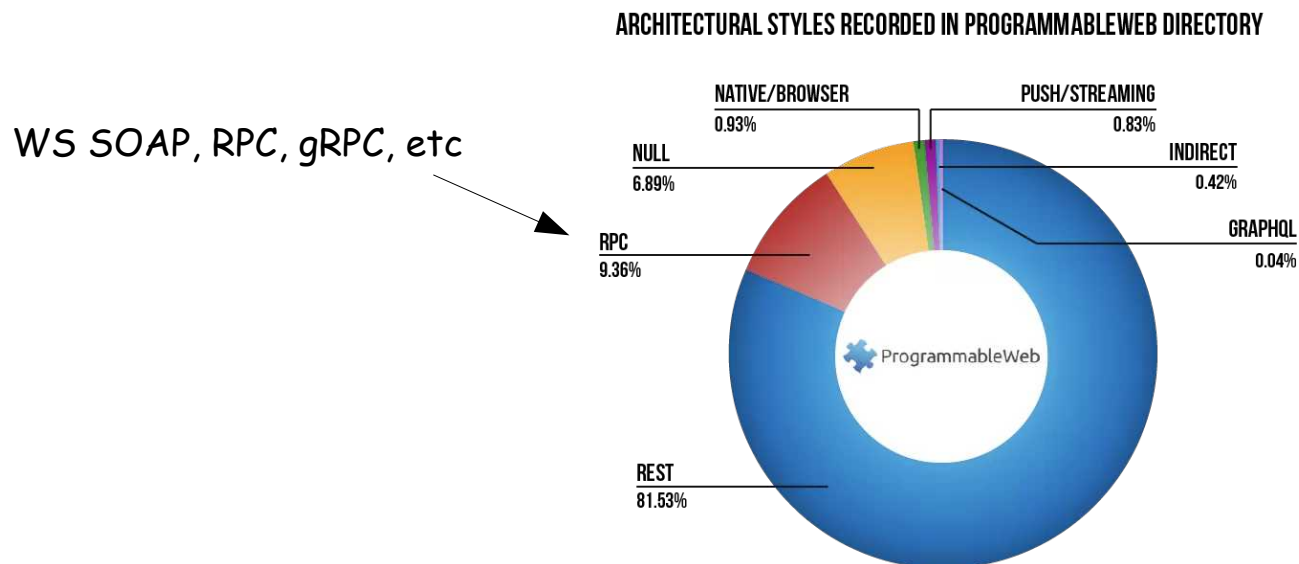


Servicios Web REST

- Servicios Web REST (o Restful)
- SOAP vs Rest
- Principios REST
- Acceso a recursos mediante URLs
- Operaciones basadas en métodos HTTP y códigos de respuesta
- Orientación a presentación
- HATEOAS

Servicios Web REST

- Los servicios Web REST están sustituyendo rápidamente a los servicios tradicionales basados en SOAP e incluso no pierden terreno frente a opciones más recientes y potentes como GraphQL



Tecnologías usadas en Servicios Web
(fuente: Programmable Web)

Servicios Web Rest: pros y contras

- Su éxito se debe a:
 - Aparente sencillez (especialmente comparado con WS basados en SOAP)
 - Buena integración en el protocolo HTTP
- También tiene problemas y es criticado por:
 - Diseñar un API restful correctamente puede no ser sencillo (implica un cambio de filosofía)
 - No se adapta bien a esquemas más allá de un CRUD
 - El propio protocolo HTTP sobre el que se basa impone muchas limitaciones

SOAP vs REST

•SOAP

- Es un **protocolo de mensajería**
- WSDL describe perfectamente la interfaz del servicio. Es posible implementar un servicio desde cero contando únicamente con esta descripción
- Mejor adaptado a servicios Web complejos que requieran seguridad extra, transacciones, operaciones asíncronas, etc.
- Basado en la generación y uso de proxies para ocultar los detalles de la comunicación
- Orientado a la ejecución de **operaciones**

•REST

- Es un **patrón de arquitectura** (no define un protocolo ni formato de mensaje concreto)
- Más integrado con el protocolo http y los servidores Web tradicionales
- Más ligero y mejor adaptado a clientes con capacidades limitadas como los móviles y adaptados o para el acceso desde una página web mediante JavaScript
- Orientado al acceso a **recursos**

Principios REST

- Codificar nuestro servicio como una serie de recursos (asegurado, hoja clinica, etc.) accesibles mediante **URLs**. **La URL nunca debe incluir referencias a operaciones a realizar sobre el recurso**
- Definir la operación a realizar en el recurso (buscar, crear, actualizar, borrar, etc.) mediante los **métodos estándar HTTP**: PUT, GET, POST, DELETE
- Usar los **códigos de respuesta estándar HTTP** para indicar el resultado de la operación (200 OK, 201 Created, 202 Accepted, etc.)
- Implementar un **servidor stateless** (sin estado), que no guarde el estado de la conversación con el cliente (i.e., sesiones). Alternativamente, implementar el estado de la conversación con el cliente mediante hipertexto (principio **HATEOAS**)

Acceso a recursos mediante URLs

- Usaremos las URLs para codificar el acceso a la información, de manera similar al acceso al resto de recursos de Internet
- Aplicaremos la filosofía REST correctamente cuando “parezca” que los elementos a los que accedemos existen “estáticamente” en el servidor
- Un URL debe identificar unívocamente a un recurso en el servidor

```
POST http://localhost:8080/servicios/clinica HTTP/1.1

<?xml version="1.0"?>
<soap:Envelope xmlns:soap="http://www.w3.org/2001/12/soap-envelope">
  <soap:Header/>
  <soap:Body>
    <buscarAseguradoRequest>
      <numSeg>17</numSeg>
    </buscarAseguradoRequest>
  </soap:Body>
</soap:Envelope>
```

Peticion SOAP
(orientado
a ejecución de
operaciones)

```
GET http://localhost:8080/clinica/asegurados/17 HTTP/1.1
```

Petición REST (orientado a recursos)

Operaciones basadas en métodos HTTP

- En REST las operaciones sobre los recursos se uniformizan utilizando los métodos del protocolo HTTP:
 - GET → Operación de sólo lectura que no modifica el estado interno del servidor
 - PUT → Actualizar un recurso ya existente en el servidor
 - DELETE → Eliminar recurso en el servidor. Modifica el estado interno del servidor
 - POST → Crea un recurso nuevo en el servidor

Operaciones basadas en métodos HTTP

- Ventajas:

- Uniformidad de acceso
- Familiaridad (filosofía de construcción similar al de las aplicaciones web)
- Interoperabilidad → Todos los lenguajes y plataforma disponen de bibliotecas para enviar/recibir peticiones HTTP por distintos métodos
- Eficiencia y escalabilidad → p. ej. los métodos basados en GET pueden ser cacheados

Operaciones basadas en métodos HTTP

Obtener hojas abiertas

```
GET http://localhost:8080/clinica/hojas?abiertas=true
```

Obtener las hojas del asegurado 3

```
GET http://localhost:8080/clinica/asegurados/3/hojas
```

Crear el asegurado con código 7

```
POST http://localhost:8080/clinica/asegurados/7 HTTP/1.1
Content-Type: application/json

{"nombre": "Miguel Álvarez"}
```

Crear una hoja clínica sin indicar código

```
POST http://localhost:8080/clinica/hojas HTTP/1.1
Content-Type: application/json

{"asegurado": http://localhost:8080/clinica/asegurados/11",
 "sintomas": "Fiebre alta",
 "diagnostico": "Gripe aguda",
 "tratamiento": "Reposo"}
```

Borrar la hoja 5

```
DELETE http://localhost:8080/clinica/hojas/5
```

Códigos de respuesta

• Usar los códigos estándar de respuesta HTTP para notificar el resultado de la operación (Ojo, no hay un consenso claro de cual usar en cada caso)

- 200 → OK
- 201 → Created
- 202 → Accepted
- 400 → Bad request
- 401 → Unauthorized
- 403 → Forbidden
- 404 → Not Found
- 406 → Not Acceptable
- 409 → Conflict
- 500 → Internal Server Error
- 501 → Not implemented

Obtención de la hoja 28


```
GET http://localhost:8080/clinica/hojas/28
```



Respuesta

```
HTTP/1.1 404 NOT FOUND
```

Recomendaciones API RESTful

Method	Resource Path or URI	I	S	Description	Query String	Returns	RCs 
GET	{resource}	Y	Y	Get an array of resources of the resource type <i>(recommend a default limit to the number of rows returned that can be overridden with the limit parameter)</i>	Optional: offset = 0 - n limit = 0 - n columns=[CSV] where = [key]=[CSV] orderby=[CSV]	Array of JSON Objects or, when the columns parameter is present, an array of JSON maps	200 204
GET	{resource}/{ID}	Y	Y	Get a specific resource matching the ID	Optional: columns=[CSV]	Array of JSON Objects or, when the columns parameter is present, an array of JSON maps	200 204
POST	{resource}	N	N	Create a new resource of the type	JSON Object	Location URI of new resource in HTTP header	201
POST	{resource}/{id}	*	*	Update a specific resource (use PUT instead)			405 Not Allowed
PUT	{resource}/{ID}	Y	N	Replace specific resource with a new copy	JSON Object	Location URI of updated resource in HTTP header	204
PATCH	{resource}/{ID}	Y	N	Update one or more attributes of a resource	JSON Object containing modified or deleted (null) attributes		200
DELETE	{resource}/{ID}	Y	N	Remove a specific resource			204
DELETE	{resource}	Y	N	Remove selected resources of a type (must be qualified with where)	Required: where = [key]=[CSV]		204 (405 if no where)

Errores comunes

- ⚡ Meter verbos o indicaciones de la operación a realizar en la propia URL

```
GET http://localhost:8080/clinica/buscarHojaClinica?num=7
```

- ⚡ Usar el método inapropiado, generando confusión (p. ej., usar GET para registrar un asegurado)

```
GET http://localhost:8080/clinica/asegurados  
{ "numSeg": 11800900, "nombre": "Juan España España" }
```

- ⚡ Usar parámetros de la URL para indicar valores de atributos durante una creación o actualización

```
PUT http://localhost:8080/clinica/asegurados/11800900?numSeg=11800900&nombre="Juan  
Andalucía Andalucía"
```

- ⚡ Tener un juego de URLs sobre un recurso no consistentes

```
GET http://localhost:8080/clinica/asegurado/11800900  
  
POST http://localhost:8080/clinica/listaAsegurados  
  
PUT http://localhost:8080/clinica/asegurados/11800900  
  
DELETE http://localhost:8080/clinica/asegurado/borrar/11800900
```

- ⚡ Realizar consultas con resultados potencialmente largos, sin ningún tipo de filtrado

```
GET http://localhost:8080/clinica/asegurado
```

HATEOAS

- HATEOAS (Hiptertext As The Engine Of Application State) es un principio de Rest que recomienda usar URLs en las respuestas para conducir la conversación con el cliente
- Si una respuesta referencia a otro recurso, incluir la propia URL del recurso en lugar de códigos o identificadores

```
HTTP/1.1 200 OK
Content-Type: application/json

{ "num": "542",
  "fechaApertura": "10/12/2013 10:20",
  "asegurado": "http://localhost:8080/clinica/asegurados/11",
  "sintomas": "Fiebre alta",
  "diagnostico": "Gripe aguda",
  "tratamiento": "Reposo",
  "ingresado": "false"
}
```



HATEOAS

Consultar las hojas del asegurado

```
GET http://localhost:8080/clinica/asegurados/7/hojas
```



```
HTTP/1.1 200 OK
```

```
Content-Type: application/json
```

```
[  
  "http://localhost:8080/clinica/hojas/112",  
  "http://localhost:8080/clinica/hojas/1283",  
  "http://localhost:8080/clinica/hojas/4300"  
]
```

HATEOAS

- Implementa el estado de la conversación con el cliente mediante URLs (p. ej., paginación en una consulta)

Consultar todas las hojas de la clínica (¡¡ muchas !!)

```
GET http://localhost:8080/clinica/hojas
```



Listado por páginas

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "listado": [
    "http://localhost:8080/clinica/hojas/1",
    "http://localhost:8080/clinica/hojas/2",
    "http://localhost:8080/clinica/hojas/3",
    "http://localhost:8080/clinica/hojas/4",
    "http://localhost:8080/clinica/hojas/5"
  ],
  "anteriores": "http://localhost:8080/clinica/hojas?desde=1&hasta=5",
  "siguientes": "http://localhost:8080/clinica/hojas?desde=6&num=11"
}
```