# slimHTTP

***Release v1.0-beta.002***

**Nov 08, 2020**

# Programming Guide

**slimHTTP** is a simple, minimal and flexible HTTP server.
It supports REST api routes, WebSocket[1] traffic and native Python imports as vhost endpoints.

Here's a demo using minimal setup:

```
import slimHTTP

http = slimHTTP.server(slimHTTP.HTTP)
http.run()
```

Some of the features of slimHTTP are:

- **No external dependencies or installation requirements.** Runs without any external requirements or installation processes.
- **Single threaded.** slimHTTP takes advantage of *select.epoll() (select.select() on Windows)* to achieve blazing speeds without threading the service. Threads are allowed and welcome, but the core code relies on using as few threads and overhead as possible.

[1] WebSocket support is provided by using a *@app.on_upgrade* hook and parsed by a separate library, like spiderWeb

CHAPTER 1

# Installation

**Note:** These instructions apply to slimHTTP .

slimHTTP is a pure python library, so no special steps are required for installation. You can install it in a variety of ways described below though for your convenience.

## 1.1 Using *pip*

```
pip install slimHTTP
```

## 1.2 Clone using *git*

```
git clone https://github.com/Torxed/slimHTTP.git
```

But most likely you'll want to submodule this in a project. To do that, I would recommend not following master as it's actively developed. Any release/tag should be good enough for production.

```
cd project/dependencies
git submodule add -b v1.0 https://github.com/Torxed/slimHTTP.git
```

Which would follow the stable release branch of *v1.0* where tests *should* be done before release.

## 1.3 Manually unpacking source

The source code archives *(including git)* include examples. Archives are available on Github:

```
unzip slimHTTP-x.x.x.zip
cd slimHTTP-x.x.x
python examples/http_server.py
```

CHAPTER 2

# Configuration

Configuration is done by supplying slimHTTP with a *dict* of options.
A complete example can be found under *Example configuration*.

**Warning:**

There's startup-sensitive configuration options.
Those are *addr* and *port* to set the listening interface.

To delcare *addr* and *port* - you have to do it from the startup code:

```
import slimHTTP

http = slimHTTP.server(slimHTTP.HTTP, addr='127.0.0.1', port=8080)
http.run()
```

Trying to set it in the runtime configuration will fail, as the server has already setup the *socket.bind((addr, port))*

**Note:** All following config options are runtime friendly, they can be changed whenever during normal operation without needing to reload the server. The format for the configuration is a valid python *dict*:

```
{
    'key-one' : 'value',
    'key-two' : 'value'
}
```

Where the *key* is any of the below options, and the value is whatever corresponds to that particular key or option.

## 2.1 Example configuration

```
import slimHTTP

http = slimHTTP.server(slimHTTP.HTTP)

@http.configuration
def config(instance):
    return {
        'web_root' : './vhosts/default',
        'index' : 'index.html',
        'vhosts' : {
            'hvornum.se' : {
                'web_root' : './vhosts/hvornum.se',
                'index' : 'index.html'
            },
            'slimhttp.hvornum.se' : {
                'module' : './vhosts/internal_tests/vhost.py'
            }
        }
    }

http.run()
```

Here, configuration changes after the server has finished starting up.
The same configuration *could* be given on startup, but is not mandatory.

The configuration changes the default web-root as well as some minor changes
to *vhost* specific resources.

## 2.2 Global configuration options

Below follows some of the configuration options that are available at all configuration levels.
These can there for be set in *vhost* scope as well as the *base*/*global* scope.

### 2.2.1 web_root

As all other variables, Web roots can be configured in the global and *vhost* scope.
The paths them selves can be relative or absolute, they will be resolved in runtime.

```
{'web_root' : './path'}
```

### 2.2.2 index

*index* can be either a single *str* of a filename, or
it can be a *list* of files in which slimHTTP will try them in cronological order.

```
{'index' : ['index.html', 'main.py']}
```

## 2.3 Vhost specific configuration

### 2.3.1 vhosts

*vhosts* key should be placed in the *base* configuration and be directly followed by a *key* representing the name of the domain *(FQDN)* that slimHTTP should react to.
And the value should be a *dict* containing any valid slimHTTP configuration.
For instance, for the *FQDN* https://slimhttp.hvornum.se/ the config would be:

```
{
    'vhosts' : {
        'slimhttp.hvornum.se' : {
            // config options for slimhttp.hvornum.se
        }
    }
}
```

Where the configuration specifics for that domain would be placed instead of the "comment".
for instance *'index' : 'index.html'* could be added.

### 2.3.2 module

**Note:**

module mode is also activated when a client requests a URL that ends with *.py*.

The *module* is a key which can tell slimHTTP that instead of using *reverse proxy* mode or a normal *look for a index* mode.
slimHTTP should import the script in question, and return the data given by that module. Here's an example:

```
{
    'vhosts' : {
        'slimhttp.hvornum.se' : {
            'module' : './vhosts/hvornum.se/vhost_slimhttp.py'
        }
    }
}
```

The exact structure of the module can be anything.
But there are two main entry functions slimHTTP will look for.

**Warning:**

The module is **reloaded** each request.
This means that persistant data or information has to be stored away on each request.
To use a in-memory storage, you *could* altho not recommended, use something like this in *vhost_slimhttp.py* from the above example.

```
if not 'MyMemStorage' in __builtins__: __builtins__['MyMemStorage'] = {}
if not 'counter' in MyMemStorage: MyMemStorage['counter'] = 0

print(f"The module ran with counter value {MyMemStorage['counter']}. Incremeting␣
↪value!"")

MyMemStorage['counter'] += 1
```

Or you could use pickle.dumps or a database to store the data you need between sessions. Although they will be a bit slower considering they're not working within the application memory space.

### on_request

**if the function** *on_request* is defined *(using 'hasattr('on_request', <module>)')*, slimHTTP will automatically call it upon each request to that vhost.

**Warning:** if *@app.route('/...', vhost='example.com')* is defined, that will take precedence over the *on_request* **if** *on_request* returns data. Otherwise the *@app.route* will be a fallback.

### @app.route

It's possible to set up *vhost* specific routes. These acts as normal *REST*-like endpoints.
The key difference is that *@app.route* takes a additional keyword, *vhost=:str*. And to access it, you need to get the current server instance so you can decorate it.

```
import slimHTTP

http = slimHTTP.instances[':80']

@http.route('/', vhost='example.com')
def route_handler(request).
    print(request)
```

This will server */* but only for the given *vhost*.
And this could serve as a entry-point for vhost specific modules.

**Note:** Note that the instance depends on the *addr* and *port* used, a *"listening on every interface on port 80"* would be *:80* in this case.

### 2.3.3 proxy

Reverse proxy support can be enabled in any vhost.
The reverse proxy will kick in once a valid HTTP header with the *Host: <host>* field defined.
Upon which slimHTTP will switch from a `HTTP_REQUEST` to a `HTTP_PROXY_REQUEST`.

**Warning:** The `HTTP_REQUEST` object has two pitfalls. One, if the proxy is slow to respond all concurrent HTTP requests to slimHTTP will become slow, since we're single threaded, it means that the proxy response has to be parsed in full before other requests can come in. The second pitfall being Issue #11.

```
{
    'vhosts' : {
        'internal.hvornum.se' : {
            'proxy' : '192.168.10.10:80',
            'ssl' : False
        }
    }
}
```

Here, *http://internal.hvornum.se* requests are proxied down to *192.168.10.10* on port *80*.

**Note:** The *'ssl' : False'* is optional and the default behavior.

CHAPTER 3

# Websockets

WebSockets are supported by slimHTTP, but enabled by a plugin.
You'll need to install slimWS one way or another.
After that, simply plug in the upgrader to slimHTTP:

```
import slimHTTP
import slimWS

http = slimHTTP.host(slimHTTP.HTTP)
websocket = spiderWeb.WebSocket()

@http.on_upgrade
def upgrade(request):
    new_identity = websocket.WS_CLIENT_IDENTITY(request)
    new_identity.upgrade(request) # Sends Upgrade request to client
    return new_identity

http.run()
```

**Note:** slimWS has a rudimentary API support, which can be viewed on the slimWS documentation.

The following example will catch any Connection: upgrade request,
and then proceed to in-memory replace the `HTTP_CLIENT_IDENTITY` with a slimWS.WS_CLIENT_IDENTITY.

Identities are usually one-shot-sessions, but since WebSockets in general are a session based connection, the slimWS.WS_CLIENT_IDENTITY persists over requests - as there are no *socket.close()* event for that protocol.
slimHTTP honors the *keep-alive* in the identity and doesn't touch the socket after each response.

CHAPTER 4

# Basic example

As shown in the overview on GitHub, the most basic example would be:

```
import slimHTTP

http = slimHTTP.host(slimHTTP.HTTP)
http.run()
```

Which uses the .. _config.default: configuration.

# CHAPTER 5

# REST

By leveraging *@app.route* we can setup mock endpoints.
These endpoints will get one parameter, the `HTTP_REQUEST` object.

**Warning:** The following example is for non-vhost entries. This is useful for simpel setups. Read below for a *REST* Vhost option.

```
@http.route('/')
def main_entry(request):
    print(request.headers)

    return request.build_headers() + b'<html><body>Test body</body></html>
```

This is a minimal example of how to respond with some default basic headers and a default content.

## 5.1 Methods and headers

Unlike many other frameworks, slimHTTP does not currently support *method='POST'* filtering
in the *@http.route* functionality. Instead, the *method* is given or found in *request.method*
in each request object *(or for the raw request data, also in 'request.headers[b"METHOD"]').*

An example to react to *PUT* requests:

```
@http.route('/')
def main_entry(request):
    if request.method == 'PUT':
        print('We got a PUT request with headers:', request.headers)
```

## 5.2 REST with Virtual Hosts

When creating virtual hosts in your configuration, the router needs to know that you want to insert a route to a specific virtual host. Which can be done by doing the following:

> **Warning:** You first need to grab the *http* instance object, since virtual host entry-points are usually defined in a separate file from where the *http* variable was created.
>
> This example also shows you how to grab that instance.

```
import slimHTTP

http = slimHTTP.instances[':80']

@http.route('/', vhost='example.com')
def main_entry(request):
    print(request.headers)

    return request.build_headers() + b'<html><body>Test body</body></html>
```

This example will not trigger on the default hosted site, but instead only trigger on the web-root of *example.com* in this example.

## 5.3 REST with JSON

By default, slimHTTP will *try* to parse incoming data labled with *Content-Type: application/json* as JSON. But ultimately it's up to the developer to verify.

To convert and work with the request data, you could do something along the lines of:

```
@http.route('/')
def main_entry(request):
    data = json.loads(request.payload.decode('UTF-8'))
    print(data['key'])
```

And to respond, you could build ontop of it by doing:

```
@http.route('/')
def main_entry(request):
    data = json.loads(request.payload.decode('UTF-8'))
    print(data['key'])

    return request.build_headers({'Content-Type' : 'application/json'}) + bytes(json.
↪dumps({"key" : "a value"}, 'UTF-8')
```

Which would instruct slimHTTP to build a basic header response with one additional header, the *Content-Type* and utilize json.dumps to dump a dictionary structure.

**Note:**

But a more future proof way would be to use the ~slimHTTP.HTTP_RESPONSE object as a return value.
This enables you to avoid building the headers yourself as well as concatinate the payload and format it.

CHAPTER 6

# Virtual Hosts

**Note:** SNI is Currently, as of v1.0.1rc3, not supported

slimHTTP supports working with hosts.
The vhosts have three different modes, which we'll try to explain here.

## 6.1 Static content mode

Normal operation mode for slimHTTP is to statically deliver anything under *web_root* using *index* whenever directory listing is attempted.
This mode is there for the **default** unless no other mode is specified, and thus one configuration option is required, and that is _web_root.

```
import slimHTTP

http = slimHTTP.server(slimHTTP.HTTP)

@http.configuration
def config(instance):
    return {
        'vhosts' : {
            'slimhttp.hvornum.se' : {
                'web_root' : './vhosts/hvornum.se'
            }
        }
    }
```

This will deliver anything under *./vhosts/hvornum.se* and jail all requests to that folder[1].

## 6.2 reverse proxy mode

To configure a reverse proxy, the proxy definitions **must** consist of two things,
an *addr* and a *port* in the format: *"addr:port"*.
A simple example would be:

```
import slimHTTP

http = slimHTTP.server(slimHTTP.HTTP)

@http.configuration
def config(instance):
    return {
        'vhosts' : {
            'internal.hvornum.se' : {
                'proxy' : '192.168.10.10:80'
            }
        }
    }
```

Which will allow outside clients to connect to a internal resource on the *192.168.10.10* IP, via slimHTTP.

**Note:** There's an optional flag to *proxy* definitions, which can be seen under _modules.

## 6.3 module mode

The module is a special python import mechanic.
It supports absolute or relative paths to a module.
The module itself will be *import <module>* imported with a bit of trickery.

Some more information regarding module entry points can be found under _modules.
But to specify a vhost as a module, simply configure the following:

```
import slimHTTP

http = slimHTTP.server(slimHTTP.HTTP)

@http.configuration
def config(instance):
```

(continues on next page)

[1] Security issues aside.

(continued from previous page)

```
    return {
        'vhosts' : {
            'slimhttp.hvornum.se' : {
                'module' : './vhosts/hvornum.se/vhost_slimhttp.py'
            }
        }
    }
```

**Note:**

module mode is also activated when a client requests a URL that ends with *.py*.

### 6.3.1 Entry point

There's no requirements on the module itself.
It can be any valid Python code and it will be executed as if someone did *import module*. However, there are a optional entry point.

```
def on_request(request).
    print(request)
```

*on_request* will be called if it's defined, otherwise it won't.
To access current service instances for decorators, simply import slimHTTP and access the ~slimHTTP.instances.

```
import slimHTTP
print(slimHTTP.instances)

http = slimHTTP.instances[':80']

@http.route('/', vhost='example.com')
def handler(request):
    print(request)
```

**Warning:** Just make sure you define a *vhost*=..., otherwise you'll replace the default context handler.

CHAPTER 7

# Discord

There's a discord channel which is frequent by the contributors.

To join the server, head over to discord.com/slimHTTP and join in. There's not many rules other than common sense and treat others with respect.

There's the *@Party Animals* role if you want notifications of new releases which is posted in the *#Release Party* channel. Another thing is the *@Contributors* role which you can get by writing *!verify* and verify that you're a contributor.

Hop in, I hope to see you there! : )

CHAPTER 8

# Issue tracker

Issues whould be reported over at GitHub/issues.

General questions, enhancements and security issues can be reported over there too. For quick issues or if you need help, head over the to the Discord.