ADSCCD : flask / python json et jsonschema / libcurl / jansson, exemple d'outils permettant de développer un client/serveur WEB de type REST

M2-CNS, parcours SA et SR
Université Paris-Saclay, site d'Evry, UEVE

Patrice LUCAS
CEA-DAM, patrice.lucas@cea.fr
PAST, département informatique de l'UEVE

Flask : présentation

https://flask.palletsprojects.com

- Flask is a lightweight WSGI web application framework in python.
  - Flask s’appuie sur une palette de composants (https://palletsprojects.com):
    - Jinja : template,
    - Werkzeug : coeur WSGI
    - Click : ligne de commande
    - ...
  - Flask est conçu pour une prise en main rapide.
  - Flask permet également de concevoir des projets plus poussés.
- Il permet par exemple pour débuter :
  - De rattacher des urls à l’exécution de fonctions pré-déterminées,
  - D’accéder à l’application en cours et la requête en cours,
  - De façonner une réponse http/https appropriée, de lever des exceptions

Flask : création d'une application

- `class flask.Flask(…)` : Application Object

- On déclare une application aussi simplement que :

```
from flask import Flask

app = Flask(__name__)
```

- Flask import_name parameter : name of the application package
  - `__name__` convient très bien pour un module simple.
  - Pour un package, il est préférable d'utiliser le nom du package, en dur ou sous la forme :

```
__name__.split('.')[0]
```

Flask : lancement de l’application

- `app.run()` : l’usage du serveur intégré n’est pas recommandé en production, notamment pour des raisons de performance lorsqu’une bonne scalabilité devient requise.
  - Peut-être utilisé par exemple pour des tests au sein d’un bloc “ `if __name__ == ‘__main__’` ”
- CGI : “`yourapplication.cgi`” file to integrate to Apache or other web server configuration.

```
#!/usr/bin/python
from wsgiref.handlers import CGIHandler
from yourapplication import app

CGIHandler().run(app)
```

- FastCGI, uWSGI (nginx, lighttpd, and cherokee), mod_wsgi (Apache)

Flask : lancement de l’application

- A partir de la version de flask 0.11 :

```
export FLASK_APP=my_app
flask run
```

Ou

```
export FLASK_APP=my_app
python –m flask
```

- Le serveur par défaut est en mode débug et restreint à localhost.
  - Pour l’ouvrir au monde entier :

```
flask run –host=0.0.0.0
```

Flask : enregistrement d’une fonction pour une url

- Une règle (rule) fait correspondre un url (endpoint) avec l’exécution d’une fonction (view_function)

Flask : @route

- Décorateur “route”
  - Paramètre
  - Converter : string, int, float, path, uuid
  - Warning : “escape” les données d’entrée peut-être une bonne pratique

```
#!/usr/bin/env python3

from flask import Flask
from markupsafe import escape

app = Flask(__name__)

@app.route('/simple_hello')
def simple_hello():
    return "Simple hello world !\n"

@app.route('/named_hello/')
@app.route('/named_hello/<name>')
def named_hello(name="Unknown"):
    return f"Hello \"{name}\" !\n"

@app.route('/repeat_hello/<int:nb>')
def repeat_hello(nb):
    msg=[]
    for _ in range(nb):
        msg.append("Hello !")

   return ' '.join(msg)

@app.route('/escape/<dangerous_param>')
def escape_dangerous_param(dangerous_param):
    return f"We take care of this param: {escape(dangerous_param)} !\n"

app.run()
```

Flask : exemple

- flask_hello_world

Flask : enregistrement d'une fonction pour une url

- La déclaration d'une route correspond à l'objet "rule" de werkzeug que l'on peut paramétrer encore plus finement.
  - Ex : restriction des méthodes avec le paramètre `methods`

```
app.add_url_rule('/set_hello',
                 'set_hello',
                 set_hello,
methods=['POST','PUT'])
```

Flask : construction inverse des urls

- url_for : permet de retrouver l'URL associé à la fonction d'une vue
  - Résiste aux futures modifications d'URL dans le code
  - Permet de gérer le déplacement de l'URL root de l'application et l'URL générée est toujours absolue

```
#!/usr/bin/env python3

from flask import Flask, url_for
from markupsafe import escape

app = Flask(__name__)

@app.route('/la_route/de_ma_fonction/<le_param>')
def ma_fonction(le_param):
    return f"Merci pour escape(le_param)"

with app.test_request_context():
    print(url_for('ma_fonction', le_param="voici_mon_param"))
```

Flask : exemple

- flask_url_for

Flask : fichier statique

- Répertoire ‘static’ à la racine du package pour héberger par exemple les CSS et autres JavaScripts

```
url_for(‘static’, filename=’style.css’)
```

Flask : template

- Fonction render_template permet de rendre des templates Jinja2 (https://jinja.palletsprojects.com/template)
  - L'échappement des variables est automatique.

```
#!/usr/bin/env python3
from flask import Flask, render_template

app = Flask(__name__)

@app.route('/hello/')
@app.route('/hello/<name>')
def hello_template(name=None):
    return render_template('hello.html', name=name)

app.run()
```

- Répertoire /templates hébergeant les templates dans le package

```
<!doctype html>
<title>My first hello example</title>
{% if name %}
  <h1>Hello {{ name }} !</h1>
{% else %}
  <h1>Hello handsome stranger !</h1>
{% endif %}
```

Flask : exemple

- flask_hello_template

Flask : accès à la requête courante, à l'application courante

- `flask.request` : la requête courante
  - method, get_json/get_data, content_type, form/args, origin/remote_addr, path/root_path/query_string, cookies …
- `flask.current_app` : l'application courante, utile si par exemple on stocke un état courant ou des paramètres au sein de l'objet application

Flask : formatage d'une réponse

- Les fonctions de vue peuvent renvoyer :
  - une string,
  - des bytes,
  - un dictionnaire qui est transformé en json,
  - un tuple (body, status, headers), (body, status), or (body, headers)
    - body : corps de la réponse selon les types précédents
    - status : statut de retour http/https
    - headers : dictionnaire ou liste `(clé, valeur)`
  - un objet `flask.Response`

```
class flask.Response(response=None, status=None, headers=None, mimetype=None, content_type=None, direct_passthrough=False)
```

- On peut "pré-manipuler une réponse" en utilisant la fonction `make_response()`

Flask : exemple

- flask_message_server

Flask : exemple d'une réponse json

```
class JsonResponse(Response): # pylint: disable=too-
many-ancestors
    """Utility wrapper to return HTTP responses with a
json body"""


    def __init__(self, json_data, status=200):
        """Return an HTTP response with a given `status`
code and serialize
        `json_data` as the body of the response.
        """
        super(JsonResponse, self).__init__(
            response=json.dumps(json_data),
            status=status,
            mimetype="application/json"
        )
```

Flask : json “natif”

- Encoding :
  - Renvoyer un dictionnaire : celui-ci sera converti en une réponse en json
  - Sérialiser un objet compatible JSON avec la fonction `jsonify()`
- Decoding:
  - `Request.get_json(force=False, silent=False, cache=True)`
    - Renvoie `None` si le mimetype n’est pas “`application/json`” et sinon renvoie le json parsé
    - `force`: parse même si le mimetype n’est pas “`application/json`”
    - `silent`: renvoie `None` si une erreur de parsing survient au-lieu de l’exception `BadRequest`

Python json encoding/decoding

- import json
  - Librarie standard
- json.dumps(['foo', {'bar': ('baz', None, 1.0, 2)}])
- json.loads('["foo", {"bar":["baz", null, 1.0, 2]}]')

Python jsonschema

- import jsonschema
  - https://python-jsonschema.readthedocs.io

```
>>> from jsonschema import validate

>>> # A sample schema, like what we'd get from json.load()
>>> schema = {
...     "type" : "object",
...     "properties" : {
...         "price" : {"type" : "number"},
...         "name" : {"type" : "string"},
...     },
... }

>>> # If no exception is raised by validate(), the instance is valid.
>>> validate(instance={"name" : "Eggs", "price" : 34.99}, schema=schema)

>>> validate(
...     instance={"name" : "Eggs", "price" : "Invalid"}, schema=schema,
... )
Traceback (most recent call last):
    ...
ValidationError: 'Invalid' is not of type 'number'
```

Flask/JSON/libcurl/jansson : exemple

- flask_json_libcurl_catalog avec serveur python/flask/json

Libcurl : easy-to-use client-side URL transfer library

- API C permettant d'exécuter des requêtes clientes HTTP/HTTPS mais pas que …
  - DICT, FILE, FTP, FTPS, GOPHER, GOPHERS, HTTP, HTTPS, IMAP, IMAPS, LDAP, LDAPS, MQTT, POP3, POP3S, RTMP, RTMPS, RTSP, SCP, SFTP, SMB, SMBS, SMTP, SMTPS, TELNET and TFTP
- Des "bindings" existent pour d'autres langages que le C
- Une commande "curl"

- https://curl.se/

Libcurl : séquence basique de download (1/2)

- `#include <curl/curl.h>`
  - Édition de lien avec “`-lcurl`”
- `CURLcode curl_global_init(long flags)`
  - une unique fois au début : `CURL_GLOBAL_ALL`
- `CURL *curl_easy_init()`
  - Interface synchrone (pour des opérations asynchrones, utiliser “`multi`”)
  - Initialisation d’un handle : ouverture de la connexion
  - Il faut réutiliser au maximum les handles
- `CURLcode curl_easy_setopt(CURL *handle, CURLoption option, parameter)`
  - Positionne le comportant du handle à l’aide de constante et leur valeur que l’on souhaite affecter

```
curl_easy_setopt(handle, CURLOPT_URL,
"http://domain.com/");
```

Libcurl : séquence basique de download (2/2)

- `size_t write_data(void *buffer, size_t size, size_t nmemb, void *userp)`
  - Fonction d'écriture de la réponse reçue en retour de la requête
    - Par défaut, écriture sur `stdout` ou le `FILE *` spécifié comme `CURLOPT_WRITEDATA`
- `curl_easy_setopt(easyhandle, CURLOPT_WRITEFUNCTION, write_data);`
- `curl_easy_setopt(easyhandle, CURLOPT_WRITEDATA, &internal_struct);`
- `CURLcode curl_easy_perform(CURL *handle)`
  - Réalise de manière synchrone la requête : l'appel est bloquant et l'on en sort que lorsque la requête est finie
- `CURLcode curl_easy_getinfo(CURL *curl, CURLINFO info, ... )`
  - Nous informe sur la réponse reçue
- `void curl_global_cleanup(void)`
  - A n'appeler qu'une fois en fin d'utilisation de curl

Libcurl : séquence basique d'upload

- `size_t read_function(char *bufptr, size_t size, size_t nitems, void *userp)`
- `curl_easy_setopt(easyhandle, CURLOPT_READFUNCTION, read_function);`
- `curl_easy_setopt(easyhandle, CURLOPT_READDATA, &filedata);`
- `curl_easy_setopt(easyhandle, CURLOPT_UPLOAD, 1L);`
- `curl_easy_setopt(easyhandle, CURLOPT_INFILESIZE_LARGE, file_size);`
- `rc = curl_easy_perform(easyhandle);`

Libcurl : exemple

- libcurl_download

jansson : json en C

- https://jansson.readthedocs.io

- Une API en C pour décoder, encoder et manipuler des JSONs.
- `#include <jansson.h>`
- Édition de lien avec “`-ljansson`”

jansson : json_t et types json

- json_t
  - Un type unique pour stocker des objets json:
    - object, array, string, number, boolean, and null
- int json_typeof(const json_t *json)
- json_is_object(const json_t *json)
- json_is_array(const json_t *json)
- json_is_string(const json_t *json)
- json_is_integer(const json_t *json)
- json_is_real(const json_t *json)
-
- json_is_null(const json_t *json)
- json_is_number(const json_t *json)
-

jansson : json_t type et reference count

- json_t
  - Un type unique pour stocker des objets json:
    - object, array, string, number, boolean, and null
- json_t *json_incref(json_t *json)
- void json_decref(json_t *json)
- Adding with a copy :

```
json_t *array, *integer;

array = json_array();
integer = json_integer(42);

json_array_append(array, integer);
json_decref(integer);
```

- Adding by stealing ref :

```
json_t *array = json_array();
json_array_append_new(array, json_integer(42));
```

jansson : boolean

- json_t *json_true(void)
- json_t *json_false(void)
- json_t *json_boolean(val)
- json_is_true(const json_t *json)
- json_is_false(const json_t *json)
- json_is_boolean(const json_t *json)

jansson : string

- json_t *json_string(const char *value)
- json_t *json_stringn(const char *value, size_t len)
- const char *json_string_value(const json_t *string)
- size_t json_string_length(const json_t *string)
- json_t *json_sprintf(const char *format, …)
- int json_string_set(json_t *string, const char *value)
- int json_string_setn(json_t *string, const char *value, size_t len)

jansson : number

- json_int_t
  - “long long”, or “long”, JSON_INTEGER_IS_LONG_LONG
  - Printf with JSON_INTEGER_FORMAT
  - json_t *json_integer(json_int_t value)
  - json_int_t json_integer_value(const json_t *integer)
  - int json_integer_set(const json_t *integer, json_int_t value)
- json_t *json_real(double value)
  - double json_real_value(const json_t *real)
  - int json_real_set(const json_t *real, double value)
  - double json_number_value(const json_t *json) : cast si nécessaire

jansson : array

- Ensemble ordonné de json
- json_t *json_array(void)
  - size,
  - get,
  - set, set_new,
  - append, append_new
  - insert, insert_new
  - remove, clear
  - extend
  - json_array_foreach(array, index, value)

jansson : object

- Dictionnaire de json
- json_t *json_object(void)
  - size,
  - get,
  - set, set_new,
  - append, append_new
  - insert, insert_new
  - del, clear
  - update[_new], update_existing[_new], update_missing[_new], update_recursive
  - json_object_foreach(object, key, value)

jansson : error

- json_error_t
  - text, source, line, column, position
- enum json_error_code json_error_code(const json_error_t *error)

jansson : encoding

- String : char *json_dumps(const json_t *json, size_t flags)
  - Buffer : json_dumpb
  - Stream : json_dumpf
  - File Descriptor : json_dumpfd
  - Fichier à l’aide d’un path : json_dump_file

jansson : decoding

- String : json_t *json_loads(const char *input, size_t flags, json_error_t *error)
  - Buffer : json_loadb
  - Stream : json_loadf
  - File Descriptor : json_loadfd
  - Fichier à l’aide d’un path : json_load_file

jansson : packing

- json_t *json_pack(const char *fmt, ...)

```
/* Create the JSON integer 42 */
json_pack("i", 42);


/* Create the JSON array ["foo", "bar",
true] */
json_pack("[ssb]", "foo", "bar", 1);


/* Build the JSON array [[1, 2],
{"cool": true}] */
json_pack("[[i,i],{s:b}]", 1, 2,
"cool", 1);
```

jansson : unpacking

- `int json_unpack(json_t *root, const char *fmt, …)`
- Validation : `int json_unpack_ex(json_t *root, json_error_t *error, size_t flags, const char *fmt, ...)`

```
/* root is the JSON integer 42 */
int myint;
json_unpack(root, "i", &myint);
assert(myint == 42);


/* root is the JSON object {"foo": "bar", "quux": true} */
const char *str;
int boolean;
json_unpack(root, "{s:s, s:b}", "foo", &str, "quux", &boolean);
assert(strcmp(str, "bar") == 0 && boolean == 1);


/* root is the JSON array [[1, 2], {"baz": null} */
json_error_t error;
json_unpack_ex(root, &error, JSON_VALIDATE_ONLY, "[[i,i], {s:n}]", "baz");
/* returns 0 for validation success, nothing is extracted */
```

jansson : equality and copying

- int json_equal(json_t *value1, json_t *value2)
- json_t *json_copy(json_t value)
- json_t *json_deep_copy(const json_t *value)

Flask/JSON/libcurl/jansson : exemple

- flask_json_libcurl_catalog avec client C en libcurl et jansson

Flask/JSON/libcurl/jansson : exercice

- flask_json_curl_key_value : implémentation d’un client serveur REST stockant des paires de clé/valeur (serveur en python/flask/json, client en C/libcurl/jansson)