Retica

Release 1.0.0

Cyrocoders

Oct 29, 2022

CONTENTS:

CHAPTER

ONE

INTRODUCTION TO RETICA

1.1 Installing Retica

The easiest way to install Retica is to use the pip command line tool.

```
$~ pip install retica
```

1.2 Creating A Retica Server

Once you have installed Retica, you can import it into your Python environment. The server class is used to create a server, in which you can add endpoints(locations) and open HTTP(s) ports.

```
import Retica
retica = Retica.Server(__name__)
```

1.3 Creating An Endpoint

Endpoints are functions that are assigned to a location and are called when a request is made to that location.

```
@retica.create_endpoint("/hello/{name}")
def index(request: Retica.Request.request, response: Retica.Response.response, **data):
    response.body = f"Hello {data['name']}"
```

1.4 Creating A Socket

Sockets are used to create a server that listens for incoming connections. The server will listen on the specified port and host. Sockets can use 2 protocols:

1. HTTP
2. HTTPS (Certificate & key files are required)

- You can also create your own protocols(In Development).

```
http_socket = Retica.Sockets.HTTP_Socket("localhost", 80)
https_socket = Retica.Sockets.HTTPS_Socket("localhost", 443, "cert.pem", "key.pem")
```

1.5 Running the Server

To run the server, you must call the run method on the server. An array of sockets should be passed in as an argument.

```
if __name__ == "__main__":
    retica.run([
        http_socket,
        https_socket
    ])
```

1.6 Boilerplate

This is the boilerplate code that you will need to create your own server.

```
import Retica
retica = Retica.Server(__name__)

@retica.create_endpoint("/hello/{name}")
def index(request: Retica.Request.request, response: Retica.Response.response, **data):
    response.body = f"Hello {data['name']}"

http_socket = Retica.Sockets.HTTP_Socket(Retica.Sockets.gethostname(), 80)

if __name__ == "__main__":
    retica.run([http_socket])
```

CHAPTER

TWO

RETICA REQUEST CLASS

The Request class is used to build & parse requests.

2.1 Creating A Request

```
import Retica

request = Retica.Request.request()
```

2.2 Parsing A Request

An encoded string can be parsed and stored into the request object using the `request.parse()` method.4

```
>>> request.parse('GET /?name=Retica HTTP/1.1\r\nHost: www.example.com\r\n\r\npost=1&
↪name=Retica')
>>> print(type(request))
<class 'Retica.Request.request'>
```

2.3 Getting Data From The Request

The Request class has the following attributes:

1. **method** - The status of the request.
2. **path** - The body of the request.
3. **protocol** - The protocol of the request.
4. **headers** - The headers of the request.
5. **data** - The data of the request (Changing this variable doesn't modify the compiled response).

```
>>> print(request.method)
GET
>>> print(request.path)
/
>>> print(request.protocol)
HTTP/1.1
```

(continues on next page)

(continued from previous page)

```
>>> print(request.headers)
{'Host': 'www.example.com'}
>>> print(request.data['GET'])
{'name': 'Retica'}
>>> print(request.data['POST'])
{'post': '1', 'name': 'Retica'}
>>> print(request.data['COOKIE'])
{}
```

2.4 Compiling The Request Into A String

The request can be compiled using the `request.compile()` method.

```
>>> request.compile()
'GET /?name=Retica HTTP/1.1\r\nHost: www.example.com\r\n\r\npost=1&name=Retica'
```

CHAPTER

THREE

RETICA RESPONSE CLASS

The Response class is used to build & parse responses.

3.1 Creating A Response

```
import Retica

response = Retica.Response.response()
```

3.2 Modifying The Response

The Response class has the following attributes:

1. **status** - The status of the response.
2. **headers** - The headers of the response.
3. **body** - The body of the response.
4. **protocol** - The protocol of the response.
5. **content_type** - The content type of the response.
6. **connection** - The connection status after the response has been received.

```
response.status = 200
response.headers = {'set-cookie': 'session=123456789'}
response.body = '<h1>Hello World!</h1>'
response.protocol = 'HTTP/1.1'
response.content_type = 'text/html'
response.connection = 'keep-alive'
```

3.3 Parsing A Response

An encoded string can be parsed and stored into the response object using the `response.parse()` method.4

```
>>> response.parse('HTTP/1.1 200 OK\r\nContent-Type: text/html\r\nConnection: keep-alive\
↪r\nSet-Cookie: session=123456789\r\n\r\n<h1>Hello World!</h1>')
>>> print(type(response))
```

3.4 Compiling The Response Into A String

The response can be compiled using the `response.compile()` method.

```
>>> response.compile()
'HTTP/1.1 200 OK\r\nContent-Type: text/html\r\nConnection: keep-alive\r\nSet-Cookie:␣
↪session=123456789\r\n\r\n<h1>Hello World!</h1>'
```