

---

# **Hyper2Web Documentation**

***Release 0.0.4***

**Xuanzhe Wang**

**Aug 25, 2017**



---

## Contents

---

<b>1</b>	<b>Hyper2Web</b>	<b>3</b>
1.1	Installation . . . . .	3
1.2	Dependency . . . . .	3
1.3	Quick Start . . . . .	3
1.4	Docs . . . . .	4
1.5	Example . . . . .	4
1.6	Test . . . . .	4
1.7	Misc . . . . .	5
1.7.1	Why did I create this framework? . . . . .	5
<b>2</b>	<b>Installation</b>	<b>7</b>
2.1	Stable release . . . . .	7
2.2	From sources . . . . .	7
<b>3</b>	<b>Tutorials</b>	<b>9</b>
3.1	Chapter 1: Set Up the Server . . . . .	9
3.2	Chapter 2: Static File Server . . . . .	10
3.3	Chapter 3: REST . . . . .	10
3.4	Chapter 4: Parameterized REST . . . . .	11
3.5	Chapter 5: Persistent Storage . . . . .	11
<b>4</b>	<b>How-to Examples</b>	<b>13</b>
<b>5</b>	<b>API References</b>	<b>15</b>
5.1	hyper2web package . . . . .	15
5.1.1	Submodules . . . . .	15
5.1.2	hyper2web.abstract module . . . . .	15
5.1.3	hyper2web.app module . . . . .	16
5.1.4	hyper2web.cli module . . . . .	17
5.1.5	hyper2web.exceptions module . . . . .	17
5.1.6	hyper2web.http module . . . . .	17
5.1.7	hyper2web.router module . . . . .	18
5.1.8	hyper2web.server module . . . . .	18
5.1.9	hyper2web.sslsocket module . . . . .	19
5.1.10	Module contents . . . . .	19
<b>6</b>	<b>Discussion</b>	<b>21</b>

<b>7</b>	<b>Contributing</b>	<b>23</b>
7.1	Types of Contributions . . . . .	23
7.1.1	Report Bugs . . . . .	23
7.1.2	Fix Bugs . . . . .	23
7.1.3	Implement Features . . . . .	23
7.1.4	Write Documentation . . . . .	24
7.1.5	Submit Feedback . . . . .	24
7.2	Get Started! . . . . .	24
7.3	Pull Request Guidelines . . . . .	25
7.4	Tips . . . . .	25
<b>8</b>	<b>Credits</b>	<b>27</b>
8.1	Development Lead . . . . .	27
8.2	Contributors . . . . .	27
<b>9</b>	<b>History</b>	<b>29</b>
9.1	0.0.0 (2017-06-08) . . . . .	29
<b>10</b>	<b>Indices and tables</b>	<b>31</b>
	<b>Python Module Index</b>	<b>33</b>

Contents:



Super Fast HTTP2 Framework for Progressive Web Application

## Installation

To install Hyper2Web, run this command in your terminal:

```
$ # due to a known issue, please install Curio manually
$ pip install git+https://github.com/dabeaz/curio.git
$ pip install hyper2web
```

This is the preferred method to install Hyper2Web, as it will always install the most recent stable release.

If you don't have `pip` installed, this [Python installation guide](#) can guide you through the process.

## Dependency

Python3.6

h2

curio

## Quick Start

Assuming you have a directory structure like:

```
your project/
--public/
  --index.html
```

```
--index.js
...
--app.py
```

Your `app.py` looks like

```
from hyper2web import app

if __name__ == '__main__':

    # A basic callback style API is provided
    # Function name is up to you
    async def post_echo(request, response):
        # Send the data received back to the client
        await response.send(request.stream.data)

    app = app.App(port=5000)
    app.post('name', post_echo)
    app.up()
```

Then run this script

```
$ python app.py
```

That's it!

If you just want to serve static files, it's just 2 lines!

```
from hyper2web import app
app.App(port=5000).up()
```

## Docs

Documentation is hosted on [hyper2web.readthedocs.io](https://hyper2web.readthedocs.io).

## Example

See the example folders for examples.

## Test

```
$ python -m unittest discover test
```

Run all tests under `test/` dir.



## Misc

### Why did I create this framework?

April 23rd, 2017, Sunday, I woke up and felt bored and decided to create my own HTTP2 web framework.

Since I had little or some prior web knowledge, this would be a super learning and fun project for me.



### Stable release

To install Hyper2Web, run this command in your terminal:

```
$ pip install hyper2web
```

This is the preferred method to install Hyper2Web, as it will always install the most recent stable release.

If you don't have [pip](#) installed, this [Python installation guide](#) can guide you through the process.

### From sources

The sources for Hyper2Web can be downloaded from the [Github repo](#).

You can either clone the public repository:

```
$ git clone git://github.com/CreatCodeBuild/hyper2web
```

Or download the [tarball](#):

```
$ curl -OL https://github.com/CreatCodeBuild/hyper2web/tarball/master
```

Once you have a copy of the source, you can install it with:

```
$ python setup.py install
```



## Chapter 1: Set Up the Server

In this tutorial, we will create a simple HTML5 game together. The game will teach you most aspects of our framework. We will only focus on backend. Frontend code will be provided.

Our framework works on both Linux and Windows systems. I will use Unix/Linux conventions/terms in this tutorial.

First, we need to create our project. Create a new directory/folder named `Game`. Under it, create a Python script named `app.py` and a directory named `public`.

`app.py` will contains all backend code and all frontend code will go into `./public` directory.

Now, put this piece of code in `app.py`.

```
from hyper2web import app

# let's only bind ip address to localhost for now. Later we will change it.
# port number is up to you. any number larger than 1000 should be fine.
app = app.App(address="localhost", port=5000)

# up() starts the server.
app.up()
```

Next, let's write the frontend. Create a `index.html` file in `./public`. Put this piece of code in it.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>The Game</title>
</head>
<body>
  Congratulations, you have set up the server correctly! <br>
  We will start to create our game next!
```

```
</body>
</html>
```

As you might know, Hyper2Web only uses HTTP/2. H2 uses HTTPS by default. Therefore, you need to have a pair of ssl keys in your top level directory. You can either generate your own keys or copy the key files in the [example](#). Copy and paste files with the name `localhost.*` to your Game directory.

Now, let's start the server. Open your terminal under Game directory and type

```
$ python app.py
```

Now open your browser and go to `https://localhost:5000`. You should be able to see the webpage you just wrote.

Congratulations! Now our server is running. The next chapter will teach you some basic RESTful routing.

## Chapter 2: Static File Server

Although you might want your App to be as dynamic as possible, you have to first understand how a static website is served.

The convention is to have a `public` directory under your project directory and have all your static files there.

For example, if the client does `GET /some_file.format`, the framework will try to read the path `public/some_file.format` and send what it read to the client. If the path does not exist, then the framework simply reports resource not found.

Therefore, you can easily just put all your frontend code under the public directory and start serving your application.

## Chapter 3: REST

Web is built on HTTP and HTTP is all about semantics. While there are thousands of ways to build HTTP API, the one which our framework embraces is REST. Since our audience's experience varies, I do not want to confuse you by explaining too much about REST. The only thing you need to know is that HTTP requests are all about semantics and a REST API is a semantic API.

Let's dive into the example code and you will understand it.

First, let's see the frontend code:

```
fetch('/top10', {method: 'GET'});
```

`fetch()` is the new browser API which does async HTTP requests. It is better than `XMLHttpRequest` in almost every aspects. It has a cleaner interface which fits RESTful API design.

This line creates a HTTP GET request with `:path = /top10`. How to respond to this request is 100% up to the server. Now, in `app.py`, write this piece of code:

```
# define API's callback function
async def top10_api(request, response):
    await response.send("some data")

# register this function
app.get("top10", top10_api)
# you can also do
# app.get("/top10", top10_api)
```

---

Now, whenever the client does such a request, `top10_api` will be run by the framework. We call this function the endpoint of a REST API. You can define the function as whatever you need, but have to include the `async` and `await` key words.

## **Chapter 4: Parameterized REST**

## **Chapter 5: Persistent Storage**





## CHAPTER 4

---

### How-to Examples

---



## hyper2web package

### Submodules

#### hyper2web.abstract module

This module defines all Abstract Types in the framework. Mainly for Better Organization of Code and For Better IDE Type Inferences

**class** `hyper2web.abstract.AbstractApp`

Bases: `object`

This class is the base class of any App classes. The framework provides an *App* class which implements all methods of this class.

A user can also implement this class if the user find that the *App* class is insufficient. But, the creator recommends user to extend *App* class instead of *AbstractApp* class.

**get** (*route: str, handler*)

**handle\_route** (*http, stream*)

This function has to be async.

**post** (*route: str, handler*)

**up** ()

**class** `hyper2web.abstract.AbstractHTTP`

Bases: `object`

The *HTTP* class implements this class.

All methods declared here are async.

**handle\_event** (*event*)

**send** (*stream\_id, headers, data*)

```
class hyper2web.abstract.AbstractRequest  
    Bases: object
```

```
class hyper2web.abstract.AbstractResponse  
    Bases: object
```

A Response should be constructed by the router. The router passes Stream information such as stream id to the Response object and Response object is passed to the end point function for top level users' use.

The flow control is handled by the HTTP class. Therefore, a send method always ends the response.

All send methods should call HTTP's send method

**send** (*data: bytes*)

send this response. :param headers: HTTP headers :param data: Body of the response. Has to be bytes  
That means, if you want to send some string, you have to convert the string to bytes. The framework does not do type conversion for you. It just fails if incorrect type is passed.

**send\_file** (*file\_path*)

**send\_status\_code** (*status\_code*)

```
class hyper2web.abstract.AbstractRouter  
    Bases: object
```

**handle\_route** (*http, stream*)

**register** (*method: str, route: str, handler*)

## hyper2web.app module

```
class hyper2web.app.App (address='0.0.0.0', port=5000, root='./public', auto_serve_static_file=True,  
                        default_file='index.html', router=<class 'hyper2web.router.Router'>)  
    Bases: hyper2web.abstract.AbstractApp
```

This class is the main class which users should be interact with.

This is the only class which users should construct.

**get** (*route: str, handler*)

Register a GET handler.

### Parameters

- **route** – A string which represent a RESTful route with optional parameters
- **handler** – A handler function. Has to be async.

**handle\_route** (*http: hyper2web.http.HTTP, stream: hyper2web.http.Stream*)

When the framework gets a incoming request, handle this request to corresponding routing handler.

Only used by the framework. Users should never call it.

**post** (*route: str, handler*)

The same as self.get except that it's for POST

**up** ()

Start the server. This is the last function users should call.

Users only call this function after set up all routing handlers.

```
hyper2web.app.default_get (app)
```

This function is the default handler for GET request whose :path is registered in the router.

To be more clear, a user does not have to register GET /index.html or GET /any\_static\_file.xxx. Any :path which is not found in the router will initiate this method.

This method treats all requests as a GET /static\_file. If :path is not a existing file path, it returns status code 404.

Users should not use this function.

```
hyper2web.app.get_index(app)
```

The default handler for GET /.

The default behavior for GET / is GET /index.html.

If a user specifies a default\_file in the constructor of App, the behavior becomes GET /default\_file

Users should not use this function.

## hyper2web.cli module

## hyper2web.exceptions module

Exceptions in hyper2web

```
exception hyper2web.exceptions.DifferentStreamIdException
```

Bases: Exception

```
exception hyper2web.exceptions.RouteNotRegisteredException
```

Bases: Exception

## hyper2web.http module

This module implements HTTP methods for end user

I currently think that they should be synchronized since they should not do IO Where as endpoint module is designed for IO

```
class hyper2web.http.HTTP(app: hyper2web.abstract.AbstractApp, sock, connection:
                             h2.connection.H2Connection)
```

Bases: *hyper2web.abstract.AbstractHTTP*

This class further implements complete HTTP2 on top of h2

```
data_received(event: h2.events.DataReceived)
```

Handle received data for a certain stream. Currently used for POST

```
handle_event(event: h2.events.Event)
```

```
request_received(event: h2.events.RequestReceived)
```

Handle a request

```
send(stream_id: int, headers, data: bytes = None)
```

send the response to the client :param stream\_id: the stream id associated with this request/response :param headers: HTTP headers. a sequence(tuple/list) of tuples

```
((:status', '200'), ('content-length', '0'), ('server', 'hyper2web'))
```

**Parameters data** – HTTP response body. Has to be bytes(binary data).

It's users' responsibility to encode any kinds of data to binary.

```
wait_for_flow_control(stream_id)
```

Blocks until the flow control window for a given stream is opened.

**window\_updated** (*event*)

Unblock streams waiting on flow control, if needed.

**class** `hyper2web.http.Request` (*stream, para*)

Bases: `hyper2web.abstract.AbstractRequest`

**class** `hyper2web.http.Response` (*stream\_id: int, http: hyper2web.http.HTTP*)

Bases: `hyper2web.abstract.AbstractResponse`

**send** (*data: bytes*)

**send\_file** (*file\_path*)

**send\_status\_code** (*status\_code*)

**set\_header** (*field, value*)

**set\_headers** (*headers*)

**update\_headers** (*headers*)

**class** `hyper2web.http.Stream` (*stream\_id: int, headers: dict*)

Bases: `object`

As the code is right now, many stream implementation is done in `endpoint.EndpointHandler`. Am moving those functionality to this class

The current design is that application will only return complete stream to top level api. But, since a user might also want to program on a live stream. For example, the client may send a giant file 1GB, the user will want to write this stream to disk in real time. Also, buffering 1GB in memory is kind of stupid.

But nonethelss, the current focus is on better organization of code instead of more API or performance.

**finalize** ()

concat all data chunks in this handler to one bytes object

**update** (*event: h2.events.DataReceived*)

assume only POST stream will call this one

## hyper2web.router module

**class** `hyper2web.router.Router` (*default\_get*)

Bases: `hyper2web.abstract.AbstractRouter`

User should never construct Router

**find\_match** (*path: str*)

'user/{userId}' should match 'user/abc' `userId = abc` return a tuple (matched, parameters) matched is the route which matches the incoming path parameters is a dict of parameters and their values

**handle\_route** (*http: hyper2web.http.HTTP, stream: hyper2web.http.Stream*)

**register** (*method: str, route: str, handler*)

## hyper2web.server module

A fully-functional HTTP/2 server written for curio.

Requires Python 3.5+.

```
class hyper2web.server.H2Server (sock, app: hyper2web.abstract.AbstractApp)
    Bases: object
```

This class just connects to socket and that's about it. Most heavy lifting is done by http.HTTP

```
run ()
    Loop over the connection, managing it appropriately.
```

```
hyper2web.server.h2_server (address, certfile, keyfile, app: hyper2web.abstract.AbstractApp)
    Create an HTTP/2 server at the given address.
```

## hyper2web.sslsocket module

```
hyper2web.sslsocket.create_listening_ssl_socket (address, certfile, keyfile)
    Create and return a listening TLS socket on a given address.
```

## Module contents





## CHAPTER 6

---

### Discussion

---



Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given. You can contribute in many ways:

### Types of Contributions

#### Report Bugs

Report bugs at <https://github.com/CreatCodeBuild/hyper2web/issues>.

If you are 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.

#### Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with “bug” and “help wanted” is open to whoever wants to implement it.

#### Implement Features

Look through the GitHub issues for features. Anything tagged with “enhancement” and “help wanted” is open to whoever wants to implement it.

## Write Documentation

Hyper2Web could always use more documentation, whether as part of the official Hyper2Web docs, in docstrings, or even on the web in blog posts, articles, and such.

## Submit Feedback

The best way to send feedback is to file an issue at <https://github.com/CreteCodeBuild/hyper2web/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 contributions are welcome :)

## Get Started!

Ready to contribute? Here's how to set up *hyper2web* for local development.

1. Fork the *hyper2web* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/hyper2web.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv hyper2web
$ cd hyper2web/
$ python setup.py develop
```

4. Create a branch for local development:

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

Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
$ flake8 hyper2web tests
$ python setup.py test or py.test
$ tox
```

To get flake8 and tox, just pip install them into your virtualenv.

6. 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
```

7. Submit a pull request through the GitHub website.

## Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 2.6, 2.7, 3.3, 3.4 and 3.5, and for PyPy. Check [https://travis-ci.org/CreatCodeBuild/hyper2web/pull\\_requests](https://travis-ci.org/CreatCodeBuild/hyper2web/pull_requests) and make sure that the tests pass for all supported Python versions.

## Tips

To run a subset of tests:

```
$ py.test tests.test_hyper2web
```



## CHAPTER 8

---

### Credits

---

#### Development Lead

- Xuanzhe Wang <wangxuanzhealbert@gmail.com>

#### Contributors

None yet. Why not be the first?





## CHAPTER 9

---

### History

---

#### **0.0.0 (2017-06-08)**

- First release on PyPI.



## CHAPTER 10

---

### Indices and tables

---

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



### h

- `hyper2web`, [19](#)
- `hyper2web.abstract`, [15](#)
- `hyper2web.app`, [16](#)
- `hyper2web.cli`, [17](#)
- `hyper2web.exceptions`, [17](#)
- `hyper2web.http`, [17](#)
- `hyper2web.router`, [18](#)
- `hyper2web.server`, [18](#)
- `hyper2web.sslsocket`, [19](#)



## A

AbstractApp (class in hyper2web.abstract), 15  
AbstractHTTP (class in hyper2web.abstract), 15  
AbstractRequest (class in hyper2web.abstract), 15  
AbstractResponse (class in hyper2web.abstract), 16  
AbstractRouter (class in hyper2web.abstract), 16  
App (class in hyper2web.app), 16

## C

create\_listening\_ssl\_socket() (in module hyper2web.sslsocket), 19

## D

data\_received() (hyper2web.http.HTTP method), 17  
default\_get() (in module hyper2web.app), 16  
DifferentStreamIdException, 17

## F

finalize() (hyper2web.http.Stream method), 18  
find\_match() (hyper2web.router.Router method), 18

## G

get() (hyper2web.abstract.AbstractApp method), 15  
get() (hyper2web.app.App method), 16  
get\_index() (in module hyper2web.app), 17

## H

h2\_server() (in module hyper2web.server), 19  
H2Server (class in hyper2web.server), 18  
handle\_event() (hyper2web.abstract.AbstractHTTP method), 15  
handle\_event() (hyper2web.http.HTTP method), 17  
handle\_route() (hyper2web.abstract.AbstractApp method), 15  
handle\_route() (hyper2web.abstract.AbstractRouter method), 16  
handle\_route() (hyper2web.app.App method), 16  
handle\_route() (hyper2web.router.Router method), 18  
HTTP (class in hyper2web.http), 17

hyper2web (module), 19  
hyper2web.abstract (module), 15  
hyper2web.app (module), 16  
hyper2web.cli (module), 17  
hyper2web.exceptions (module), 17  
hyper2web.http (module), 17  
hyper2web.router (module), 18  
hyper2web.server (module), 18  
hyper2web.sslsocket (module), 19

## P

post() (hyper2web.abstract.AbstractApp method), 15  
post() (hyper2web.app.App method), 16

## R

register() (hyper2web.abstract.AbstractRouter method), 16  
register() (hyper2web.router.Router method), 18  
Request (class in hyper2web.http), 18  
request\_received() (hyper2web.http.HTTP method), 17  
Response (class in hyper2web.http), 18  
RouteNotRegisteredException, 17  
Router (class in hyper2web.router), 18  
run() (hyper2web.server.H2Server method), 19

## S

send() (hyper2web.abstract.AbstractHTTP method), 15  
send() (hyper2web.abstract.AbstractResponse method), 16  
send() (hyper2web.http.HTTP method), 17  
send() (hyper2web.http.Response method), 18  
send\_file() (hyper2web.abstract.AbstractResponse method), 16  
send\_file() (hyper2web.http.Response method), 18  
send\_status\_code() (hyper2web.abstract.AbstractResponse method), 16  
send\_status\_code() (hyper2web.http.Response method), 18

`set_header()` (`hyper2web.http.Response` method), [18](#)  
`set_headers()` (`hyper2web.http.Response` method), [18](#)  
`Stream` (class in `hyper2web.http`), [18](#)

## U

`up()` (`hyper2web.abstract.AbstractApp` method), [15](#)  
`up()` (`hyper2web.app.App` method), [16](#)  
`update()` (`hyper2web.http.Stream` method), [18](#)  
`update_headers()` (`hyper2web.http.Response` method), [18](#)

## W

`wait_for_flow_control()` (`hyper2web.http.HTTP` method),  
[17](#)  
`window_updated()` (`hyper2web.http.HTTP` method), [17](#)