



PONTIFICIA
UNIVERSIDAD
CATÓLICA
DE CHILE

IIC3103

Taller de Integración

Profesores

Arturo Tagle / Daniel Darritchon



Resumen clase anterior

API's

Una API es una interfaz que entrega un software para ser extendido (definición genérica).

Cuando hablemos de servicios o servicios web: Colección de servicios que permiten interactuar con un sistema/aplicación (por ejemplo, Spotify API).



Integración por servicios

Módulo 2: Servicios - Parte 2

**Clasificación de APIs y
Autenticación / Autorización**

Richardson maturity model

Clasificando a las API's

REST

SOAP

Glory of REST

Level 3: Hypermedia Controls

Level 2: HTTP Verbs

Level 1: Resources

Level 0: The Swamp of POX



Características de los niveles de madurez



1. **Nivel 0 – URL única:** Todo se realiza en una única URL, identificando el “proceso” a ejecutar dentro del mensaje.
2. **Nivel 1 – recursos:** Maneja la complejidad “dividiendo para conquistar”, se crean múltiples recursos para diferenciar objetos.
3. **Nivel 2 – verbos:** Introduce el uso de “verbos” para manejar situaciones similares de la misma forma.
4. **Nivel 3 – Hypermedia:** Introduce el descubrimiento, proveyendo un protocolo autodocumentado (HATEOAS).

Ejemplos API SOAP



REST

Representational State Transfer

REST

Estilo de arquitectura y set de restricciones para servicios web.

Fielding, R. (2000)

Es decir, REST no nos impone un estándar formal ni permite saber si un servicio es Restful o no. Es, más bien, un conjunto de restricciones con objetivos específicos.



Principios clave de diseño de servicios REST

- Deben ser simples e intuitivos
- Deben ser eficientes y simples de mantener
- Deben usar los estándares web (cuando hagan sentido)
- Deben ser amigables con el desarrollador (¡servicios claros y bien documentados!)



Los servicios se basan en objetos o recursos

- Los servicios REST son el símil al CRUD (create-read-update-delete) de las bases de datos
- Los servicios REST exponen los recursos, objetos o entidades de manera que sean explorables y trabajables por quién los utiliza.
- Los recursos u objetos pueden tener sub-recursos y relaciones entre ellos

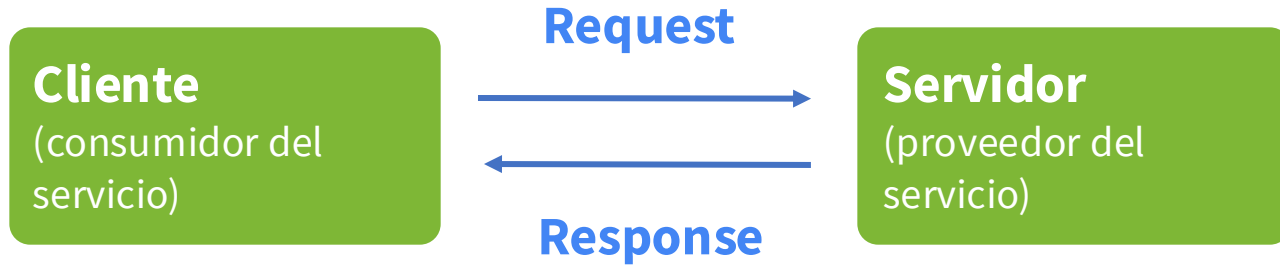




Características de los servicios REST

1. Arquitectura cliente servidor
2. Sistema por capas
3. Caché
4. Servidor stateless
5. Interfaz uniforme
 - a. Identificación de los recursos
 - b. Manipulación de recursos
 - c. Mensajes autodescriptivos

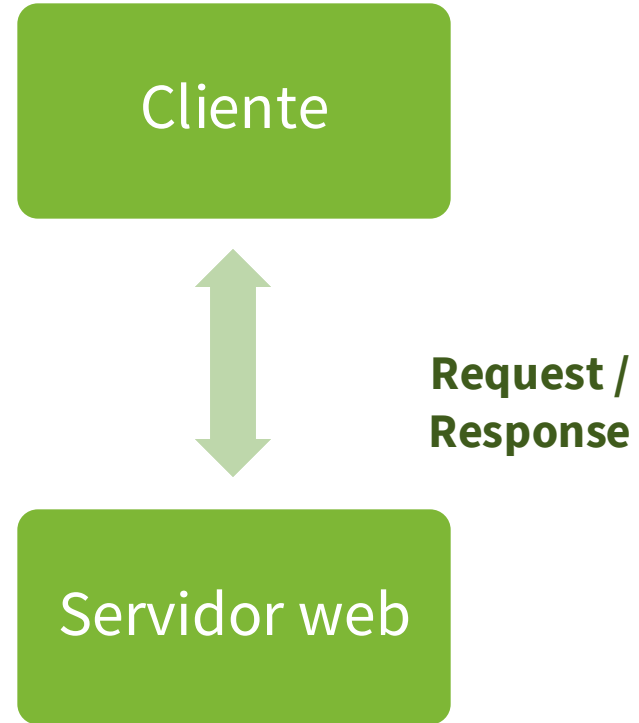
Arquitectura cliente - servidor



Sistema por capas

Existe una interfaz uniforme de acceso al servicio, sin importar el funcionamiento interno del servidor.

La arquitectura interna es una “caja negra” para el consumidor del servicio.

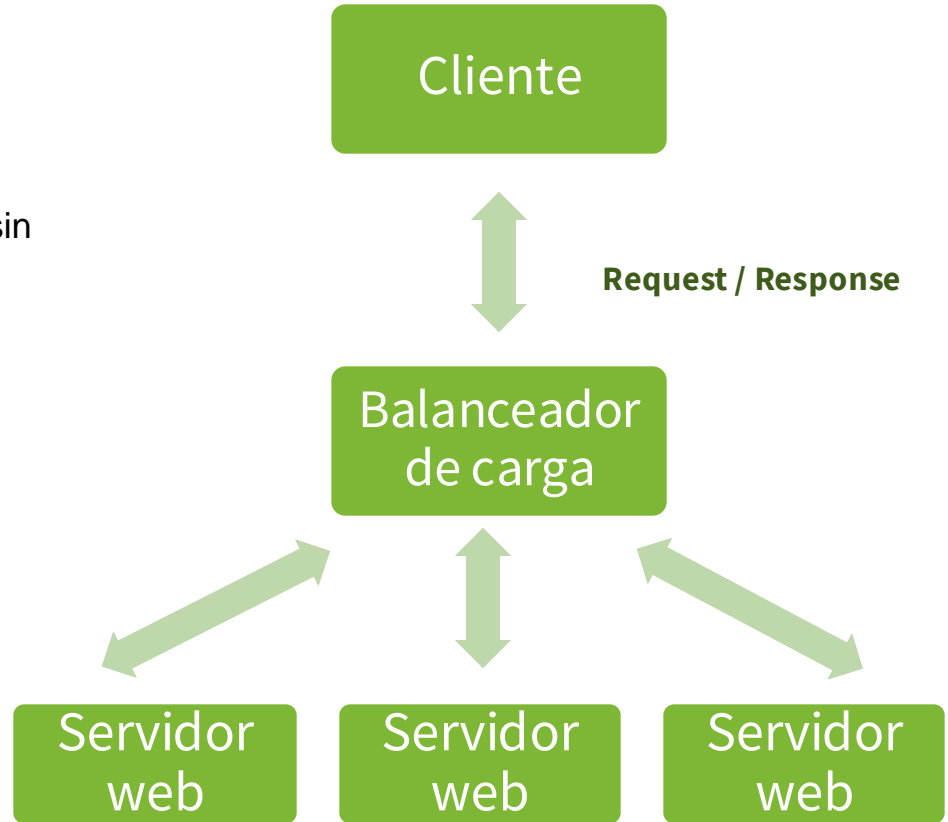


Cliente y servidor web, sin capas intermedias. Comunicación directa.

Sistema por capas

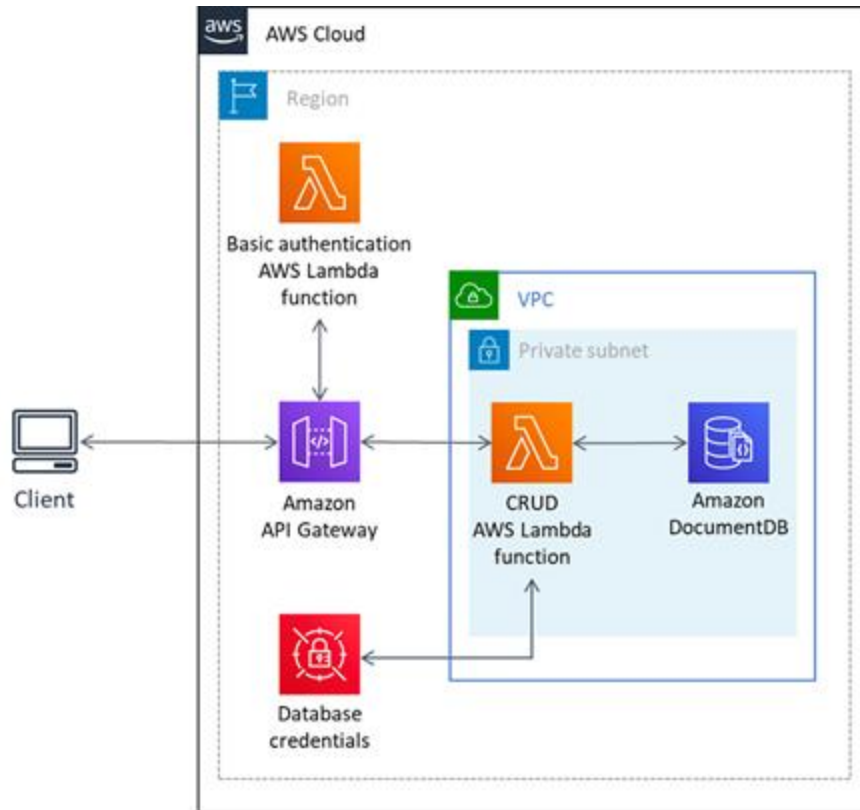
Existe una interfaz uniforme de acceso al servicio, sin importar el funcionamiento interno del servidor.

La arquitectura interna es una “caja negra” para el consumidor del servicio.

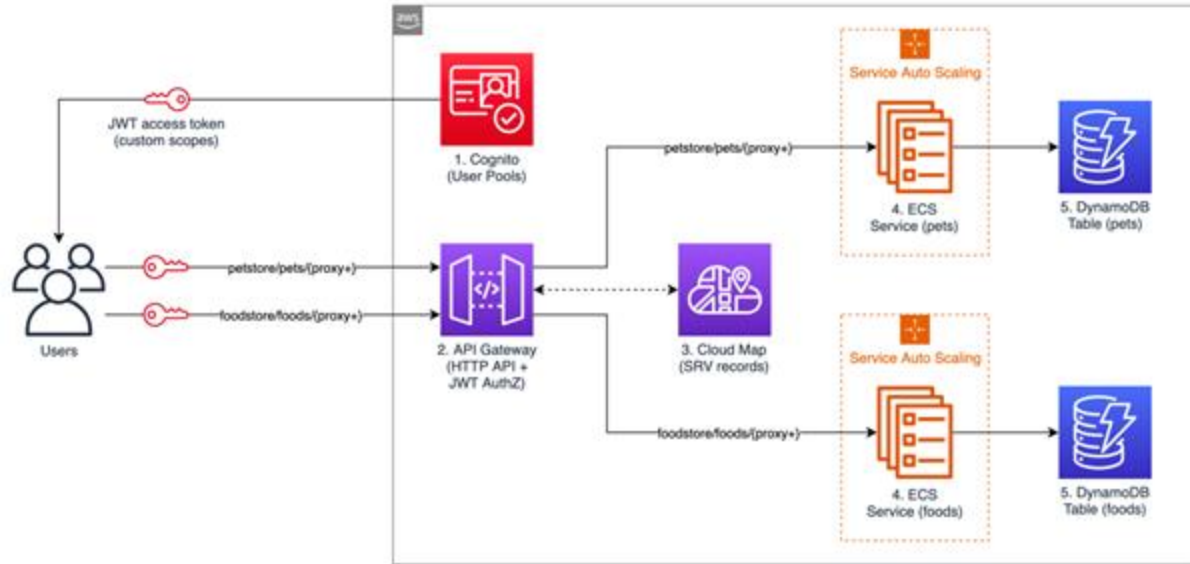


Servicio con balanceador de carga y múltiples servidores capaces de procesar el request.

Sistema por capas



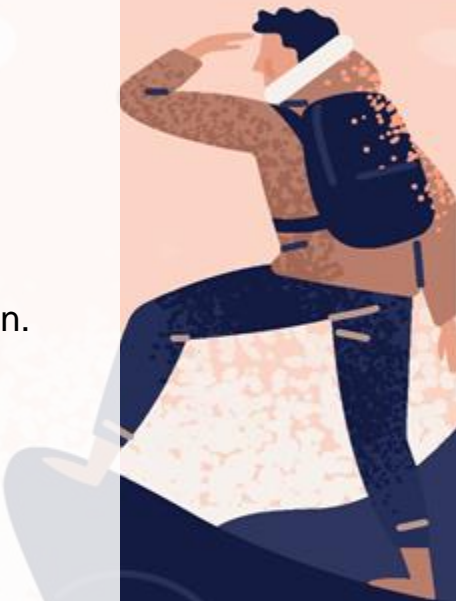
Sistema por capas



Caché

El cliente tiene permitido reutilizar la información.

Esto dependerá del tipo de servicio y lo que especifique la respuesta.



¿Por qué usar caché?

- Reducir ancho de banda necesario
 - Reducir latencia
 - Reducir carga en servidores
 - Mayor tolerancia a fallas de red
-
- Los mensajes pueden ser marcados explícitamente como “cacheables” o “no cacheable”
 - Además, se puede setear la duración del caché.
 - Se evita duplicar llamadas ya que el cliente puede usar respuestas anteriores que siguen siendo válidas
 - Recursos marcados como públicos pueden ser cacheados por proxys u otros agentes intermedios, mientras que los privados sólo por el cliente final

Servidor stateless

El servidor no guarda información de la sesión del cliente.

Todos los requests deben contener toda la información necesaria para autenticar y ejecutar la acción.

Cada request debe ser “stand-alone”, y no debe verse afectado por una secuencia de llamadas anteriores del mismo cliente.

No confundir servidor stateless con recurso stateless:

- Estado del servidor (o aplicación) es el estado o interacciones previas que ha realizado un determinado cliente.
- Estado de un recurso (u objeto) es el estado actual de un recurso en el tiempo, y no tiene relación con la interacción de un cliente con el servidor.

Ventajas

Escalabilidad: Cualquier servidor puede procesar cualquier request.

Menor complejidad: Se remueve toda la lógica de sincronización por el lado del servidor.

Cache más simple: No hay incertezas respecto de un estado anterior para determinar el cache.

Aplicación “no se pierde”: el estado actual lo determina el cliente. Este envía toda la información necesaria para ejecutar la acción.

An illustration of a business meeting. A man with a beard and glasses, wearing a blue sweater, stands on the right, pointing at a flipchart. The flipchart contains a flowchart, a bar chart, and a pie chart. Two people, a man in a suit and a woman in a green vest, are seated at a table on the left, working on laptops. A lightbulb icon is above the man's head, and a computer monitor icon is above the woman's head. The background is a stylized blue and grey abstract shape.

Ejemplo

Interfaz Uniforme

Recursos u objetos son fácilmente identificables y son manipulados usando (a lo menos) 4 operaciones estándar. Los mensajes deben ser auto-descriptivos.

Revisar artículo HTTP Methods de [restfulapi.net](https://restfulapi.net/http-methods/):

<https://restfulapi.net/http-methods/>



Identificación

- `https://url/nombre/id?parametros=1`

Operaciones

- GET: Obtener un objeto.
- POST: Crear un objeto u otras acciones.
- PUT: Crear o actualizar un objeto completo.
- DELETE: Borrar un objeto.
- PATCH: Actualizar un objeto parcialmente.
- HEAD: Muestra información del objeto.
- OPTIONS: Muestra info del servicio.

Códigos de respuesta estándar

(todos los códigos [aquí](#))

- 200 OK
- 404 Not Found
- 500 Internal Server Error

An illustration of a business meeting. A man with a beard and glasses, wearing a blue sweater, stands on the right, pointing at a flipchart. The flipchart contains a flowchart, a bar chart, and a pie chart. Two people, a man in a suit and a woman in a green vest, are seated at a table on the left, working on laptops. A lightbulb icon is above the man's head, and a computer screen icon is above the woman's head. The background is a dark blue wavy shape. The word "Ejemplo" is written in white in the center.

Ejemplo



RESPONSE

200 Ok

```
{
  "id": 1,
  "name": "Rick Sanchez",
  "status": "Alive",
  "species": "Human",
  "type": "",
  "gender": "Male",
  "hair": "Blue"
}
```

Obtiene el personaje ID = 1

REQUEST BODY

```
{  
  "hair": "White"  
}
```

RESPONSE

```
200 Ok  
{  
  "id": 1,  
  "name": "Rick Sanchez",  
  "status": "Alive",  
  "species": "Human",  
  "type": "",  
  "gender": "Male",  
  "hair": "White"  
}
```

Actualiza la propiedad *hair* del personaje ID = 1



RESPONSE

200 Ok

```
{  
  "id": 1,  
  "name": "Rick Sanchez",  
  "status": "Alive",  
  "species": "Human",  
  "type": "",  
  "gender": "Male",  
  "hair": "White"  
}
```

Obtiene el personaje ID = 1 (con propiedad *hair* cambiada)

DELETE characters/1

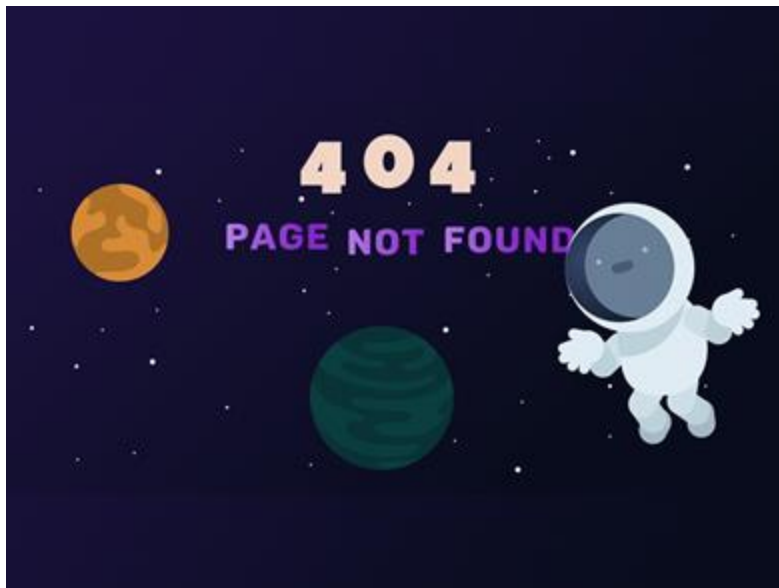
REQUEST BODY

RESPONSE

204 No Content

Borra el personaje ID = 1

GET characters/1



RESPONSE

404 Not Found

Obtiene el personaje ID = 1 (no existe porque se borra en paso anterior)



RESPONSE

200 Ok

```
{  
  "id": 2,  
  "name": "Morty",  
  "status": "Alive",  
  "species": "Human",  
  "type": "",  
  "gender": "Male",  
  "hair": "Brown"  
}
```

Obtiene el personaje ID = 2 (recién creado)

POST characters/

REQUEST

```
{  
  "name": "Summer",  
  "species": "Human",  
  "gender": "Female",  
  "hair": "Orange"  
}
```

RESPONSE

```
201 Created  
  
{  
  "id": 3,  
  "name": "Summer",  
  "status": "Alive",  
  "species": "Human",  
  "type": "",  
  "gender": "Female",  
  "hair": "Orange"  
}
```

Crea un nuevo personaje. Variables como ID u otras se llenan



RESPONSE

200 Ok

```
{  
  "id": 3,  
  "name": "Summer",  
  "status": "Alive",  
  "species": "Human",  
  "type": "",  
  "gender": "Female",  
  "hair": "Orange"  
}
```

Obtiene el personaje ID = 3

Interfaz Uniforme: Idempotencia

¿A qué nos referimos cuando hablamos de idempotencia?

- a) Una llamada a una API Rest debe contener toda la información necesaria para autenticar y ejecutar la acción
- b) Una misma llamada produce siempre el mismo resultado en el servidor (sin efectos colaterales)

Interfaz Uniforme: Idempotencia

Una misma llamada produce siempre el mismo resultado en el servidor (sin efectos colaterales)

La respuesta puede cambiar debido a que un recurso puede haber cambiado entre requests

Además, definimos una llamada como “segura” cuando no actualiza o borra recursos

GET, OPTIONS y HEAD

- Idempotente – Seguro
- Sólo realizan consultas del objeto, read-only, no se realizan cambios

PUT, PATCH y DELETE

- Idempotente – No Seguro
- Update de elementos con los mismos parámetros tiene el mismo resultado

POST

- No idempotente – No Seguro
- Cambia el estado del objeto

El estándar sólo existe para la comunicación entre partes y las operaciones que pueden realizar

No existe estándar para los mensajes en sí

Browsable API's

Browsable API's

¿Y si no necesitáramos documentación de la API?

¿Si sólo con consumirla pudiera saber todas las rutas?



- La raíz de la API muestra todos los recursos y las url's para obtenerlos
- Cada recurso viene con las url's de sus recursos relacionados y subrecursos.
- Esto permite navegar la API sin necesidad de documentación
- Permite a la API mutar rutas sin aviso
- Ejemplo: [Rick & Morty API](#)



GET /api

RESPONSE

200 Ok

```
{  
  "characters": "https://rickandmortyapi.com/api/character",  
  "locations": "https://rickandmortyapi.com/api/location",  
  "episodes": "https://rickandmortyapi.com/api/episode"  
}
```

Obtiene la raíz de la API

GET api/character/1



```
rickandmortyapi.com/api/character/1

{
  id: 1,
  name: "Rick Sanchez",
  status: "Alive",
  species: "Human",
  type: "",
  gender: "Male",
  - origin: {
    name: "Earth (C-137)",
    url: "https://rickandmortyapi.com/api/location/1"
  },
  - location: {
    name: "Citadel of Ricks",
    url: "https://rickandmortyapi.com/api/location/3"
  },
  image: "https://rickandmortyapi.com/api/character/avatar/1.jpeg",
  - episode: [
    "https://rickandmortyapi.com/api/episode/1",
    "https://rickandmortyapi.com/api/episode/2",
    "https://rickandmortyapi.com/api/episode/3",
    "https://rickandmortyapi.com/api/episode/4",
    "https://rickandmortyapi.com/api/episode/5",
    "https://rickandmortyapi.com/api/episode/6",
    "https://rickandmortyapi.com/api/episode/7",
    "https://rickandmortyapi.com/api/episode/8",
    "https://rickandmortyapi.com/api/episode/9",
    "https://rickandmortyapi.com/api/episode/10",
    "https://rickandmortyapi.com/api/episode/11",
    "https://rickandmortyapi.com/api/episode/12"
  ]
}
```

Obtiene el personaje ID = 1, y las rutas de otros recursos asociados

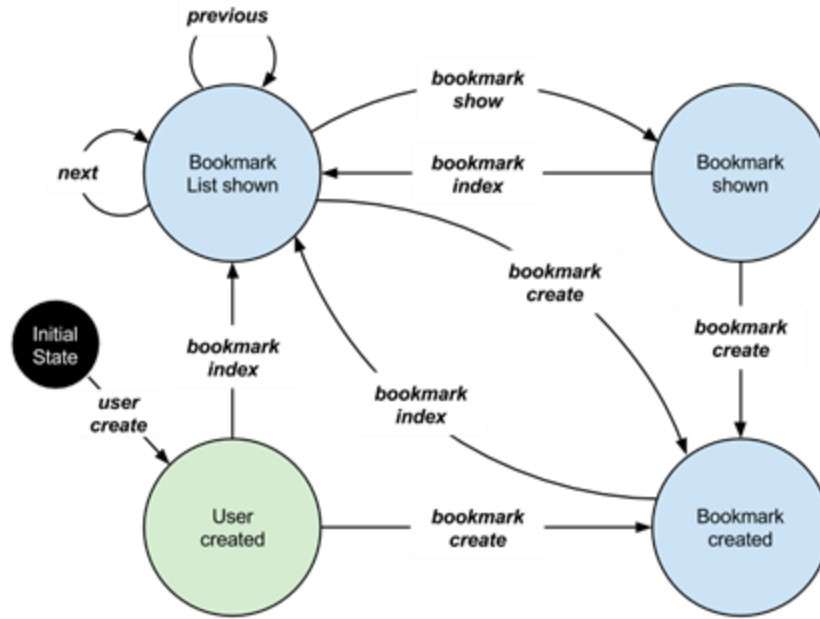
Hypermedia

The engine of application state

Hipermedia

¿Y si cada acción nos dijera todos los posibles caminos o acciones posibles?





1 punto de entrada

9 acciones posibles

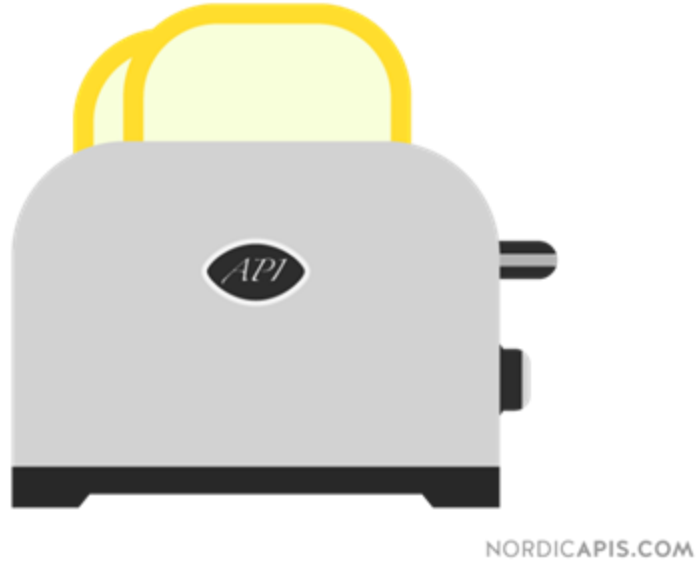
Infinitos caminos o cambios de estado posibles

- Hipermedia dentro de las respuestas de la API son las indicaciones de los caminos posibles a seguir (o estados).
- Sólo necesito conocer el punto de entrada para ir navegando los caminos posibles que me da cada estado.
- Incorpora el concepto de máquina de estados a una API, de forma que sea autodocumentada
- Cada estado tiene una serie de acciones posibles que me permiten ir al siguiente estado.



Ejemplo: IoT Toaster





Tostador inteligente, controlado por una API HATEOAS: sólo conocemos el punto inicial de la API, pero no conocemos la forma de operar el tostador.

Ejemplo tomado de Nordicapis.com (link en bibliografía)

GET /toaster/1

Acción inicial: Obtener tostador id = 1.

HTTP 200 OK

```
{
  "id": "/toaster/1",
  "state": "off",
  "operations": [{
    "rel": "state",
    "method": "PATCH",
    "href": "/toaster/1",
    "expects": {
      "state": "on"
    }
  }]
}
```

PATCH /toaster/1

```
{  
  "state": "on"  
}
```

Cambiar estado a “on”

HTTP 200 OK

```
{  
  "id": "/toaster/1",  
  "state": "on",  
  "strength": 0,  
  "operations": [{  
    "rel": "state",  
    "method": "PATCH",  
    "href": "/toaster/1",  
    "expects": {  
      "state": "off" }  
    }, {  
    "rel": "strength",  
    "method": "PATCH",  
    "href": "/toaster/1",  
    "expects": {  
      "strength": [1,2,3,4,5,6] }  
    }  
  ]  
}
```


PATCH /toaster/1

```
{  
  "strength": 6  
}
```

Setear intensidad en 6

HTTP 200 OK

```
{  
  "id": "/toaster/1",  
  "state": "heating",  
  "strength": 6,  
  "operations": [{  
    "rel": "state",  
    "method": "PATCH",  
    "href": "/toaster/1",  
    "expects": {  
      "state": "off" }  
  }, {  
    "rel": "strength",  
    "method": "PATCH",  
    "href": "/toaster/1"  
    "expects": {  
      "strength": [1,2,3,4,5,6]  
    }  
  }  
]  
}
```

El lenguaje y sintaxis no es aleatoria

Existen algunas convenciones / Estándares para la Hipermedia, como JSON HAL (JSON Hiptertext Application Language)

```
{
  "_links": {
    "self": {
      "href": "http://example.com/api/book/hal-cookbook" },
    "next": {
      "href": "http://example.com/api/book/hal-case-study" },
    "prev": {
      "href": "http://example.com/api/book/json-and-beyond" },
    "first": {
      "href": "http://example.com/api/book/catalog" },
    "last": {
      "href": "http://example.com/api/book/upcoming-books" }
  }
}
```

Autenticación / Autorización

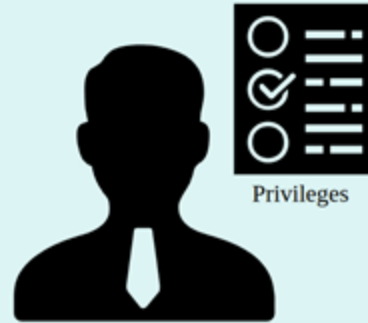
Cómo proteger las APIs

Autorización / Autenticación



Authentication

are you who you say you are?



Authorization

are you allowed to do this action?

APIs Públicas

A grandes rasgos, una API puede ser de uso interno (usada para el funcionamiento de la propia App) o de uso externo (para interactuar con otros actores, por ejemplo, clientes u otras aplicaciones), a las que llamaremos **API interna** y **API externa** respectivamente.

Otra característica importante es su **accesibilidad**, pudiendo ser públicas o privadas. Una **API pública está disponible abiertamente en internet** (cualquier persona con acceso a internet puede verla), mientras que una **API privada está disponible sólo para ciertos usuarios**.

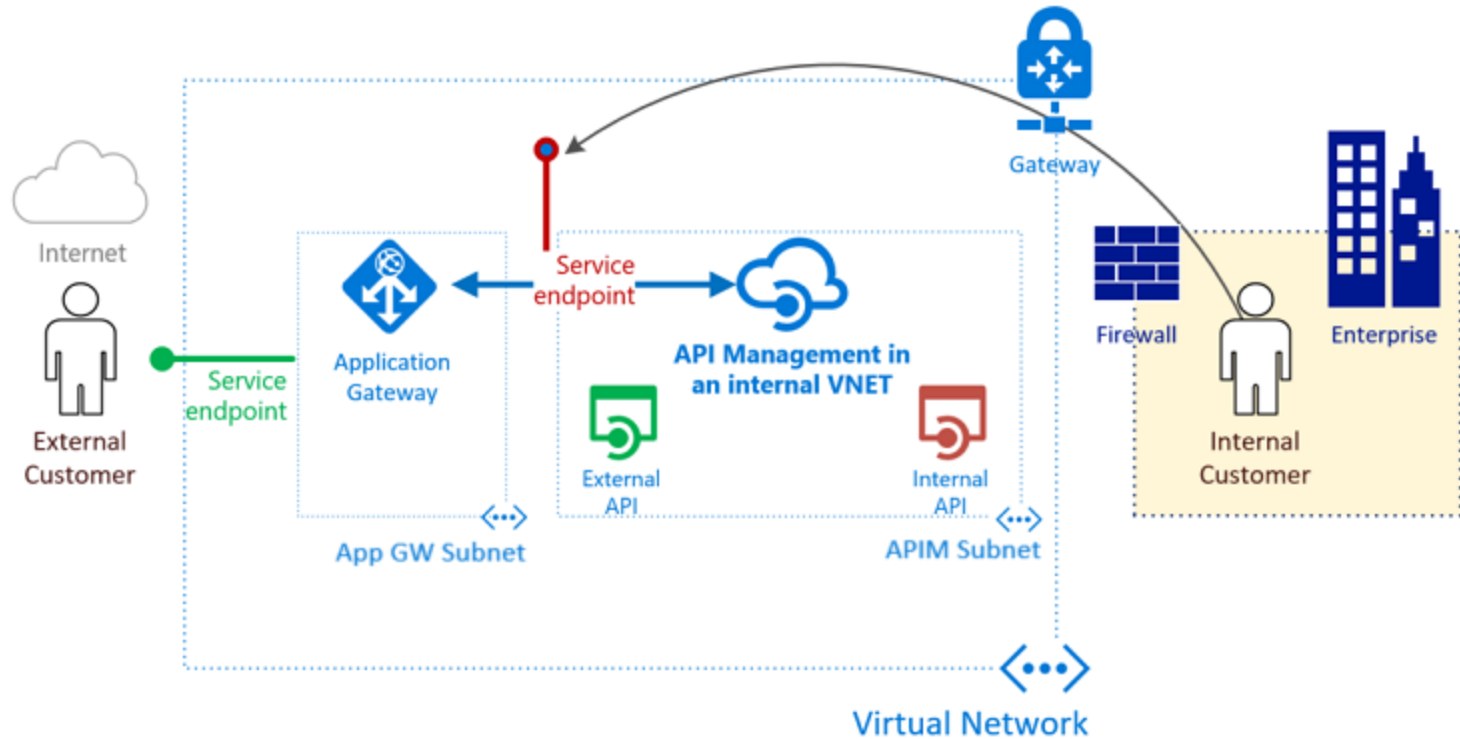
- API Pública del [Instituto de arte de Chicago](https://api.artic.edu/api/v1/artworks) que muestra información y fotos sobre las obras de arte en su colección: <https://api.artic.edu/api/v1/artworks>
- API Pública del servicio meteorológico de Singapur con mediciones de la temperatura del aire en tiempo real: <https://api.data.gov.sg/v1/environment/air-temperature>

APIs Privadas



Para consumir una API sin autenticación, no necesitarás ningún tipo de credenciales, en cambio, una API con autenticación requiere que el usuario demuestre que tiene permiso para usarla, en su versión más simple, usando un usuario y contraseña. Si intentas acceder a la API sin estar autenticado, esta te responderá con un error de permisos.

APIs Privadas



APIs Privadas

Una API privada requiere algún tipo de autenticación para validar que tienes permisos para usarla.

Ejemplos:

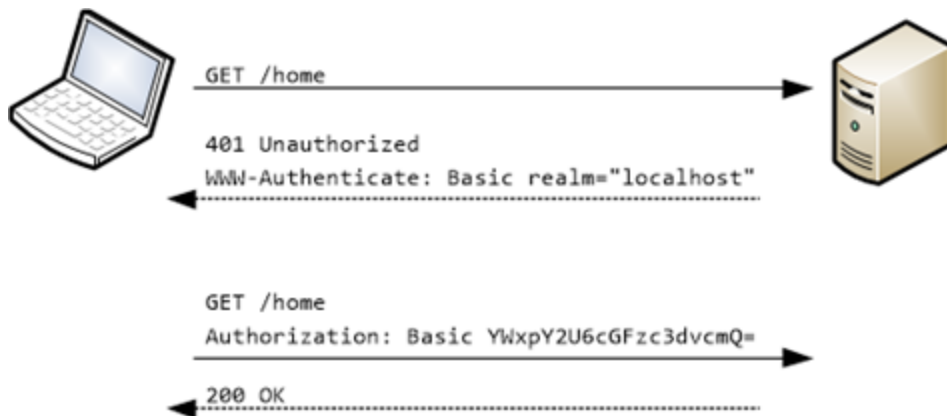
- API de Spotify:
<https://developer.spotify.com/documentation/general/guides/authorization/use-access-token/>
- API de Slack:
<https://api.slack.com/authentication/basics#calling>
- API de Meta
<https://developers.facebook.com/docs/facebook-login/guides/access-tokens>
- API de Salesforce:
https://developer.salesforce.com/docs/atlas.en-us.api_rest.meta/api_rest/using_resources_working_with_records.htm
- API de Beetrack: <https://apidoc.beetrack.com>
- API de Adereso: <https://api.adere.so/docs>

Tipos de Autenticación

Entre las formas más simples de autenticación encontramos:

- API Key: `"x-api-key": "13qiwjoi12he87h21e7y2ue9023e"`
- Bearer Token: `"Authorization": "Bearer p09wi1298wu872ye8732ye8"`

- Basic Auth



Basic Auth

Basic authentication is a simple authentication scheme built into the HTTP protocol. The client sends HTTP requests with the Authorization header that contains the word Basic word followed by a space and a base64-encoded string username:password. For example, to authorize as demo / p@55w0rd the client would send


b64encode(demo:p@55w0rd) => ZGVtbzpwQDU1dzByZA==

Authorization: **Basic ZGVtbzpwQDU1dzByZA==**


OAuth: Open Authorization

https://github.com/login/oauth/authorize?response_type=code&client_id=0215e013a47011c6abf1f8redire

3rdPartApp wants to access your Google Account

 some@email.com

This will allow **3rdPartApp** to:

-  View and edit events on all your calendars

Make sure you trust 3rdPartApp




You may be sharing sensitive info with this site or app. Learn about how calendly.com will handle your data by reviewing its [terms of service](#) and [privacy policies](#). You can always see or remove access in your [Google Account](#)

[Learn about the risks](#)

[Cancel](#) [Allow](#)

Authorize Thalion Herohtar

Thalion Herohtar by Herohtar wants to access your Herohtar account

-  Personal user data Full access
-  Repositories Public and private
- Organization access
 -  EpicGames ✓

[Authorize Herohtar](#)

Log In with Google

Log In with Apple

Log In with Github

Allow "Maps" to access your location while you are using the app?

Your current location will be displayed on the map and used for directions, nearby search results, and estimated travel times.

[Don't Allow](#) [Allow](#)

Search for a place or address

OAuth: Open **A**uthorization

OAuth 1.0 was largely based on two existing proprietary protocols: Flickr's authorization API and Google's AuthSub

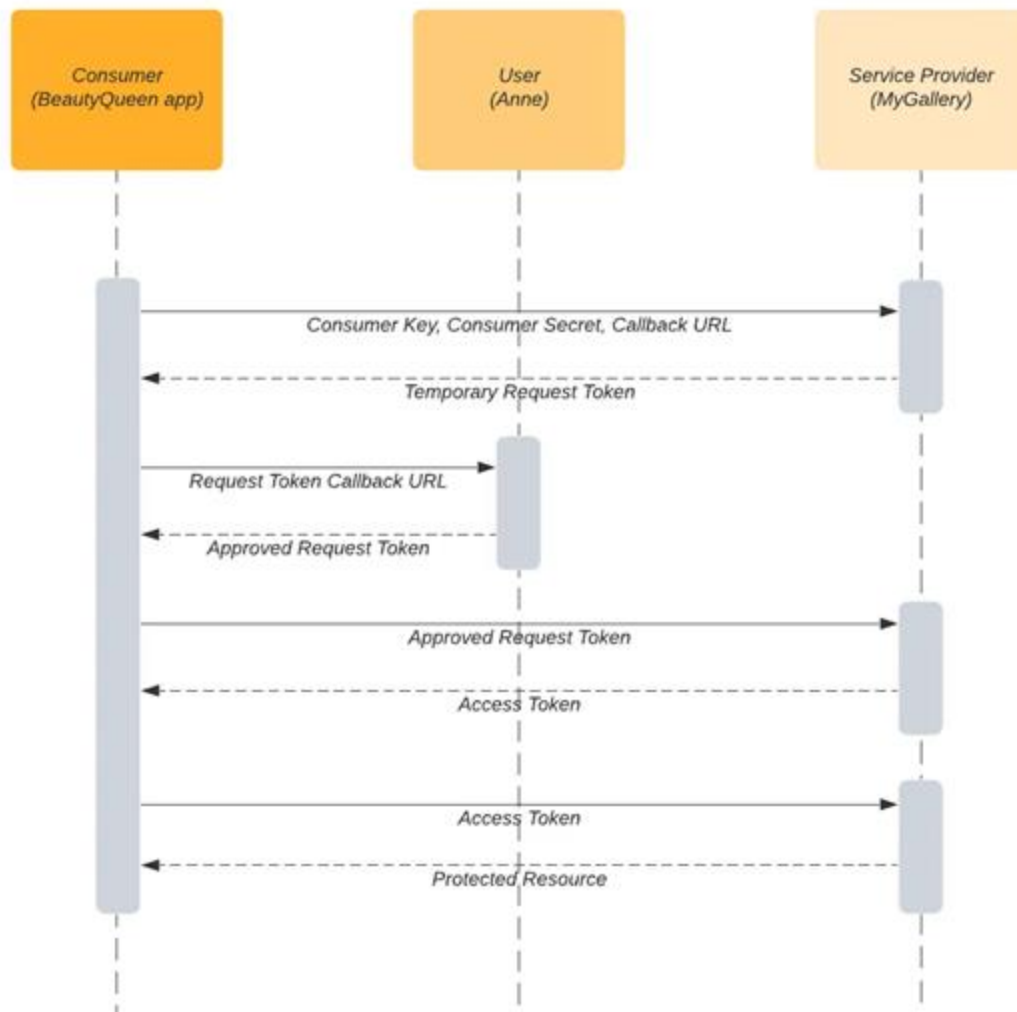
Over a few years of many companies building OAuth 1 APIs, and many developers writing code to consume the APIs, the community learned where the protocol was proving challenging to people.

Several specific areas were identified as needing improvement because they were either limiting the abilities of the APIs, or were too challenging to implement.

OAuth 2.0 represents years of discussions between a wide range of companies and individuals including Yahoo!, Facebook, Salesforce, Microsoft, Twitter, Deutsche Telekom, Intuit, Mozilla and Google.

OAuth 1.0

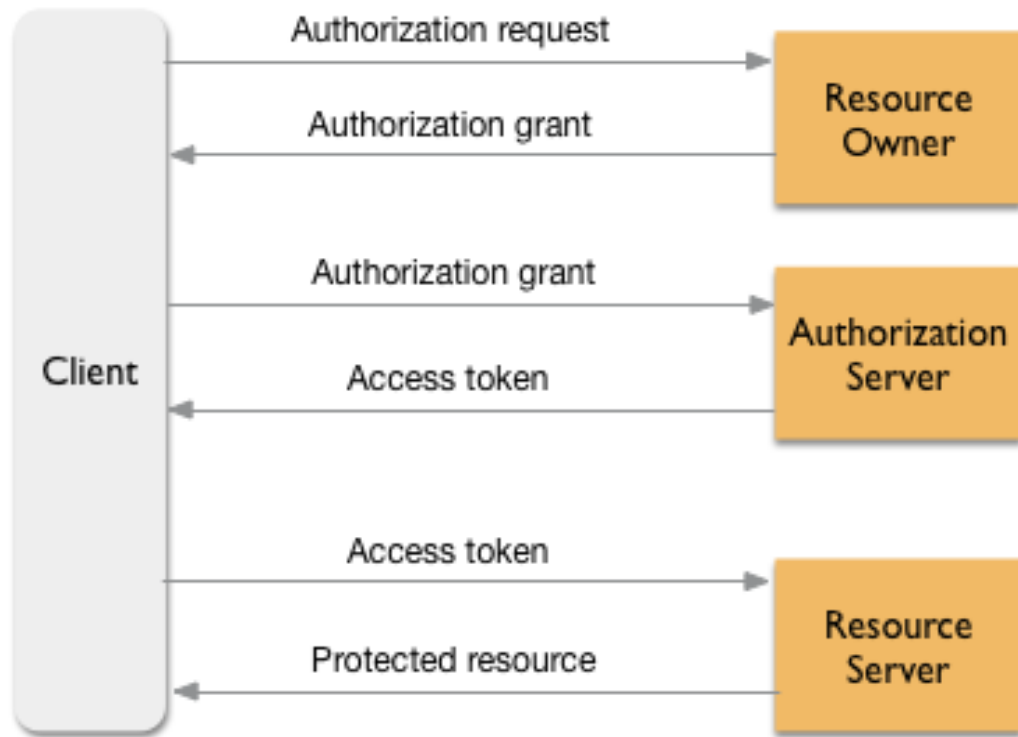
- Designed for web applications
- Generates a signature on every API call to the server (created before HTTPS)
- No decoupling between Resource Provider and Authorization Provider.
- Long lived access tokens



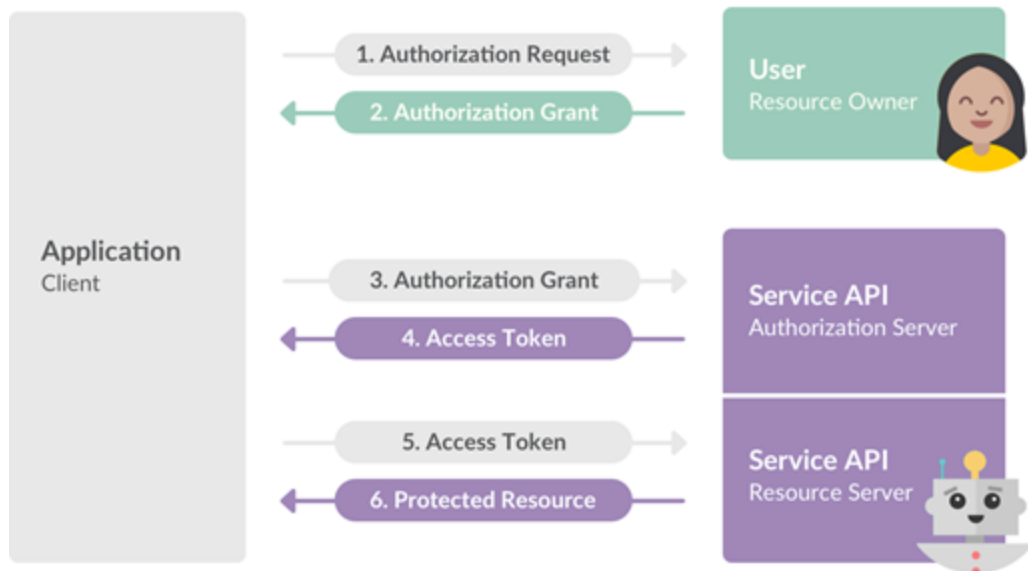
OAuth 2.0

- Better support for non-browser applications
- Reduced complexity in signing requests, uses HTTPS for communication.
- The separation of roles
- The short-lived access token and the refresh token

OAuth 2.0 is a complete rewrite of OAuth 1.0 and it's not backward compatible with OAuth 1.0 *



Ejemplo Flujo OAuth 2.0



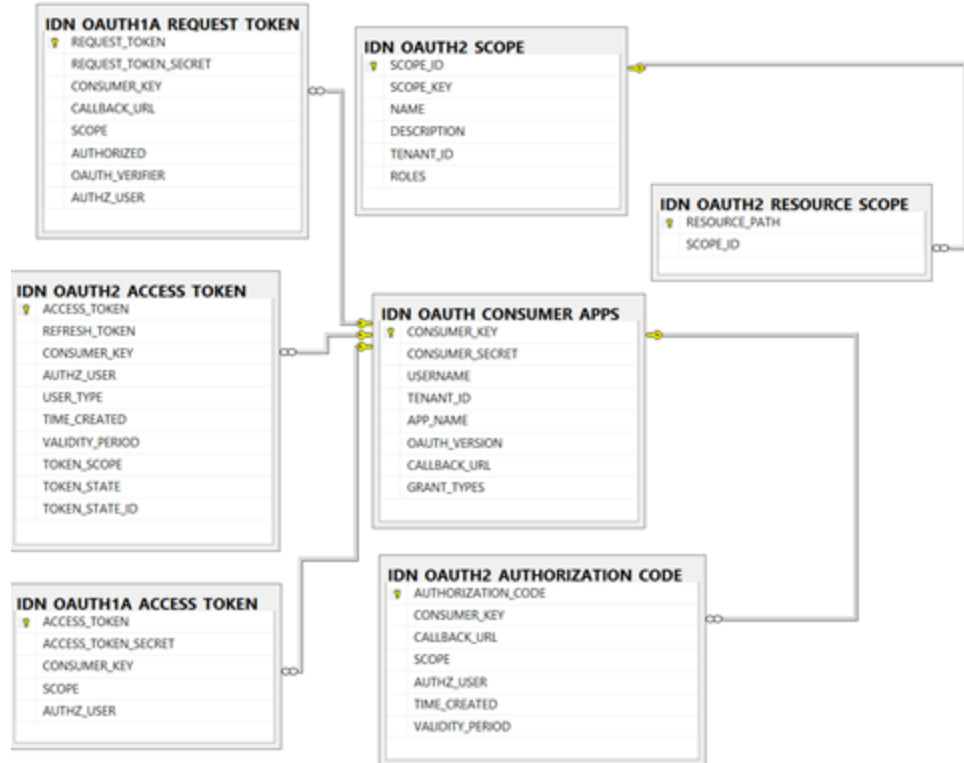
Implementación de Autenticación / Autorización

Ventajas:

- Tienes mucho control. Se pueden invalidar tokens de acceso desde el servidor.

Desventajas:

- Alta complejidad.
- Bases de datos.
- No permite autenticar entre servicios sin comunicarse entre ellos.



JSON Web Tokens

header.payload.signature

<https://jwt.io>

Header

base64UrlEncode



```
{  
  "alg": "HS256",  
  "typ": "JWT"  
}
```

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9

Payload

`base64UrlEncode`



```
{  
  "iss": "tallerdeintegracion.cl",  
  "exp": 1426420800,  
  "company": "Taller de Integración",  
  "awesome": true  
}
```

```
eyJpc3MiOiJ0YWxsZXJkZWludGVncmFjaW9uLmNslwiZXhwIjoxNDI2NDIwODAwLCJjb21wYW55IjojVGFsbGVyIGRlEludGVncmFjacOzbiIsImF3ZXNvbWUiOnRydWV9
```

Signature

```
encodedContent = base64UrlEncode(header) + "." + base64UrlEncode(payload);  
signature = hashHmacSHA256(encodedContent, secret);
```

JSON Web Tokens

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJ0YWxsZXJkZWludGVncmFjaW9uLmNslwiZ_{XhwljoxNDI2NDIwODAwLCJjb21wYW55IjojVGFs}
bGVyIGRIIEludGVncmFjacOzbilslmF3ZXNvbWUi
OnRydWV9.j8J1ilFP083Mvu2dTWdqmsYqqaEeC
P8Qrn4FUFCzFF0

Secret usado para esta firma: kXp2s5v8y/B?E(H+MbQeThWmYq3t6w9z

JSON Web Tokens

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJ0YWxsZXJkZWludGVncmFjaW9uLmNslwiZ_{XhwljoxNDI2NDIwODAwLCJjb21wYW55Ijo}iVGFs_{bGVyIGRIIEludGVncmFjacOzbilslmF3ZXNvbWUi}
OnRydWV9.p1JPtrCiy6YTs8l8_1hWyhuTmukAmJ
KwqmjWpKYBy84

Usando otro **secret**: fThWmZq4t7w!z%C*F-JaNdRgUkXn2r5u

JSON Web Tokens

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJ0YWxsZXJkZWludGVncmFjaW9uLmNslwiZ_{XhwljoxNDI2NDIwODAwLCJjb21wYW55IjojVGFs}
bGVyIGRIIEludGVncmFjacOzbilslmF3ZXNvbWUi
OnRydWV9.j8J1ilFP083Mvu2dTWdqmsYqqaEeC
P8Qrn4FUFCzFF0

Secret usado para esta firma: kXp2s5v8y/B?E(H+MbQeThWmYq3t6w9z

JSON Web Tokens

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJ0YWxsZXJkZWludGVncmFjaW9uLmNslwiZ
XhwljoxNDI2NDIwOTAwLCJjb21wYW55IjoiaVGFs
bGVyIGRIIEludGVncmFjacOzbilslmF3ZXNvbWUi
OnRydWV9.PpQZzNsjjLsHCFSUSKwlqxNu3owU
Jw-v6huXJCoFU-o

Modificando payload: "exp": 1426420800 -> "exp": 1426420900

JSON Web Tokens

Ventajas:

- La autenticación entre servicios es muy liviana y rápida ya que no necesita comunicación entre ellos. (Basta con que ambos compartan el **secret** para validar que fue hecho por el otro).
- No hay que mantener bases de datos ni lógicas complejas de autenticación / Autorización.

Desventajas:

- El payload debe ser pequeño, de lo contrario los tokens quedan muy largos.
- No se puede invalidar un token desde el servidor



PONTIFICIA
UNIVERSIDAD
CATÓLICA
DE CHILE

IIC3103

Taller de Integración

Profesores

Arturo Tagle / Daniel Darritchon



Bibliografía

Richardson maturity model

Richardson Maturity Model, <https://restfulapi.net/richardson-maturity-model/>

Richardson Maturity Model, steps toward the glory of REST,
<https://martinfowler.com/articles/richardsonMaturityModel.html>

Act Three: The Maturity Heuristic:
<https://www.crummy.com/writing/speaking/2008-QCon/act3.html>

Implementing HATEOAS with Django REST framework:
<https://blog.majsky.cz/implementing-hateoas-django-rest-framework/>



Bibliografía

HATEOAS

HATEOAS 101: Introduction to a REST API Style (video & slides), <https://apigee.com/about/blog/technology/hateoas-101-introduction-rest-api-style-video-slides>

Haters gonna HATEOAS, <http://timelessrepo.com/haters-gonna-hateoas>

HATEOAS Driven REST APIs: <https://restfulapi.net/hateoas/>

Why HATEOAS is useless and what that means for REST: <https://medium.com/@andreasreiser94/why-hateoas-is-useless-and-what-that-means-for-rest-a65194471bc8>

How to Improve API Experience Using Hypermedia, <https://nordicapis.com/improve-api-experience-using-hypermedia/>

Designing a True REST State Machine, <https://nordicapis.com/designing-a-true-rest-state-machine/>

Hypertext Application Language: https://en.wikipedia.org/wiki/Hypertext_Application_Language

A Pragmatic Take On REST Anti Patterns: <https://nordicapis.com/a-pragmatic-take-on-rest-anti-patterns/>



Bibliografía

Autenticación / Autorización

Imagen Autenticación/Autorización:

https://idratherbewriting.com/learnapidoc/docapis_more_about_authorization.html

Imagen API interna: <https://learn.microsoft.com/en-us/azure/api-management/api-management-howto-integrate-internal-vnet-appgateway>

Imagen HTTP 403 Forbidden: <https://dev.to/abbeyperini/beginners-guide-to-http-part-5-authentication-3p2p>

Imagen Basic Auth: <https://learn.microsoft.com/es-es/aspnet/web-api/overview/security/basic-authentication>

Basic Auth:

<https://swagger.io/docs/specification/authentication/basic-authentication/#:~:text=Basic%20authentication%20is%20a%20simple,%2Dencoded%20string%20username%3Apassword%20>

Oauth:

<https://cloud.google.com/apigee/docs/api-platform/security/oauth/oauth-introduction?hl=es-419>

<https://www.rfc-editor.org/rfc/rfc6749>

<https://www.oauth.com/oauth2-servers/differences-between-oauth-1-2/>

<https://medium.com/identity-beyond-borders/oauth-1-0-vs-oauth-2-0-e36f8924a835>

JWT:

<https://jwt.io>

<https://www.rfc-editor.org/rfc/rfc7519>

<https://www.toptal.com/web/cookie-free-authentication-with-json-web-tokens-an-example-in-laravel-and-angularjs>