
self.wiki Documentation

Release 0.7.1

Paul Ollivier

Dec 20, 2018

Contents:

| | | |
|----------|---|-----------|
| 1 | Important security note: | 3 |
| 2 | Installation | 5 |
| 3 | Configuration | 7 |
| 4 | Usage | 9 |
| 4.1 | Keyboard shortcuts | 9 |
| 4.2 | Todos | 9 |
| 4.3 | Git integration | 10 |
| 5 | Advanced usage | 11 |
| 6 | Special thanks | 13 |
| 7 | Contributing to this project | 15 |
| 7.1 | Ways you can contribute | 15 |
| 7.2 | Contributing bug reports and feature requests | 15 |
| 7.3 | Contributing code, tests | 16 |
| 8 | API Documentation | 17 |
| 8.1 | self_wiki | 17 |
| 8.2 | self_wiki.wiki | 17 |
| 8.3 | self_wiki.todo | 19 |
| 8.4 | self_wiki.utils | 19 |
| 9 | Indices and tables | 21 |
| | Python Module Index | 23 |

[self.wiki](#) is a wiki and todo manager. I wanted to be able to write notes, documentation, and tasks from a simple (understand: minimal) interface, using mostly keyboard shortcuts.

Here's what my feature list draft looked like:

- [x] Create wikis directly from URL (`ctrl+l`, then type stuff, on most browsers)
 - [x] On any URLs. i should not be restricted to naming stuff (restricted names are `/todo`, `/**/edit`, `/**/edit/save`, `/**/edit/delete`)
- [x] Wikis should be more or less standard extended markdown
- [x] Wikis should be stored on the filesystem **as-is**, no database or stuff like that. I should be able to read them using `less` when i want to.
- [x] Should not make calls to the outside world, except on linked stuff.
- [x] At least todo operations should have keyboard shortcuts
 - [x] add
 - [] delete
 - [] update status
- [x] I should be able to navigate links without using weird browsers like [uzbl](#)

That's more or less it.

I looked at several other solutions, including [tiddlywiki](#), but it was too mouse-based for my tastes.

CHAPTER 1

Important security note:

`self.wiki` should *not* be publicly accessible! Any potential mean-inclined person could steal valuable secrets from your computer via this application! That is because we allow a potential attacker to request files outside of `self.wiki`'s `CONTENT_ROOT`.

If you don't specify a `--host` argument, `self.wiki` will listen only on the local computer, and should therefore be safe to use.

CHAPTER 2

Installation

Install using pip, using the master branch, or by picking a release in the [releases](#) tab:

```
pip install https://vit.am/gitea/paulollivier/self-dot-wiki/archive/master.tar.gz
```

Then, simply run the included script:

```
$ self.wiki --help
usage: self.wiki [-h] [--debug] [--host HOST] [-p PORT]

optional arguments:
  -h, --help            show this help message and exit
  --debug               Turns on debug mode
  --host HOST           address to bind on
  -p PORT, --port PORT  Port to listen on
```


CHAPTER 3

Configuration

Configuration is kept to a minimum, and uses environment variables to achieve its goals.

| Environment name | variable | default | note |
|------------------------|----------|--------------------------------------|---|
| SELF_WIKI_CONTENT_ROOT | | <code>./self.wiki</code> | self.wiki will store its markdown files there. |
| SELF_WIKI_FAVICON_PATH | | <code>static/ favicon.ico</code> | Path to the favicon to use. Must be relative to the CONTENT_ROOT. |
| SELF_WIKI_TITLE_PREFIX | | <code>"self.wiki "</code> | Page <title> prefix. |

After having started `self.wiki`, go to your navigator, open up <http://localhost:4000/>. Help should be available at <http://localhost:4000/help>.

If a page is not available, you will be redirected to its edit page, which is simply `/path/to/page/edit`.

4.1 Keyboard shortcuts

We make heavy use of `accesskeys` to navigate the page. In fact, `self.wiki` autogenerates those on every link present on any page.

On firefox, you can activate these keys by pressing `alt+shift+key`.

There are also some keyboard shortcuts available on a more general manner.

| Keys | Context | Effect |
|--------------------------|---------|---|
| <code>ctrl+c n</code> | any | create a new todo item |
| <code>ctrl+c d</code> | view | delete current page |
| <code>alt+shift+f</code> | edit | send a file, sibling to the current edited file |
| <code>alt+shift+s</code> | edit | save current edited file |

4.2 Todos

To create a todo item, use the keyboard shortcut (please see above). You will be prompted for a text that will be shown.

To mark a todo item as done (but not remove it completely), click on its text. The text will be striked, representing completion.

To delete a todo item, click on its `del` button.

NOTE: if a todo item is deleted, when also marked as done, we will write this item to a special page, `/journal/year/month/day.md`.

4.3 Git integration

If a `.git` repository is present at the root of the `SELF_WIKI_CONTENT_ROOT`, `self.wiki` will try to commit changes.

Please note that they won't be pushed or pulled to a remote repository! I might add it in the future

CHAPTER 5

Advanced usage

Instead of running the included `self.wiki` script, you may use any WSGI-compatible server. This will increase the performance of loading the pages.

For instance, using `gunicorn`:

```
gunicorn -b localhost:4000 self_wiki:app
```

I have yet to run benchmarks to measure the real-world improvements.

CHAPTER 6

Special thanks

This project uses many open-source libraries:

- flask
- pymarkdown
- milligram
- mousetrap.js
- sphinx

Special thanks to those.

Contributing to this project

Being Open-Source, this project welcomes ideas and changes. However, some rules are in place:

1. Please be nice. The maintainers don't have time for rude people.
2. Before submitting your issue, search if a similar issue has not been raised. If yes, please add your comment to the existing issue instead
3. Don't be rude.
4. That's it. I just like lists.

7.1 Ways you can contribute

You can:

- Submit bugs, feature requests or other
- Contribute code, in the form of Pull Requests
- Just tell the maintainers that this project is helpfull. It *will* be appreciated.

7.2 Contributing bug reports and feature requests

When writing bug reports, please remember to include as much data as possible with your request. That will help tremendously.

When writing feature requests, please specify your use case, and what behaviour you expect. If you can, writing unit tests matching your feature request is a huge help.

7.3 Contributing code, tests

7.3.1 Architecture

The main code is in `self_wiki`, the tests are in `tests`, and the documentation resides in `docs`.

7.3.2 Environment

You may use the provided [Pipfile] to manage the environment for this project. If you add or remove packages, please remember to commit the resulting `Pipfile.lock`.

When writing code in this project, remember to add tests! We use `py.test` as test runner, and the whole suite is run against multiple versions of python, via `tox`.

7.3.3 Misc.

If you are adding features, remember to update the configuration accordingly.

Should you lack the skills to contribute, we will be happy to help.

8.1 self_wiki

self_wiki is an opinionated Wiki engine & task manager.

8.2 self_wiki.wiki

Contains wiki-related stuff.

For instance, `:py:class:Page` may be used to manipulate `.md` files on disk.

class `self_wiki.wiki.Page` (*path*, *root=""*, *level=0*)
Container for a markdown file.

Basically, all manipulation on `.md` files should go via this

load()

Load the markdown data from disk.

Also sets object properties according to filesystem state.

path

Return the full path to the markdown document.

Return type `str`

relpath

Return the page's path, relative to the configured content root.

Return type `str`

render()

Render the markdown to HTML, using the object's converter.

Return type `str`

save()

Persist the Page object on disk and update the recent files list.

note: this method does not update a RecentFileManager object!

title

Return the title of the page.

This is computed either from the markdown's metadata ('Title:' as one of the pages' header), or the first level 1 header, or the pages' path

Return type str

class self_wiki.wiki.RecentFileManager(*root, wanted_extensions=None*)

Represents a collection of files, with their age attached.

delete(*path*)

Delete :param path: from the recent files.

Parameters path (str) – The exact path we should forget.

get(*limit=None*)

Return up to *limit* recent items.

Return type List[Dict[str, Union[str, int]]]

classmethod get_recent_files(*directory, limit=20, wanted_extensions=None*)

Return the list of recent files.

This list is sorted by modification time as a UNIX timestamp (recent first), with an optional *limit*.

Return type List

Parameters

- **directory** (str) – Base directory for the search
- **limit** (int) – number of results to return
- **wanted_extensions** (Optional[List[str]]) – A list of file extensions we want.

If None, ['md'] is used. :return: a dictionary list with, where each dict has the following keys: path, mtime

re_scan(*limit=None, wanted_extensions=None*)

Re-scan the defined content root.

Parameters wanted_extensions (Optional[List[str]]) – a list of file extensions we want to include.

Specifying [''] will include everything. Defaults to ['md'] :type limit: Optional[int] :param limit: limit the number of results to this :return:

root

Return the path we consider as root.

Return type str

update(*path*)

Update the recency of the file designated by :param path:.

Note that said file is not required to exist.

Parameters **path** (*str*) – path to the file, relative to RecentFileManager.root, or not.

8.3 self_wiki.todo

Models related to todos stuff.

class `self_wiki.todo.TODOList` (*serialization_path*)

A container for a collection of Todos.

from_json (*j*)

Insert an element from a dictionary object.

Tries to compensate for eventual missing id.

Parameters **j** (*dict*) – A dictionary containing at least a ‘text’ key

load ()

Load a serialized collection from disk.

save ()

Persist current collection on disk.

todos

Return the internal object list.

Why not rename self._todos to self.todos? No idea.

8.4 self_wiki.utils

Some useful classes and functions related to self.wiki.

`self_wiki.utils.write_todo_to_journal` (*basepath, todo*)

Write the object to a Page, denoted as journal in the URL.

The given item should have the following keys: ‘id’, ‘text’

CHAPTER 9

Indices and tables

- `genindex`
- `modindex`
- `search`

S

- `self_wiki`, [17](#)
- `self_wiki.todo`, [19](#)
- `self_wiki.utils`, [19](#)
- `self_wiki.wiki`, [17](#)

D

`delete()` (`self_wiki.wiki.RecentFileManager` method), 18

F

`from_json()` (`self_wiki.todo.TODOList` method), 19

G

`get()` (`self_wiki.wiki.RecentFileManager` method), 18

`get_recent_files()` (`self_wiki.wiki.RecentFileManager` class method), 18

L

`load()` (`self_wiki.todo.TODOList` method), 19

`load()` (`self_wiki.wiki.Page` method), 17

P

`Page` (class in `self_wiki.wiki`), 17

`path` (`self_wiki.wiki.Page` attribute), 17

R

`re_scan()` (`self_wiki.wiki.RecentFileManager` method), 18

`RecentFileManager` (class in `self_wiki.wiki`), 18

`relpath` (`self_wiki.wiki.Page` attribute), 17

`render()` (`self_wiki.wiki.Page` method), 17

`root` (`self_wiki.wiki.RecentFileManager` attribute), 18

S

`save()` (`self_wiki.todo.TODOList` method), 19

`save()` (`self_wiki.wiki.Page` method), 17

`self_wiki` (module), 17

`self_wiki.todo` (module), 19

`self_wiki.utils` (module), 19

`self_wiki.wiki` (module), 17

T

`title` (`self_wiki.wiki.Page` attribute), 18

`TODOList` (class in `self_wiki.todo`), 19

`todos` (`self_wiki.todo.TODOList` attribute), 19

U

`update()` (`self_wiki.wiki.RecentFileManager` method), 18

W

`write_todo_to_journal()` (in module `self_wiki.utils`), 19