# Rest ou WebSocket

Work4Graph  CATI Sysmics

RÉPUBLIQUE FRANÇAISE
*Liberté*
*Égalité*
*Fraternité*

INRAE

# Rest ou WebSocket

- REST (representational state transfer)  2000
  - Service Web basé sur le protocole HTTP et les méthodes associées GET/POST/DELETE/PUT…
  - Utilise une ressource web (web serveur) pour échanger des données textuelles (json)
  - Pas de notion d'état tout comme le web (http)
  - Le principe de base : c'est un client qui va "picorer" de l'information sur un serveur
- WebSocket  2011
  - Creation de canal de communication full-duplex entre le client et le serveur web
  - Connexion permanente
  - C'est le serveur qui previent d'un changement et va "gaver" le client d'informations
  - Pas encore complètement implementé par tous le navigateurs

# Comment implementer simplement REST

- REST
  - Est un standard qui est implementé dans tous les langages avec des librairies ou des FrameWork.
    - Ex : Flask en python avec des librairies additionnelles comme FlaskRestFull et implementation de swagger avec flask-restx…

**PYTHON CODE**

```
from flask import Flask
app = Flask(__name__)
app.run(host="0.0.0.0",  port=5000,   threaded=True, debug=True)
```

# Comment implementer simplement REST

## PYTHON CODE 1/2

```
from flask import Flask, request
from flask_restx import Api, Resource, fields

app = Flask(__name__)
api = Api(app, version='1.0', title='tuto API',
    description='A Cool API' )

ns = api.namespace('species', description='species operations')

species = api.model('speciesSpec', {
    'id':   fields.Integer(required=True,
                           description='species identifier'),
    'name': fields.String(required=True,
                           description='species name')
})
```


## PYTHON CODE 2/2

```
class SpeciesRetreiver(object):
    def __init__(self):
        self.speciesList = []

    def get(self, id):
        for species in self.speciesList:
            if species['id'] == id:
                return species

    def create(self, id, name):
       species = { 'id': id, 'name': name}
        self.speciesList.append(species)
        return species

speciesR = SpeciesRetreiver()
speciesR.create(1,'AThaliana')
speciesR.create(2,'Tomate')

@ns.route('/')
class SpeciesList(Resource):
    '''Shows a list of all species'''
    @ns.doc('list_species')
    @ns.marshal_list_with(species)
  def get(self):
        '''List all species'''
        return speciesR.speciesList


app.run(host="0.0.0.0",  port=5000,   threaded=True, debug=True)
```

# Comment implementer simplement REST (Serveur)

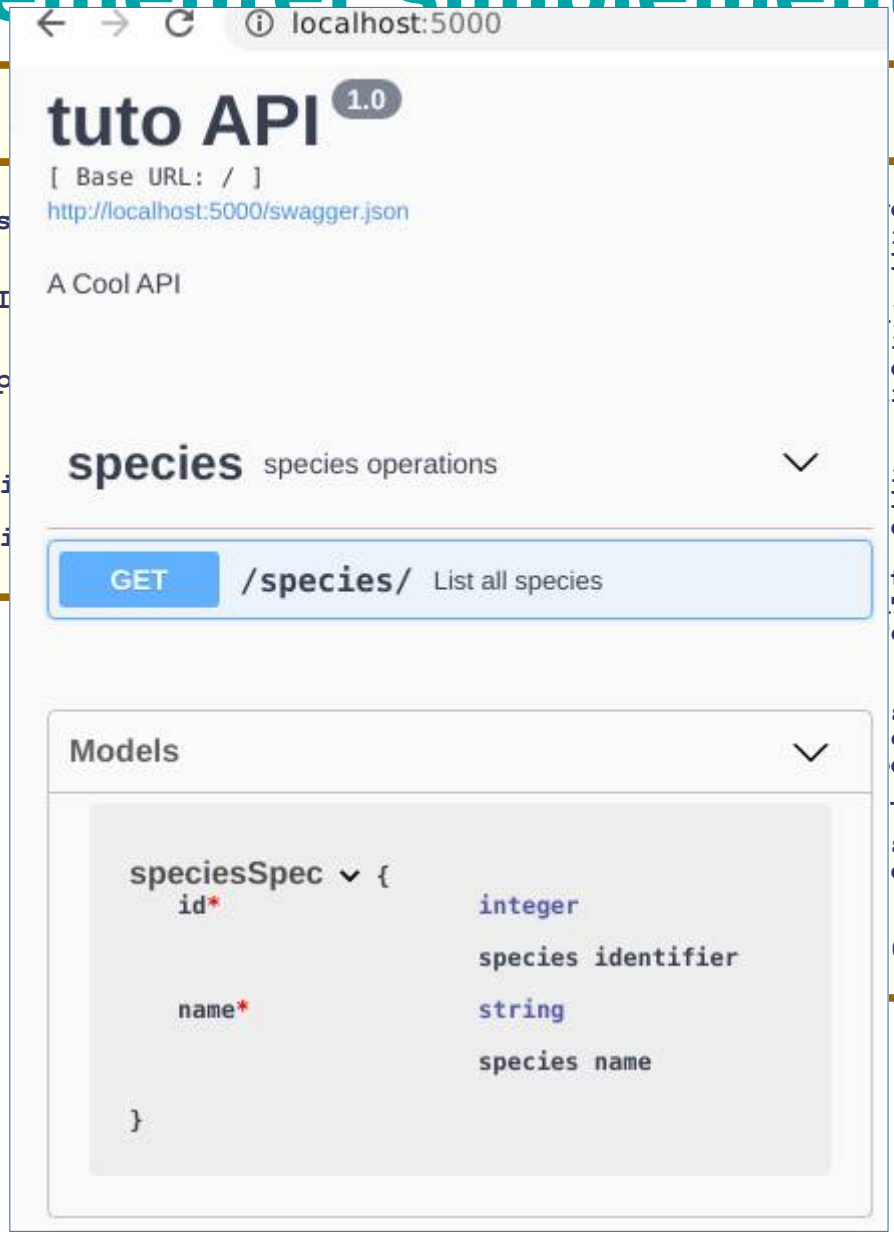
## PYTHON CODE 1/2

```
from flask import Flask, request
from flask_restx import Api, Resource, fields

app = Flask(__name__)
api = Api(app, version='1.0', title='tuto API
    description='A Cool API' )

ns = api.namespace('species', description='sp

species = api.model('speciesSpec', {
    'id':   fields.Integer(required=True,
                           description='speci
    'name': fields.String(required=True,
                           description='speci
})
```


## PYTHON CODE 2/2

```
er(object):
f):
List = []

):
in self.speciesList:
es['id'] == id:
rn species

 id, name):
id': id, 'name': name}
List.append(species)
es

treiver()
Thaliana')
omate')


source):
of all species'''
ecies')
_with(species)

species'''
esR.speciesList


0",  port=5000,   threaded=True, debug=True)
```

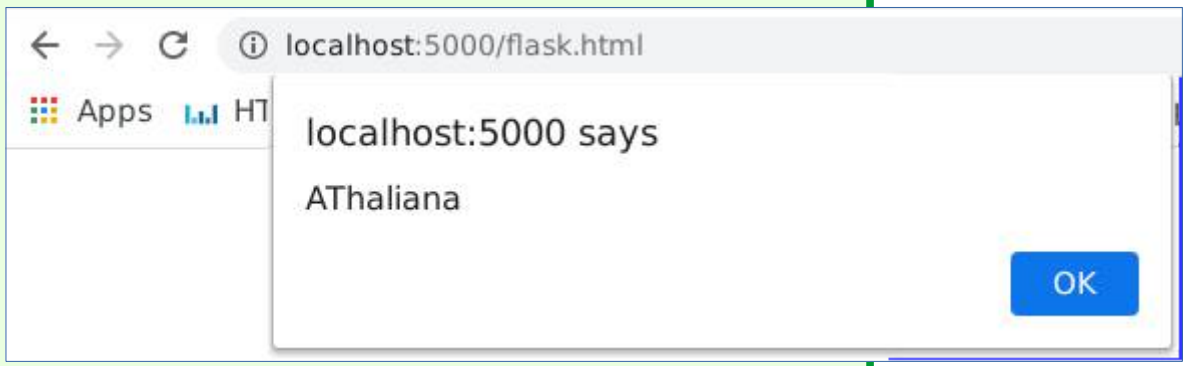
# Comment implementer simplement REST (Client)

## HTML CODE

```
<HTML>
<HEAD>
	<title>Flask test</title>
	<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>
	<script>

	$('document').ready(function() {

		$.ajax({	// ATTENTION FONCTIONNE EN ASYNCHRONE
			url: "http://localhost:5000/species",
			type: 'GET',
			datatype: 'json',
			success: function(data) {
				alert(data[0].name);
			}
		});
	console.log("coucou");
	});
	</script>
</HEAD>
<BODY>
</BODY>
</HTML>
```



# Comment implementer simplement WebSocket

- **WebSocket**
  - Est un standard qui est implementé dans la plus part des langages.
  - C’est le serveur qui envoie les données et pas le client qui vient les chercher.
  - Possibilité de broadcaster ou de faire des groupes de clients.
  - Connexion continue, bi-directionnelle.
  - Attention les clients n’implémentent pas toute la norme (comme au bon vieux temps de javascript).

# Comment implementer simplement WebSocket

## HTML CODE

```
<HTML>
<HEAD>
	<title>Flask test</title>
	<script src="//cdnjs.cloudflare.com/ajax/libs/socket.io/2.2.0/socket.io.js" ></script>

	<script type="text/javascript" charset="utf-8">
		var socket = io().connect('http://localhost:5000');;
		socket.on('connect', function() {
			socket.emit('hi!', {data: 'hello'} ) ;
		});
	</script>
</HEAD>
<BODY>
</BODY>
</HTML>
```

## PYTHON CODE Serveur

```
from flask import Flask, render_template
from flask_socketio import SocketIO, emit


@socketio.on('connect')
def connection():
    print ("un demande de connection");

@socketio.on('hi!')
def hellomessage(message):
   print(message)



if __name__ == '__main__':
    socketio.run(app)
```

```
WebSocket transport not available. Install eventlet or gevent and gevent-websoc
et for improved performance.
 * Serving Flask app "websocket" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deploym
nt.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
un demande de connection
127.0.0.1 - - [03/Dec/2020 14:36:01] "GET /socket.io/?EIO=3&transport=polling&t=
NOeqHjn HTTP/1.1" 200 -
{'data': 'hello'}
127.0.0.1 - - [03/Dec/2020 14:36:01] "POST /socket.io/?EIO=3&transport=polling&
=NOeqHju&sid=744f6861751a4e03a364202bfb1f749f HTTP/1.1" 200 -
```

# Comment implementer simplement WebSocket

## HTML CODE

```
<HTML>
<HEAD>
	<title>Flask test</title>
	<script src="//cdnjs.cloudflare.com/ajax/libs/socket.io/2.2.0/socket.io.js" ></script>

	<script type="text/javascript" charset="utf-8">
		var socket = io().connect('http://localhost:5000');;
		socket.on('connect', function() {
			socket.emit('hi!', {data: 'hello'} ) ;
		});
	</script>
</HEAD>
<BODY>
</BODY>
</HTML>
```

## PYTHON CODE Serveur

```
from flask import Flask, render_template
from flask_socketio import SocketIO, emit


@socketio.on('connect')
def connection():
    print ("un demande de connection");

@socketio.on('hi!')
def hellomessage(message):
   print(message)



if __name__ == '__main__':
    socketio.run(app)
```

# Comment implementer simplement WebSocket

## HTML CODE

```
<HTML>
<HEAD>
	<title>Flask test</title>
	<script src="//cdnjs.cloudflare.com/ajax/libs/socket.io/2.2.0/socket.io.js" ></script>

	<script type="text/javascript" charset="utf-8">
		var socket = io().connect('http://localhost:5000');;
		socket.on('connect', function() {
			socket.emit('hi!', {data: 'hello'} ) ;
		});

		socket.on("species", function(sp) {
			alert("SPecies : "+sp[1].name);
		});
	</script>
</HEAD>
<BODY>
</BODY>
</HTML>
```


## PYTHON CODE Serveur

```
from flask import Flask, render_template
from flask_socketio import SocketIO, emit

species = [{'id': 1, 'name': 'AThalina' },
           {'id': 2, 'name': 'Tomate' }]

@socketio.on('connect')
def connection():
    print ("un demande de connection");

@socketio.on('hi!')
def hellomessage(message):
   print(message)
   emit("species", species);


if __name__ == '__main__':
    socketio.run(app)
```

```
un demande de connection
127.0.0.1 - - [03/Dec/2020 14:54:16] "GET /socket.io/?EIO=3&transport=polling&t=
NOeuT9y HTTP/1.1" 200 -
127.0.0.1 - - [03/Dec/2020 14:54:16] "GET /favicon.ico HTTP/1.1" 404 -
{'data': 'hello'}
127.0.0.1 - - [03/Dec/2020 14:54:16] "POST /socket.io/?EIO=3&transport=polling&t
=NOeuTA5&sid=666718c2596b400c86d7bf403c89144c HTTP/1.1" 200 -
127.0.0.1 - - [03/Dec/2020 14:54:16] "GET /socket.io/?EIO=3&transport=polling&t=
NOeuTA6&sid=666718c2596b400c86d7bf403c89144c HTTP/1.1" 200 -
```

# Comment implementer simplement WebSocket

## HTML CODE

```
<HTML>
<HEAD>
	<title>Flask test</title>
	<script src="//cdnjs.cloudflare.com/ajax/libs/socket.io/2.2.0/socket.io.js" ></script>

	<script type="text/javascript" charset="utf-8">
		var socket = io().connect('http://localhost:5000');;
		socket.on('connect', function() {
			socket.emit('hi!', {data: 'hello'} ) ;
		});

		function cb(a, b) {
			console.log(a);
			console.log(b);
		};


		socket.on("species", function(sp) {
			alert("SPecies : "+sp[1].name);

			socket.emit('info', {data: 'give me more'}, cb );
		});

	</script>
</HEAD>
<BODY>
</BODY>
</HTML>
```

## PYTHON CODE Serveur

```
from flask import Flask, render_template
from flask_socketio import SocketIO, emit

species = [{'id': 1, 'name': 'AThalina' },
           {'id': 2, 'name': 'Tomate' }]

@socketio.on('connect')
def connection():
    print ("un demande de connection");

@socketio.on('hi!')
def hellomessage(message):
   print(message)
   emit("species", species);


@socketio.on("info")
def demandeInfo(message):
   print(message)
   return ("des infos", "pour le cb")



if __name__ == '__main__':
    socketio.run(app)
```

# Rest vs WebSocket

- **Websocket est prometteur mais sans doute encore jeune pour pouvoir profiter complètement de cette technologie.**
- **REST est une technologie robuste, fiable et mature**
- **Les philosophies des deux sont très différentes, nos projets comportants souvent beaucoup de données, websocket deviendra sans doute une bonne solution. Quid de la charge côté serveur ???**