



# API Rest Consideraciones

---

UNJu - Desarrollo y Arquitecturas Avanzadas de Software

*Ing. José Zapana*



# ¿Qué es Rest?

- Es un estilo arquitectónico para la Web que permite diseñar servicios web bajo ciertos parámetros
- Devuelve un json de resultado, estatus y headers

The screenshot shows a REST client interface for the Star Wars API. The request is a GET to `https://swapi.dev/api/people`. The response is a JSON object with the following structure:

```
1 {
2   "count": 82,
3   "next": "https://swapi.dev/api/people/?page=2",
4   "previous": null,
5   "results": [
6     {
7       "name": "Luke Skywalker",
8       "height": "172",
9       "mass": "77",
10      "hair_color": "blond",
11      "skin_color": "fair",
12      "eye_color": "blue",
13      "birth_year": "198BY",
14      "gender": "male",
15      "homeworld": "https://swapi.dev/api/planets/1/",
16      "films": [
17        "https://swapi.dev/api/films/1/",
```

The interface also shows the status of the response: 200 OK, Time: 1219 ms, Size: 5.8 KB.



# Restricciones de Rest

---

- Es un esquema cliente / servidor: solicitud http y respuesta http
- Interfaces estandarizadas:
  - Recursos (o entidades):
  - Representaciones: formato (json o xml)
  - Mensajes descriptivos: explican la intención
- Es stateless: no hay memoria de ejecuciones previas



## Otras Restricciones

- Puede ser cacheable. Para solicitudes comunes para que funcione más rápido, puede estar en el servidor o en el cliente.
- Código bajo demanda (opcional). Que sea ejecutado por el cliente
- Sistema de capas.





# Consejos para diseñar API Rest

---

1. Representa recursos y no acciones. La imagen siguiente muestra acciones y no son representaciones correctas

```
GET /deleteProduct?id=1234  
GET /deleteProduct/1234  
POST /products/1234/delete
```

Lo correcto es que exista un recurso llamado producto y mediante los métodos HTTP se pueda indicar la acción a realizar

```
DELETE /products/1234
```

Aquí estamos indicando el método http DELETE.



# ¿Cómo se definen los recursos?

Existen 4 tipos de recursos

- **Colecciones:** representan un conjunto de datos, lista de todos los productos
- **Documentos:** por ejemplo 1234 es un documento dentro de la colección de productos.
- **Stores:** recursos adicionales generalmente controlados desde el cliente
- **Controladores:** acciones





# Ejemplo

---

**Colecciones (plural)**

`/products`

**Stores (plural)**

`/users/20/favorites`

**Documentos (singular)**

`/products/1`

`/products/pencil`

**Controladores (verbos)**

`/users/20/reset-password`

Stores: muestra la lista de favoritos del usuario 20

Controladores: resetear el pass del usuario 20, normalmente se usan métodos POST.



## Consejo 2 - No devuelvas siempre 200 OK.

Ejemplo

```
POST /customers
HTTP/1.1 200 OK
Date: Fri, 28 Feb 2020 09:09:09 GMT
Content-Type: application/json

{
  "error": true,
  "message": "Invalid ID",
}
```

Utilizar los códigos HTTP para dar significado a la respuesta. Por ejemplo

- 201: recurso creado
- 202: Solicitud recibida. En proceso.
- 204: Solicitud exitosa. Respuesta sin contenido (por ejemplo, en un delete)
- 401: No autorizado.
- 403: Acceso prohibido (por ejemplo, no puede eliminar)
- 404: Recurso no encontrado
- 405: método no permitido
- 500: error interno del servidor.





## Consejo 3 - No hagas todo con POST

---

```
POST /customers/1234/delete
```

Solución: indicar los métodos HTTP para indicar la intención:

- GET: Obtener un recurso (colección o un dato)
- POST: crear un nuevo recurso
- PUT: actualizar un recurso existente
- DELETE: Eliminar recursos
- PATCH: actualiza un recurso existente. Solo actualiza los campos que se quieren modificar.
- OPTIONS: Obtener metadatos para interactuar



## Consejo 4 - Asegurar la API

---

Mediante autenticación

- API Key
- Bearer Token
- OAuth 1.0, etc



## Consejo 5 - Versionar la API

---

Ejemplo de la API de Google para traer videos

```
https://www.googleapis.com/youtube/v3/videos
```

Crear una nueva versión de la API y dejar la anterior hasta que los clientes puedan migrar a la nueva versión y luego quitar la versión anterior.



# Referencias

---

- [Building a RESTful Web Service](#)
- [The Spring @Controller and @RestController Annotations](#)
- [Building REST services with Spring](#)