data: {}".format(e.response.status_code,
e.response.text))

    # Output the results
    print("File successfully uploaded to book {}.".format(book_id))
    print(" - page ID: {}".format(page['id']))
    print(" - page Name: {}".format(page['name']))



# Run this when called on command line
if __name__ == '__main__':
    # Check arguments provided
    if len(sys.argv) < 2:
        error_out("<callback> arguments need to be provided")

    bs_api_opts = gather_api_options()
    callback = sys.argv[1]

    # callback
    possibles = globals().copy()
    possibles.update(locals())
    method = possibles.get(callback)
    if not method:
         raise NotImplementedError("Method %s not implemented" % method_name)
    method()
```

Created 2023-04-27 06:56:59 UTC by Sander
Updated 2023-04-27 07:03:13 UTC by Sander