Python auto-generated documentation — 3 tools that will help document your project

Creating an up-to-date, meaningful, easily usable documentation is not trivial. This article shortly reviews 3 tools that could help automate the process. I focus only on Python tools that can be used for internal documentation.

Bartlomiej Skwira · Follow
Published in blueriders
10 min read · Jul 17, 2020

Listen Share

When someone asks if there's a documentation for this project
thib02 commited on 30 Dec 2018

Source https://thecodinglove.com/when-someone-asks-if-theres-a-documentation-for-this-project

I think we've all been there:

- didn't create any documentation for a project (the chances are pretty high that it was a startup ;))
- created documentation that went obsolete really fast
- created documentation that nobody needed
- ignored existing documentation — there was *some* documentation, but we didn't read it

Personally I'm guilty of all the "sins" above.

> "Let him who created a project with perfect documentation be the first to throw an exception at the server." — Bartek Skwira

Creating a good, usable, up-to-date documentation is not that easy. But is it worth to create documentation in the first place?

Pros of creating good documentation:

1. **Increases information exchange** between team members— this single reason is just so *powerful*!
2. **Decreases onboarding** time of new members

3. Helps to organize big projects (helps to **see the big picture**)

4. **Increases team member awareness** of how the whole project is organized

5. **Increases development speed** — finding information is faster and thus development is faster

6. **Promotes standards** and consistency

Cons of creating documentation?

1. **Requires time** (and money) — sometimes a project can't afford to spend time on documentation.

2. It's **hard to keep it up-to-date**, especially in startup projects with rapid changes

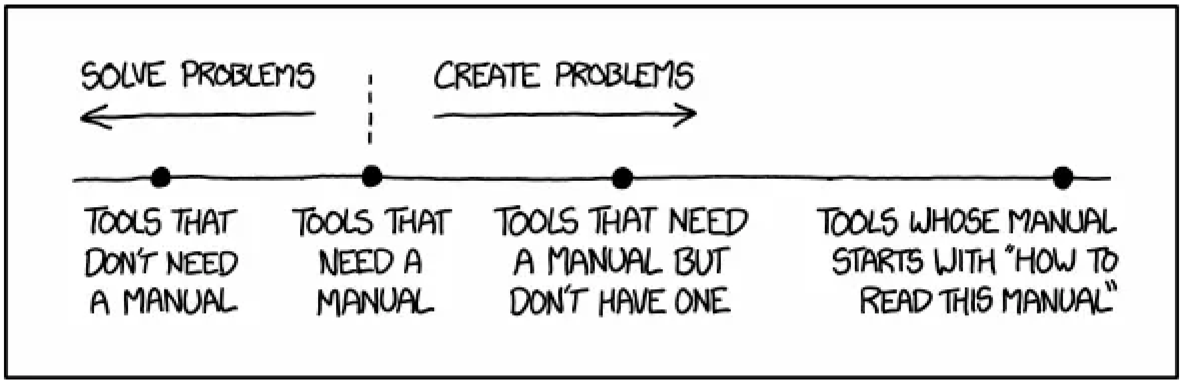
3. Creating documentation is **not a "pleasant" activity** for the developers (compared to creating code) — some developers don't like to create the documentation, they will be demotivated when asked to do it.

Cons of bad documentation?

1. "Out-of-date" documentation **can lead to misunderstandings** and slower development

2. Can get fragmented — it's hard to maintain one, consistent documentation.

Taking the pros and cons into account it would seem sensible to either create good, up-to-date documentation or not create it at all. But there are tools that can help and decrease the human factor. Autogenerated documentation tools require less effort from people and can create valuable documentation automatically.


SOLVE PROBLEMS
CREATE PROBLEMS
TOOLS THAT DON'T NEED A MANUAL
TOOLS THAT NEED A MANUAL
TOOLS THAT NEED A MANUAL BUT DON'T HAVE ONE
TOOLS WHOSE MANUAL STARTS WITH "HOW TO READ THIS MANUAL"

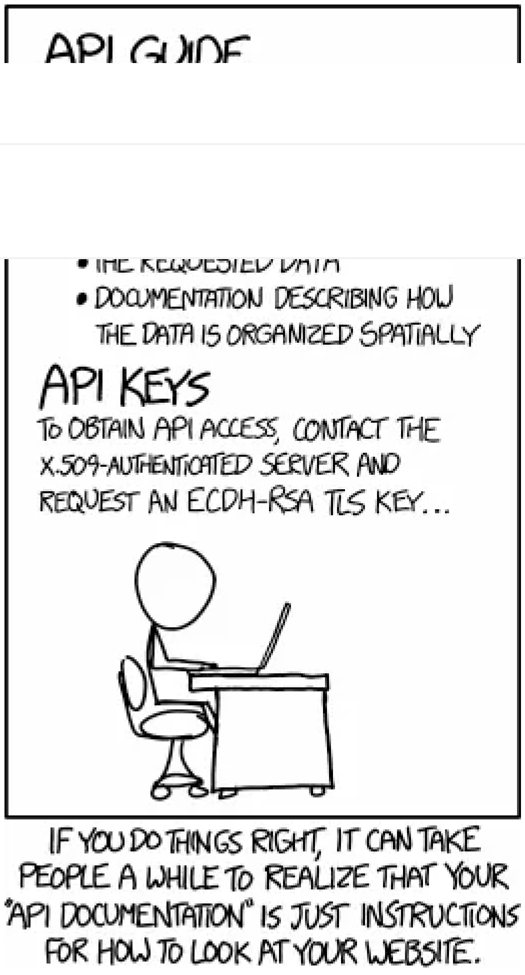
Source https://xkcd.com/1343/

2. Three types of autogenerated documentation:

By *autogenerated* I mean documentation which is built with some tool (maybe cli) and the output is instantly browsable (in a form of a web page, a pdf etc..). We can distinguish 3 types of these:

- fully automatic — no need to do anything manually
- semi-automatic — it works without doing anything manually, but it can be better when you put in some extra effort
- manual — you write the documentation by yourself, the output webstie/pdf is generated automatically

3. Swagger UI — the fully automatic solution


API GUIDE
THE REQUESTED DATA
• DOCUMENTATION DESCRIBING HOW THE DATA IS ORGANIZED SPATIALLY
API KEYS
TO OBTAIN API ACCESS, CONTACT THE X.509-AUTHENTICATED SERVER AND REQUEST AN ECDH-RSA TLS KEY...
IF YOU DO THINGS RIGHT, IT CAN TAKE PEOPLE A WHILE TO REALIZE THAT YOUR "API DOCUMENTATION" IS JUST INSTRUCTIONS FOR HOW TO LOOK AT YOUR WEBSITE.

Source: https://xkcd.com/1481/

Swagger delivers multiple tools:

- Swagger Editor — helps design an API
- Swagger UI — creates an API documentation
- Swagger Codegen — generates server stubs and client SDKs for an API

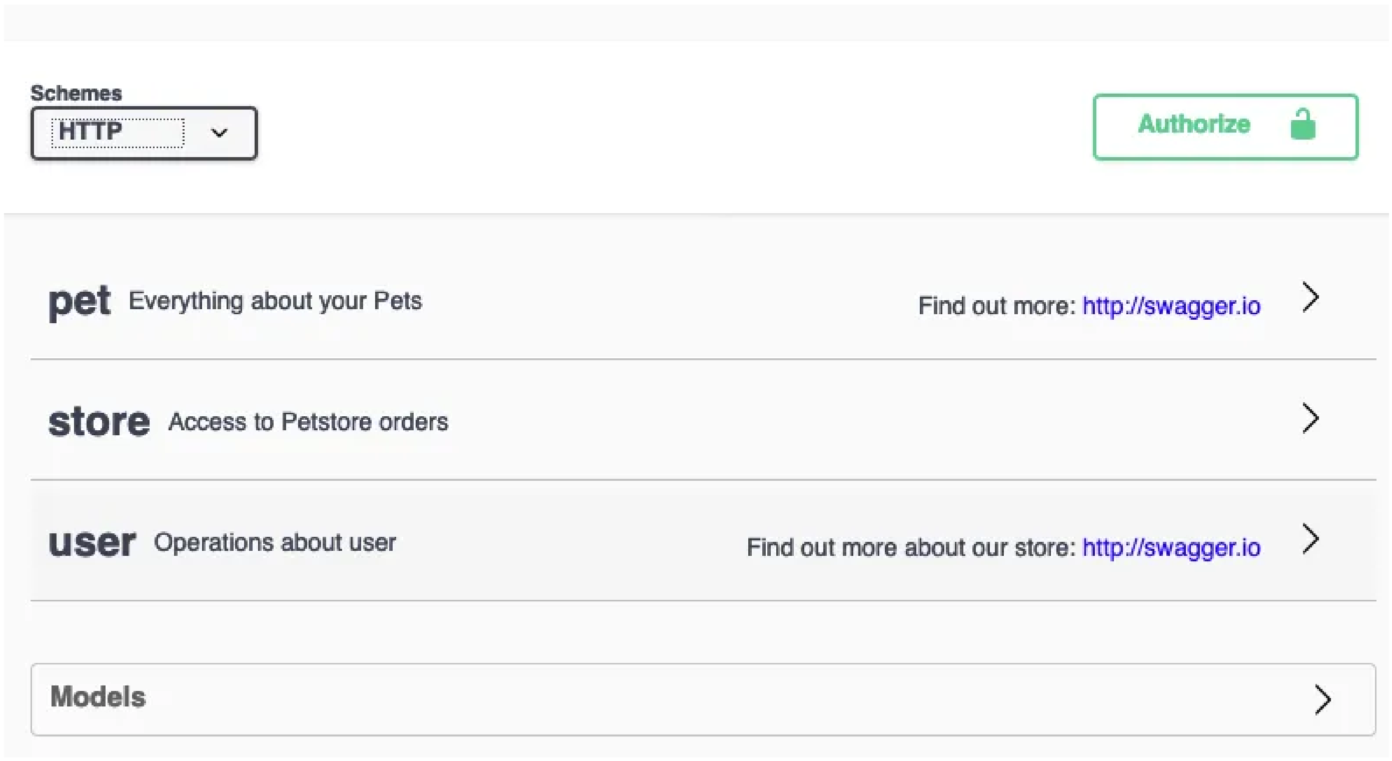
All are licensed under Apache 2.0 license.

I'll focus only on Swagger UI. What does it do?

> *Swagger UI allows anyone — be it your development team or your end consumers — to visualize and interact with the API's resources without having any of the implementation logic in place. It's automatically generated from your OpenAPI (formerly known as Swagger) Specification, with the visual documentation making it easy for back end implementation and client side consumption.*

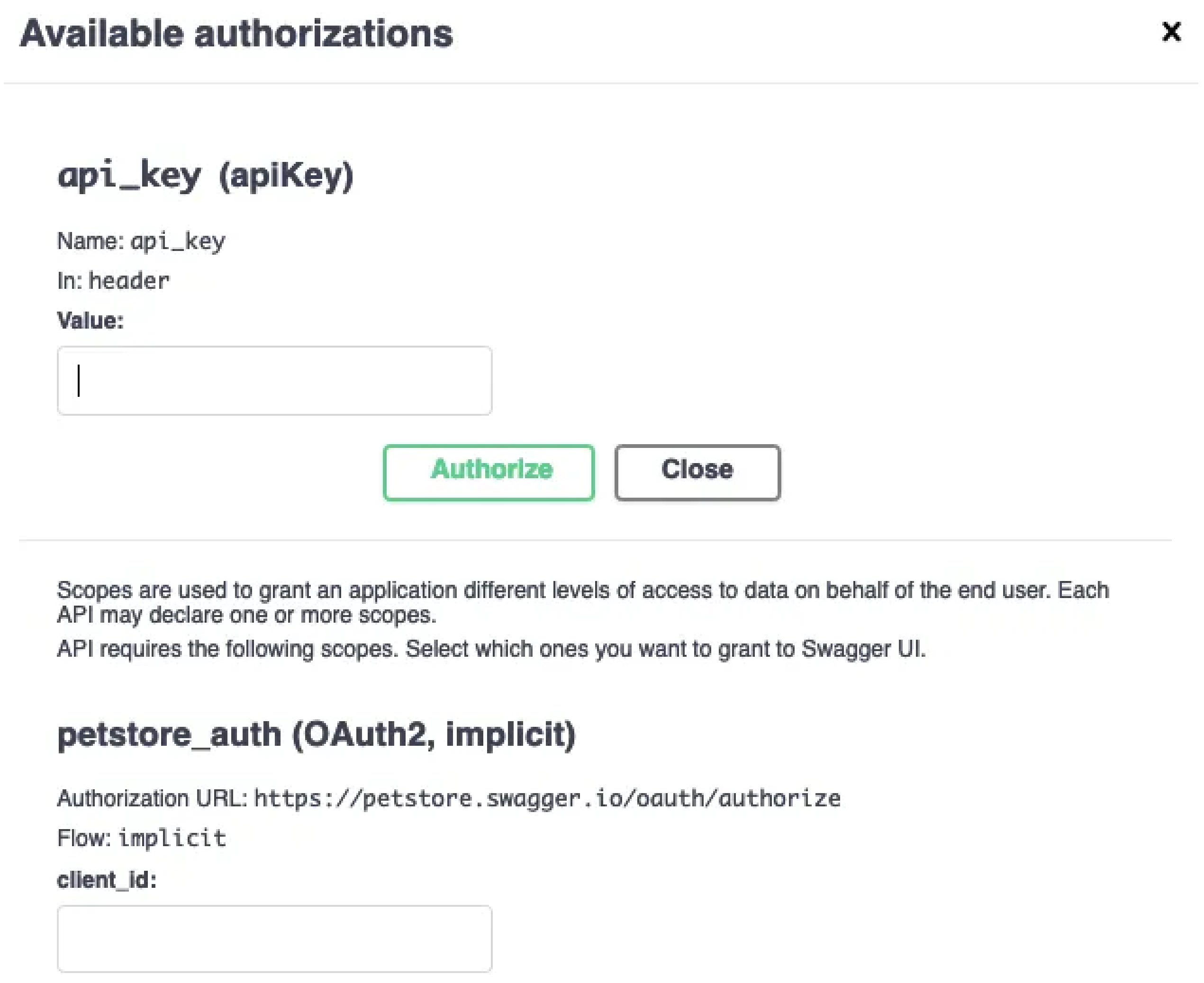
So it **automatically documents, visualizes and helps interact with an API. Try the Petstore demo page**

Overview of the API:


Schemes
HTTP
Authorize
pet Everything about your Pets
Find out more: http://swagger.io
store Access to Petstore orders
user Operations about user
Find out more about our store: http://swagger.io
Models

You can see what API namespaces/groups are defined (pet, store, user) and see if the API handles http/https schemes.

When you click the "Authorize" button:


Available authorizations

api_key (apiKey)

Name: api_key

In: header

Value:

Scopes are used to grant an application different levels of access to data on behalf of the end user. Each API may declare one or more scopes.

API requires the following scopes. Select which ones you want to grant to Swagger UI.

petstore_auth (OAuth2, implicit)

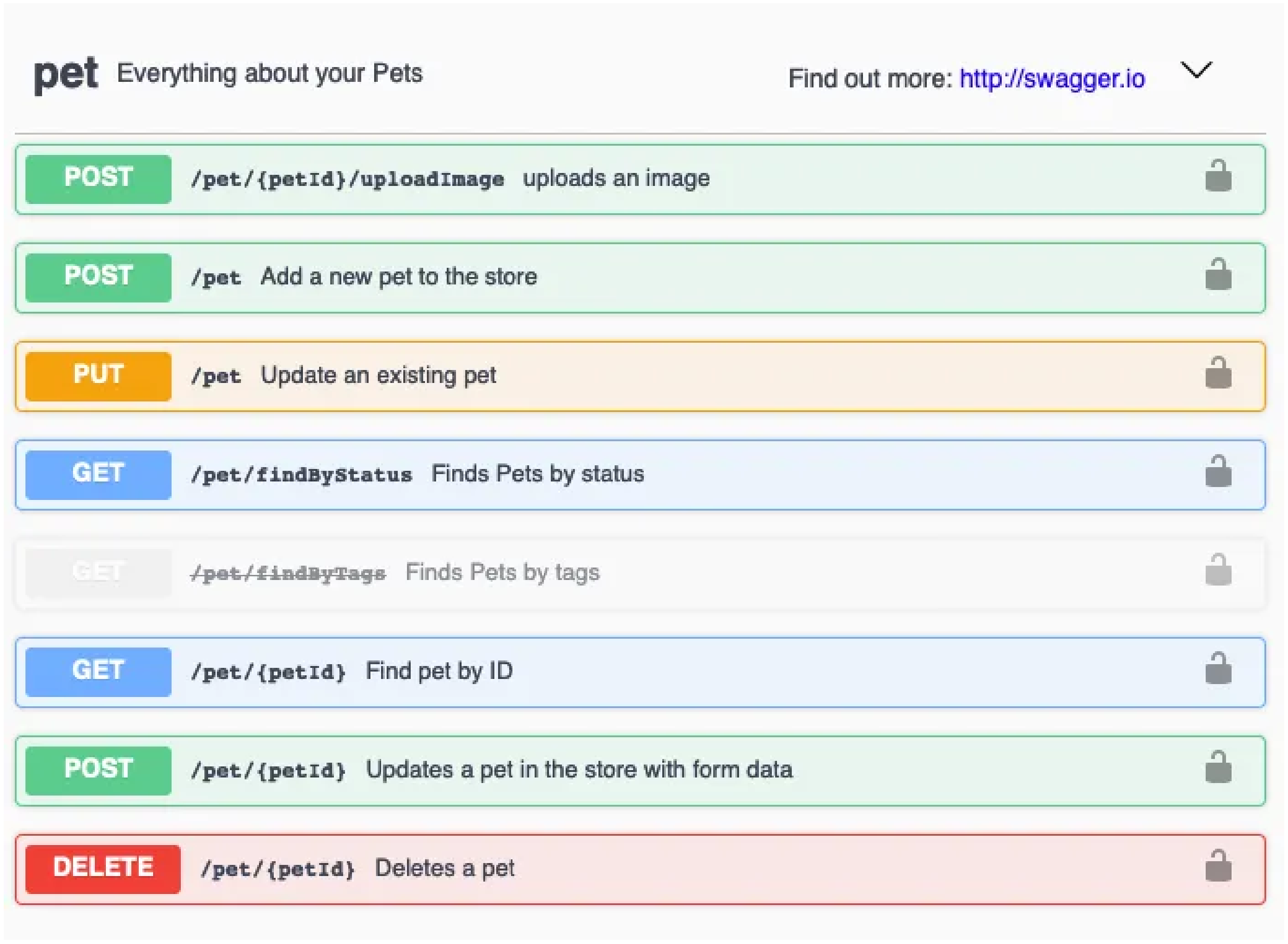
Authorization URL: https://petstore.swagger.io/oauth/authorize

Flow: implicit

client_id:

The **"Value" and "client_id" inputs are interactive — you can type a value and click "Authorize"** (there is no real authorization behind it, it's just a stub).

When you click on the group name ("pet"), you get a list of all the endpoints defined:


pet Everything about your Pets — Find out more: http://swagger.io

POST `/pet/{petId}/uploadImage` uploads an image

POST `/pet` Add a new pet to the store

PUT `/pet` Update an existing pet

GET `/pet/findByStatus` Finds Pets by status

GET ~~`/pet/findByTags`~~ Finds Pets by tags

GET `/pet/{petId}` Find pet by ID

POST `/pet/{petId}` Updates a pet in the store with form data

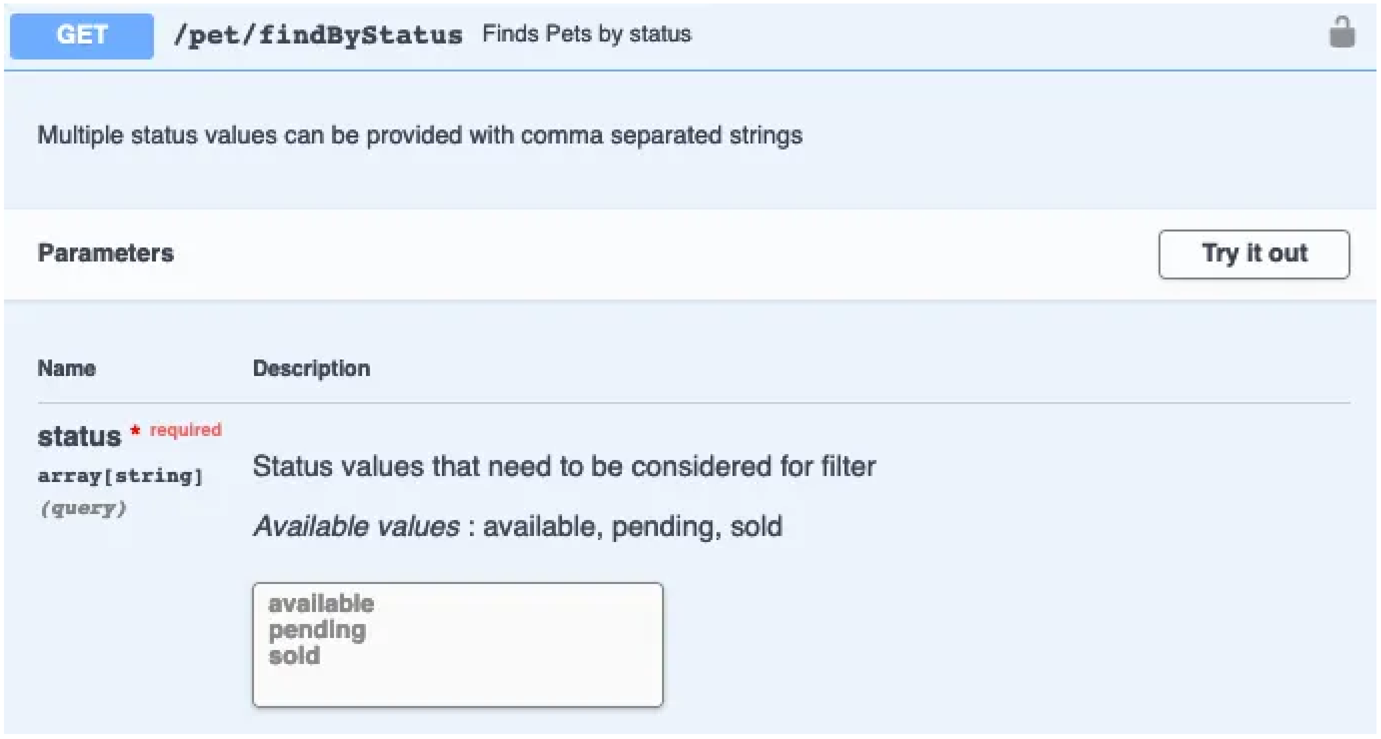
DELETE `/pet/{petId}` Deletes a pet

You can see the URL, HTTP verb and path parameters. The lock icon on the right side highlights authorization options.

The greyed-out and strikethrough endpoint is deprecated.

(* For SEO reasons I would recommend creating URLs with hyphens instead of camelCase)

Expanding a GET endpoint:



GET `/pet/findByStatus` Finds Pets by status

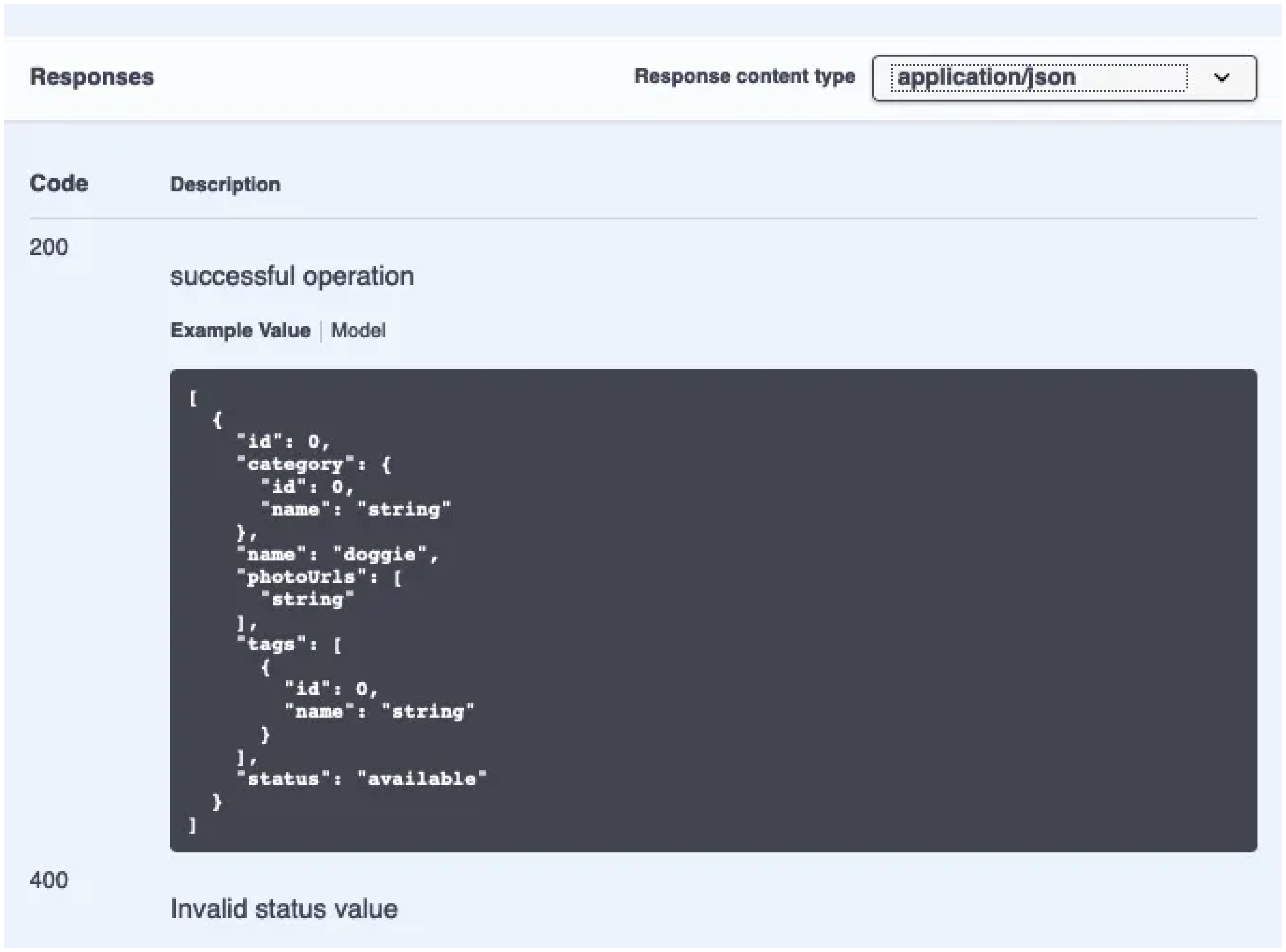
Multiple status values can be provided with comma separated strings

Parameters

Name	Description
status * required array[string] (query)	Status values that need to be considered for filter *Available values* : available, pending, sold

```
available
pending
sold
```

The “*status*” parameter is documented and has available values listed.



Responses — Response content type: application/json

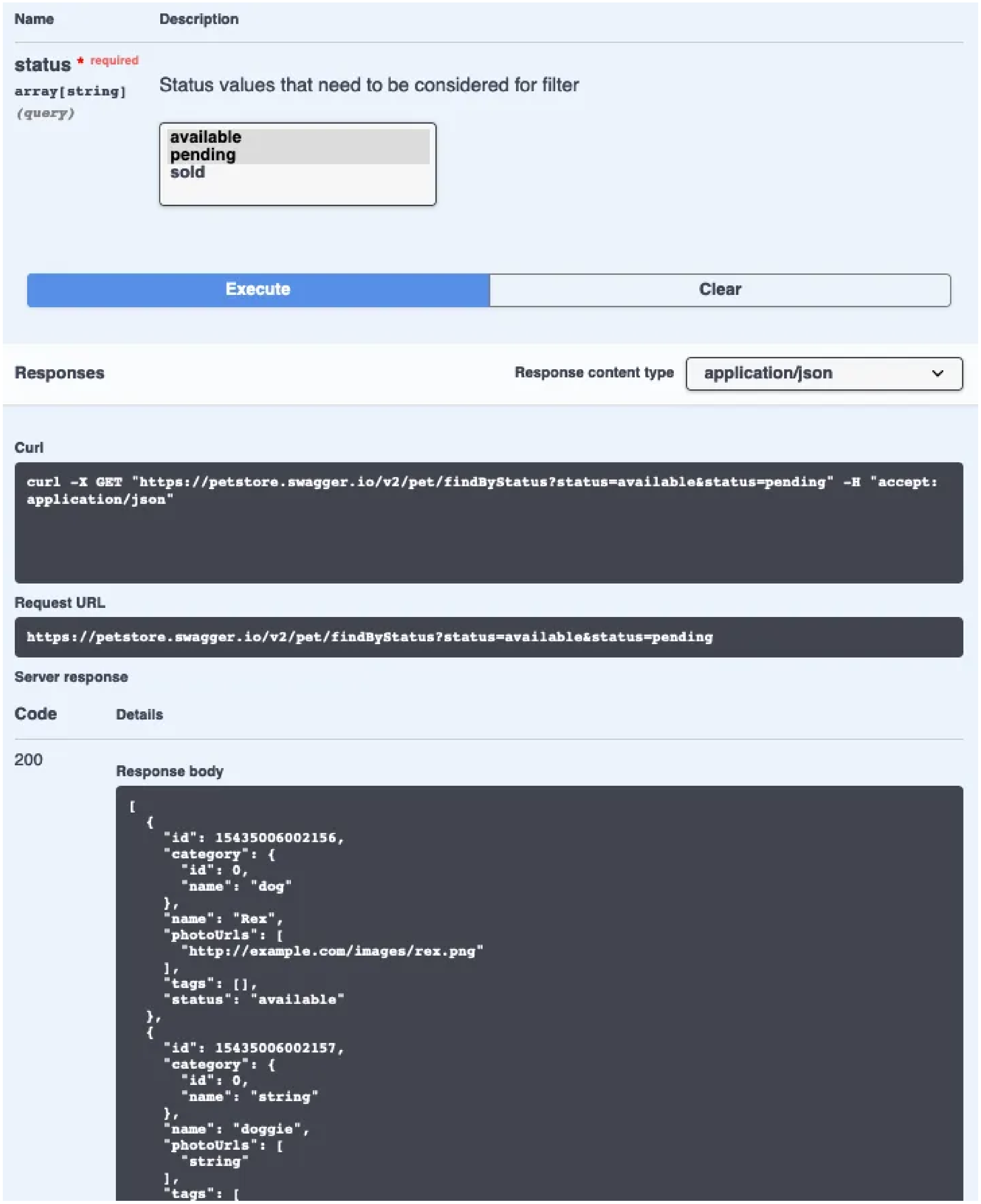
Code	Description
200	successful operation
400	Invalid status value

Example Value | Model

```
[
  {
    "id": 0,
    "category": {
      "id": 0,
      "name": "string"
    },
    "name": "doggie",
    "photoUrls": [
      "string"
    ],
    "tags": [
      {
        "id": 0,
        "name": "string"
      }
    ],
    "status": "available"
  }
]
```

The response lists possible HTTP codes (200,400) and an example JSON response.

Clicking *“Try it”* enables **interactive mode in which you can modify the parameter, execute the query and see results, that’s nice!**



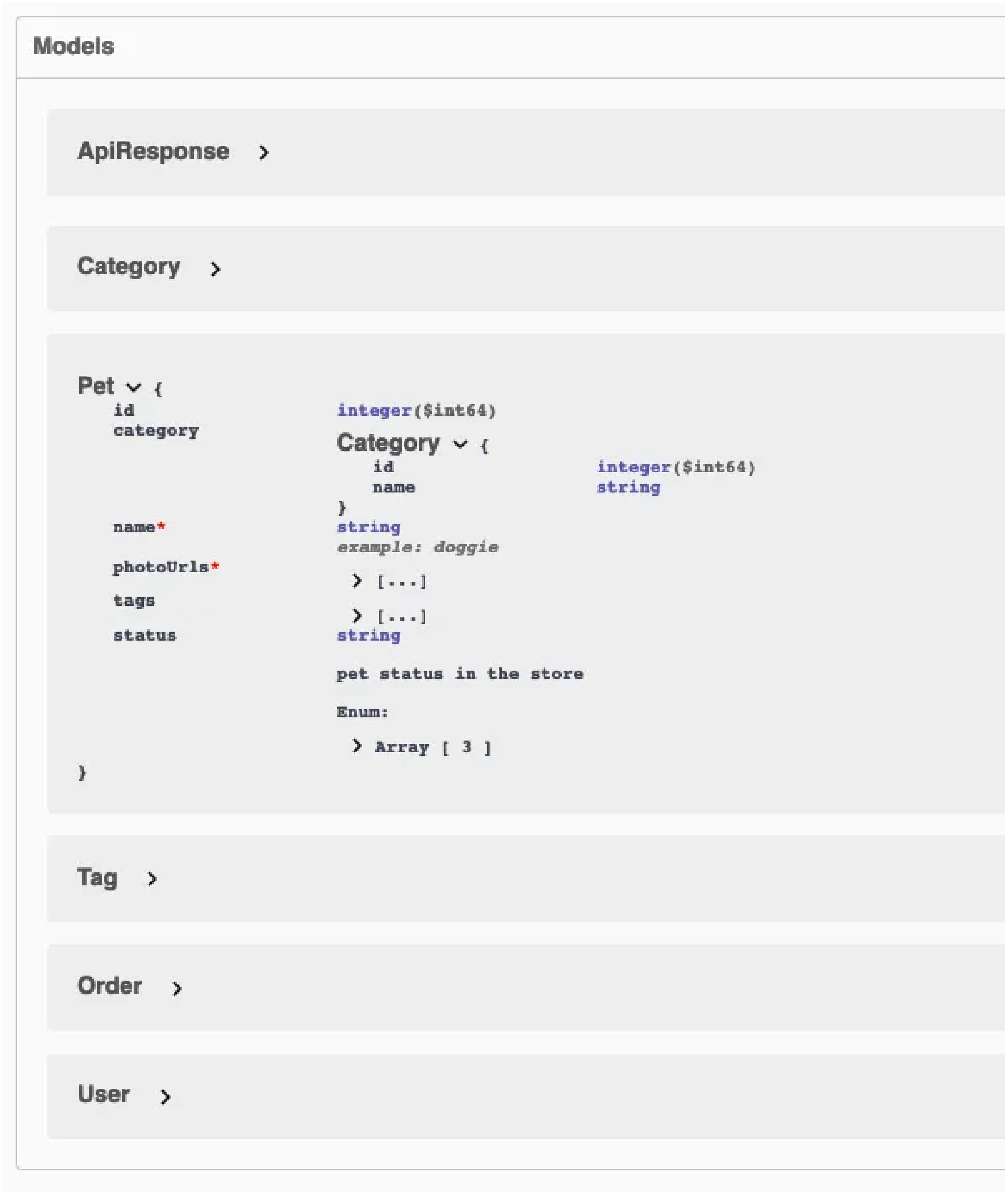
Status values that need to be considered for filter

```
curl -X GET "https://petstore.swagger.io/v2/pet/findByStatus?status=available&status=pending" -H "accept:
application/json"
```

```
https://petstore.swagger.io/v2/pet/findByStatus?status=available&status=pending
```

```
[
  {
    "id": 15435006002156,
    "category": {
      "id": 0,
      "name": "dog"
    },
    "name": "Rex",
    "photoUrls": [
      "http://example.com/images/rex.png"
    ],
    "tags": [],
    "status": "available"
  },
  {
    "id": 15435006002157,
    "category": {
      "id": 0,
      "name": "string"
    },
    "name": "doggie",
    "photoUrls": [
      "string"
    ],
    "tags": [
```

Models view — you can see the nesting and types:



```
Models

ApiResponse >

Category >

Pet v {
    id              integer($int64)
    category        Category v {
                        id              integer($int64)
                        name            string
                    }
    name*           string
                    example: doggie
    photoUrls*       > [...]
    tags             > [...]
    status          string

                    pet status in the store

                    Enum:
                     > Array [ 3 ]
}

Tag >

Order >

User >
```

Swagger creates this beautiful documentation automagically, but your **API needs to follow the OpenAPI Specification** (originally known as the Swagger Specification).

Installation:

The installation will vary depending on how you serve your API. If you are using a framework, then a ready-to-go library:

- Django REST Swagger
- Flask-RESTPlus
- falcon-swagger-ui
- FastAPI has Swagger-UI built in — no need for external libraries

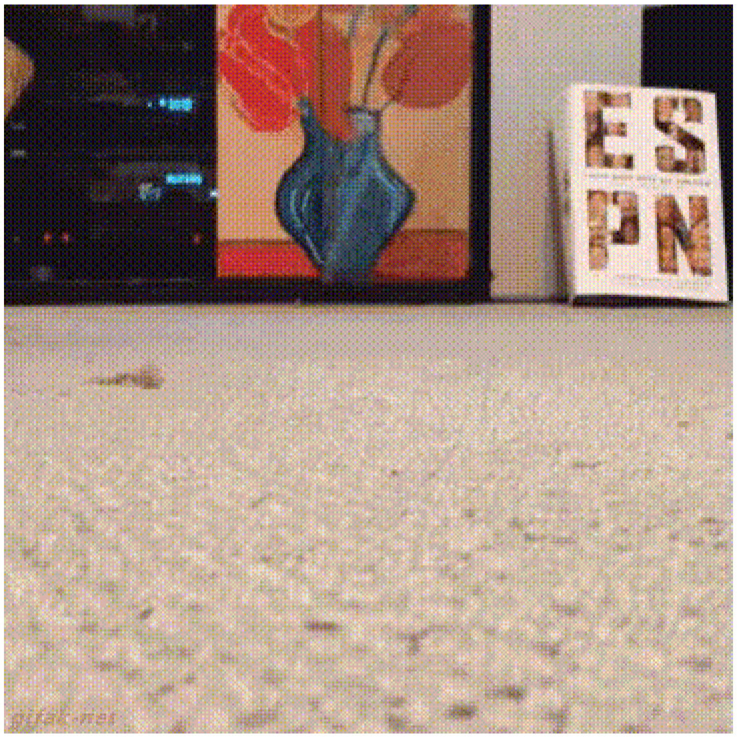
Or you can try a standalone installation.

4. pdoc3 — the semi-automatic solution

What does it do?

> *Python package `pdoc` provides types, functions, and a command-line interface for accessing public documentation of Python modules, and for presenting it in a user-friendly, industry-standard open format […]*
>
> *`pdoc` extracts documentation of:*
>
> *- modules (including submodules),*
>
> *- functions (including methods, properties, coroutines …),*
>
> *- classes, and*
>
> *- variables (including globals, class variables, and instance variables)*
>
> *`pdoc` only extracts* public API *documentation ([…] if their* identifiers don't begin with an underscore *'_')*

So `pdoc` **takes you code (modules, functions/methods, classes, variables) and creates a browsable (html/plaintext) documentation.** It's semi-automatic because it uses your code to create the main docs, but it will add more useful info if you have docstrings.



When I use a framework without reading its documentation

Other features:

- Docstrings for objects can be disabled, overridden, or whitelisted with a special module-level dictionary `__pdoc__`
- Supports multiple docstring formats: pure Markdown (with extensions), numpydoc, Google-style and some reST directives
- LaTeX math syntax is supported when placed between recognized delimiters
- Linking to other identifiers in your modules
- Programmatic usage — control `pdoc` using Python
- Custom templates - override the built-in HTML/CSS or plaintext
- With CLI params you can: change the output directory, omit the source code preview, target documentation at specific modules, filter identifiers that will be documented
- Create output formatted in Markdown-Extra, compatible with most Markdown-(to-HTML-)to-PDF converters

- Local HTTP server (**it was throwing exceptions for me)*
- Requires Python 3.5+
- **License GNU AGPL-3.0** (**make sure you double-check how you use pdoc3 in a commercial product, read more)*

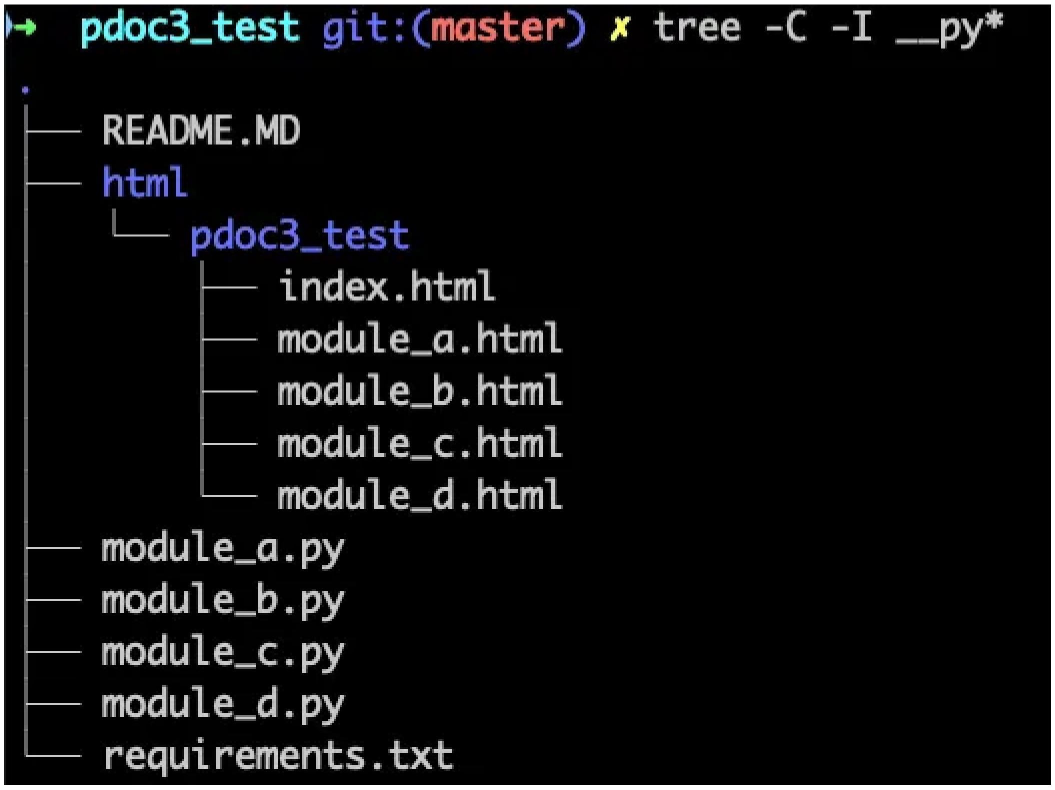
Installation:

```
pip install pdoc3
```

Usage (inside your Python project):

```
pdoc --html .
```

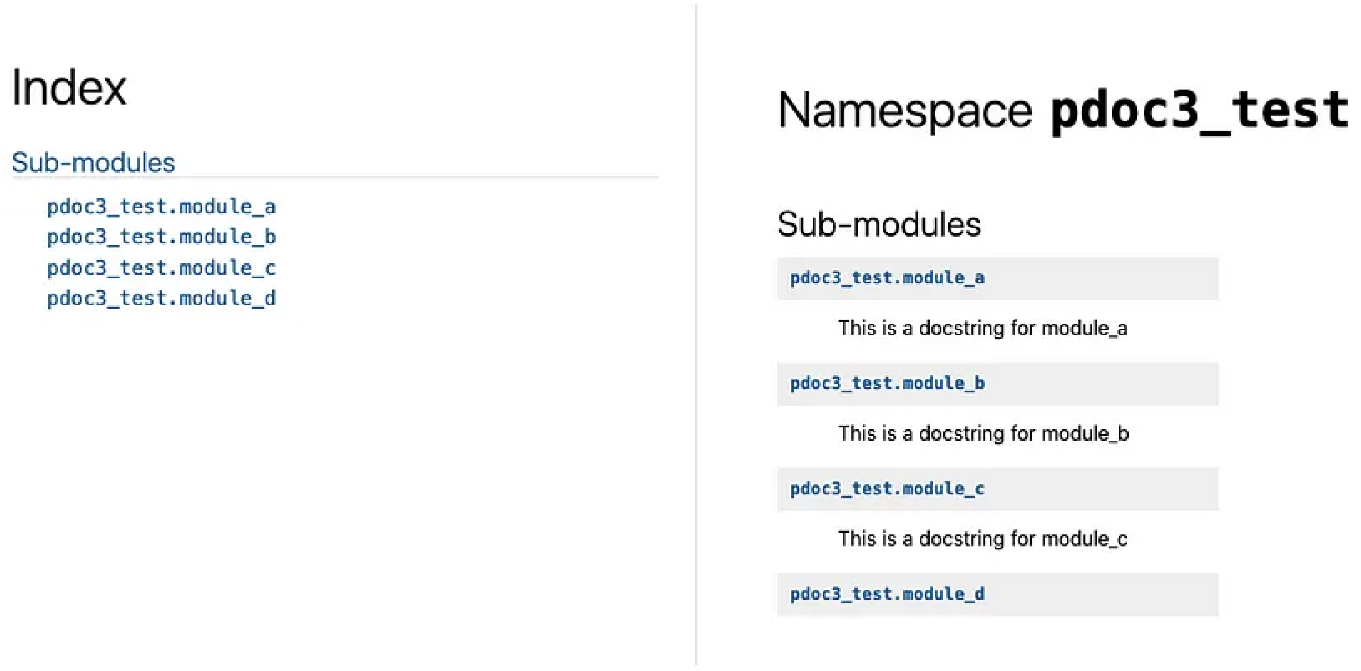
This will create a directory called `html` containing another directory (named the same way as your project dir) and inside you will find `.html` files with your Python modules documented. Here is the output of `pdoc` ran on my example Python code



```
➜ pdoc3_test git:(master) ✗ tree -C -I __py*
.
├── README.MD
├── html
│   └── pdoc3_test
│       ├── index.html
│       ├── module_a.html
│       ├── module_b.html
│       ├── module_c.html
│       └── module_d.html
├── module_a.py
├── module_b.py
├── module_c.py
├── module_d.py
└── requirements.txt
```

The blue “html” directory is the output of running “pdoc”.

The `index.html` file:



Index

Sub-modules

pdoc3_test.module_a
pdoc3_test.module_b
pdoc3_test.module_c
pdoc3_test.module_d

Namespace **pdoc3_test**

Sub-modules

pdoc3_test.module_a

This is a docstring for module_a

pdoc3_test.module_b

This is a docstring for module_b

pdoc3_test.module_c

This is a docstring for module_c

pdoc3_test.module_d

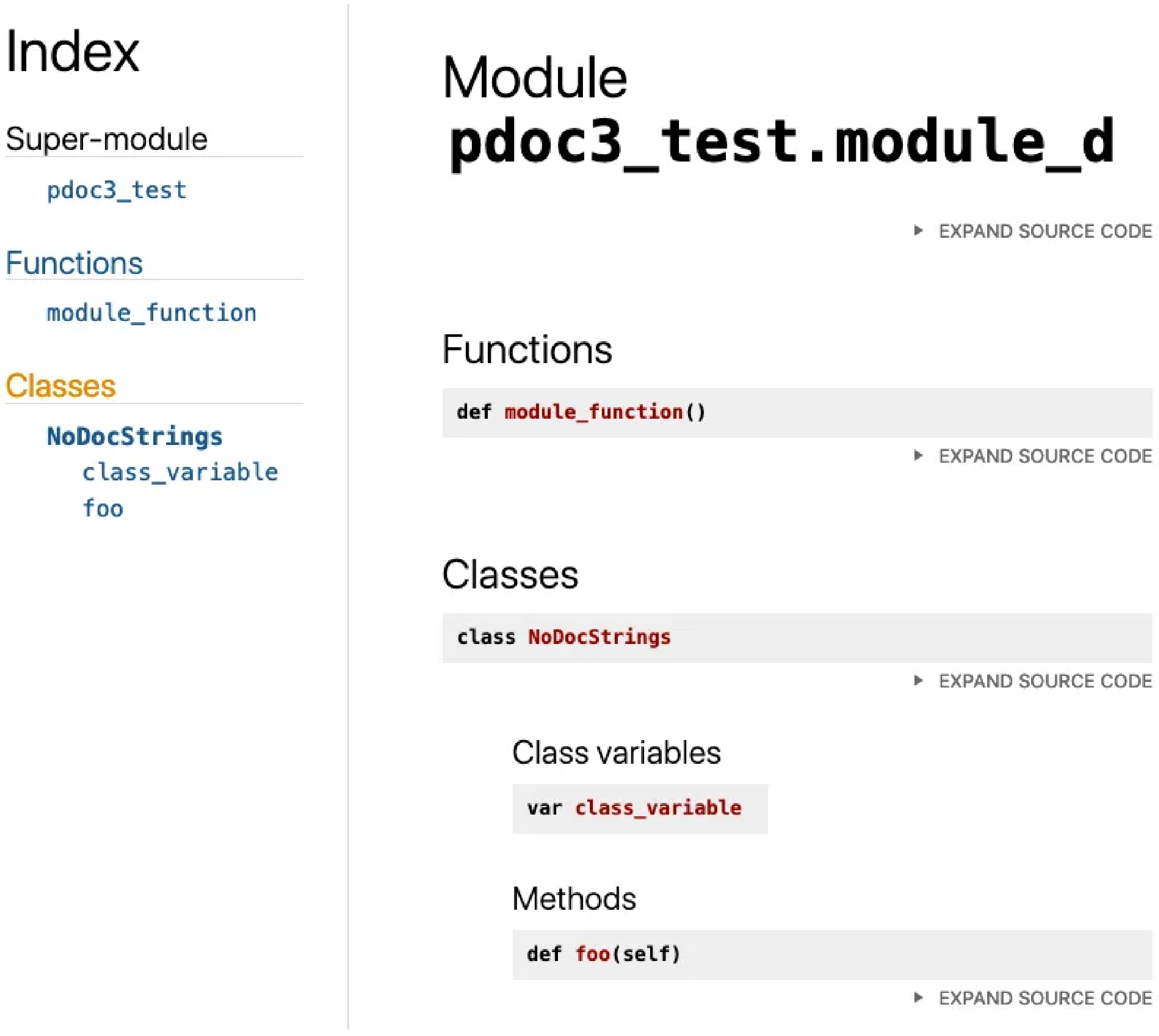
“pdoc” index.html file opened in a browser.

We have all our modules indexed on the left. What is important to notice `pdoc` **also documented code without docstrings. This is huge — you don’t need to have docstrings and you will still benefit from `pdoc`.**

No docstring code:

```
module_variable = 1

class NoDocStrings:
    class_variable = 2

    def __init__(self):
        self.instance_variable = 3

    def foo(self):
        pass

    def _private_method(self):
        pass

    def __name_mangled_method(self):
        pass


def module_function():
    pass
```

The result:


Index

Super-module

pdoc3_test

Functions

module_function

Classes

NoDocStrings

class_variable

foo

Module pdoc3_test.module_d

▸ EXPAND SOURCE CODE

Functions

```
def module_function()
```

▸ EXPAND SOURCE CODE

Classes

```
class NoDocStrings
```

▸ EXPAND SOURCE CODE

Class variables

```
var class_variable
```

Methods

```
def foo(self)
```

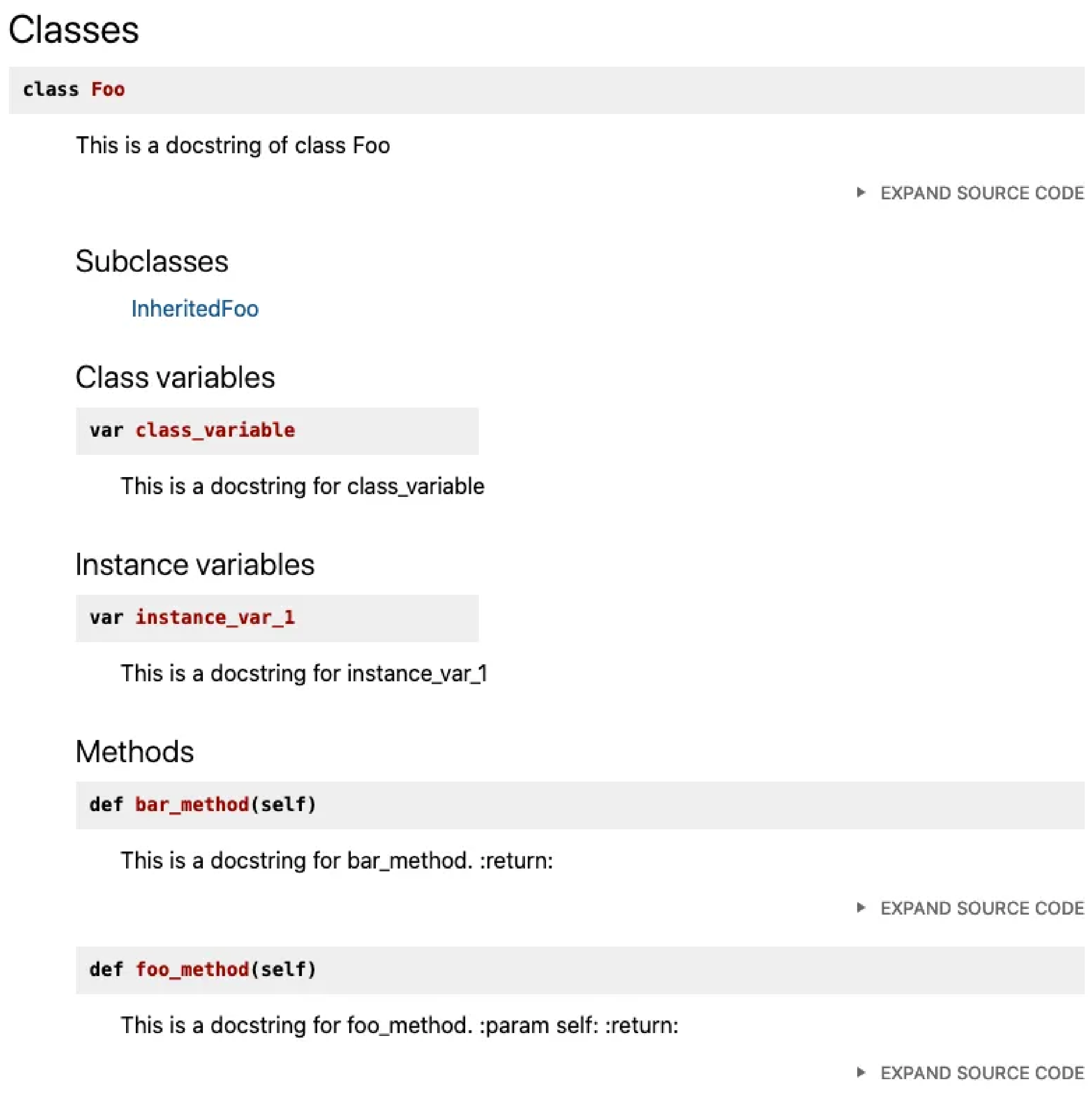
▸ EXPAND SOURCE CODE

Code without docstrings is also indexed by “pdoc”!

A class with docstrings:

```
    class Foo:
        """
    This is a docstring of class Foo
    """
    class_variable = 3
    """This is a docstring for class_variable"""

    def __init__(self):
        self.instance_var_1 = 1
        """This is a docstring for instance_var_1"""
        self.instance_var_2 = 2

    def foo_method(self):
        """
        This is a docstring for foo_method.
        :param self:
        :return:
        """

    def bar_method(self):
        """
        This is a docstring for bar_method.
        :return:
        """

    def _private_method(self):
        """
        This is a docstring for _private_method
        :return:
        """

    def __name_mangled_method(self):
        """
        This is a docstring for __name_mangled_method
        :return:
        """
```

The result:


Classes

`class Foo`

This is a docstring of class Foo

▸ EXPAND SOURCE CODE

Subclasses

InheritedFoo

Class variables

`var class_variable`

This is a docstring for class_variable

Instance variables

`var instance_var_1`

This is a docstring for instance_var_1

Methods

`def bar_method(self)`

This is a docstring for bar_method. :return:

▸ EXPAND SOURCE CODE

`def foo_method(self)`

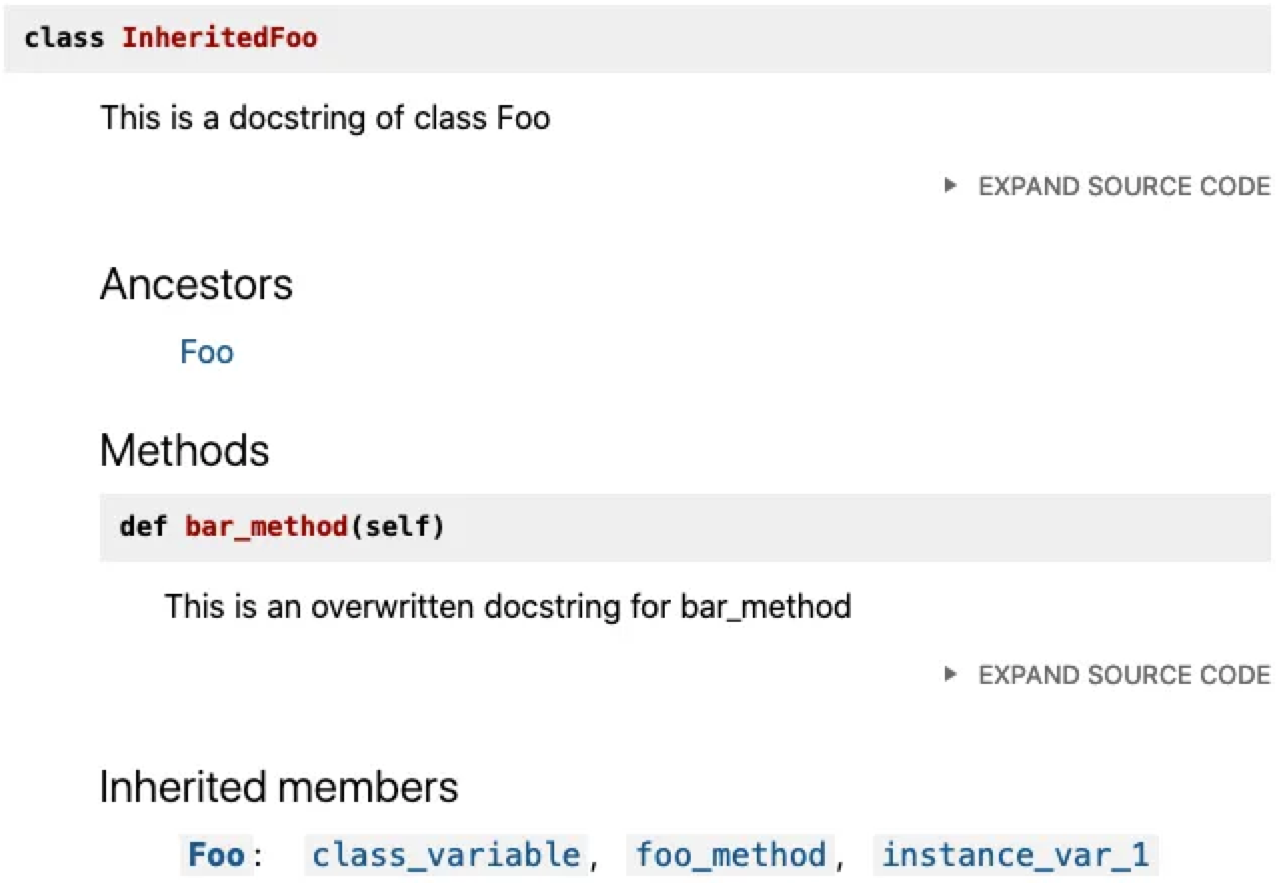
This is a docstring for foo_method. :param self: :return:

▸ EXPAND SOURCE CODE

A class which has no docstrings, but inherits from a class with a docstring:

```
class InheritedFoo(Foo):

    def foo_method(self):
        pass

    def bar_method(self):
        """This is an overwritten docstring for bar_method"""
        pass
```

The result:



```
class InheritedFoo
```

This is a docstring of class Foo

EXPAND SOURCE CODE

Ancestors

Foo

Methods

```
def bar_method(self)
```

This is an overwritten docstring for bar_method

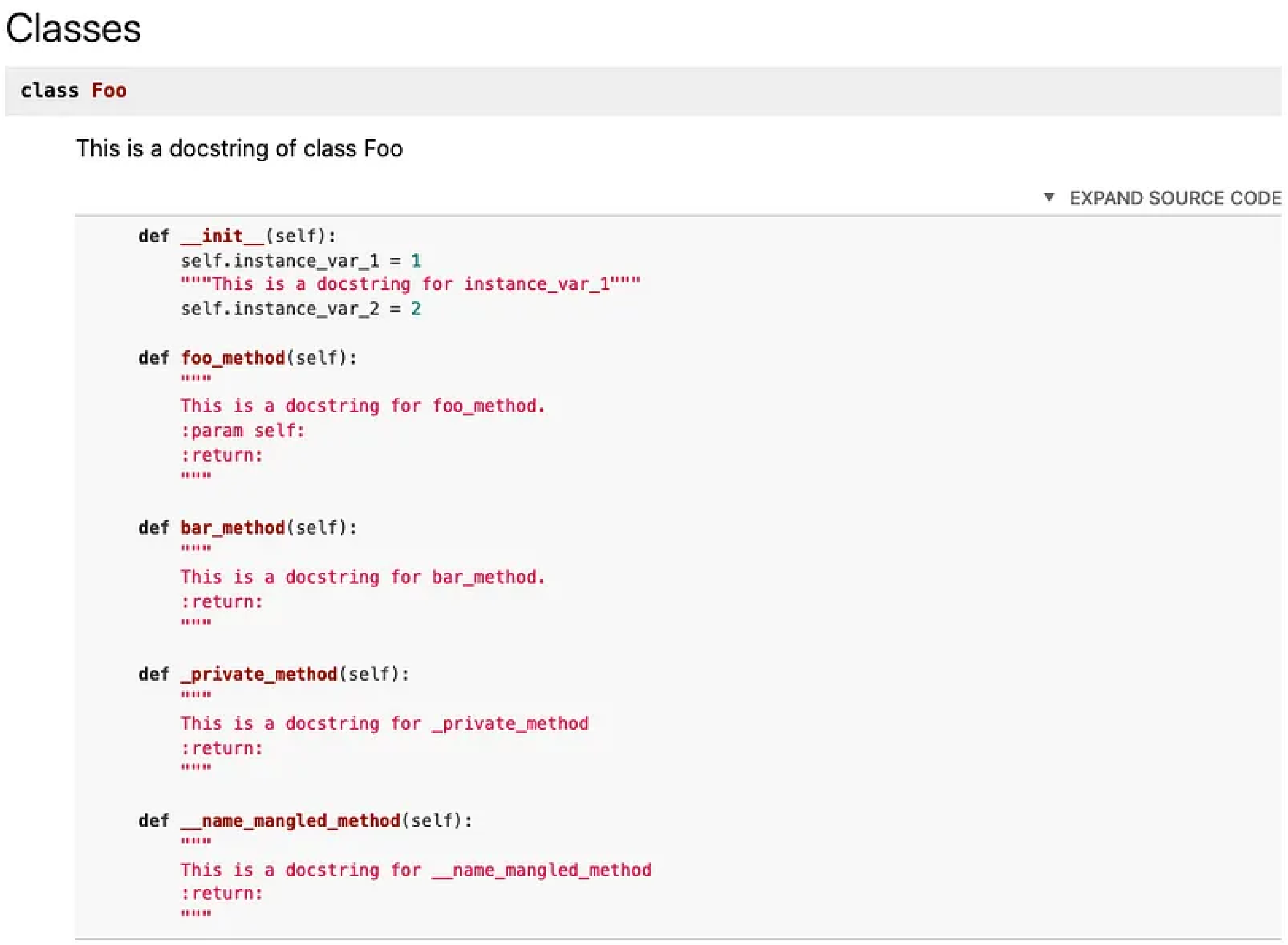
EXPAND SOURCE CODE

Inherited members

`Foo`: `class_variable`, `foo_method`, `instance_var_1`

The `bar_method` docstring was overwritten.

Private and name-mangled methods are not documented but you can see them when you click “Expand source code”:



Classes

```
class Foo
```

This is a docstring of class Foo

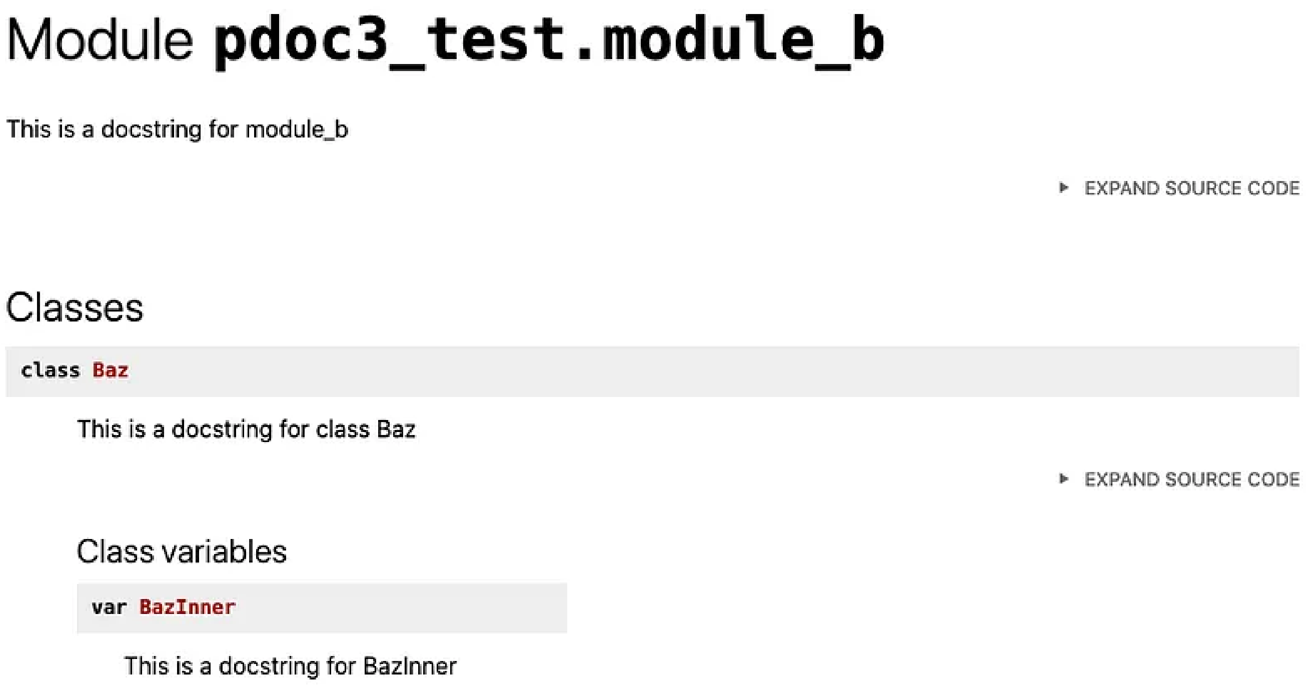
▼ EXPAND SOURCE CODE

```
    def __init__(self):
        self.instance_var_1 = 1
        """This is a docstring for instance_var_1"""
        self.instance_var_2 = 2

    def foo_method(self):
        """
        This is a docstring for foo_method.
        :param self:
        :return:
        """

    def bar_method(self):
        """
        This is a docstring for bar_method.
        :return:
        """

    def _private_method(self):
        """
        This is a docstring for _private_method
        :return:
        """

    def __name_mangled_method(self):
        """
        This is a docstring for __name_mangled_method
        :return:
        """
```

Nested classes:

```
class Baz:
    """
    This is a docstring for class Baz
    """

    class BazInner:
        """
        This is a docstring for BazInner
        """
```

The result:


Module **pdoc3_test.module_b**

This is a docstring for module_b

▸ EXPAND SOURCE CODE

Classes

`class Baz`

This is a docstring for class Baz

▸ EXPAND SOURCE CODE

Class variables

`var BazInner`

This is a docstring for BazInner

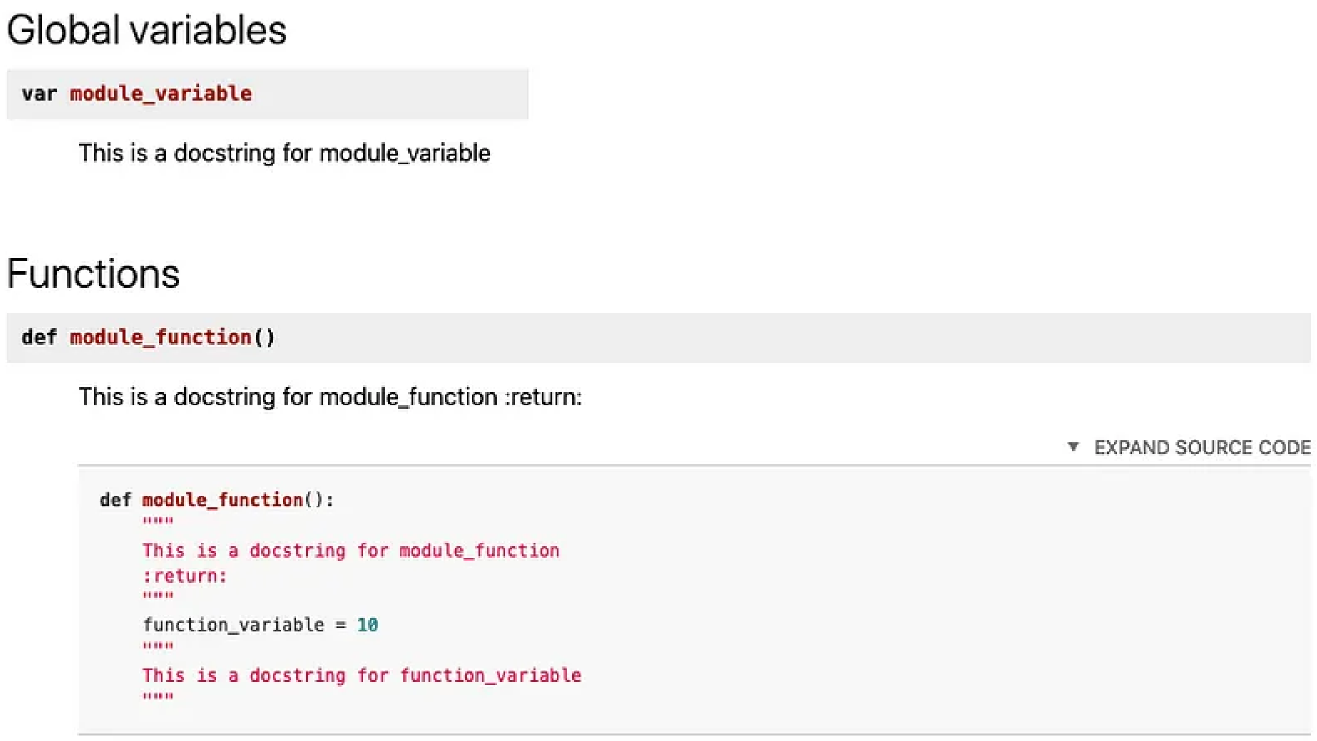
Inner class "BazInner" was indexed as a variable, *wish* `pdoc` *would also indicate that it is a class ;)*

Module-level variables and functions:

```
"""
This is a docstring for module_c
"""

module_variable = 100
"""
This is a docstring for module_variable
"""

def module_function():
    """
    This is a docstring for module_function
    :return:
    """
    function_variable = 10
    """
    This is a docstring for function_variable
    """

def _private_module_function():
    """
    This is a docstring for _private_module_function
```

```
        :return:
        """

    def __name_mangled_function():
        """
        This is a docstring for __name_mangled_function
        :return:
        """
```

The result:



Global variables

`var module_variable`

This is a docstring for module_variable

Functions

`def module_function()`

This is a docstring for module_function :return:

```
def module_function():
    """
    This is a docstring for module_function
    :return:
    """
    function_variable = 10
    """
    This is a docstring for function_variable
    """
```

You can see more examples on `pdoc3` docs page — they documented their own code with `pdoc` :)

5. MkDocs — the manual solution

What does it do?

> *MkDocs is a* ***fast, simple*** *and* ***downright gorgeous*** *static site generator that's geared towards building project documentation. Documentation source files are written in Markdown, and configured with a single YAML configuration file.*

Out of the 3 tools I'm describing this one is the least automatic, **it only autogenerates a nice-looking documentation website.** All of the content is created manually.

Features:

- Source files are written in Markdown
- Preview site with a dev server
- Host anywhere like GitHub pages or Amazon S3

- Plugins
- Custom themes (also check out the community themes — GitBook is my favourite)
- Python versions 3.5, 3.6, 3.7, 3.8, and pypy3
- MkDocs License — BSD, each theme may have its own license, i.e, ReadTheDocs theme is MIT-licensed.

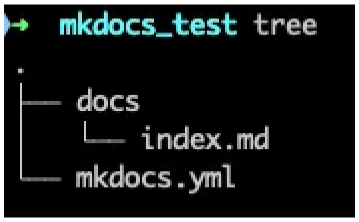
Installation:

```
pip install mkdocs
```

Usage:

```
mkdocs new mkdocs_test
```

Result:



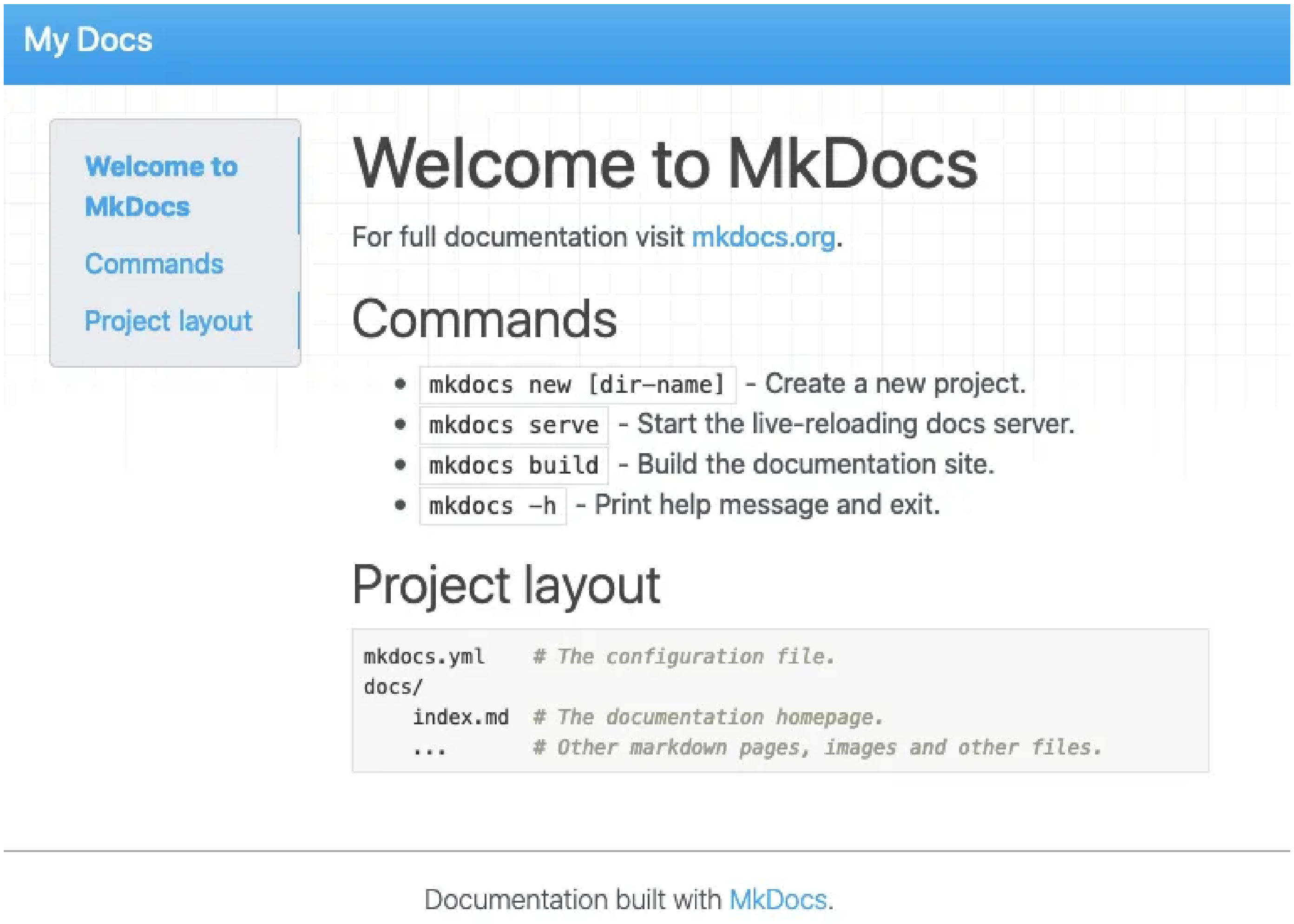
```
→ mkdocs_test tree
.
├── docs
│   └── index.md
└── mkdocs.yml
```

- *mkdocs.yml* — configuration
- *index.md* — the default docs page

To run the **dev server:**

```
mkdocs serve
```

And go to http://127.0.0.1:8000 (by default)



Welcome to MkDocs

For full documentation visit mkdocs.org.

Commands

- `mkdocs new [dir-name]` - Create a new project.
- `mkdocs serve` - Start the live-reloading docs server.
- `mkdocs build` - Build the documentation site.
- `mkdocs -h` - Print help message and exit.

Project layout

```
mkdocs.yml    # The configuration file.
docs/
    index.md  # The documentation homepage.
    ...       # Other markdown pages, images and other files.
```

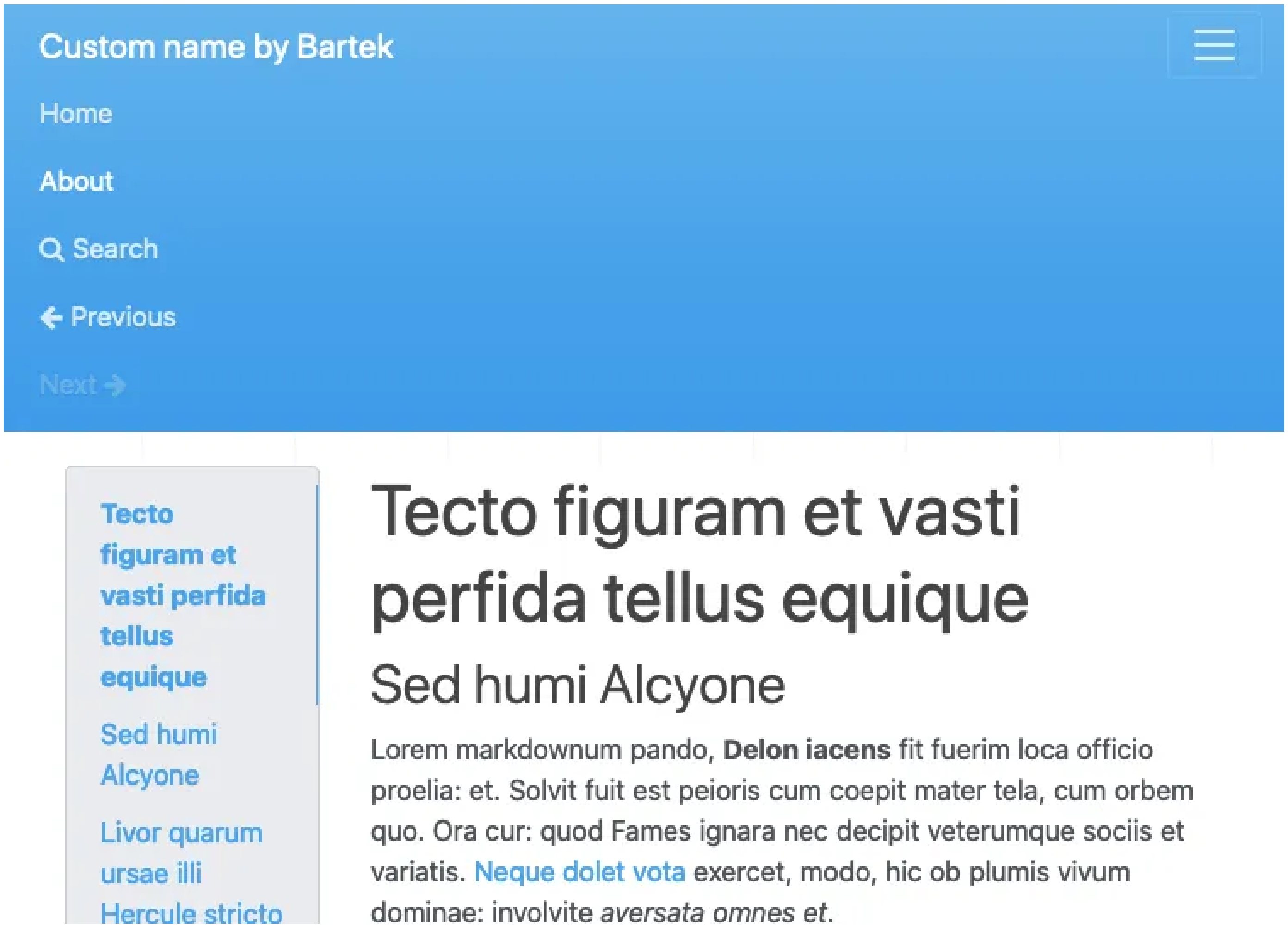
Documentation built with MkDocs.

The server will auto-reload the page whenever you change the configuration or documented pages.

Adding a new page:

Create a `.md` file in `docs/` dir and link it in the configuration file in `nav` section:

```
nav:
    - Home: index.md
    - About: about.md
```



Tecto figuram et vasti perfida tellus equique

Sed humi Alcyone

Lorem markdownum pando, **Delon iacens** fit fuerim loca officio proelia: et. Solvit fuit est peioris cum coepit mater tela, cum orbem quo. Ora cur: quod Fames ignara nec decipit veterumque sociis et variatis. Neque dolet vota exercet, modo, hic ob plumis vivum dominae: involvite *aversata omnes et*.

You also get *“search”, “previous”, “next”* buttons for free.

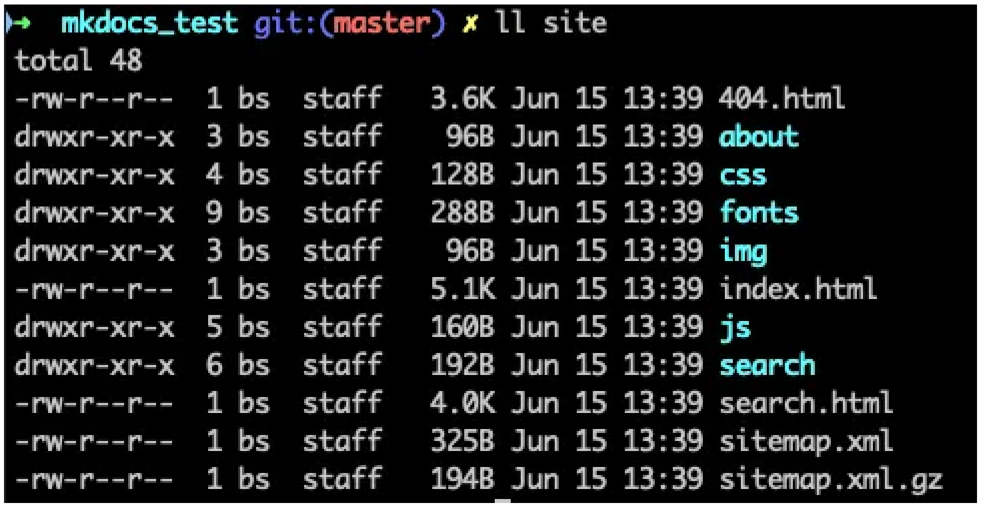
Changing the theme is as easy as (in the config file):

```
theme: readthedocs
```

Building the site (in cli):

```
mkdocs build
```

This will create a static html site located int `site` directory.



```
➜  mkdocs_test git:(master) ✗ ll site
total 48
-rw-r--r--  1 bs  staff   3.6K Jun 15 13:39 404.html
drwxr-xr-x  3 bs  staff    96B Jun 15 13:39 about
drwxr-xr-x  4 bs  staff   128B Jun 15 13:39 css
drwxr-xr-x  9 bs  staff   288B Jun 15 13:39 fonts
drwxr-xr-x  3 bs  staff    96B Jun 15 13:39 img
-rw-r--r--  1 bs  staff   5.1K Jun 15 13:39 index.html
drwxr-xr-x  5 bs  staff   160B Jun 15 13:39 js
drwxr-xr-x  6 bs  staff   192B Jun 15 13:39 search
-rw-r--r--  1 bs  staff   4.0K Jun 15 13:39 search.html
-rw-r--r--  1 bs  staff   325B Jun 15 13:39 sitemap.xml
-rw-r--r--  1 bs  staff   194B Jun 15 13:39 sitemap.xml.gz
```

Deploying:

> *The documentation site that you just built only uses static files so you'll be able to host it from pretty much anywhere. GitHub project pages and Amazon S3 may be good hosting options, depending upon your needs.*

6. Alternatives:

What other tools are available in the Python ecosystem that help with documentation:

- The offical Python documentation pages use reStructuredText (as markup language) and Sphinx, (**I find Markdown a bit simpler than rST but it's a personal choice*)
- Doxygen —generates documentation from annotated sources
- Portray — Python3 command-line tool and library that helps you create great documentation websites for your Python projects with as little effort as possible
- Pycco — Python port of Docco: the original quick-and-dirty, hundred-line-long, literate-programming-style documentation generator. It produces HTML that displays your comments alongside your code.

7. Other resources

DocumentationTools - Python Wiki

This page is primarily about tools that help, specifically, in generating documentation for software written in Python...

wiki.python.org

Summary

If you are creating an **API then Swagger-UI is a must.**

With **very little effort** you can create module/class/function documentation using **pdoc3**. If the developers write **docstrings** then you will **benefit even** more.

Writing manual documentation takes more time, but things like architecture overview, installation etc should be (at least briefly) described. **MkDocs** makes it easy to create **simple and beautiful documentation.**

Just remember that having some documentation is **not an excuse for creating bad code. Self-documenting code is an absolute priority.**

Hey, do atheists go to hell?
No.
How about
people who don't provide documentation for their code
Straight to hell. To the boiler room of hell. All the way down.

Source https://www.reddit.com/r/ProgrammerHumor/comments/glbjhf/documentation_is_a_must/

What are Your experiences with documentation? Do you document your project? Do you use other tools to create documentation? Let me know in the comments section :)

Python Documentation Swagger Mkdocs Python Technology

Follow

Written by Bartlomiej Skwira

29 Followers · Writer for blueriders

Python Developer. Pychology and neuroscience enthusiast.