# restAPY Documentation

***Release latest***

**Jun 30, 2020**

# Contents

# CHAPTER 1

# Introduction

RestAPY is a module made to easily create Rest APIs in Python.

It therefore can be used to make JSON data from your projects easily accessible through a webbrowser or through API requests.

**Note**: This documentation contains all the useful information about version 1.2.0

# CHAPTER 2

# Dependencies

All of the modules restAPY is based on are part of the default python install and therefore do not need to be installed seperately. The following list contains all of those modules:

- socket
- threading
- json

# CHAPTER 3

# Code example

The following example will first create an API instance that will be accessible through the URL "http://localhost:8000/".

Then it will tell the API what data to return when the user requests this domain at the root path ("/")

After which the API starts listening for incoming connections.

```
import restAPY

api = restAPY.API(8000, “localhost”)

api.set_path(“/”, {“celsius”:5, “fahrenheit”:41})

api.run()
```

CHAPTER 4

# Methods in depth

## 4.1 API class

The API class is the main building block of restAPY as it contains all of the necessary code to configure and start your rest APIs

An instance of it can be created like this:

> api = restAPY.API(port, url)

Both the port and the url argument are optional. Their default values are:

- port = 80
- url = "0.0.0.0"

This means that any http connection to the API's server will be treated as a request.

## 4.2 Configuring the API

### 4.2.1 Debugging (on by default)

This will print events like invalid invalid requests to the terminal. (Turn off in deployed version)

> api.debug = False

To then test if your API works you can either put the link to it into your webbrowser or run the following command inside your terminal:

> curl [link]

### 4.2.2 Configuring what data gets send/processed (set_path)

To tell the API what data to return to the user after they made a request the set_path method is used.

#### Static Data

The function takes a path (String) and JSON compatible data (arrays, dicts, numbers, strings) as arguments and tells the API to return the given data when the giben path is requested.

```
data = {“celsius”:5, “fahrenheit”:41}

api.set_path(“/data”, data)
```

This example would tell the API to return the value of the data variable when path /data is requested (http://domain/data)

#### Dynamic Data (status code: “200 OK”)

When you want the API to return dynamic data (for example when it receives a POST request) you put still use the set_path function, but it now takes a function as an argument instead of the returnable data itself.

```
import json

def foo(request):

    if request[“Type”] == “POST”:

        return json.loads(request[“JSON”])

    elif request[“Type”] == “GET”:

        return [1,2,3,4]

api.set_path(“/dynamic”, foo)
```

In this case the API will return all JSON information about the HTTP request when a POST-Request is made, whilst just returning [1,2,3,4] when a GET-Request is made.

**NOTE1**: The data the function returns needs to be convertible into JSON.

**NOTE2**: Do NOT put the “()” after the function name when giving a function as an argument to set_path()

**NOTE3**: Your function needs to take request as an argument as this variable will contain all the information about the request the user made

#### Dynamic Data with custom status code

Functions that dynamically work with requests, and also return status codes other than “200 OK”, work just as those stated above regarding their arguments aswell as its connection to an url. The only difference is their return value. Instead of just returning the JSON data a dictionary with the same structure as in the following example needs to be returned:

```
{ “http_status_code” : “200 OK”, “response_content” : [1,2,3,4] }
```

### 4.2.3 Configuring how the data is presented

#### Indentation

To make the JSON response from the API more readable the default indentation of it is set to 4. This can be adjusted as follows:

```
api.json_indent = integer_value
```

### Sorting the response

By default the JSON response from the API is not sorted. This can be change by doing the following:

api.sort_json = True

**NOTE:** This can cause complications when using datatypes that can't be compared with each other like Strings and Integers

## 4.2.4 Configuring the APIs network settings

### Changing the APIs port

api.port = new_port_number

### Changing the APIs url

api.url = new_url_string

### Changing the maximum number of connections

By default the API can handle 16 simultaneous connections. To change this you can do the following:

api.max_connections = new_connection_limit_integer

## 4.2.5 Encryption (HTTPS only)

The following steps need to be taken to set up encryption

**Activate encryption (off by default)**

api.use_tls = True

**Set cerificate (.cert or .pem)**

api.certchain = "path/to/certificate.cert"

**Set private key**

api.privkey = "path/to/key.pem"

**Turn HTTP Redirect on/off (on by default)**

api.redirect_http = True

**Choose the https port (443 by default)**

api.https_port = 443