Protocols of IoT: HTTP (Interactive Session)

e-Yantra Team

Embedded Real-Time Systems (ERTS) Lab
Indian Institute of Technology, Bombay

IIT Bombay
February 10, 2021

Agenda for Discussion

1. HTTP with Examples
2. requests Library
3. Implementing APIs
4. Thingsboard API

HTTP with Examples

Request Message Format

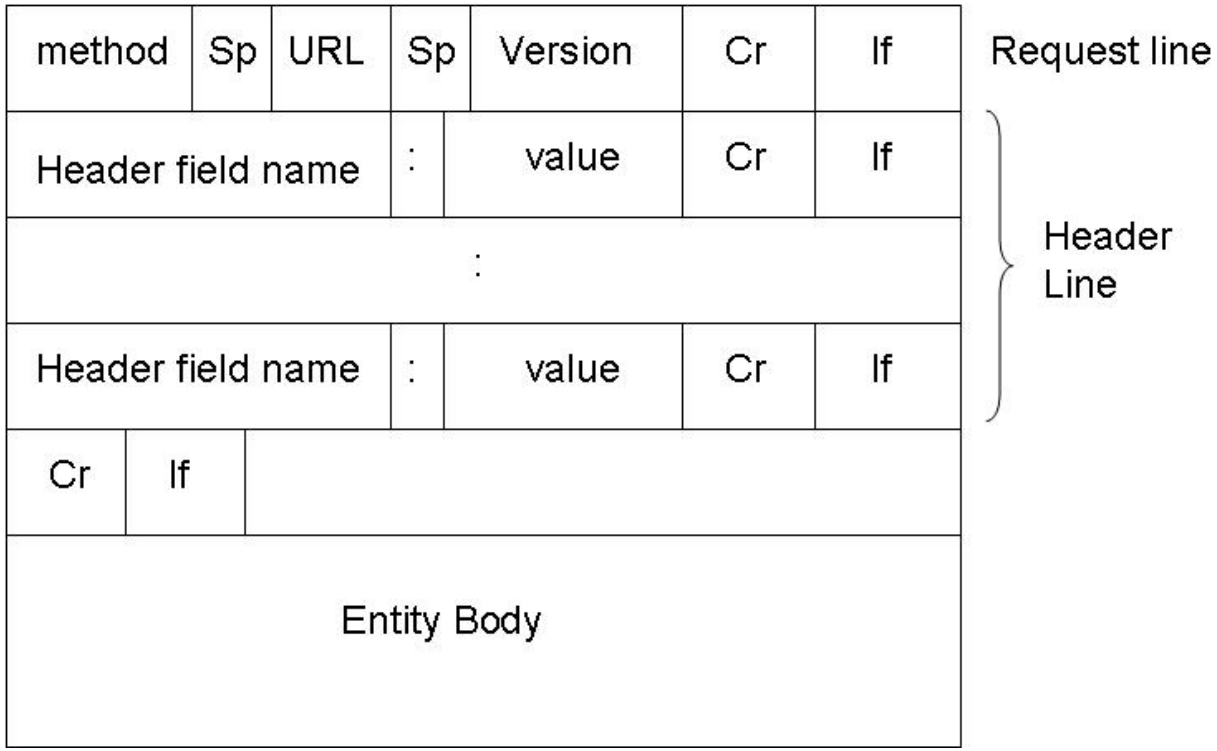

Figure 1: HTTP Request Message

Request Message Format

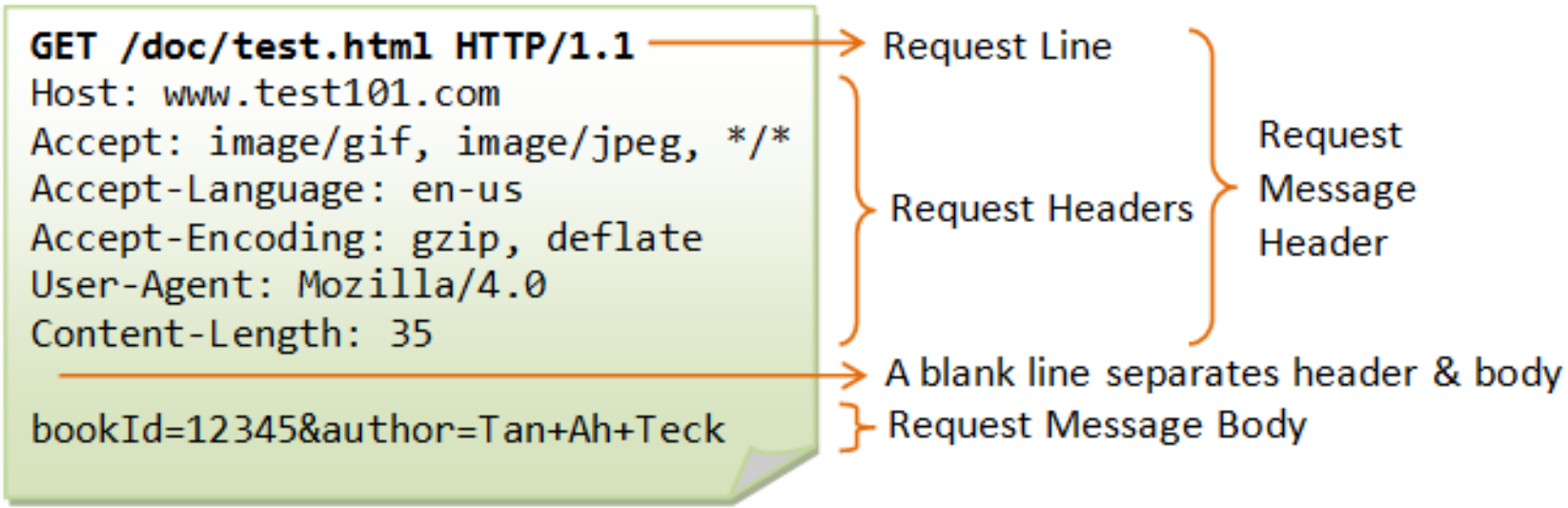

Figure 2: HTTP Request Message

Response Message Format

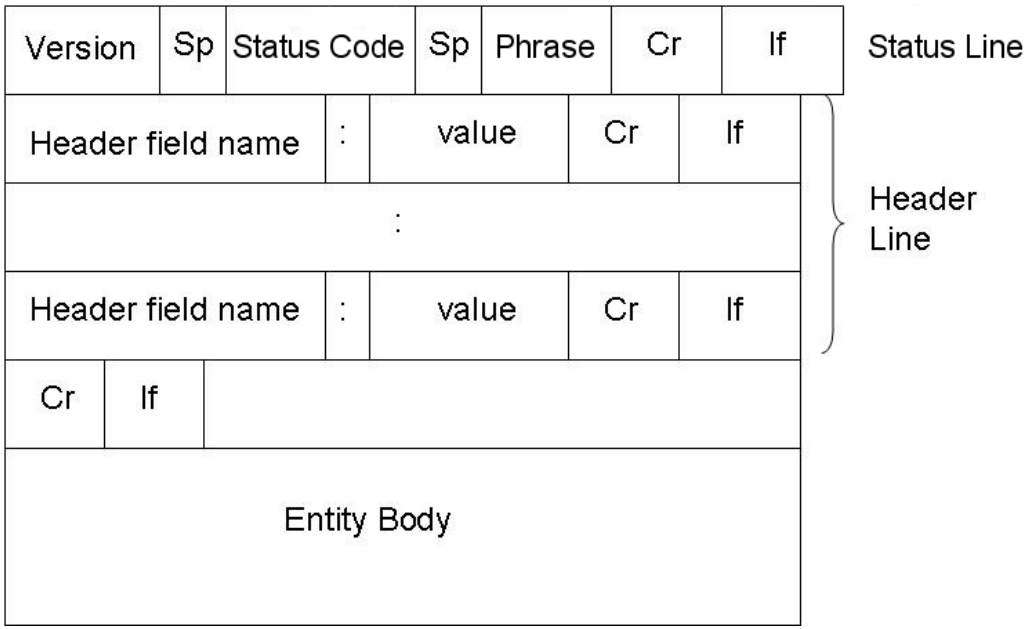

Figure 3: HTTP Response Message

Response Message Format

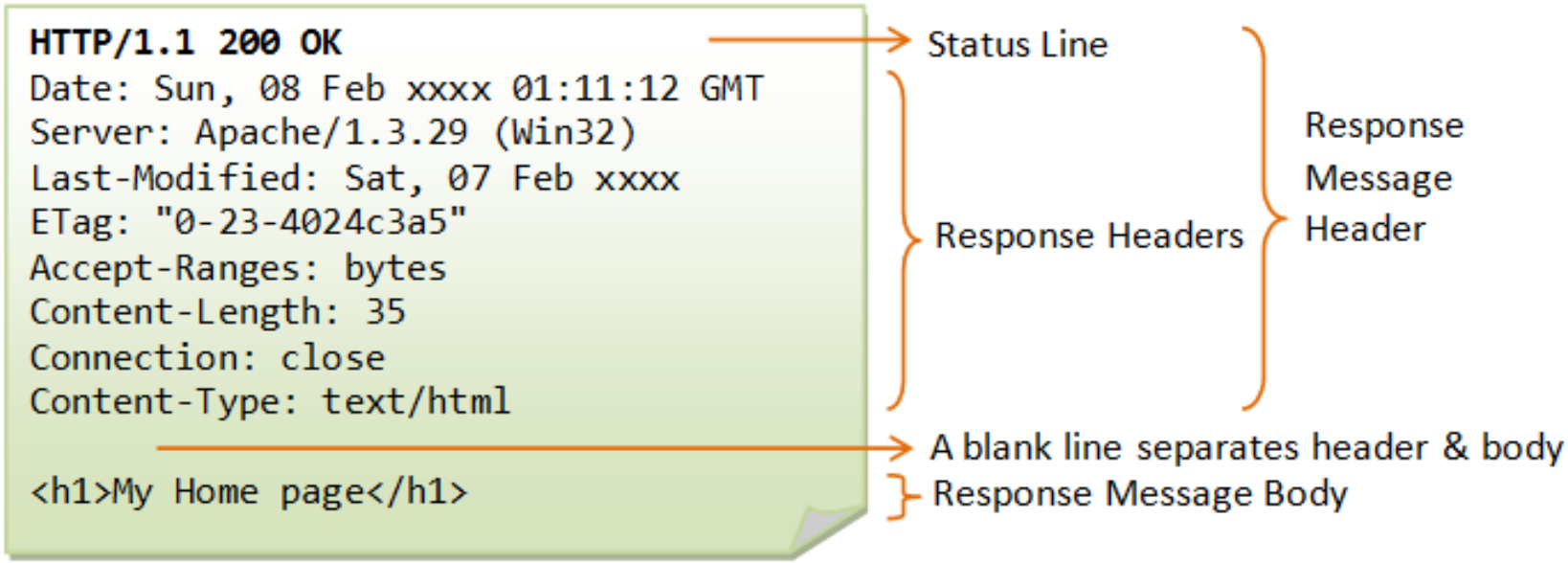

Figure 4: HTTP Response Message

Let's see HTTP messages

Open your terminal and type

```
curl -v www.google.com
```

Parts of a resource locator

scheme://authority:port/path?query#fragment

1. Scheme (http or https)
2. Authority (www.google.com)
3. Port (80, 8080, 3001, etc.)
4. Path
5. Query (key1=value1&key2=value2)
6. Fragment

Resource

Examples

https://www.google.com/search?sxsrf=ACYBGNQAZ9ZC_m8Sg5kohN3z4WTYUI1WGg%3A1580104209880&source=hp&ei=EXouXsvdM7-e4-EPpeSGiAY&q=hello+world&oq=hello+world&gs_l=psy-ab.3..0l10.726.3433..3579...0.0..0.301.1861.5j5j2j1.....0....1..gws-wiz.......35i39j0i131j0i67j0i20i263.1IvM-ut4B1Y&ved=0ahUKEwiLqovxiqPnAhU_zzgGHSWyAWEQ4dUDCAU&uact=5

https://www.e-yantra.org/#eLSI

http://eyic.e-yantra.org/#section2

Methods

Also known as HTTP verbs

1. GET
2. POST
3. PUT
4. DELETE
5. PATCH

Find more at:
`https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods`

Response

HTTP Response consists of

1. Status Code
2. Headers
3. Body

Status Codes

Indicates whether the HTTP request completed successfully or not.

Responses are grouped in five classes:

1. Informational responses (100 - 199)
2. Successful responses (200 - 299)
3. Redirects (300 - 399)
4. Client errors (400 - 499)
5. Server errors (500 - 599)

Status Codes

Most recurring ones are:

1. Success Codes:
   1. 200 OK
   2. 201 Created
2. Redirection Messages:
   1. 301 Moved Permanently
   2. 304 Not Modified
   3. 307 Temporary Redirect

Status Codes

Most recurring ones are:

3. Client Errors:
    1. 400 Bad Request
    2. 401 Unauthorized
    3. 403 Forbidden
    4. 404 Not Found
    5. 405 Method not Allowed
    6. 408 Request Timeout

Status Codes

Most recurring ones are:

4. Server Errors:
    1. 500 Internal Server Error
    2. 502 Bad Gateway
    3. 503 Service Unavailable
    4. 504 Gateway Timeout

HTTP Headers

Headers let the client and the server pass additional information with an HTTP request or response.

It's definition consists of it's case-insensitive name followed by a colon (:), then by its value.

Proprietary headers use "X-" prefix

Types of headers:

1. General
2. Request
3. Response
4. Entity

Headers

1. Cookie (Request)
2. Accept (Request)
3. Authorization (Request)
4. Set-Cookie (Response)
5. Content-Type (Entity)

Request & Response Body

Not all requests and responses have a body. Requests fetching resources, like GET, HEAD, DELETE, or OPTIONS, usually don't need one.

Most common request-response body types (specified using Content-Type header):

1. text/html
2. application/json
3. application/x-www-form-urlencoded
4. multipart/form-data
5. image/png
6. text/css

Inspecting browser requests

1. Web apps for analysis
    1. https://github.com
    2. https://*.e-yantra.org
    3. https://www.google.com
    4. https://discord.com
2. On chrome or firefox, press "ctrl + shift + i".
3. Go to the "network" tab.
4. Perform some operations.
5. Inspect requests
6. Additional Instructions
    - You can "Clear" logs before performing an operation.
    - You can use "preserve logs" option to preserve requests from old webpage.

requests Library

requests

A 3rd party HTTP Client library for Python.

Easier to use than the standard libraries available in Python.

Documentation at `https://requests.readthedocs.io/en/master/`

To install run `pip install requests` or
`python -m pip install requests`

Example code

```
import requests

r = requests.get('https://httpbin.org/get', params={'
    queryParam1': 'value1', 'queryParam2': 'value2'})
r.text

r = requests.post('https://httpbin.org/post', data = {'key':
    'value'}, headers={'Authorization': 'Bearer myAccessToken
    '})
print(r.status_code)

r = requests.put('https://httpbin.org/put', json = {'jsonKey'
    : 'jsonValue'})
r.json()

requests.delete('https://httpbin.org/delete')
```

Implementing APIs

Implementing APIs

Two HTTP APIs

1. **Thingsboard** HTTP API for fetching and publishing different kinds of data at thingsboard.

   API documentation
   `https://thingsboard.e-yantra.org/swagger-ui.html`

2. **IoT HTTP Server (homework)** is another simple API for an HTTP Server used to store IoT data.

   API documentation Included with homework.

Using API Documentation

You need 7 things at max

1. URL - The resource.
2. Method - Is it GET or POST
3. Path parameters
4. Query parameters
5. Headers - What headers do you need to send?
6. Data
7. Data Format

Thingsboard API

Base URL

https://thingsboard.e-yantra.org/api/

Docs

https://thingsboard.io/docs/reference/http-api/

API Reference

https://thingsboard.e-yantra.org/swagger-ui.html

Thingsboard API

- Thingsboard API is divided into **Device API** and **Administration Rest API**.
- **Device API** allows for uploading telemetry, RPC and reporting/reading attributes.
- **Administration Rest API** is used for creating and managing entities, such as creating devices, rules or fetching telemetry.

Publish client-side attributes

Publish client-side attributes

URL: **/v1/$ACCESS_TOKEN/attributes**

Publish client-side attributes

URL: **/v1/$ACCESS_TOKEN/attributes**

Method: **POST**

Publish client-side attributes

URL: **/v1/$ACCESS_TOKEN/attributes**

Method: **POST**

Path parameter: **ACCESS_TOKEN**

Publish client-side attributes

URL: **/v1/$ACCESS_TOKEN/attributes**

Method: **POST**

Path parameter: **ACCESS_TOKEN**

Data with Content-Type application/json:

```
{
    "attribute1": "value1",
    "attribute2": true,
    "attribute3": 43.27
}
```

Publish client-side attributes

```
response = requests.post(f'{base_url}/v1/{ACCESS_TOKEN}/
    attributes', json={
    'energyConsumed': 'kWh',
    'instantaneousVoltage': 'V',
    'instantaneousCurrent': 'A',
    'instantaneousPower': 'kW'
})
```

Fetch client-side or shared attributes

Fetch client-side or shared attributes

URL: **/v1/$ACCESS_TOKEN/attributes**

Fetch client-side or shared attributes

URL: **/v1/$ACCESS_TOKEN/attributes**

Method: **GET**

Fetch client-side or shared attributes

URL: **/v1/$ACCESS_TOKEN/attributes**

Method: **GET**

Path parameter: **ACCESS_TOKEN**

Fetch client-side or shared attributes

URL: **/v1/$ACCESS_TOKEN/attributes**

Method: **GET**

Path parameter: **ACCESS_TOKEN**

Query parameter:
clientKeys=attribute1,attribute2&sharedKeys=shared1,shared2

Fetch client-side or shared attributes

URL: **/v1/$ACCESS_TOKEN/attributes**

Method: **GET**

Path parameter: **ACCESS_TOKEN**

Query parameter:
clientKeys=attribute1,attribute2&sharedKeys=shared1,shared2

Data with Content-Type application/json:

```
response = requests.get(f'{base_url}/v1/{ACCESS_TOKEN}/
    attributes')
```

Telemetry upload API

Telemetry upload API

URL: **/v1/$ACCESS_TOKEN/telemetry**

Telemetry upload API

URL: **/v1/$ACCESS_TOKEN/telemetry**

Method: **POST**

Telemetry upload API

URL: **/v1/$ACCESS_TOKEN/telemetry**

Method: **POST**

Path parameter: **ACCESS_TOKEN**

Telemetry upload API

URL: **/v1/$ACCESS_TOKEN/telemetry**

Method: **POST**

Path parameter: **ACCESS_TOKEN**

Data with Content-Type application/json:

```
{
    "energyConsumed": 50,
    "instantaneousCurrent": 3,
    "instantaneousVoltage": 232,
    "instantaneousPower": 0.696
}
```

Telemetry upload API

```
response = requests.post(f'{base_url}/v1/{ACCESS_TOKEN}/
    telemetry', json={
    'energyConsumed': 50,
    'instantaneousCurrent': 3,
    'instantaneousVoltage': 232,
    'instantaneousPower': 0.696
})
```

Login (Administration API)

Login (Administration API)

URL: **/auth/login**

Login (Administration API)

URL: **/auth/login**

Method: **POST**

Login (Administration API)

URL: **/auth/login**

Method: **POST**

Data with Content-Type application/json:

```
{
    "username": "omkar@e-yantra.org",
    "password": "12345"
}
```

Login (Administration API)

```
response = requests.post(f'{base_url}/auth/login', json={
    'username': 'omkar@e-yantra.org',
    'password': '12345'
})
```

Sending RPC commands (Administration API)

Sending RPC commands (Administration API)

URL: **/plugins/rpc/{one|two}way/$DEVICE_ID**

Sending RPC commands (Administration API)

URL: **/plugins/rpc/{one|two}way/$DEVICE_ID**

Method: **POST**

Sending RPC commands (Administration API)

URL: **/plugins/rpc/{one|two}way/$DEVICE_ID**

Method: **POST**

Path parameter: **DEVICE_ID**

Sending RPC commands (Administration API)

URL: **/plugins/rpc/{one|two}way/$DEVICE_ID**

Method: **POST**

Path parameter: **DEVICE_ID**

Data with Content-Type application/json:

```
{
    "method": "setAC",
    "params": false
}
```

Sending RPC commands (Administration API)

```
response = requests.post(f'{base_url}/plugins/rpc/oneway/{
    DEVICE_ID}', json={
    'method': 'setAC',
    'params': False
}, headers={
    'X-Authorization': f'Bearer {TOKEN}'
})
```

Intercept RPC commands

Intercept RPC commands

URL: **/v1/$ACCESS_TOKEN/rpc**

Intercept RPC commands

URL: **/v1/$ACCESS_TOKEN/rpc**

Method: **GET**

Intercept RPC commands

URL: **/v1/$ACCESS_TOKEN/rpc**

Method: **GET**

Path parameter: **ACCESS_TOKEN**

Intercept RPC commands

URL: **/v1/$ACCESS_TOKEN/rpc**

Method: **GET**

Path parameter: **ACCESS_TOKEN**

```
response = requests.get(f'{base_url}/v1/{ACCESS_TOKEN}/rpc',
    params={
    'timeout': 20000
})
```

Result of RPC

Result of RPC

URL: **/v1/$ACCESS_TOKEN/rpc/$ID**

Result of RPC

URL: **/v1/$ACCESS_TOKEN/rpc/$ID**

Method: **POST**

Result of RPC

URL: **/v1/$ACCESS_TOKEN/rpc/$ID**

Method: **POST**

Path parameter: **ACCESS_TOKEN**, **ID**

Result of RPC

URL: **/v1/$ACCESS_TOKEN/rpc/$ID**

Method: **POST**

Path parameter: **ACCESS_TOKEN**, **ID**

```
response = requests.post(f'{base_url}/v1/{ACCESS_TOKEN}/rpc/{
    rpc_request["id"]}', json={
    'result': 'ok'
})
```

Fetch time-series data (Administration API only)

Fetch time-series data (Administration API only)

URL:
/plugins/telemetry/DEVICE/$DEVICE_ID/values/timeseries

Fetch time-series data (Administration API only)

URL:
/plugins/telemetry/DEVICE/$DEVICE_ID/values/timeseries

Method: **GET**

Fetch time-series data (Administration API only)

URL:
/plugins/telemetry/DEVICE/$DEVICE_ID/values/timeseries

Method: **GET**

Path parameter: **$DEVICE_ID**

Fetch time-series data (Administration API only)

URL:
/plugins/telemetry/DEVICE/$DEVICE_ID/values/timeseries

Method: **GET**

Path parameter: **$DEVICE_ID**

Query parameters: **startTs**, **endTs**

More query parameters:
`https://thingsboard.io/docs/user-guide/telemetry/`

Fetch time-series data (Administration API only)

```
response = requests.get(f'{base_url}/plugins/telemetry/DEVICE
    /{DEVICE_ID}/values/timeseries', params={
    'startTs': '1422026157000',
    'endTs': '1604128783099'
}, headers={
    'X-Authorization': f'Bearer {TOKEN}'
})
```

About the homework

- Read the API documentation
- Use requests documentation to find the syntax of method that will make the http request
- Fill the functions given in `ServerApi` class following the instructions in comments
- Use `self.session()` object in methods of `ServerApi`.
- Functions return `response.json()` mostly, unless otherwise specified.
- Pay **close attention to the input and return types of methods** in **ServerApi**

An Example: Implement the login function

API documentation from **api.html**

Login

Resource for user login. The returned **authToken** should be sent in **Authorization** header as **Bearer <token>**.

Curl | HTTP

```
curl -X POST -H "Accept: application/json" -H "Content-Type: application/json" -d '{
    "username": "manjrekarom@gmail.com",
    "password": "abcd"
}' "https://apihptu.e-yantra.org/api/login"
```

Wrong Password | Login Successful

Status	200 OK

Figure 5: Login Resource

An Example: Implement the login function

Base URL: **https://apihptu.e-yantra.org/api**

An Example: Implement the login function

Base URL: **https://apihptu.e-yantra.org/api**

URL: **/login**

An Example: Implement the login function

Base URL: **https://apihptu.e-yantra.org/api**

URL: **/login**

Method: **POST**

An Example: Implement the login function

Base URL: **https://apihptu.e-yantra.org/api**

URL: **/login**

Method: **POST**

Data with Content-Type application/json:

An Example: Implement the login function

Base URL: **https://apihptu.e-yantra.org/api**

URL: /**login**

Method: **POST**

Data with Content-Type application/json:

```
{
    "username": <username>,
    "password": <password>
}
```

An Example: Implement the login function

Python Code

An Example: Implement the login function

Python Code

```
def login(self, username: str, password: str) -> None:
    """
    User login.

    Raises exception if unsuccessful by calling
        raise_exceptions.
    """
    login_url = f'{self.base_url}/login'
    response = self.session.post(login_url, json={
        'username': username,
        'password': password
    })
    self.raise_exceptions(response)
    self.session.headers.update({'Authorization': f'Bearer {
        response.json()["authToken"]}'})
```

Too many requests!

Too many requests!

Control the rate of http requests to the web server by adding delays.

Use `time.sleep()`

```
import time
...

login(...) # some function making http request
time.sleep(0.5) # delay of 0.5 seconds
create_thing(...) # another function making http request
time.sleep(1) # delay of 1 seconds

...
```

A delay of 300ms to 1.5s is enough.

References

1. MDN https://developer.mozilla.org/en-US/
2. cURL https://curl.haxx.se/
3. requests https://requests.readthedocs.io/en/master/
4. JSON Placeholder https://jsonplaceholder.typicode.com/
5. json-server https://github.com/typicode/json-server
6. httpbin http://httpbin.org/
7. Thingsboard HTTP API
   https://thingsboard.io/docs/reference/http-api/
8. Thingsboard RPC
   https://thingsboard.io/docs/user-guide/rpc/

Thank You!

Post your queries on: helpdesk@e-yantra.org