WEB. Arquitectura, tecnologías y herramientas

Autor: Lic. Danilo Zecchin. UNLP

Arquitectura cliente-servidor

- Protocolo HTTP: Hypertext Transfer Protocol

  - Protocolo de request-response
  - Sus recursos se identifican con URLs
  - Posee un *Header*
  - Se transmite texto plano

REST

- REST significa:
  - **RE**presentational
  - **S**tate
  - **T**ransfer

¿Representational State Transfer?

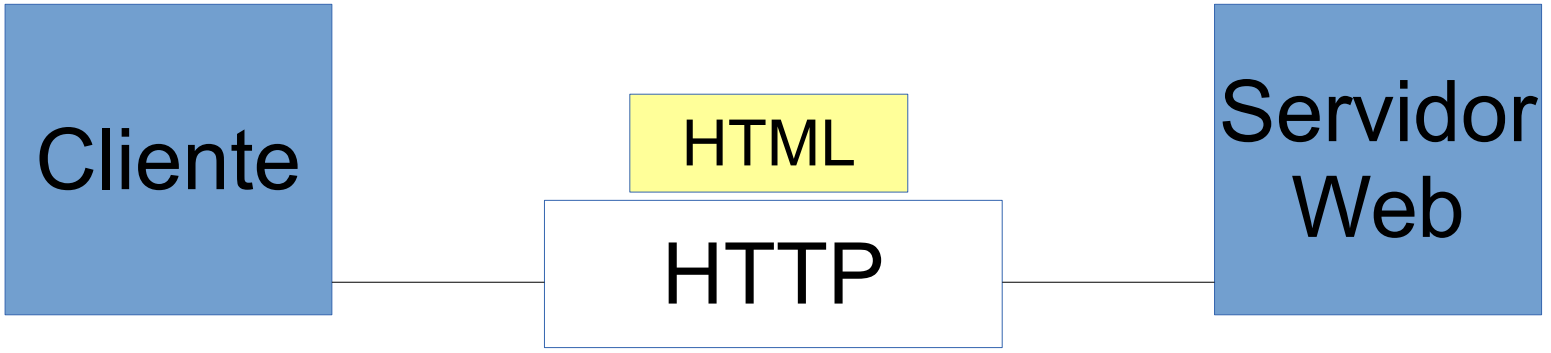
- Representa el estado de una base de datos en un momento dado
- Es un tipo de arquitectura que está basada en los estándares Web y HTTP
- En las arquitecturas REST todo es un **recurso**
- **Sin estado (stateless).** Excelente para sistemas distribuidos

REST - Recursos

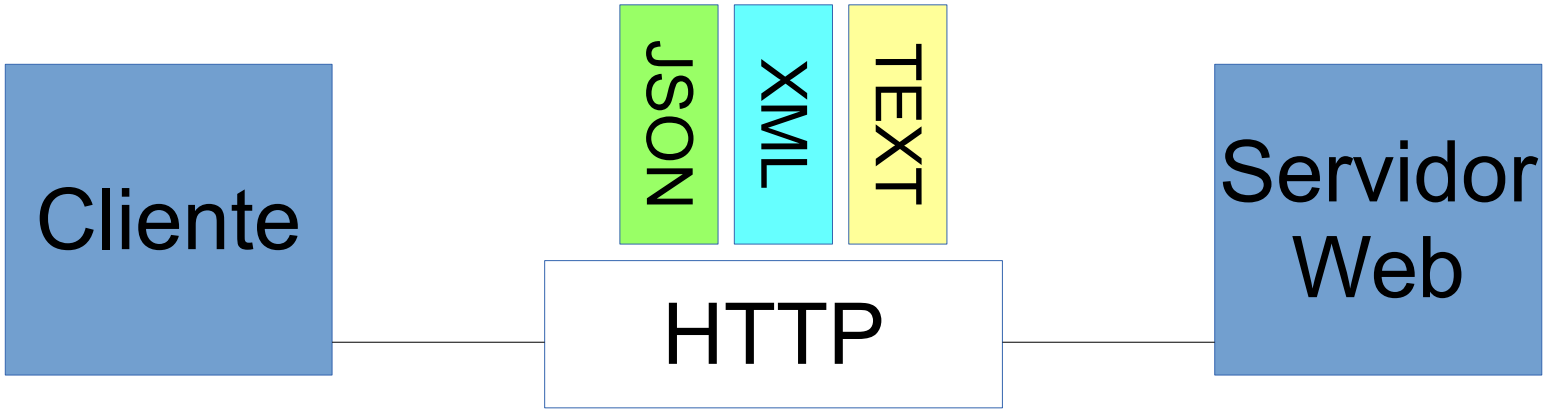
- Es accedido a través de una API común basada en los métodos HTTP.
- Son identificados por IDs globales → URLs/URIs
- REST permite diferentes representaciones para los recursos: texto plano, XML, JSON

REST – Modelo cliente/servidor

- Sitios Web:



- RESTful Web Services



CRUD

- Las cuatro operaciones básicas sobre un almacenamiento persistente son:
  - **C**REATE
  - **R**EAD
  - **U**PDATE
  - **D**ELETE

REST – CRUD - Métodos HTTP

CRUD	HTTP	RESTful WS	Descripción
INSERT	POST/ PUT	POST	Agrega un recurso nuevo
UPDATE	PUT/ POST/ PATCH	PUT	Actualiza o sobre-escribe un recurso existente
SELECT	GET	GET	Recupera un recurso. El recurso nunca se modifica a través de un GET
DELETE	DELETE	DELETE	Elimina un recurso

REST HTTP Request

Cada request debe especificar una URI para identificar el recurso

GET http://www.misistema.com/**api/v1/usuarios/58**

Recuperamos el usuario cuyo ID es el 58

```
GET /usuarios/58 http/1.1
Host: http://www.misistema.com/api/v1
Content-Type: application/json
Accept-Language: us-en
```

REST HTTP Response

HTTP/1.1 200 OK (285ms)
Date: Fri, 21 Apr 2019 10:00:10 GMT
Server: Apache/2.4.4 (CentOS) OpenSSL/1.01e-fips PHP/7.9
Content-Lenght: 100
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-type: application/json;charset=UTF-8

```
{
    “id”: 58,
    “nombre”: “Juan”,
    “apellido”: “Perez”
}
```

REST HTTP Request

POST http://www.misistema.com/**api/v1/usuarios**

Creamos un nuevo usuario. La información del mismo viaja en el cuerpo del request

```
POST /usuarios http/1.1
Host: http://www.misistema.com/api/v1
Content-Type: application/json
Accept-Language: us-en

{
    “nombre”: “Juan”,
    “apellido”: “Perez”
}
```

REST HTTP Response

HTTP/1.1 201 Created (285ms)
Date: Fri, 21 Apr 2019 10:00:10 GMT
Server: Apache/2.4.4 (CentOS) OpenSSL/1.01e-fips PHP/7.9
Content-Lenght: 100
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-type: application/json;charset=UTF-8

```
{
    “id”: 58,
    “resource”: “/usuarios/58”
}
```

REST HTTP Request

XXXX http://www.misistema.com/**api/v1/usuarios/58**

- **DELETE**: Borro al usuario con ID 58 del sistema
- **PUT**: Modifico el usuario con ID 58 en el sistema (los datos van en el body)

REST HTTP Request

GET http://www.misistema.com/**api/v1/usuarios**

- **GET**: Pido la lista de usuarios al server
- **DELETE**: Borro todos los usuarios?

REST API

Reglas para la definición de la API

- Usar sustantivos y no verbos para definir la estructura de la API que permite acceso a los recursos:

Propósito	Método	Incorrecto	Correcto
Recuperar lista de usuarios	GET	/getAllUsers	/users
Crear un nuevo usuario	POST	/createUser	/users
Borrar un usuario	DELETE	/deleteUser	/users/10
Obtener la s direcciones de un usuario	GET	/getUserAdresses	/users/10/adresses

REST API

Reglas para la definición de la API

- Indicar jerarquía con “/”
- Usar sustantivos plurales para todos los recursos:
  - /users en lugar de /user
  - /products en lugar de /product
- Usar sub-recursos para las relaciones:
  - GET /devices/10/sensors
  - GET /devices/10/sensors/1
- No usar “/” al final
- Usar “-” para que sea más fácil de leer y nombres cortos
- Usar parámetros en la URL (GET) para:
  - Filtrado → GET /devices?type=industrial
  - Ordenación → GET /devices?sort=-type
  - Paginación → GET /devices?offset=20&limit=5

REST API

Reglas para la definición de la API

- Manejar errores con los códigos de error de HTTP

Código	Descripción	Código	Descripción
200	Ok	403	Prohíbido (el server no permite la ejecución)
201	Ok (recurso creado)	404	No encontrado
204	Ok (recurso borrado)	405	Método no soportado
400	Request inválido (la razón viaja en el payload de respuesta)	408	Timeout
401	No autorizado (requiere autenticación)	5xx	Errores del servidor

Diseño de servicios RESTful

1)Identificar los recursos a ser expuestos como servicios
2)Modelar las relaciones entre los recursos con links que pueden ser seguidos para obtener más detalle o hacer cambios de estado
3)Definir URIs según las buenas prácticas y reglas para direccionar esos recursos
4)Entender como funcionarán los métodos HTTP GET, POST, PUT, DELETE para cada recurso
5)Diseñar y documentar la representación de cada recurso
6)Implementar y desplegar en un servidor Web
7)Probar con un Navegador u otra herramienta (Postman)

RESTful services con Flask

- Flask es un micro framework para construir aplicaciones Web en Python
- Core simple
- Extensible con módulos
- Soporta creación de servicios RESTful mediante el mapeo de URL
- Soporta built-in server
- Instalación:

```
sudo apt-get install python3-flask
```

Flask “Hello World”

- Una aplicación minimalista lucirá como:

```
from flask import Flask
app = Flask(__name__)

@app.route('/')
def hello_world():
    return 'Hello World!'

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

- Para ejecutar:

```
python flaskMinimalist.py
* Running on http://127.0.0.1:5000/
```

Flask - Mapeo de URIs - @route

- El decorador “route()” es usado para vincular un método a una URI:

```
@app.route('/users')
def get_users():
    allUsers = model.getAllUsers()
    response = app.response_class(
        response=json.dumps(allUsers),
        status=200,
        mimetype='application/json'
    )
    return response
```

Flask - Mapeo de URIs - @route

- Se pueden definir partes de las URIs como variables:

```
@app.route('/users/<userId>')
def get_user(userId):
    aUser = model.getUser(userId)
    response = app.response_class(
          response=json.dumps(aUser),
          status=200,
          mimetype='application/json'
      )
      return response
```

- Se puede definir el método HTTP soportado por la función. Por defecto, siempre es GET.

```
@app.route('/users/<userId>', methods=['GET','PUT'])
def get_user(userId):
```

Flask - Mapeo de URIs - @route

- Acceso a los datos del Request:

```
from flask import request
@app.route('/users/<userId>', methods=['GET','PUT'])
def get_user(userId):
    if request.method == 'GET':
        aUser = model.getUser(userId)
        …
        return response
    elif request.method == 'PUT':
        aUser = model.updateUser(userId,request.body)
       ...
    else:
         return 'Method not supported'
```

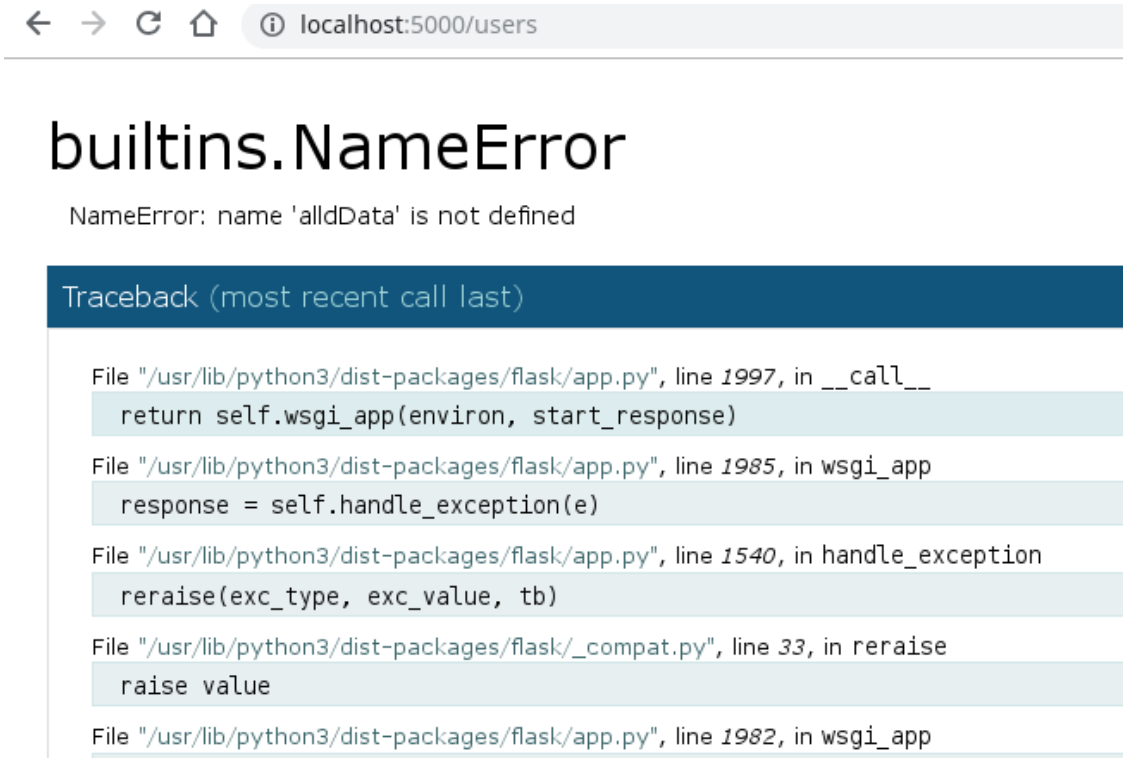
Flask - Interceptores antes y después del request

```
import sqlite3
from flask import g
@app.before_request
def before_request():
    g.db = sqlite3.connect(…)

@app.after_request
def after_request(response):
    g.db.close()
    return response
```

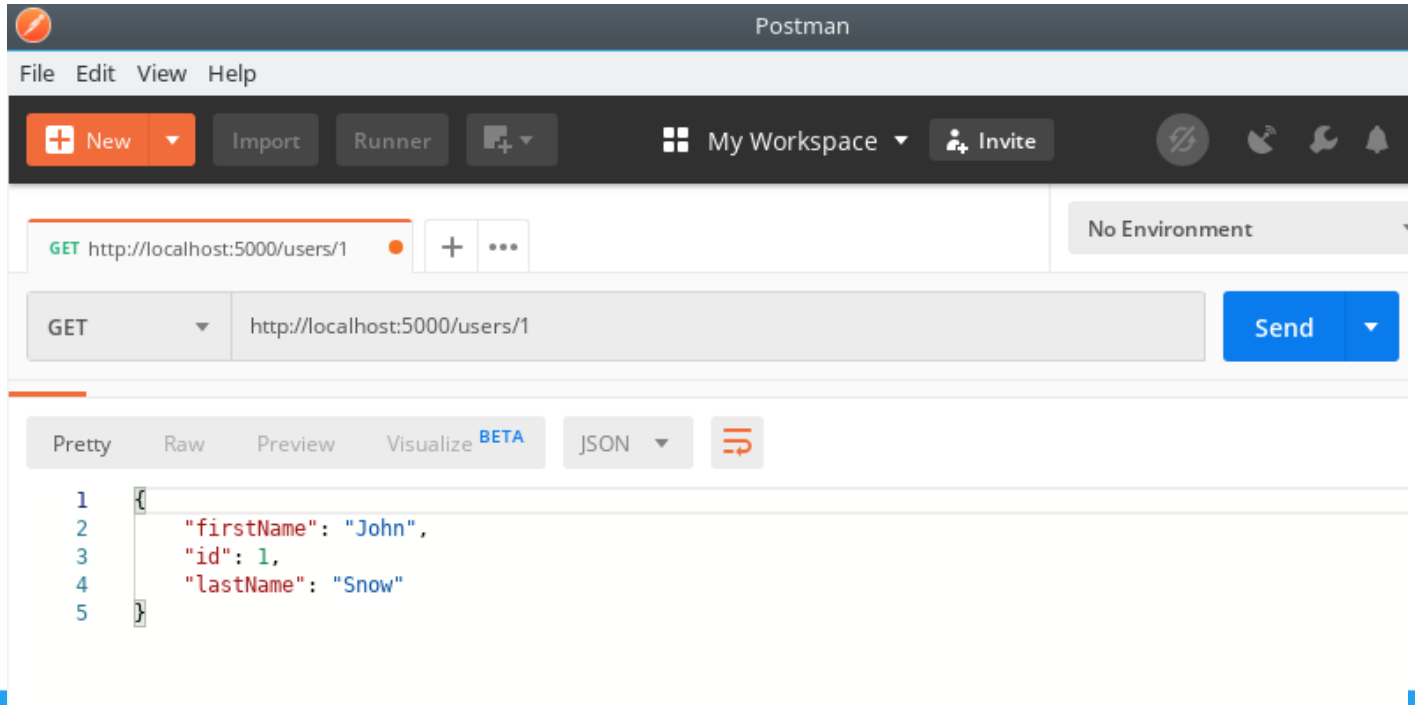
Flask

- Ventajas:
  - Rápido desarrollo y test
  - Se pueden cambiar los scripts sin detener el server → CGI
  - Consola de log Web



Probando con Postman

- Postman es una plataforma que asiste en el desarrollo y test de APIs Web
- Es un cliente que soporta diferentes arquitecturas: REST, SOAP, GrapQL



Arquitectura orientada a servicios

- Estilo de diseño de software donde los servicios son provistos a otros componentes mediante componentes de aplicación, a través de un protocolo de comunicación
- Un servicio es una unidad funcional discreta que puede ser accedida remotamente
- Tiene cuatro propiedades:
  1)Representa una actividad o lógica de negocios
  2)Es auto contenido
  3)Funciona como caja negra
  4)Puede estar compuesto por otros servicios subyacentes
- Integra componentes de software distribuidos

SOA + RESTful

- Un servicio Web es un servicio construido con tecnologías Web
- RESTful es un patrón de diseño que permite la implementación de servicios Web
- Sin embargo, RESTful no permite implementar todos los principos de SOA:
  - No existe la definición de un contrato de servicio
  - Necesita orquestación para la composición de procesos

Bibliografía

- Fielding, Roy T.; Gettys, James; Mogul, Jeffrey C.; Nielsen, Henrik Frystyk; Masinter, Larry; Leach, Paul J.; Berners-Lee, Tim (June 1999). Hypertext Transfer Protocol – HTTP/1.1

- Practical Internet of Things with JavaScript. Arvind Ravulavaru. 2017. Packt>

- https://restfulapi.net

- Flask https://flask.palletsprojects.com/
- https://en.wikipedia.org/wiki/Service-oriented_architecture
- https://publications.opengroup.org/standards/soa
- http://dret.net/netdret/docs/soa-rest-www2009/