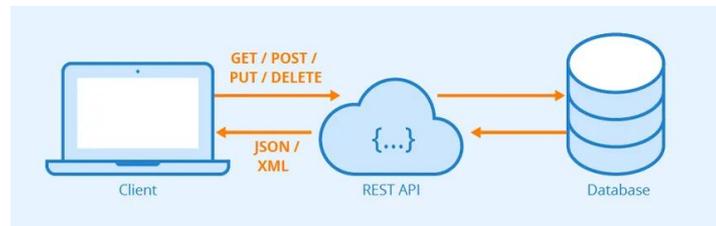




API Rest



UNJu - Programación Orientada a Objetos



¿Qué es REST?

- **R**epresentational **S**tate **T**ransfer
- Es un estilo arquitectónico para la Web que permite diseñar servicios web bajo ciertos parámetros
- Devuelve un json de resultado, estatus y headers
- Apis públicas
 - <https://swapi.dev/api/people>
 - <https://petstore.swagger.io/#/>



Aclarando conceptos

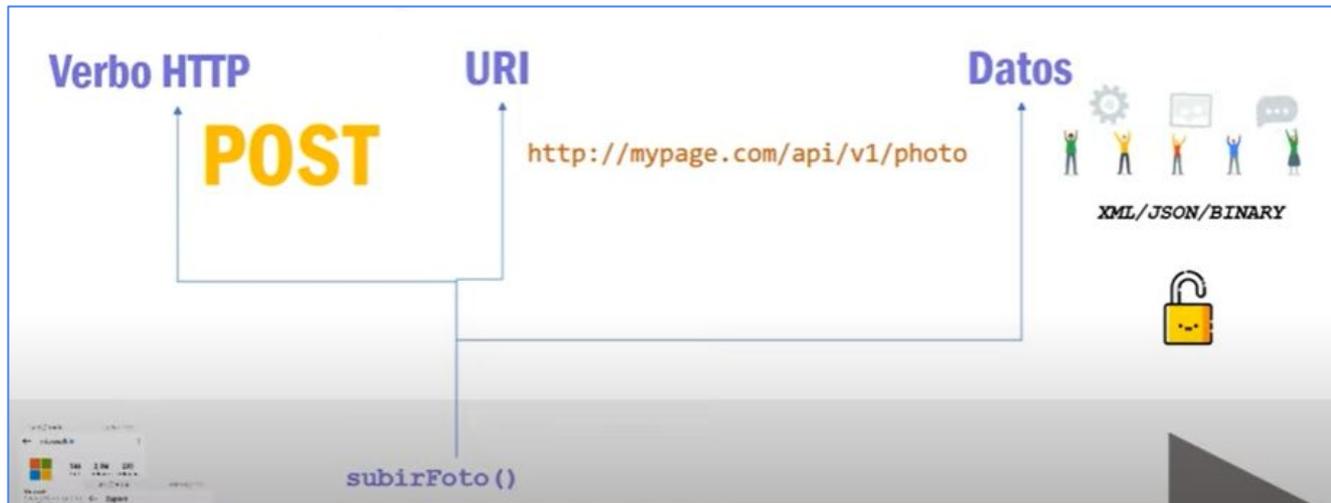
- **API:** Es una abstracción de funciones y procedimientos
- **Rest:** Es una lógica de restricciones y recomendaciones bajo la cual se construye la API (estilo de arquitectura)
- **Rest Full API:**
 - Es una api implementada construida utilizando la lógica de Rest.
 - Funcionan en una arquitectura cliente servidor
 - Utilizan el protocolo HTTP





Partes de una API Rest

- En Rest cada procedimiento de la api está formado por:
 - Verbo HTTP
 - Dirección URI única
 - Información necesaria





Verbos HTTP

Son indicadores que identifican el objetivo del requerimiento, por ejemplo:

- **GET:** devolver información al cliente
- **POST:** agregar un registro en la base de datos
- **PUT / PATCH:** Editar un registro
- **DELETE:** eliminar un registro



Tecnologías para implementar una API Rest

- No interesa la tecnología del lado del servidor ni del lado del cliente.





Formato de intercambio de datos

Debe ser un formato flexible tales como:

- JSon
- XML
- Binario
- Texto plano



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.





Consideraciones para diseñar API Rest

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

```
DELETE /deleteProduct?id=1234
```

```
DELETE /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 cuatro tipo 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`

Controladores (verbos)

`/users/20/reset-password`

`/products/pencil`

- **Stores:** muestra la lista de favoritos del usuario 20
- **Controladores:** resetear el pass del usuario 20, normalmente se usan métodos POST.



Mensajes de respuesta

No devolver 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.



No hacer 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



Seguridad de la API

- Asegurar la API. Mediante autenticación y autorización en los endpoints, se puede utilizar:
 - Json Tokens (JWT)
 - API Key (estática)
 - Bearer Token
 - OAuth 1.0, etc



Versionar la API

- 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.



Utilizar https

- Garantiza que la información se encripta y viaja de manera segura
- Se pueden obtener los certificados desde [Let's script](#)



Referencias

- [Crear API REST con Spring](#)
- [Let's script](#)
- [Métodos de petición HTTP](#)