yippi

Release 0.1.1

Rendy Arya Kemal

Jul 26, 2022

CONTENTS

CHAPTER

ONE

OVERVIEW

An (a)sync e621 API wrapper library.

- Free software: GNU Lesser General Public License v3 (LGPLv3)

1.1 Installation

```
pip install yippi
```

You can also install the in-development version with:

```
pip install git+ssh://git@github.com/rorre/yippi.git@master
```

1.2 Quickstart

1.2.1 Sync

```
>>> import requests
>>> from yippi import YippiClient
>>>
>>> session = requests.Session()
>>> client = YippiClient("MyProject", "1.0", "MyUsernameOnE621", session)
>>> posts = client.posts("m/m zeta-haru rating:s") # or ["m/m", "zeta-haru", "rating-s"],
↪ both works.
[Post(id=1383235), Post(id=514753), Post(id=514638), Post(id=356347), Post(id=355044)]
>>> posts[0].tags
{'artist': ['zeta-haru'],
 'character': ['daniel_segja', 'joel_mustard'],
 'copyright': ['patreon'],
 'general': ['5_fingers', ..., 'spooning'],
 'invalid': [],
 'lore': [],
 'meta': ['comic'],
 'species': ['bird_dog', ... ]}
```

1.2.2 Async

```
>>> import aiohttp
>>> from yippi import AsyncYippiClient
>>>
>>> session = aiohttp.ClientSession()
>>> client = AsyncYippiClient("MyProject", "1.0", "MyUsernameOnE621", session)
>>> posts = await client.posts("m/m zeta-haru rating:s") # or ["m/m", "zeta-haru",
↪"rating-s"], both works.
[Post(id=1383235), Post(id=514753), Post(id=514638), Post(id=356347), Post(id=355044)]
>>> posts[0].tags
{'artist': ['zeta-haru'],
 'character': ['daniel_segja', 'joel_mustard'],
 'copyright': ['patreon'],
 'general': ['5_fingers', ..., 'spooning'],
 'invalid': [],
 'lore': [],
 'meta': ['comic'],
 'species': ['bird_dog', ... ]}
```

Examples are available in examples directory.

1.3 Documentation

Documentation is available on readthedocs: https://yippi.readthedocs.io/

1.4 Development

To run the all tests run:

```
tox
```

Note, to combine the coverage data from all the tox environments run:

Windows	`set PYTEST_ADDOPTS=--cov-append` `tox`
Other	`PYTEST_ADDOPTS=--cov-append tox`

CHAPTER

TWO

INTRODUCTION

2.1 Installation

Yippi does not require many dependency,It only asks for the client as its dependency.

Currently there are two clients available:

- `aiohttp` (Async)
- `requests` (Sync)

To install, you can use `pip`.:

```
pip install yippi
```

And you should be ready to go.

2.2 Calling the API

Currenly, only GET routes are available for now. To initialize the client, you have to supply your project name, version, and your username on e621.:

```
from yippi import YippiClient, AsyncYippiClient

client = YippiClient("MyProject", "1.0", "MyUsername")
client = AsyncYippiClient("MyProject", "1.0", "MyUsername")
```

The only core functions are as follows.:

```
client.posts()      # Searches e621
client.post()       # Fetch for a post
client.notes()      # Searches for notes
client.flags()      # Searches for flags
client.pools()      # Searches for pools
```

If you're using the async client, you only need to prepend the `await` statement.:

```
await client.posts()
await client.post()
await client.notes()
await client.flags()
await client.pools()
```

An example is available on the Overview page.

CHAPTER

THREE

ADVANCED

3.1 Adding Custom Client

The module is written to be as easily portable as possible. Therefore, any custom client should be easily written to this library. The *yippi.AbstractYippi.AbstractYippi* class should be enough for you to inherit. And what you need to do is override the abstract functions.

For example, here's a snippet from AsyncYippiClient.

```
from typing import List
from typing import Optional
from typing import Union

import aiohttp
from aiohttp import BasicAuth
from aiohttp import FormData

from .AbstractYippi import AbstractYippi
from .AbstractYippi import limiter
from .Classes import Flag
from .Classes import Note
from .Classes import Pool
from .Classes import Post
from .Exceptions import APIError
from .Exceptions import UserError


class AsyncYippiClient(AbstractYippi):
    def __init__(
        self,
        project_name: str,
        version: str,
        creator: str,
        loop=None,
        session: aiohttp.ClientSession = None,
    ) -> None:
        self._loop = loop
        self._session: aiohttp.ClientSession = session or aiohttp.ClientSession()
        super().__init__(project_name, version, creator)

    async def close(self) -> None:
```

(continues on next page)

(continued from previous page)

```
        await self._session.close()

    async def __aenter__(self) -> "AsyncYippiClient":
        return self

    async def __aexit__(self, exc_type, exc_val, exc_tb) -> None:
        await self.close()

    @limiter.ratelimit("call_api", delay=True)
    async def _call_api(
        self,
        method: str,
        url: str,
        data: Union[dict, FormData] = None,
        file=None,
        **kwargs
    ) -> Optional[Union[List[dict], dict]]:
        auth = None
        if self._login != ("", ""):
            auth = BasicAuth(*self._login)

        if file:
            file = file["upload[file]"]
            formdata = FormData()
            formdata.add_field(
                "upload[file]", file[1], filename=file[0], content_type=file[2]
            )
            formdata.add_fields(data)
            data = formdata

        query_string = self._generate_query_keys(**kwargs)
        url += "?" + query_string
        r = await self._session.request(
            method, url, data=data, headers=self.headers, auth=auth
        )
        await self._verify_response(r)
        if not r.status == 204:
            return await r.json()

        return None

    async def _verify_response(self, r) -> None:
        if 300 <= r.status < 500:
            res = await r.json()
            if r.status >= 400:
                raise UserError(res.get("message") or res.get("reason"), json=res)

        elif r.status >= 500:
            raise APIError(r.reason)

        if r.status != 204 and (
            not r.headers.get("Content-Type")
```

(continues on next page)

(continued from previous page)

```
            or "application/json" not in r.headers.get("Content-Type")
        ):
            res = await r.text()
            if "Not found." in res:
                raise UserError("Not found.")
            raise UserError("Invalid input or server error.")

    async def posts(
        self,
        tags: Union[List, str] = None,
        limit: int = None,
        page: Union[int, str] = None,
    ) -> List[Post]:
        response = await self._get_posts(tags, limit, page)  # type: ignore
        posts = [Post(p, client=self) for p in response["posts"]]
        return posts

    async def post(self, post_id: int) -> Post:
        api_res = await self._get_post(post_id)  # type: ignore
        return Post(api_res["post"], client=self)

    async def notes(
        self,
        body_matches: str = None,
        post_id: int = None,
        post_tags_match: Union[List, str] = None,
        creator_name: str = None,
        creator_id: int = None,
        is_active: bool = None,
        limit: int = None,
    ) -> List[Note]:
        response = await self._get_notes(
            body_matches,
            post_id,
            post_tags_match,
            creator_name,
            creator_id,
            is_active,
            limit,
        )  # type: ignore
        result = [Note(n, client=self) for n in response]
        return result

    async def flags(
        self,
        post_id: int = None,
        creator_id: int = None,
        creator_name: str = None,
        limit: int = None,
    ) -> List[Flag]:
        response = await self._get_flags(post_id, creator_id, creator_name)  # type:␣
↪ignore
```

(continues on next page)

(continued from previous page)

```
        result = [Flag(f, client=self) for f in response]
        return result

    async def pools(
        self,
        name_matches: str = None,
        id_: Union[int, List[int]] = None,
        description_matches: str = None,
        creator_name: str = None,
        creator_id: int = None,
        is_active: bool = None,
        is_deleted: bool = None,
        category: str = None,
        order: str = None,
        limit: int = None,
    ) -> List[Pool]:
        response = await self._get_pools(
            name_matches,
            id_,
            description_matches,
            creator_name,
            creator_id,
            is_active,
            is_deleted,
            category,
            order,
            limit,
        )  # type: ignore
        result = [Pool(p, client=self) for p in response]
        return result

    async def pool(self, pool_id: int) -> Pool:
        response = await self._get_pool(pool_id)  # type: ignore
        return Pool(response, client=self)
```

CHAPTER

FOUR

REFERENCE

4.1 yippi package

4.1.1 yippi.AbstractYippi module

class yippi.AbstractYippi.**AbstractYippi**(*project_name: str*, *version: str*, *creator: str*)

Bases: abc.ABC

An abstract class (abc) for all the Yippi's client.

Generally you don't really need to use this, except if you want to use different implementation for the client.

Parameters

- **project_name** – Your project's name where this library is going to be used.
- **version** – You project's version number.
- **creator** – Your e621 username.
- **session** – The HTTP client session object.
- **loop** – The event loop to run on. This is only required on async client.

VALID_CATEGORY = ('series', 'collection')

VALID_ORDER = ('name', 'created_at', 'updated_at', 'post_count')

abstract flags(*post_id: Optional[int] = None*, *creator_id: Optional[int] = None*, *creator_name: Optional[str] = None*, *limit: Optional[int] = None*) → Union[List[*yippi.Classes.Flag*], Awaitable[List[*yippi.Classes.Flag*]]]

Search for flags

Parameters

- **post_id** – The ID of the flagged post.
- **creator_id** – The user's ID that created the flag.
- **creator_name** – The user's name that created the flag.
- **limit** – Limits the amount of notes returned to the number specified.

Returns list of *Flag* of the flags.

login(*username: str*, *api_key: str*) → None

Supply login credentials to client.

Parameters

- **username** – Your e621 username.
- **api_key** – Your API key. Find it under "Account" on e621.

abstract notes(*body_matches: Optional[str] = None*, *post_id: Optional[int] = None*, *post_tags_match: Optional[Union[List, str]] = None*, *creator_name: Optional[str] = None*, *creator_id: Optional[int] = None*, *is_active: Optional[bool] = None*, *limit: Optional[int] = None*) → Union[List[*yippi.Classes.Note*], Awaitable[List[*yippi.Classes.Note*]]]

Search for notes.

Parameters

- **body_matches** – The note's body matches the given terms. Use a * in the search terms to search for raw strings.
- **post_id** – The post where the note is located.
- **post_tags_match** – The note's post's tags match the given terms. Meta-tags are not supported.
- **creator_name** – The creator's name. Exact match.
- **creator_id** – The creator's user id.
- **is_active** – Can be `True` or `False`.
- **limit** – Limits the amount of notes returned to the number specified.

Returns `list` of *Note* of the notes.

abstract pool(*pool_id: int*) → Union[*yippi.Classes.Pool*, Awaitable[*yippi.Classes.Pool*]]

Fetch for a pool.

Parameters **pool_id** – The pool's ID to look up.

Returns *Pool* of the pool.

abstract pools(*name_matches: Optional[str] = None*, *id_: Optional[Union[int, List[int]]] = None*, *description_matches: Optional[str] = None*, *creator_name: Optional[str] = None*, *creator_id: Optional[int] = None*, *is_active: Optional[bool] = None*, *is_deleted: Optional[bool] = None*, *category: Optional[str] = None*, *order: Optional[str] = None*, *limit: Optional[int] = None*) → Union[List[*yippi.Classes.Pool*], Awaitable[List[*yippi.Classes.Pool*]]]

Search for pools.

Parameters

- **name_matches** – Search for pool names.
- **id_** – Search for a pool ID. Multiple IDs are fine. .. warning:: Take note of the underscore (_) mark!
- **description_matches** – Search for pool descriptions.
- **creator_name** – Search for pools based on creator name.
- **creator_id** – Search for pools based on creator ID.
- **is_active** – If the pool is active or hidden. (True/False)
- **is_deleted** – If the pool is deleted. (True/False)
- **category** – Can either be "series" or "collection".
- **order** – The order that pools should be returned, can be any of: `name`, `created_at`, `updated_at`, `post_count`. If not specified it orders by `updated_at`.

- **limit** – The limit of how many pools should be retrieved.

Returns `list` of *Pool* of the pools.

abstract post(*post_id: int*) → Union[*yippi.Classes.Post*, Awaitable[*yippi.Classes.Post*]]

Fetch for a post.

Parameters **post_id** – The post's ID to look up.

Returns *Post* of the posts.

abstract posts(*tags: Optional[Union[List, str]] = None*, *limit: Optional[int] = None*, *page: Optional[Union[int, str]] = None*) → Union[List[*yippi.Classes.Post*], Awaitable[List[*yippi.Classes.Post*]]]

Search for posts.

Parameters

- **tags** – The tags to search.
- **limit** – Limits the amount of notes returned to the number specified.
- **page** – The page that will be returned.

Returns `list` of *Post* of the posts.

4.1.2 yippi.AsyncYippi module

class yippi.AsyncYippi.**AsyncYippiClient**(*project_name: str*, *version: str*, *creator: str*, *loop=None*, *session: Optional[aiohttp.client.ClientSession] = None*)

Bases: *yippi.AbstractYippi.AbstractYippi*

async close() → None

async flags(*post_id: Optional[int] = None*, *creator_id: Optional[int] = None*, *creator_name: Optional[str] = None*, *limit: Optional[int] = None*) → List[*yippi.Classes.Flag*]

Search for flags

Parameters

- **post_id** – The ID of the flagged post.
- **creator_id** – The user's ID that created the flag.
- **creator_name** – The user's name that created the flag.
- **limit** – Limits the amount of notes returned to the number specified.

Returns `list` of *Flag* of the flags.

async notes(*body_matches: Optional[str] = None*, *post_id: Optional[int] = None*, *post_tags_match: Optional[Union[List, str]] = None*, *creator_name: Optional[str] = None*, *creator_id: Optional[int] = None*, *is_active: Optional[bool] = None*, *limit: Optional[int] = None*) → List[*yippi.Classes.Note*]

Search for notes.

Parameters

- **body_matches** – The note's body matches the given terms. Use a * in the search terms to search for raw strings.
- **post_id** – The post where the note is located.

- **post_tags_match** – The note's post's tags match the given terms. Meta-tags are not supported.
- **creator_name** – The creator's name. Exact match.
- **creator_id** – The creator's user id.
- **is_active** – Can be `True` or `False`.
- **limit** – Limits the amount of notes returned to the number specified.

Returns `list` of *Note* of the notes.

async pool(*pool_id: int*) → *yippi.Classes.Pool*

Fetch for a pool.

Parameters **pool_id** – The pool's ID to look up.

Returns *Pool* of the pool.

async pools(*name_matches: Optional[str] = None*, *id_: Optional[Union[int, List[int]]] = None*, *description_matches: Optional[str] = None*, *creator_name: Optional[str] = None*, *creator_id: Optional[int] = None*, *is_active: Optional[bool] = None*, *is_deleted: Optional[bool] = None*, *category: Optional[str] = None*, *order: Optional[str] = None*, *limit: Optional[int] = None*) → List[*yippi.Classes.Pool*]

Search for pools.

Parameters

- **name_matches** – Search for pool names.
- **id_** – Search for a pool ID. Multiple IDs are fine. .. warning:: Take note of the underscore (_) mark!
- **description_matches** – Search for pool descriptions.
- **creator_name** – Search for pools based on creator name.
- **creator_id** – Search for pools based on creator ID.
- **is_active** – If the pool is active or hidden. (True/False)
- **is_deleted** – If the pool is deleted. (True/False)
- **category** – Can either be "series" or "collection".
- **order** – The order that pools should be returned, can be any of: `name`, `created_at`, `updated_at`, `post_count`. If not specified it orders by `updated_at`.
- **limit** – The limit of how many pools should be retrieved.

Returns `list` of *Pool* of the pools.

async post(*post_id: int*) → *yippi.Classes.Post*

Fetch for a post.

Parameters **post_id** – The post's ID to look up.

Returns *Post* of the posts.

async posts(*tags: Optional[Union[List, str]] = None*, *limit: Optional[int] = None*, *page: Optional[Union[int, str]] = None*) → List[*yippi.Classes.Post*]

Search for posts.

Parameters

- **tags** – The tags to search.
- **limit** – Limits the amount of notes returned to the number specified.
- **page** – The page that will be returned.

Returns `list` of *Post* of the posts.

4.1.3 yippi.YippiSync module

class yippi.YippiSync.**YippiClient**(*project_name: str*, *version: str*, *creator: str*, *session: Optional[requests.sessions.Session] = None*)

Bases: *yippi.AbstractYippi.AbstractYippi*

flags(*post_id: Optional[int] = None*, *creator_id: Optional[int] = None*, *creator_name: Optional[str] = None*, *limit: Optional[int] = None*) → List[*yippi.Classes.Flag*]

Search for flags

Parameters

- **post_id** – The ID of the flagged post.
- **creator_id** – The user's ID that created the flag.
- **creator_name** – The user's name that created the flag.
- **limit** – Limits the amount of notes returned to the number specified.

Returns `list` of *Flag* of the flags.

notes(*body_matches: Optional[str] = None*, *post_id: Optional[int] = None*, *post_tags_match: Optional[Union[List, str]] = None*, *creator_name: Optional[str] = None*, *creator_id: Optional[int] = None*, *is_active: Optional[bool] = None*, *limit: Optional[int] = None*) → List[*yippi.Classes.Note*]

Search for notes.

Parameters

- **body_matches** – The note's body matches the given terms. Use a * in the search terms to search for raw strings.
- **post_id** – The post where the note is located.
- **post_tags_match** – The note's post's tags match the given terms. Meta-tags are not supported.
- **creator_name** – The creator's name. Exact match.
- **creator_id** – The creator's user id.
- **is_active** – Can be `True` or `False`.
- **limit** – Limits the amount of notes returned to the number specified.

Returns `list` of *Note* of the notes.

pool(*pool_id: int*) → *yippi.Classes.Pool*

Fetch for a pool.

Parameters **pool_id** – The pool's ID to look up.

Returns *Pool* of the pool.

pools(*name_matches: Optional[str] = None*, *id_: Optional[Union[int, List[int]]] = None*, *description_matches: Optional[str] = None*, *creator_name: Optional[str] = None*, *creator_id: Optional[int] = None*, *is_active: Optional[bool] = None*, *is_deleted: Optional[bool] = None*, *category: Optional[str] = None*, *order: Optional[str] = None*, *limit: Optional[int] = None*) → List[*yippi.Classes.Pool*]

Search for pools.

Parameters

- **name_matches** – Search for pool names.
- **id_** – Search for a pool ID. Multiple IDs are fine. .. warning:: Take note of the underscore (_) mark!
- **description_matches** – Search for pool descriptions.
- **creator_name** – Search for pools based on creator name.
- **creator_id** – Search for pools based on creator ID.
- **is_active** – If the pool is active or hidden. (True/False)
- **is_deleted** – If the pool is deleted. (True/False)
- **category** – Can either be "series" or "collection".
- **order** – The order that pools should be returned, can be any of: `name`, `created_at`, `updated_at`, `post_count`. If not specified it orders by `updated_at`.
- **limit** – The limit of how many pools should be retrieved.

Returns `list` of *Pool* of the pools.

post(*post_id: int*) → *yippi.Classes.Post*

Fetch for a post.

Parameters **post_id** – The post's ID to look up.

Returns *Post* of the posts.

posts(*tags: Optional[Union[List, str]] = None*, *limit: Optional[int] = None*, *page: Optional[Union[int, str]] = None*) → List[*yippi.Classes.Post*]

Search for posts.

Parameters

- **tags** – The tags to search.
- **limit** – Limits the amount of notes returned to the number specified.
- **page** – The page that will be returned.

Returns `list` of *Post* of the posts.

4.1.4 yippi.Classes module

class yippi.Classes.**Flag**(*json_data=None*, **args*, ***kwargs*)

Bases: yippi.Classes._BaseMixin

Representation of e621's Flag object.

Parameters

- **data** (*optional*) – The json server response of a `post_flags.json` call.
- **client** (*optional*) – The yippi client, used for api calls.

Variables **All** (*Any*) – Refer to e621 API docs for available attributes.

classmethod create()

get_post() → Union[*Post*, Awaitable[*Post*]]

Fetch the post linked with this flag.

Returns *yippi.Classes.Post* – The post linked with this flag.

class yippi.Classes.**Note**(*json_data=None*, **args*, ***kwargs*)

Bases: yippi.Classes._BaseMixin

Representation of e621's Note object.

Parameters

- **data** (*optional*) – The json server response of a `/notes.json` call.
- **client** (*optional*) – The yippi client, used for api calls.

Variables **All** (*Any*) – Refer to e621 API docs for available attributes.

classmethod create(*post: Union[*Post*, int]*, *x: int*, *y: int*, *width: int*, *height: int*, *body: str*, *client:* AbstractYippi) → *Note*

delete() → None

get_post() → Union[*Post*, Awaitable[*Post*]]

Fetch the post linked with this note.

Returns *yippi.Classes.Post* – The post linked with this note.

revert(*version_id: str*) → dict

Reverts note to specified version_id. **This function has not been tested.**

Parameters **version_id** – Target version to revert.

Raises ***UserError*** – Raised if - Note does not come from `notes()`. - `client` kwags was not supplied.

Returns *dict* – JSON status response from API.

update() → dict

Updates the note. **This function has not been tested.**

Returns *dict* – JSON status response from API.

upload() → dict

Uploads the note. **This function has not been tested.**

Returns *dict* – JSON status response from API.

class yippi.Classes.**Pool**(*json_data=None*, **args*, ***kwargs*)

Bases: yippi.Classes._BaseMixin

Representation of e621's Pool object.

Parameters

- **data** (*optional*) – The json server response of a `pools.json` call.
- **client** (*optional*) – The yippi client, used for api calls.

Variables **All** (*Any*) – Refer to e621 API docs for available attributes.

classmethod create()

get_posts() → Union[List[*Post*], Awaitable[List[*Post*]]]

Fetch all posts linked with this pool.

If the client is an async client, it will automatically call *get_posts_async()*.

Returns `list` of *yippi.Classes.Post* – All the posts linked with this pool.

async get_posts_async() → List[*Post*]

Async representation of `get-posts()`

Returns `list` of *yippi.Classes.Post* – All the posts linked with this pool.

revert(*version_id: str*) → dict

Reverts note to specified version_id. **This function has not been tested.**

Parameters **version_id** – Target version to revert.

Raises ***UserError*** – Raised if - Pool does not come from `pools()`. - `client` kwargs was not supplied.

Returns *dict* – JSON status response from API.

update()

class yippi.Classes.**Post**(*json_data=None*, **args*, ***kwargs*)

Bases: yippi.Classes._BaseMixin

Representation of e621's Post object.

Parameters

- **data** (*optional*) – The json server response of a `/posts.json` call.
- **client** (*optional*) – The yippi client, used for api calls.

Variables **All** (*Any*) – Refer to e621 API docs for available attributes.

favorite() → Union[dict, Awaitable[dict]]

classmethod from_file(*path*) → *Post*

classmethod from_url(*url*) → *Post*

unfavorite() → None

update(*reason: Optional[str] = None*) → Union[List[dict], dict, Awaitable[Union[List[dict], dict]]]

Updates the post. **This function has not been tested.**

Parameters **reason** (*optional*) – Reasoning behind the edit. Defaults to None.

Raises *UserError* – If the post did not come from any Post endpoint or if no changes has been made.

`upload()` → Union[dict, Awaitable[dict]]

`vote`(*score: int = 1*, *replace: bool = False*) → dict

Vote the post.

If you want to cancel your vote, repeat the same function again with same score value, but with replace set to False.

Parameters

- **score** (*optional*) – Score to be given, this could be either 1 or -1, with
- **1 represents vote up and -1 represent vote down. Defaults to 1.**
- **replace** (*optional*) – Replaces old vote or not. Defaults to False.

Raises *UserError* – Raised if - Post does not come from `post()` or `posts()`. - If the value of `score` is out of scope. - `client` kwargs was not supplied.

Returns *dict* – JSON response with keys `score`, `up`, `down`, and `our_score`. Where `dict['our_score']` is 1, 0, -1 depending on the action.

`class yippi.Classes.Tag`(*json_data=None*, **args*, ***kwargs*)

Bases: `yippi.Classes._BaseMixin`

`class yippi.Classes.TagCategory`(*value*)

Bases: `enum.IntEnum`

An enumeration.

`ARTIST = 1`

`CHARACTER = 4`

`COPYRIGHT = 3`

`GENERAL = 0`

`INVALID = 6`

`LORE = 8`

`META = 7`

`SPECIES = 5`

4.1.5 yippi.Exceptions module

`exception yippi.Exceptions.APIError`

Bases: `Exception`

`exception yippi.Exceptions.UserError`(**args*, *json=None*, ***kwargs*)

Bases: `Exception`

4.1.6 yippi.Enums module

class yippi.Enums.**Rating**(*value*)

Bases: enum.Enum

Enum for e621 rating.

EXPLICIT = 'e'

NONE = None

QUESTIONABLE = 'q'

SAFE = 's'

CHAPTER

FIVE

CONTRIBUTING

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

5.1 Bug reports

When reporting a bug please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

5.2 Documentation improvements

Yippi could always use more documentation, whether as part of the official yippi docs, in docstrings, or even on the web in blog posts, articles, and such.

5.3 Feature requests and feedback

The best way to send feedback is to file an issue at https://github.com/rorre/yippi/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that code contributions are welcome :)

5.4 Development

To set up *yippi* for local development:

1. Fork yippi (look for the "Fork" button).
2. Clone your fork locally:

```
git clone git@github.com:rorre/yippi.git
```

3. Create a branch for local development:

```
git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

4. When you're done making changes run all the checks and docs builder with tox one command:

```
tox
```

5. Commit your changes and push your branch to GitHub:

```
git add .
git commit -m "Your detailed description of your changes."
git push origin name-of-your-bugfix-or-feature
```

6. Submit a pull request through the GitHub website.

5.4.1 Pull Request Guidelines

If you need some code review or feedback while you're developing the code just make the pull request.

For merging, you should:

1. Include passing tests (run `tox`)[1].
2. Update documentation when there's new API, functionality etc.
3. Add a note to `CHANGELOG.rst` about the changes.
4. Add yourself to `AUTHORS.rst`.

5.4.2 Tips

To run a subset of tests:

```
tox -e envname -- pytest -k test_myfeature
```

To run all the test environments in *parallel* (you need to `pip install detox`):

```
detox
```

[1] If you don't have all the necessary python versions available locally you can rely on Travis - it will run the tests for each change you add in the pull request.
It will be slower though ...

CHAPTER

SIX

CHANGELOG

6.1 0.2.0 (2021-06-19)

- **Drop Python 3.6 support.** The library now requires Python 3.7 or higher.
- **Bring back rate limit.** It is now compliant to e621's docs rate limiting, that is 2 requests per second.
- **Add ability to login with API key.**
- **Add ability to search pools.**
- **Post.rating is now an Enum.**
- Better type annotations. Thank you Deer-Spangle for the help!
- Handle 204 response properly from server by returning None.
- Handle errors properly by looking up if either message or response key exists in response.
- Add ability to use custom requests session on YippiSync client.
- All classes now have reference to the client. This is important for interaction abilities.
- Add __repr__ to classes.
- **Add ability to favorite, unfavorite, and vote Post.** Note: There are other functions, but those are not tested and should not be used as of now.

6.2 0.1.0 (2020-04-19)

- Initial release.

CHAPTER

SEVEN

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

y

INDEX

R

S

T

U

V

Y