appInterface.py

This class must be used to communicate with the RESTful API. Some functions are restricted to the initial/final communication protocol between an application and the API and others to data exchange between an application and the API.

Note: To generate this doc use the command: pycco appInterface.py -p

```
import requests
from requests.structures import CaseInsensitiveDict
import json
```

```
class ApplicationInterface:
```

```
    def __init__(self, url):
        self.URL = url
        self.headers = CaseInsensitiveDict()
        self.headers["Content-Type"] = "application/json"
```

DATA EXCHANGE FUNCTIONS

Returns the data of a node based on its ID

Args: ID (str): ID of the node

Returns: json: data of the node

```
    def getDataFromID(self, ID):

        url= self.URL + "/" + str(ID)
        return self.get(url)
```

Returns the data of a node based on its IP

Args: IP (str): IP of the node

Returns: json: data of the node

```
    def getListOfMessageFromDeviceIP(self, IP):

        url= self.URL + "/sensor/?ip=" + str(IP)
        return self.get(url)
```

Returns the data of nodes based on their date

Args: date (int): timestamp of the date

Returns: json: data of the nodes

```
    def getListOfMessageByDate(self, date):

        url= self.URL + "/offload/?date=" + str(date)
        return self.get(url)
```

Returns the data of nodes based on their type

Args: typ (str): type of the nodes

Returns: json: data of the nodes

```
    def getListOfMessageFromSensorType(self, typ):

        url= self.URL + "/sensor_type/?type=" + str(typ)
        return self.get(url)
```

Returns the data of nodes based on their validation field

Args: val (Boolean): value of data's validation field

Returns: json: data of the nodes

```
    def getListOfMessageWithValidation(self, val):

        url= self.URL + "/valid_items/?valid=" + str(val)
        return self.get(url)
```

Removes data based on their ID

Args: ID (str): ID of the node

Returns: json: data of the node

```
    def deleteDataFromID(self, ID):

        url= self.URL + "/" + str(ID)
        return self.delete(url)
```

Removes data based on their date

Args: date (int): timestamp date of the data

Returns: json: data of the node

```
    def deleteListOfMessageByDate(self, date):

        url= self.URL + "/offload/?date=" + str(date)
        return self.delete(url)
```

Add data from a single device to the database of a node

Args: ip (str): IP of the sender device date (int): timestamp of the data type (str): type of the data jsonfile (str): path to the json file containing the data

Returns: json: data send to the database

```
def postDataFromSingleDevice(self, ip: str, date: int, type: str
    url = self.URL + "/"
    val = self.getLocalData(jsonfile)
    DATA = {'ip':ip, 'date':date, 'type':type, 'values':val["val
    json_object = self.dumpData(DATA)
    return self.post(url, json_object)
```

Add data from a mutiple device to the database of a node

Args: ip (str): IP of the sender device date (int): timestamp of the data type (str): type of the data dict (dict): dictionary containing the data

Returns: json: data send to the database

```
def postDataFromSingleDeviceDict(self, ip: str, date: int, type:
    url = self.URL + "/"
    try:
        DATA = {'ip':ip, 'date':date, 'type':type, 'values':dict
        json_object = self.dumpData(DATA)
        return self.post(url, json_object)
    except:
        print("[Error] - Dict must have the following form: {'va
        return None
```

Add data from multiple devices to the database of a node

Args: jsonfile (json): json file containing the data

Returns: json: data send to the database

```
def postDataFromMultipleDevice(self, jsonfile):
    url = self.URL + "/multiple/"
    DATA = self.getLocalData(jsonfile)
    json_object = self.dumpData(DATA)
    return self.post(url, json_object)
```

COMMUNICATION PROTOCOL FUNCTIONS

Send the IP of an application to the database of a node

Args: ip (str): IP of the sender device date (int): timestamp of the data type (str): type of the data (Should be : 'appIP') appname (str): name of the application Returns: json: post data send to the database

```
def postIP(self, ip: str, date: int, type: str, appname: str):
    url = self.URL + "/appIP/"
    dict = {'values': [{'id': "0", 'date': 0, 'parameterId': "0"
    try:
        DATA = {'ip':ip, 'date':date, 'type':type, 'values':dict
        json_object = self.dumpData(DATA)
        return self.post(url, json_object)
    except:
        print("[Error] - Exception occur when trying to post the
        return url
```

Delete the IP of an application from the database of a node

Args: name (str): name of the application

Returns: json: data send to the database

```
def deleteAppIPbyName(self, name):
    url = self.URL + "/appIP/?type=" + name
    return self.delete(url)
```

Returns the IP of an application in the database of a node based on its name

Args: name (str): name of the application

Returns: json: data send to the database

```
def getAppIPbyName(self, name):
    url = self.URL + "/appIP/?type=" + name
    return self.get(url)
```

GENERIC FUNCTIONS

Generic function to post data to the database

Args: url (str): particular url to post the data json_object (json): the data to post

Returns: json: data send to the database

```
def post(self, url, json_object):
    try:
        resp = requests.post(url=url, headers=self.headers, data
        if(resp.status_code in [204,200,201]):
            return resp.json()
        else:
            return None
    except Exception as inst:
        print("[Error] - ", type(inst))
        return None
```

Generic function to delete data from the database

Args: url (str): particular url to delete the data

Returns: json: data delete from the database

```
def delete(self, url):


    try:
        resp = requests.delete(url=url, headers=self.headers)
        if(resp.status_code in [204,200,201]):
            return resp.json()
        else:
            return None
    except Exception as inst:
        print("[Error] - ", type(inst))
        return None
```

Generic function to get data from the database

Args: url (str): particular url to get the data

Returns: json: data get from the database

```
def get(self, url):


    try:
        resp = requests.get(url=url, headers=self.headers)
        if(resp.status_code in [204,200,201]):
            return resp.json()
        else:
            return None
    except Exception as inst:
        print("[Error] - ", type(inst))
        return None
```

UTILITY FUNCTIONS

Returns the data of a json file

Args: path (str): path to the json file

Returns: dict: data of the json file

```
def getLocalData(self, path):


    f = open(path)
    values = json.load(f)
    return values
```

Send dictionary to json format

Args: dict (dict): dictionary to send to json format

Returns: json: dictionary in json format

```
def dumpData(self, dict):


    return json.dumps(dict, indent = 4)
```