



UT07: WEB CONTROLLERS Y CLIENTE WEB EN ODOO

ÍNDICE

1.- Web Controllers

2.- APIs REST con Web Controllers

1

WEB CONTROLLERS



Los **Web Controllers** en Odoo son componentes que permiten interactuar con el sistema a través de **rutas HTTP**.

Sus principales usos son:

- **Crear interfaces personalizadas:** se pueden crear **interfaces web** personalizadas que no están disponibles en los módulos predeterminados de Odoo.
- **Exponer datos como APIs:** permiten crear **endpoints REST** que expongan datos del sistema a aplicaciones externas
- **Procesar formularios web:** permiten recoger datos enviados desde formularios web, como registros de usuarios, encuestas o formularios.
- **Integrar servicios externos:** se pueden utilizar para interactuar con APIs de terceros.
- **Gestionar contenido web:** se pueden usar para proporcionar contenido dinámico en las páginas web del sitio construido en Odoo.

Los **Web Controllers** se definen en un archivo Python dentro del directorio controllers.



Al igual que en otras partes de Odoo (como en el modelo), se puede cambiar el nombre del fichero siempre y cuando también se actualice en el fichero `__init__.py`

```
# -*- coding: utf-8 -*-  
  
from . import controllers
```

El código básico de un Web Controller sería el siguiente:

```
from odoo import http
from odoo.http import request
```

```
class MyController(http.Controller):
    @http.route('/cpu/hello', type='http', auth='public', website=True)
    def hello_world(self, **kwargs):
        return "Hola mundo"
```

Importaciones necesarias para trabajar con HTTP

Los controladores tienen que heredar de **http.Controller**

El decorador **@http.route** se utilizará para definir las rutas HTTP

Cada ruta que definimos tendrá su propia función

← → ↻ 🛡️ 📄 localhost:8069/cpu/hello

Hola mundo

Si accedemos a la ruta que hemos indicado veremos como nos devuelve la cadena de texto de la función

Los parámetros del decorador son:

Ruta del endpoint

Tipo de autenticación, con
'public' indica que cualquier
usuario puede acceder a la URL

```
@http.route('/cpu/hello', type='http', auth='public', website=True)
```

Indica que la ruta devuelve una respuesta
HTTP estándar. La otra opción disponible es
'json' para indicar que la ruta es un endpoint
JSON-RPC (JSON Remote Procedure Call)

Indica que la ruta es parte de un
sitio Web (se renderiza como
HTML))

Podemos devolver diferentes tipos de contenido:


- Texto plano
- Página web estática utilizando una plantilla HTML
- Página web dinámica que interactúa con los datos almacenados en Odoo
- Un JSON si quiero implementar una API REST

Generar una página web estática

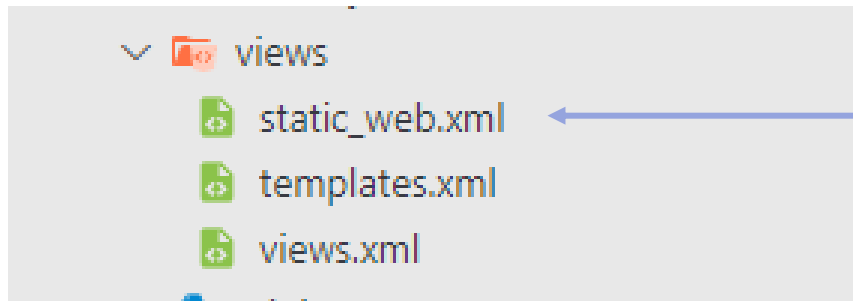
En este caso, deberemos utilizar el método **http.request.render()** para hacer referencia a una plantilla XML que tendremos en el directorio **views**.

Veamos cómo sería el código:

```
class MyController(http.Controller):
    @http.route('/cpu/static', type='http', auth='public', website=True)
    def hello_world(self, **kwargs):
        return http.request.render('cpu_management.static_web', {})
```



Este será el identificador que le daremos a la plantilla XML. Observa que la forma de referenciarlo es con el nombre del módulo, el carácter punto y el identificador de la plantilla



Creo el fichero para la plantilla en views y no me olvido de incluirlo en el archivo `__manifest__` para que lo cargue.

```
# always loaded
'data': [
    'security/ir.model.access.csv',
    'views/views.xml',
    'views/static_web.xml',
    # 'views/templates.xml',
],
```

El identificador que referencio en el modelo

Nombre que le doy a la plantilla (recuerda que en Odoo todo tiene un identificador y un nombre)

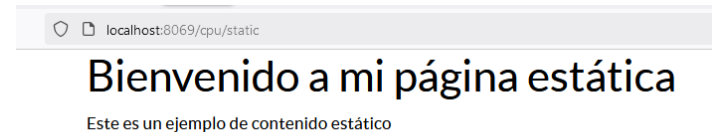
```
OdooDesarrollo > volumesOdoo > addons > cpu_management > views > static_web.xml
1  <odoo>
2      <template id="static_web" name="Una página estática">
3          <t t-call="web.html_container">
4              <div class="container">
5                  <h1>Bienvenido a mi página estática</h1>
6                  <p>Este es un ejemplo de contenido estático</p>
7              </div>
8          </t>
9      </template>
10 </odoo>
```

La plantilla **web.html_container** una estructura HTML básica

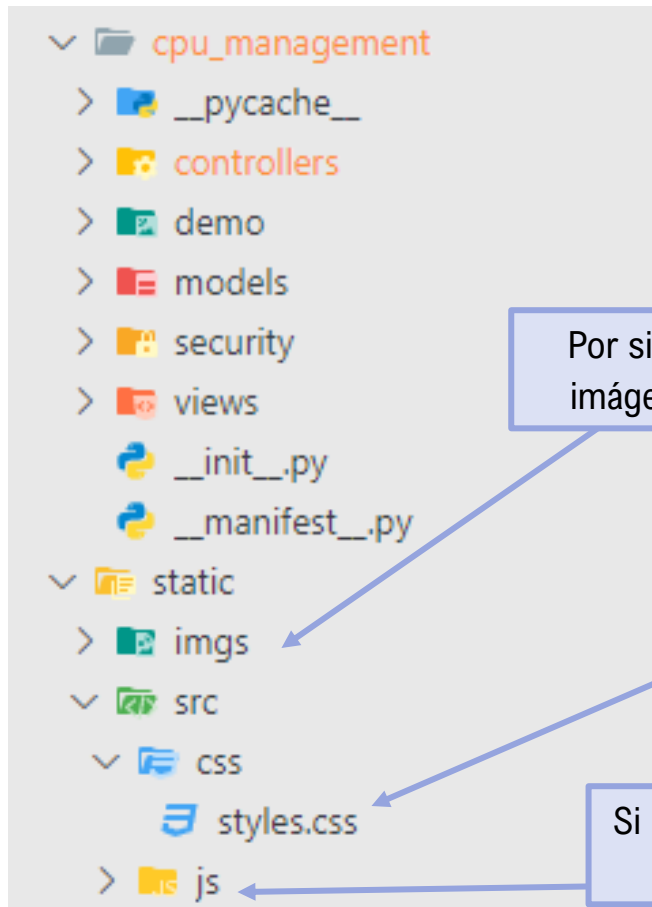
El atributo **t-call** sirve para integrar otra plantilla

Este es el contenido web estático que se mostrará

La etiqueta **<t>** es un elemento especial de las plantillas XML de **QWeb**, el motor de plantillas de Odoo. Según el atributo realizará diferentes acciones.



Si queremos añadir estilos CSS o bien contenido Javascript a la página también es posible. Veamos un ejemplo de cómo agregar estilos CSS.



Por si quiero insertar imágenes en mi web

Fichero con mis estilos CSS

Si quisiera añadir Javascript

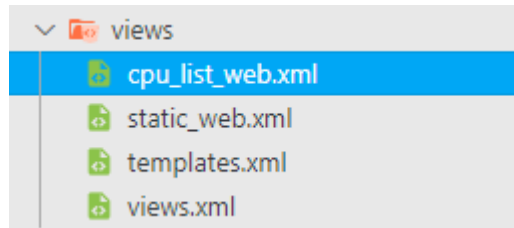
```
OdooDesarrollo > volumesOdoo > addons > static > src > css > styles.css > ...  
1  h1 {  
2      background-color: chartreuse;  
3  }  
4
```

En este ejemplo voy a cambiar el color de fondo del encabezado.

Generar una página web dinámica

Crear una web estática puede ser útil para crear una *landing page* o similar, pero lo normal al crear páginas en Odoo es generar una web dinámica que obtenga la información de la base de datos.

Veamos cómo hacerlo mediante un ejemplo creado una web que muestre los nombres de los microprocesadores y su fabricante.



Comenzamos creando el fichero XML con la plantilla y referenciándolo en el `__manifest__`

```
'data': [
    'security/ir.model.access.csv',
    'views/views.xml',
    'views/static_web.xml',
    'views/cpu_list_web.xml',
    # 'views/templates.xml',
],
```

En el controlador tenemos que leer los datos del modelo. Lo hacemos con esta línea.

```
OdooDesarrollo > volumesOdoo > addons > cpu_management > controllers > controllers.py
1
2 @http.route('/cpu/list', type='http', auth='public', website=True)
3 def cpu_list_web(self, **kwargs):
4     cpus = request.env['cpu_management.cpu'].search([])
5     return http.request.render('cpu_management.cpu_list_web', {
6         'cpus': cpus
7     })
8
```

Hay que enviar los datos que hemos leído a la plantilla, para lo que utilizamos el segundo parámetro de `http.request.render`, que recoge un diccionario con los datos que se envían

El atributo **t-foreach** sirve para iterar sobre los datos

El atributo **t-as** indica cómo llamamos a cada elemento sobre el que iteramos y nos sirve para hacer referencia a él

```
OdooDesarrollo > volumesOdoo > addons > cpu_management > views > cpu_list_web.xml
1  <odoo>
2      <template id="cpu_list_web">
3          <t t-call="web.html_container">
4              <div class="container">
5                  <h1>Listado de CPUs</h1>
6                  <t t-foreach="cpus" t-as="cpu">
7                      <div class="cpu">
8                          <span><t t-esc="cpu.name"/></span>
9                          <span><t t-esc="cpu.manufacturer"/></span>
10                     </div>
11                 </t>
12             </div>
13         </t>
14     </template>
15 </odoo>
```

El atributo **t-esc** muestra un dato tras sanitizarlo para evitar

Como habrás deducido, una parte relevante del código anterior es el objeto **request.env**. Este es un objeto disponible dentro de los controladores web de Odoo que se inicializa automáticamente con el entorno de ejecución de la solicitud HTTP en curso.

Acceso a modelos con request.env

Para acceder a un modelo de la base de datos simplemente tendremos que utilizar la siguiente sintaxis:

```
http.request.env['module.model_name']
```

Donde *module* es el nombre del módulo y *model_name* el nombre del modelo que quieres obtener

Acceso un modelo con un dominio de búsqueda

Una vez que tenemos el modelo podemos obtener todos los registros con el método **search()**.

```
http.request.env['module.model_name'].search([])
```

En el código anterior no aplicamos ningún filtro (lista vacía que se le pasa como parámetro), por lo que obtendremos todos los registros.

Listado de CPUs

Ryzen 7 3800	AMD
Ryzen 9 7950X	AMD
Core i9 10900X	Intel
Ryzen 5 5600	AMD
Core i7 12700KF	Intel

En el parámetro de **search()** podemos usar un dominio para que únicamente se obtengan los registros que cumplan una serie de condiciones. La forma de hacerlo es pasando una lista de tuplas donde cada tupla tiene 3 elementos: campo, operador de condición y valor con el que se compara.

```
ient.cpu'].search([("manufacturer", "=", "AMD")])
```

Listado de CPUs

Ryzen 7 3800	AMD
Ryzen 9 7950X	AMD
Ryzen 5 5600	AMD

Limitar el número de resultados

Podemos poner un límite al número de resultados devueltos con el parámetro **limit**.

Por ejemplo, si solo queremos el primer registro usaríamos la siguiente orden:

```
request.env['cpu_management.cpu'].search([], limit=1)
```

Listado de CPUs

Ryzen 7 3800

AMD

La otra parte que te habrá llamado la atención del código de la página web es el uso de la **etiqueta <t>**.

Esta etiqueta es un elemento especial de las plantillas XML de QWeb, y tiene diferentes usos según los atributos que tenga.

Atributo t-esc

El atributo **t-esc** renderiza una variable o expresión, escapando caracteres especiales para evitar problemas de seguridad como la inyección de código.

```
<p><b>Usuario:</b><t t-esc="request.env.user.name" /></p>
```

En este ejemplo observa que también se puede acceder al objeto **request.env** desde el fichero XML

Aquí le damos un uso diferente, no accedemos a un modelo, sino a la propiedad **user** que contiene los datos del usuario actual

Atributo t-raw

Renderiza una variable o expresión **sin escapar**. Esto es útil para textos que contienen HTML y queremos que se renderice.

Este atributo solo deberíamos utilizarlo cuando sea absolutamente imprescindible y si estamos muy seguros de la fuente del contenido, ya que puede abrir nuestro sistema a vulnerabilidades como ataques XSS

```
<t t-raw="'<strong>Texto en negrita</strong>'"
```

Atributo t-if, t-else y t-elif

Evalúa una condición y renderiza el contenido sólo si la condición es verdadera.

```
<div class="container">
  <h1>Listado de CPUs</h1>
  <t t-if="cpus">
    <t t-foreach="cpus" t-as="cpu">
      <div class="cpu">
        <span class="name"><t t-esc="cpu.name" /></span>
        <span class="manufacturer"><t t-esc="cpu.manufacturer" /></span>
      </div>
    </t>
  </t>
  <t t-else="">
    <p>No hay ninguna CPU registrada.</p>
  </t>
</div>
```

Comprobamos si cpus contiene algún registro

Para que sea sintácticamente válido, debemos pasarle la cadena vacía a **t-else**

Atributo t-foreach y t-as

Se utiliza para iterar sobre listas, tuplas o cualquier estructura iterable.


Este atributo (salvando las diferencias de sintaxis) es equivalente al for .. in .. de Python.

```
<ul>
|   <t t-foreach="[1, 2,3]" t-as="num">
|   |   <li>Número: <t t-esc="num" /></li>
|   </t>
</ul>
```

Atributo t-set y t-value

Define una variable dentro de una plantilla. Indicamos el nombre de la variable con el atributo **t-set** y el valor con el atributo **t-value**

```
<t t-set="full_name" t-value="cpu.name + ' (' + cpu.manufacturer + ')'" />  
<span><t t-esc="full_name" /></span>
```



En este ejemplo estamos utilizando una expresión para asignar el valor

2

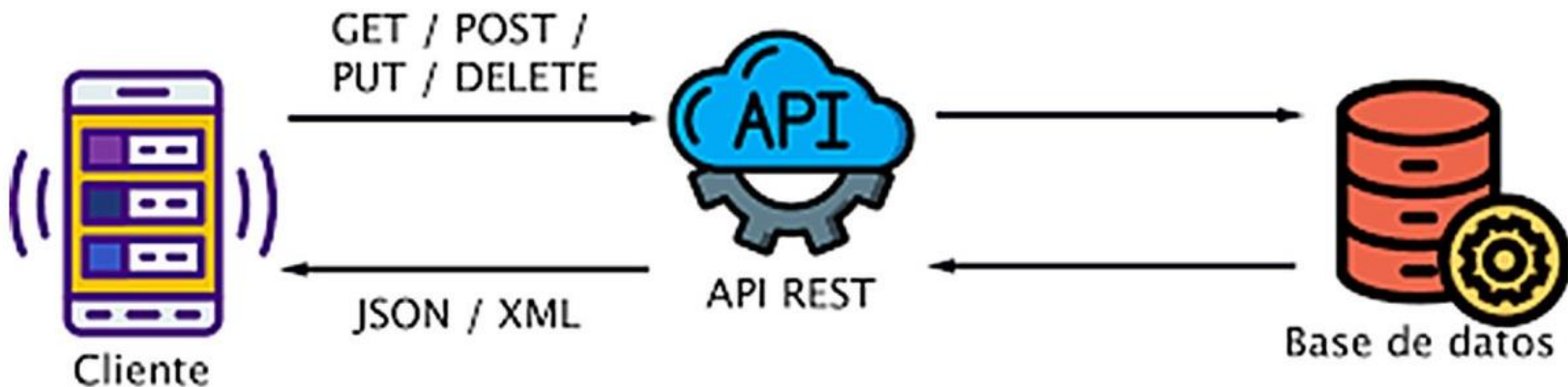
APIS REST CON WEB CONTROLLERS



REST (Representational State Transfer) es un estilo de arquitectura del software para comunicaciones cliente servidor apoyado en el protocolo HTTP.

REST se basa en tres conceptos clave:

- URLs
- Métodos HTTP
- Estados de respuesta.



URL (Uniform Resource Locator)

Una **URL** es la dirección que se le da a un recurso en la red. REST redefine este concepto utilizándolo para identificar recursos, pero también asignándoles nombres representativos.

Así, las consultas a la API son fácilmente comprensibles.

Por ejemplo:

- <https://swapi.dev/api/people/1/>
- https://api.twitter.com/2/users/:id/timelines/reverse_chronological

Métodos HTTP

Los métodos HTTP se utilizan para indicar qué se quiere hacer con un recurso determinado.

Se utilizan cuatro métodos principalmente, asociados con las operaciones CRUD:

- **GET:** para obtener o leer un recurso.
- **PUT:** actualiza o reemplaza un recurso
- **DELETE:** elimina un recurso del servidor
- **POST:** crea un recurso en el servidor

Estados de respuesta

El resultado de la consulta a la API se indica en el campo de estado de la respuesta HTTP.

Los estados definidos por el estándar HTTP son:

- 1xx Informational
- 2xx Success
- 3xx Redirection
- 4xx Client Error
- 5xx Server Error

Ejemplos:

- **200 (OK):** operación realizada con éxito
- **401 (Unauthorized):** el usuario no ha facilitado el método de autenticación requerido por la API y no tiene acceso al recurso.
- **403 (Forbidden):** el usuario no tiene permiso para acceder al recurso.
- **404 (Not Found):** indica que la API REST no puede mapear la URI con un recurso, pero puede que sí pueda en un futuro, por lo que sí se permitirían futuras solicitudes.
- **501 (Not Implemented):** el servidor no reconoce la solicitud o el método, pero probablemente será una funcionalidad futura.

**MORE
INFO**<https://restfulapi.net/http-status-codes/>

Para realizar consultas REST desde el navegador podemos utilizar el addon de Firefox **RESTClient**.

The image shows the RESTClient interface within a Firefox browser window. The interface is divided into three main sections: Request, Response, and Curl. The Request section is active, showing a GET method and a URL of http://www.example.com. The Response section is currently empty. The Curl section shows a placeholder for a curl command. The Firefox Add-ons page is also visible, showing the RESTClient extension by Chao ZHOU with a 4.1-star rating and 145 reviews.

RESTClient Interface:

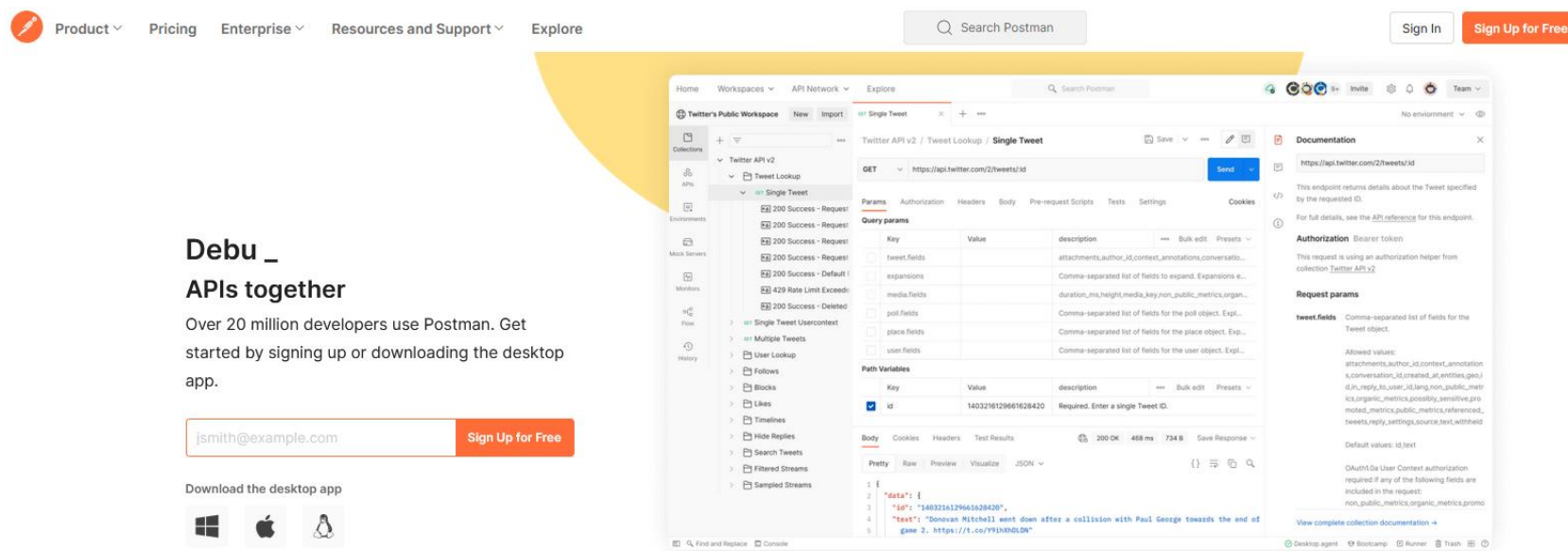
- Request:** Method: GET, URL: http://www.example.com, SEND button.
- Response:** Headers, Response tabs.
- Curl:** Command, CURL command placeholder.

Firefox Add-ons Page:

- Firefox Browser ADD-ONS**
- RESTClient, a debugger for RESTful web services.** by Chao ZHOU
- 42,710 Users**, **145 Reviews**, **4.1 Stars**
- 5 stars:** 88
- 4 stars:** 25
- 3 stars:** 10
- 2 stars:** 4
- 1 star:** 18

Footer: Home Fork me Report an issue v3.0.7

Alternativamente, se puede utilizar **Postman**, una aplicación de escritorio con muchas más opciones.






Debu _
APIs together

Over 20 million developers use Postman. Get started by signing up or downloading the desktop app.

[Sign Up for Free](#)

Download the desktop app

What is Postman?

Postman is an API platform for building and using APIs. Postman simplifies each step of the API lifecycle and streamlines collaboration so you can create better APIs—faster.



Ejemplo de consulta a StarWars API

The screenshot shows the RESTClient interface. At the top, there are tabs for Authentication, Headers, and View. On the right, there are links for Favorites, Data migration, and RESTClient. The main area is divided into three sections: Request, Response, and Curl.

Request Section:

- Method: GET
- URL: `https://swapi.dev/api/people/1`
- Body: Request Body

Response Section:

The Response section has three tabs: Headers (selected), Response, and Preview. The Headers tab displays the following response headers:

Header	Value
Status Code	: 200 OK
allow	: GET, HEAD, OPTIONS
content-type	: application/json
date	: Wed, 28 Dec 2022 07:56:35 GMT
etag	: "ee398610435c328f4d0a4e1b0d2f7bbc"
server	: nginx/1.16.1
strict-transport-security	: max-age=15768000
vary	: Accept, Cookie
x-firefox-spdy	: h2
x-frame-options	: SAMEORIGIN

Curl Section:



The Curl section shows the command used to make the request:

```
curl -X GET -k -i 'https://swapi.dev/api/people/1'
```

RESTClient

Authentication ▾ Headers ▾ View ▾ Favorites ▾ Data migration

[-] Request

Method **GET** ▾ URL  `https://swapi.dev/api/people/1251`  ▾ **SEND**

Body



Request Body

[-] Response

Headers Response Preview

Status Code	: 404 Not Found
allow	: GET, HEAD, OPTIONS
content-type	: application/json
date	: Wed, 28 Dec 2022 07:58:30 GMT
etag	: "8bee5c3ad44d6c57f19e49e8e76ee09e"
server	: nginx/1.16.1
vary	: Accept, Cookie
x-firefox-spdy	: h2
x-frame-options	: SAMEORIGIN

[-] Curl

Command  

```
curl -X GET -k -i 'https://swapi.dev/api/people/1251'
```

Algunas APIs permiten pasar parámetros en la URL

Searching

All resources support a `search` parameter that filters the set of resources returned. This allows you to make queries like:

```
https://swapi.dev/api/people/?search=r2
```

All searches will use case-insensitive partial matches on the set of search fields. To see the set of search fields for each resource,

[-] Request

Method URL

Body

Request Body

[-] Response

Headers Response Preview

```
1 {  
2   "count": 1,  
3   "next": null,  
4   "previous": null,  
5   "results": [{
```

De forma análoga a cómo creamos páginas dinámicas en Odoo, podemos crear un API REST que realice consultas contra la base de datos.

Tenemos que tener en cuenta lo siguiente:

- Se utiliza el protocolo HTTP
- Toda la información que se intercambia entre el cliente y el servidor en una API REST se hace en formato JSON
- Debemos controlar los encabezados de la respuesta para indicar el estado de la respuesta, así como el tipo de contenido

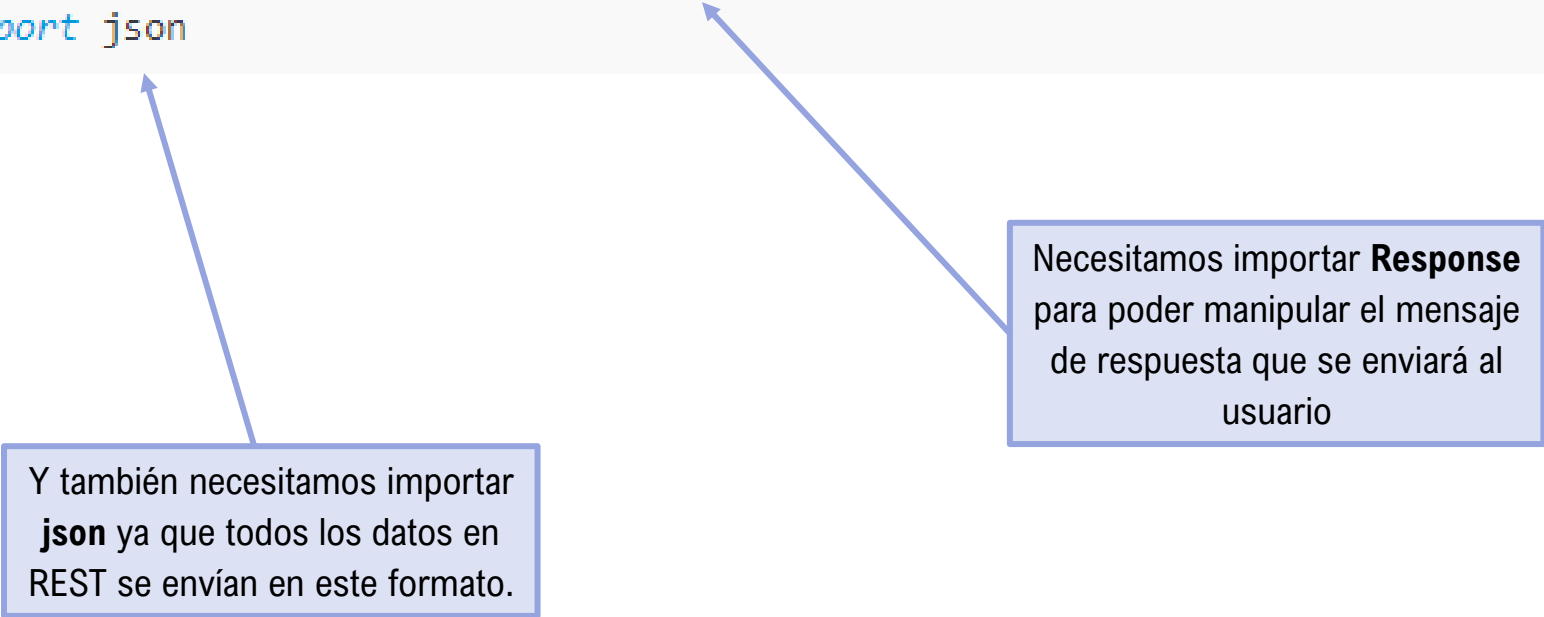
Vamos a ver cómo implementaríamos el **endpoint GET /api/cpu** para que devuelva un listado de todos los procesadores con su nombre, fabricante y número de núcleos

```
from odoo import http # type: ignore
from odoo.http import request, Response # type: ignore
import json

class MyController(http.Controller):

    @http.route("/api/cpu", type="http", methods=['GET'], csrf=False)
    def get_cpus(self, **kwargs):
        cpus = request.env['cpu_management.cpu'].search([])
        result = []
        for cpu in cpus:
            result.append({
                'name': cpu.name,
                'manufacturer': cpu.manufacturer,
                'total_cores': cpu.total_cores,
            })
        response = Response(
            json.dumps( result ),
            content_type = 'application/json',
            status = 200
        )
        return response
```

```
from odoo import http                                # type: ignore
from odoo.http import request, Response              # type: ignore
import json
```



Y también necesitamos importar **json** ya que todos los datos en REST se envían en este formato.

Necesitamos importar **Response** para poder manipular el mensaje de respuesta que se enviará al usuario

Las rutas REST tienen **nombres representativos**

```
class MyController(http.Controller):  
    @http.route("/api/cpu", type="http", methods=['GET'], csrf=False)  
    def get_cpu(self, **kwargs):
```

REST utiliza el protocolo HTTP.
Mucho cuidado aquí porque,
aunque los datos se transmiten
en JSON, la otra opción (json) es
para comunicaciones JSON RDP

El método sirve para
indicar qué operación
quiere realizar el usuario.
Los métodos son GET
(obtener datos), POST
(enviar datos), PUT
(modificar datos) y DELETE
(eliminar datos)

En este ejemplo estamos recogiendo todos los registros que hay en el modelo cpu

```
def get_cpus(self, **kwargs):  
    cpus = request.env['cpu_management.cpu'].search([])  
    result = []  
    for cpu in cpus:  
        result.append({  
            'name': cpu.name,  
            'manufacturer': cpu.manufacturer,  
            'total_cores': cpu.total_cores,  
        })
```

Como aquí no quiero todos los campos, creo un diccionario en blanco y agrego los campos que voy a devolver.

Aquí construimos la respuesta, para ello creamos una instancia de la clase **Response**

El primer parámetro de Response será el cuerpo del mensaje. Utilizamos el método **json.dumps()** que convierte una lista/diccionario de Python en una cadena JSON

Si quisiéramos realizar el paso contrario (de cadena a lista/diccionario) debemos usar el método **json.loads()**

Indicamos que el contenido del mensaje está en JSON, esto añadirá la cabecera *ContentType*:
application/json

Alternativamente podíamos haber indicado nosotros directamente en encabezado con el parámetro **headers**

```
json.dumps( result ),  
headers={"Content-Type": "application/json"},  
status = 200
```

```
response = Response(  
    json.dumps( result ),  
    content_type = 'application/json',  
    status = 200  
)  
return response
```

Con **status** indicamos el código de estado (por ejemplo, 200 para OK o 404 si el recurso no se ha encontrado)

Ejemplo pasando parámetros en la URL

```
@http.route("/api/cpu/<string:name>", type="http", methods=['GET'], csrf=False)
def get_cpu_by_name(self, name):
    try:
        cpu = request.env['cpu_management.cpu'].search([('name', '=', name)], limit=1)
        if not cpu:
            return Response(
                json.dumps({'msg': 'CPU no encontrada'}),
                content_type="application/json",
                status=404,
            )
        data = {
            "id": cpu.id,
            "name": cpu.name,
            "manufacturer": cpu.manufacturer,
        }
        return Response(
            json.dumps(data),
            content_type="application/json",
            status=200
        )
    except Exception as e:
        return Response(
            json.dumps({'msg': f"Error interno del servidor: {str(e)}"}),
            content_type="application/json",
            status=500
        )
```


Ejemplo con Query String

```
@http.route('/api/cpu', type='http', auth='none', csrf=False, methods=['GET'])
def get_cpus_by_manufacturer(self):
    try:
        manufacturer = request.params.get('manufacturer')
        if manufacturer:
            cpus = request.env['cpu_management.cpu'].sudo().search([('manufacturer', '=', manufacturer)])
        else:
            cpus = request.env['cpu_management.cpu'].sudo().search([])
        data = [
            {"id": cpu.id, "name": cpu.name, "manufacturer": cpu.manufacturer, "cores": cpu.total_cores}
            for cpu in cpus
        ]
        return Response(
            json.dumps(data),
            content_type="application/json",
            status=200
        )
    except Exception as e:
        return Response(
            json.dumps({"error": f"Internal server error: {str(e)}"}),
            content_type="application/json",
            status=500
        )
```


Veamos ahora el **endpoint POST /api/cpu**

```
@http.route("/api/cpu", type='http', methods=['POST'], csrf=False)
def create_cpu(self, **kwargs):
    try:
        raw_body = request.httprequest.data
        data = json.loads(raw_body.decode('utf-8'))
    except Exception as e:
        return Response(f"Invalid JSON: {str(e)}", status=400)
    cpu = request.env['cpu_management.cpu'].create({
        'name': data.get('name'),
        'manufacturer': data.get('manufacturer'),
    })
    return json.dumps({
        'id': cpu.id,
        'name': cpu.name,
        'manufacturer': cpu.manufacturer,
    })
```

