RESTful API

FI.UBA

Ingeniería de Software

¿ Qué NO es REST ?

- NO es un standard
- NO es un protocolo
- NO es un reemplazo de SOAP
- NO es una biblioteca

¿ Qué es REST ?

- Surge de la tesis doctoral de Roy Fielding en el año 2000
- Significa **Re**presentational **S**tate **T**ransfer
- Utiliza estándares existentes como HTTP
- Comunicación cliente - servidor
- Se presenta en 3 niveles de madurez

Características

- Arquitectura cliente - servidor
- Stateless
- Cacheable
- Expone recursos (URIs)
- Usa explícitamente los verbos HTTP
- Navegable

Stateless

- Cada request se ejecuta de forma independiente del resto
- Cada request contiene toda la información necesaria para completarse
- La API no mantiene ningún tipo de sesión
- Se promueve el uso de tokens para manejo de seguridad

Cacheable

- Reduce ancho de banda usado
- Reduce latencia
- Reduce carga en servidores
- Oculta fallos de red

Cacheable

- Lo que se define como “cacheabilidad” en los sistemas REST es la capacidad de estos sistemas para *etiquetar* de alguna forma las respuestas para que otros mecanismos intermedios funcionen como un caché.
- Estos sistemas o mecanismos intermedios (existen entre el cliente y el servidor) deben ser por lo general transparentes para los desarrolladores, no deben afectar la manera en que los servicios se consumen.

Cacheable

Expires

```
Expires: Fri, 19 Nov 2021 19:20:49 EST
```

Cache-Control

```
Cache-Control: max-age=3600
```

Last-Modified

```
Last-Modified: Fri, 19 May 2021 09:17:49 EST
```

Compresión

Las APIs suelen retornar representaciones en varios formatos, entre ellos formato plano, XML, HTML, JSON y estos formatos pueden ser comprimidos para ahorrar ancho de banda sobre la red.

Accept-Encoding

Ejemplo de cómo el cliente informa que mecanismos soporta:

```
Accept-Encoding:     gzip,compress
```

Content-Encoding

Ejemplo de cómo el server informa que mecanismo usó para encripción:

```
Content-Type:     text/html
Content-Encoding:     gzip
```

```
Ejempo Node:
```

```
const compression = require('compression');
app.use(compression()); // activa gzip si el cliente lo soporta
```

URIs

- *Uniform Resource Identifier*
- Identificación unívoca de recursos con cadenas de caracteres
- Identifica los recursos por clase o tipo
- Uso de sustantivos en plural por convención. **No verbos**
- Distinción de recursos principales y subordinados

URIs - Ejemplos

Recurso: clientes

- ***/clientes*** representa todos los clientes
- ***/clientes/1*** representa al cliente con id 1
- ***/clientes?nombre=juan*** representa a los clientes con nombre juan
- ***/clientes/1/compras*** representa a las compras del cliente 1

y si las compras son recursos primarios...?

URIs - Ejemplos

Recurso: compras

- ***/compras***
- ***/compras?cliente=1***

Verbos HTTP - Requests

- **GET**: solicita una representación de un recurso específico
- **POST**: se utiliza para enviar una entidad a un recurso en específico
- **DELETE**: borra un recurso en específico
- **PUT**: reemplaza todas las representaciones actuales del recurso de destino con la carga útil de la petición
- **PATCH**: aplica modificaciones parciales a un recurso (a diferencia de PUT)
- **OPTIONS**: es utilizado para describir las opciones de comunicación para el recurso de destino

HTTP Status Codes - Responses (I)

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

HTTP Status Codes - Responses (III)

200 - **OK**

201 - **Created (con el location en el header)**

400 - **Bad Request**

401 - **Authorization Required**

404 - **Not Found**

405 - **Method Not Allowed**

408 - **Request Time-Out**

409 - **Conflict**

422 - **Unprocessable Entity**

500 - **Internal Server Error**

502 - **Bad Gateway**

504 - **Gateway Time-Out**

REST Security Design Principles

Least Privilege: Tener el menor privilegio requerido para hacer las acciones.
Fail-Safe Defaults: Por defecto no tener acceso a los recursos
Complete Mediation: El sistema debe validar los permisos de acceso a todos los recursos
Keep it Simple
Https
Password Hashes: (PBKDF2, bcrypt, y scrypt)
Never expose information on URLs: Usernames, passwords, session tokens, y API keys deberían no aparecer en la URL para evitar ser logueadas en los logs de web server logs
Considerar agregar Timestamp en los requests.
Validación de los parámetros de entrada

Monitorear transacciones sospechosas.

Cantidad de requests por IP o por token/JWT/user para evitar problemas de denegación de servicio, o simplemente controlar o reducir el uso excesivo que puede bajar la performance de la API en general.

Limitación de velocidad, o tiempos de demora agregados entre request y request para ciertos casos, ayuda a reducir las solicitudes excesivas que ralentizarían la API, ayuda a lidiar con llamadas / ejecuciones accidentales y monitorea e identifica de manera proactiva una posible actividad maliciosa.

APIs pagas como las de google por ejemplo permiten configurar límites de uso, tarifa, para evitar sorpresas ante un mal uso o bug que genere por error multiples llamadas a la API.

Autenticación & Autorización

- Basic auth
- Api Keys
- Bearer Authentication
- OAuth
- JWT

Autenticación & Autorización
Basic Auth

- Base64 encoding
  - user: **fiuba**
  - pass: **k@X4R$KFEbCn**
  - plain-auth: **fiuba:k@X4R$KFEbCn**
  - Authorization: **Zml1YmE6a0BYNFIkS0ZFYkNu**

Base64 es fácilmente decodificable, Basic authentication solo debería usarse en conjunto con otro mecanismo de seguridad como HTTPS/SSL.

API Keys

Algunas APIs usan API keys para autorización.
Una API keyes un token que el cliente provee cuando hace la llamada
Via queryString:

```
GET /something?api_key=abcdef12345
```

Como header:

```
GET /something HTTP/1.1
X-API-Key: abcdef12345
```

Como cookie:

```
GET /something HTTP/1.1
Cookie: X-API-KEY=abcdef12345
```

API keys se supone que es secreta y que solo el cliente y servidor la conocen.
Sólo debería usarse en conjunto con otro mecanismo de seguridad como HTTPS/SSL.

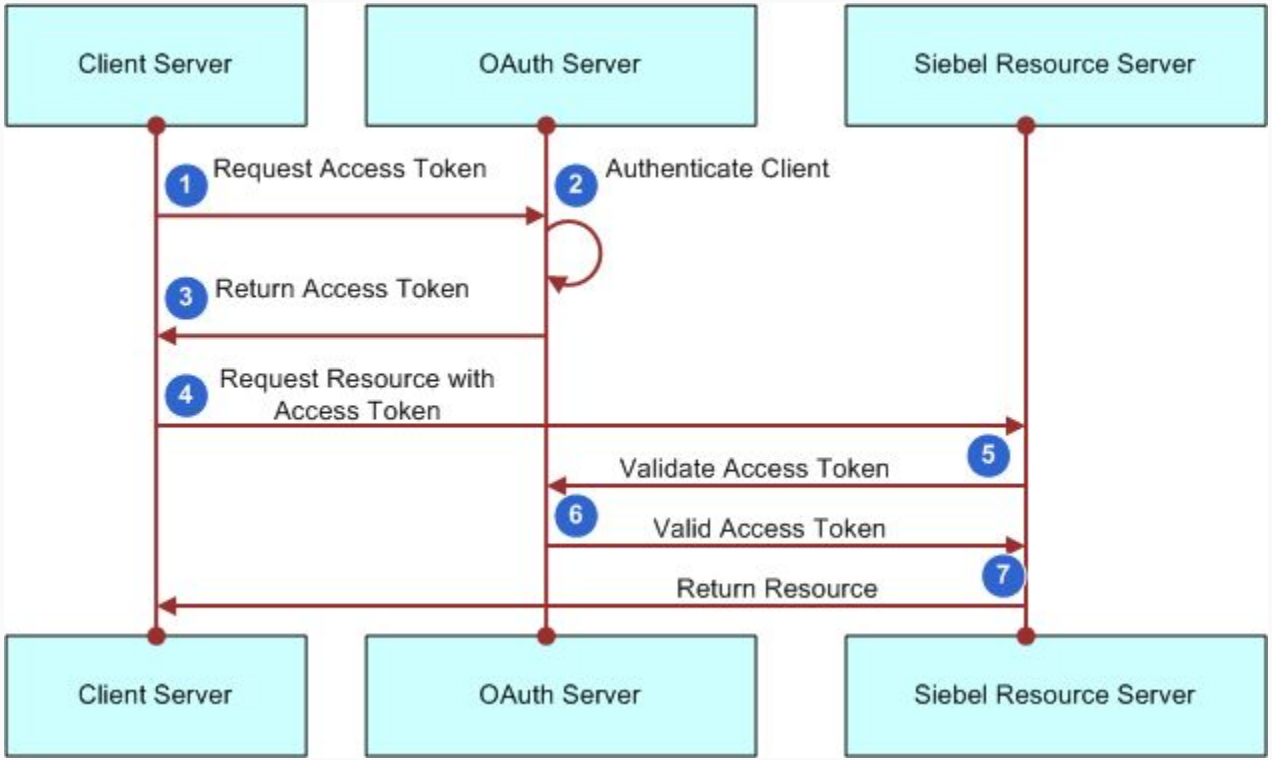
Bearer Authentication / Token Auth

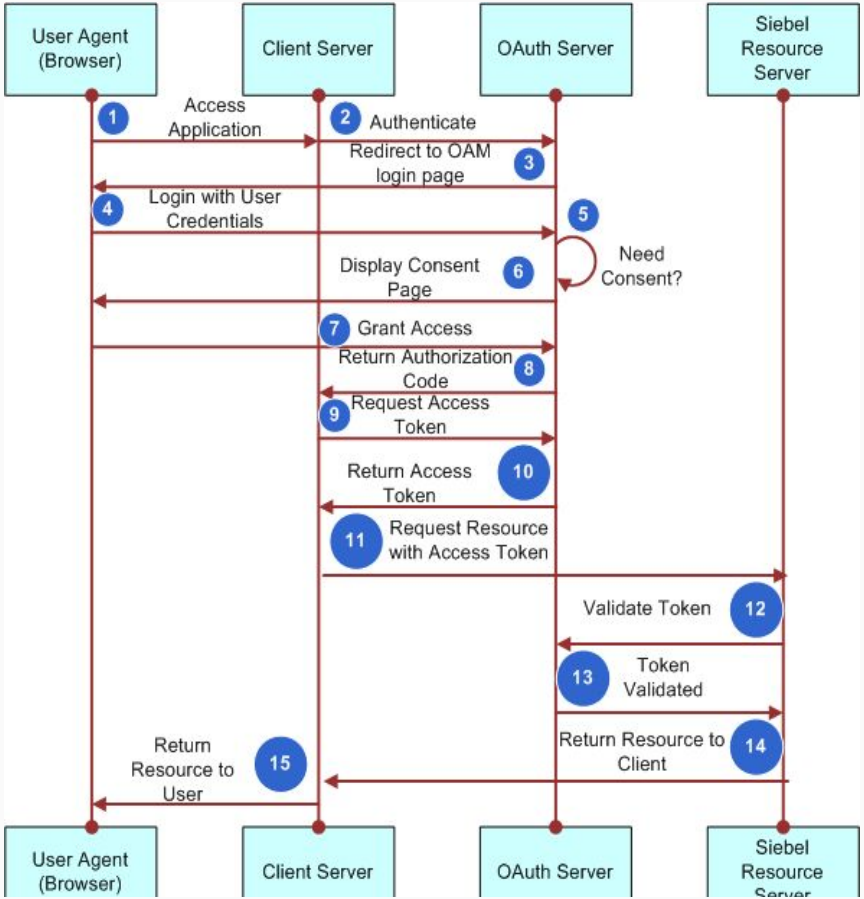
Utiliza tokens de seguridad llamados Bearer (da acceso al portador del token)
Se envía en un header de Authorization

```
Authorization: Bearer <token>
```

OAuth

- Es un protocolo de autenticación
- Consiste en delegar la autenticación de usuario al servicio que gestiona las cuentas, de modo que sea éste quien otorgue el acceso para las aplicaciones de terceros.
- OAuth 2 provee un flujo de autorización para aplicaciones web, aplicaciones móviles e incluso programas de escritorio.


Client Server
OAuth Server
Siebel Resource Server
1
Request Access Token
2
Authenticate Client
3
Return Access Token
4
Request Resource with Access Token
5
Validate Access Token
6
Valid Access Token
7
Return Resource
Client Server
OAuth Server
Siebel Resource Server


User Agent (Browser)
Client Server
OAuth Server
Siebel Resource Server
1
Access Application
2
Authenticate
Redirect to OAM login page
3
4
Login with User Credentials
5
Need Consent?
Display Consent Page
6
7
Grant Access
Return Authorization Code
8
9
Request Access Token
Return Access Token
10
11
Request Resource with Access Token
Validate Token
12
13
Token Validated
Return Resource to Client
14
Return Resource to User
15
User Agent (Browser)
Client Server
OAuth Server
Siebel Resource Server

Autenticación & Autorización
JWT

- El token se genera en el primer paso también
  - user: **fiuba** / pass: **k@X4R$KFEbCn**
  - POST -> /token { “user”: “**fiuba**”, “pass”: “**k@X4R$KFEbCn**” }
  - Response:

    eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1aWQiOjEyMzQ1Njc4OTAsIm5hbWUiOiJKb2huIERvZSIsInByb2YiOiJPV05FUiIsIm9yZyI6ODM0NzUyM30.2PWhCiy6sgDeBhGFbC1Ws1wIoGgy7eY-44uey_aR0eo
- Las credenciales del usuario viajan sólo 1 vez
- El token **no** se almacena del lado del servidor para validar
- El uso de JWT incrementa la eficiencia en las aplicaciones evitando hacer multiples llamadas a la base de datos.

Autenticación & Autorización
JWT (2)

- user: fiuba / pass: k@X4R$KFEbCn
- Authorization:
  eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1aWQiOjEyMzQ1Njc4OTAsIm5hbWUi
  OiJKb2huIERvZSIsInByb2YiOiJPV05FUiIsIm9yZyI6ODM0NzUyM30.2PWhCiy6sgD
  eBhGFbC1Ws1wIoGgy7eY-44uey_aR0eo
- Authorization - Desencriptado: https://jwt.io/#debugger-io

HEADER:ALGORITHM & TOKEN TYPE	PAYLOAD:DATA	VERIFY SIGNATURE

{
"alg": "HS256",
"typ": "JWT"
}
	{
	"uid": 1234567890,
	"name": "John Doe",
	"prof": "OWNER",
	"org" : 8347523
	}
		HMACSHA256 (
		base64UrlEncode(header) + "." +
		base64UrlEncode(payload),
		hqo245thgSD$KFJtgu#%ebvjqf?2f3idJjdivI
		)

Autenticación & Autorización
JWT (3)

Los JWT pueden ser mensajes solo firmados, solo encriptados, o ambos
Si un token es solo firmado pero no encriptado, cualquiera puede leer su contenido, pero si no se conoce la clave privada no puede modificado. Ya que al validar la firma no coincidiría.

Autenticación & Autorización
JWT (4)

El server valida la firma del JWT para saber que lo que él le envió al cliente no se modificó, y utiliza la información del mismo.
De esta forma se sigue siendo stateless, y generalmente se hace más eficiente la validación del usuario, sin tener que acceder a un medio persistente a validar si el token es válido y a quién pertenece el mismo.

Refresh Tokens

Los access token deberían tener un tiempo limitado de vida, por eso aparecen los refresh token.
Este es otro token que sirve para un solo uso y es utilizado para obtener un nuevo access token.
Es una credencial que permite obtener nuevos tokens sin necesidad de usar las credenciales de usuario y password nuevamente.

Versionado

Rest no provee un mecanismo definido para versionado pero se suelen ver estas estrategias:

- Usando la URI:

```
https://api.fi.uba.ar/v1
```

```
https://apiv1.fi.uba.ar
```

```
https://api.fi.uba.ar/20211101/
```

- Usando un Custom Header:

```
Accept-version: v1
```

```
Accept-version: v2
```

- Usando el Header Accept:

```
Accept: application/vnd.example.v1+json
```

```
Accept: application/vnd.example+json;version=1.0
```

Hateoas


{
pageNumber: 1,
totalItemCount: 25,
pageItemCount: 10,
_links: {
next: {
href: http://localhost:8080/cursos?pagina=2,
type: "application/vnd.cloud.programar.hateoas.Page"
},
previous: {
href: http://localhost:8080/cursos?pagina=0,
type: "application/vnd.cloud.programar.hateoas.Page"
},
self: {
href: http://localhost:8080/cursos?pagina=1,
type: "application/vnd.cloud.programar.hateoas.Page"
}
},
_embedded: {
items: [
{
codigo: "cod-10",
titulo: "Curso número 10",
unidadesDidacticasCompletadas: 2000,
_links: {
self: {
href: http://localhost:8080/cursos/cod-10,
type:

Respuestas

- Mantener lo más estandarizadas a las mismas.
- Reducir el tamaño de la respuesta a lo necesario
- Utilizar Código de Errores HTTP

Ejemplo uso más común:

HTTP CODE: 401 {

```
    "success": false,  // solo informativo, el error se define por el HTTP CODE
    "message": "Invalid email or password",
    "error_code": 1308,
    "data": {}
}
```

HTTP CODE: 200 {

```
    "success": true,
    "message": "User logged in successfully",  // optional in success responses
    "data": { }
}
```

Donde data es un objeto que contiene un mapa de otros objetos

```
{
    "success": true,
    "message": "User found",
    "data": {
        "user": {
            "id": 2,
            "name": "Juan",
            "email": " juan@fi.uba.ar ",
            "city": {
                "id": 3,
                "name": "Buenos Aires",
                "country": {
                    "id": 2,
                    "name": "Argentina",
                    "code_country": "AR",
                    "avatar": " //localhost:3000/api/v1/country_AR.png ",
                }
            }
        },
        "role": "client",
        "favorites": ["blue", "red", "white"]
    }
}
```

Paginado, Filtros y Ordenamientos?

HTTP CODE: 200

```
{
   "success": true,
   "metadata" : {
      "page": 5,
      "per_page": 20,
      "page_count": 20,
      "total_count": 521,
      }
   "data" : {....}
}
```

usando headers

HTTP/1.1 200
Pagination-Count: 100
Pagination-Page: 5
Pagination-Limit: 20
Content-Type: application/json

Filtros y ordenamiento suele usarse como parámetros del querystring o del body

Que pasa con los formatos, booleanos, fechas? Con la notación: camelCase o snake_case?

Logging, Health, Metrics

Tener logs, metricas y puntos de controls ayudan a detectar los problemas antes de que realmente lleguen. Se suelen agregar endpoints para verificar o monitorear que la api esta viva, y obtener datos de uso de memoria, etc.

- `/health`
- `/metrics`

OWASP API Security Top 10 (2023)

https://owasp.org/www-project-api-security/

API1: Broken Object Level Authorization

- Errores al controlar el acceso a recursos individuales.
- Ejemplo: `/users/1234` accedido por otro usuario que no es dueño.
- Prevención: controlar siempre la autorización a nivel de recurso.

API2: Broken Authentication

- Autenticación débil o mal implementada.
- Buenas prácticas: uso de JWT, OAuth2, expiración de tokens, etc.

API3: Broken Object Property Level Authorization

- Ej: se permite modificar campos que no deberían (como `isAdmin`).
- Prevención: control de permisos a nivel de atributo/campo.

OWASP API Security Top 10 (2023)

API4: Unrestricted Resource Consumption

- APIs que permiten abusos tipo DoS: muchas requests, archivos pesados, etc.
- Prevención: paginación, rate-limiting, límites de tamaño.

API6: Unrestricted Access to Sensitive Business Flows

- Automatización o abuso de funcionalidades críticas (reservas, compras, etc).
- Prevención: CAPTCHA, rate limits, monitoreo.

API8: Security Misconfiguration

- Exposición de headers innecesarios, errores con info sensible, CORS mal configurado, etc.

API9: Improper Inventory Management

- Exposición de versiones antiguas o endpoints no documentados.
- Prevención: control de versiones y documentación actualizada.

Recursos

- **Architectural Styles and the Design of Network-based Software Architectures** - https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm
- **Wikipedia** - https://es.wikipedia.org/wiki/Roy_Fielding
- **Roy Fielding personal site**- http://roy.gbiv.com/
- JWT: https://jwt.io/introduction/ https://jwt.io/#debugger-io
- https://martinfowler.com/articles/richardsonMaturityModel.html
- https://hackernoon.com/restful-api-design-step-by-step-guide-2f2c9f9fcdbf
- https://www.postman.com/
- OWASP https://owasp.org/www-project-api-security/

Recursos

- JSON API
- OPENAPI
- Swagger.io
- JSON Schema
- HATEOAS
- JSON
- GraphQL
- OWASP