

Drupal Services

Agustin Casiva

agustin@42mate.com

www.42mate.com

Intro

Hoy en día las arquitecturas web son complejas, el viejo y sencillo modelo LAMP está prácticamente obsoleto. Hoy en día el ecosistema es muy diverso, debemos integrar aplicaciones (feeds, twits, facebook, otros sitios), tenemos diversos clientes (celulares, tablets, televisores, browsers, otros sitios). Desde el backend la cosa no es muy diferente, tenemos diversos backends de datos (MySQL, Redis, Memcache, ElasticSearch, Solr, Otros sitios).

Para poder intercomunicar todos estos productos que cumplen un rol dentro de la aplicación necesitamos algún canal basado en un protocolo de comunicación para lograr el intercambio de información.

Algunos protocolos son estandares, como el protocolo de comunicación a MySQL el cual es bien conocido y requiere un driver para permitir la interacción el cual implementa el estándar establecido. Otros son específicos e implementados por la desarrolladores de la aplicación.

La forma más sencilla que tienen los developers hoy en día para crear estos canales de comunicación específicos para sus aplicaciones son los Web Services. Basándose en un protocolo bien conocido de comunicación, por ejemplo HTTP, montan sus servicios aceptando datos de entrada y devolviendo información del sistema en algún formato fácilmente procesable por otro sistema (XML, Json, CSV, etc). Existen diferentes tipos de servicios web, SOAP, REST, XML-RPC entre otros, pero sin dudas los más prácticos y más controversiales hoy en día son los servicios REST.

La filosofía de los servicios REST es reutilizar toda la infraestructura de la World Wide Web y el protocolo HTTP para interconectar aplicaciones. En criollo, así como pedimos una página web diciendo

https://www.google.com.ar/?gfe_rd=cr&ei=CkGIVq6AKsqB8Qemj7XYCQ&gws_rd=ssl#q=rest

y este devuelve una página web con la información legible para humanos, la idea es que podamos hacer la misma petición pero obtener la información en un formato legible por un sistema así el mismo puede procesarla y hacer lo que necesite con la misma.

De la misma forma, enviando peticiones http poder impactar cambios en el estado sistema remoto.

Estas URLs que expone el sistema remoto se conocen como endpoints, y el conjunto de endpoints se conoce como la API del sistema remoto.

La controversia viene de la mano que REST establece cómo deben realizarse estas peticiones y la forma en que deben implementarse los endpoints para que tengan un sentido común para la API que provee el sistema remoto. Muchas veces como desarrolladores no seguimos adecuadamente estas reglas y se generan discusiones de lo mal o bien que hicimos nuestra API. Un mal diseño de una API puede traer problemas serios si nuestro sistema crecerá en endpoints, si crecerá el número de developers que desarrollaran la API y si crece el número de clientes que tendremos. Lo mejor será seguir las reglas de REST, las cuales son sencillas, desde el día 0 y estar seguro que el equipo las conoce a la hora de extender y mantener la API.

Entonces, antes de comenzar con APIs REST deberían tener bien claro

- El protocolo HTTP, verbos, urls, headers, codigos de respuesta
- Cómo usar HTTP para hacer REST
- Autenticación y Autorización
- Seguridad

Un buen libro para conocer estas cosas es "Build APIs You Won't Hate" escrito por Phil Sturgeon.

Y Drupal ?

Drupal provee un conjunto de módulos muy prácticos para implementar APIs REST en nuestros Web Sites basados en Drupal los cuales aseguran que sigamos estos lineamientos y que sea muy fácil hacerlo. Asi y todo, puedo dar fe como algunos developers lo hacen mal de todos modos, pero si te preocupa hacerlo bien, Drupal te la hace muy fácil.

El principal módulo es Services, este módulo provee casi todo lo necesario para implementar servicios web bien implementados relativamente fácil. Services es agnóstico al protocolo a usar, que conozca posee soporte para SOAP, XML-RPC y REST, solo debemos instalar los servers que son módulos adicionales que corren sobre services y realizar la integración de los mismos. Yo solo he trabajado con REST asi que hablaré de este Server.

Por otro lado, un gran problema de los servicios REST es la autenticación y la autorización. Existen varias formas de validar quién es el que realiza las peticiones (Autenticación) y si puede o no realizar la acción y/o consultar los datos (Autorización). Existen protocolos para hacer este proceso y también hay muchas discusiones sobre cual es la mejor forma de hacerla. El mecanismo más conocido por estos días es OAuth2 el cual es un protocolo para autenticar y autorizar servicios web basándose en tokens. No soy un experto en OAuth2 pero trataré de explicar aquí lo necesario para que lo integren en su Drupal. Existe un módulo que provee soporte para OAuth2 a los endpoints expuestos por Services llamado intuitivamente OAuth2.

De aquí en más trataré de explicarles cómo configurar los módulos, les mostraré como realizar autorizaciones y autenticaciones, como implementar nuevos endpoints y como usar herramientas para probar sus endpoints.

Instalación de Módulos

Partiremos desde una instalación limpia de Drupal con un profile por defecto.

Descargaremos los módulos

- Services
- Libraries
- Chaos Tools
- Entity API
- Entity Reference
- X Autoload
- OAuth2 Server

Vamos a habilitar

- Services
- REST Server
- OAUTH2 Server
- Libraries

Libraries necesita la librería <https://github.com/bshaffer/oauth2-server-php>, hay que bajar la última versión, descomprimir y ponerla en el directorio libraries.

Con esos módulos habilitados ya estamos para comenzar a configurar los servicios.

Configurando el Servicio

Para comenzar a configurar el servicio debemos ir a Structure > Services, ahí veremos una pantalla como esta, iremos a Add para añadir una API nueva.

Home » Administration » Structure

Services

Services are collections of methods available to remote applications. They are defined in modules, and may be accessed in a number of ways through server modules. Visit the [Services Handbook](#) for help and information.

All enabled services and methods are shown. Click on any method to view information or test.

[+ Add](#) [+ Import](#)

Storage: - All - Enabled: - All - Search:

Sort by: Enabled, name Order: Up [Apply](#) [Reset](#)

NAME	STORAGE	OPERATIONS
There are no endpoints to display.		

Para crear necesitaremos completar este formulario

Home » Administration » Structure » Services

Add a new endpoint

Machine-readable name of the endpoint *

The endpoint name can only consist of lowercase letters, underscores, and numbers.

Server *

REST

Select a the server that should be used to handle requests to this endpoint.

Path to endpoint *

☐ Debug mode enabled
Useful for developers. Do not enable on production environments

Authentication

☐ OAuth2 authentication

☐ Session authentication

Choose which authentication schemes that should be used with your endpoint. If no authentication method is selected all requests will be done by an anonymous user.

[Save](#)

Primero nos pide el machine name, un identificador para la API.

Nos pide elegir un server, solo habilitamos REST.

Path al endpoint, de aquí se desprenden todos los recursos que definamos para la API, es básicamente el prefix para las urls que definimos.

Luego si queremos podemos tildar debug el cual nos dejará información en el dblog.

Por último nos pide autenticación, podemos usar basada en session (usando la cookie de session de PHP) o basada en OAuth2. Ambas funcionan y tienen sus pro y cons, veremos ambos casos, por ahora para hacerlo simple dejaremos sin autenticación.

Una vez creado lo tendemos en la lista de services que definimos, observe que podemos crear varios tipos de services, ideal para manejar diferentes versiones de nuestra API o para soportar diferentes tipos de Web Services (SOAP, XML-RPC, etc).

[Home](#) » [Administration](#) » [Structure](#)

Services

Services are collections of methods available to remote applications. They are defined in modules, and may be accessed in a number of ways through server modules. Visit the [Services Handbook](#) for help and information.

All enabled services and methods are shown. Click on any method to view information or test.

[+ Add](#) [+ Import](#)

Storage

Enabled

Search

- All -

- All -

Sort by

Order

Enabled, name

Up

Apply

Reset

NAME	STORAGE	OPERATIONS
demo	Normal	Edit Resources

En Edit Resources podremos habilitar o deshabilitar los recursos disponibles en este service.

Antes de avanzar veamos que hay en la solapa Server

Services

EDIT

SERVER

AUTHENTICATION

RESOURCES

EXPORT

REST

Response formatters *

☐ bencode

☒ json

☐ jsonp

☐ php

☐ xml

☐ yaml

Select the response formats you want to enable for the rest server.

Request parsing *

☒ application/json

☐ application/vnd.php.serialized

☐ application/x-www-form-urlencoded

☐ application/x-yaml

☐ application/xml

☐ multipart/form-data

☐ text/xml

Select the request parser types you want to enable for the rest server.

Save

En esta podemos definir, para el server REST, qué response formats soportaremos (como serán las respuestas) y de qué forma podemos definir los payloads de nuestras request. Como ven hay varias opciones disponibles (normalmente cuando hacemos algo custom no se tienen en cuenta todas estas alternativas), por ahora solo dejaremos disponibles las opciones basadas en json.

Resources

Los resources son las funcionalidades que implementamos para exponer a través del servicio. Cuando vayamos al código implementaremos resources, los resources estarán disponibles para todos los services, en cada service podemos habilitar o deshabilitar los resources que exponemos a través del service.

Para comenzar a entender este concepto veamos los recursos que services nos trae por defecto.

Por defecto, service provee resources para interactuar con los pilares de drupal, nodos, taxonomías, usuarios, files y algunas funciones del sistema. Probaremos habilitar algunos resources de nodos para probar.

Si abrimos las opciones en node veremos que tenemos

- **CRUD Operations** : La base de REST, permite Crear, Leer, Updatear o Borrar un recurso individual
- **Action** : Sirve para hacer alguna acción en los recursos, ejemplo actualizar nodos en solr.
- **Targeted actions** : Sirve para hacer una acción en un recurso particular, ejemplo publicar/despublicar nodo
- **Relationships** : Sirve para implementar acciones siendo sugar syntactic a un recurso en particular, ejemplo comentarios de un nodo.

El REST server automáticamente mapea estos conceptos con los verbos http, entonces, teniendo

C,R,U,D,I = Create, Read, Update, Delete (CRUD) and Index requests

A = Action

T = Targeted action

X = Relationship request

Para entender cómo pedir cada uno de estas operaciones podemos usar las siguientes tabla.

```
COUNT |0|1|2|3|4|N|
```

```
-----
```

```
GET  |I|R|X|X|X|X|
```

```
-----
```

```
POST |C|A|T|T|T|T|
```

```
-----
```

```
PUT  | |U| | | | |
```

```
-----
```

```
DELETE |D| | | | |
```

```
-----
```

Count se refiere a la cantidad de elementos en el path, por ejemplo, para el resource news en nuestro caso el path es **demo/api/v1/news**, hasta ahí es el recurso y el count es 0. Si ahora decimos **demo/api/v1/news/223**, el count será de 1 y 223 es el nid de la news. Si fuese una targeted action podría ser **demo/api/v1/news/223/unpublish**, el count es 2 y unpublish será usado para saber que targeted action usar, caso similar sería si fuese una relationship request.

En resumen, que para poder pegar a un recurso index debemos usar GET. Para actualizar un recurso debemos usar PUT, para borrar un recurso debemos usar DELETE. Para realizar una acción sobre node debemos usar POST. Para realizar una targeted action debemos usar POST. Lo único que nos falta saber es la url del endpoint a utilizar.

Home > Administration > Structure > Services

Services

EDIT SERVER AUTHENTICATION RESOURCES EXPORT

✓ Resources have been saved

Resources
Select the resource(s) or methods you would like to enable, and click Save.

RESOURCE	SETTINGS	ALIAS
<input type="checkbox"/> file		<input type="text"/>
<input type="checkbox"/> node		<input type="text"/>
CRUD operations		
<input checked="" type="checkbox"/> retrieve Retrieve a node		
<input type="checkbox"/> create Create a node		
<input type="checkbox"/> update Update a node		
<input type="checkbox"/> delete Delete a node		
<input checked="" type="checkbox"/> index List all nodes		

Por ahora, en resources, habilitemos retrieve e index. Ambos podrán ser utilizados con GET.

La url del resource

Dijimos previamente que el path al resource es demo/api/v1/news, veamos de donde sale. Para obtener la url del resource primero debemos tener bien presente

- El dominio / IP del site, en nuestro caso 10.11.12.201
- El prefix del Service, en nuestro caso api/demo/v1
- El nombre del resource, en este caso node
- El ID del resource en caso que queramos realizar algo sobre un recurso en particular.

Con estas cosas deduciremos la URL.

Para el caso de index de nodos será

GET http://10.11.12.201/api/demo/v1/node

Note que al dominio, añadimos el prefix y el nombre del resource, esto devolverá todos los nodos disponibles.

Esto devolverá algo como esto

```
[
  {
    "nid": "48",
    "vid": "48",
    "type": "article",
    "language": "und",
    "title": "Pneum",
    "uid": "0",
    "status": "1",
    "created": "1451724514",
    "changed": "1451784579",
    "comment": "0",
    "promote": "0",
    "sticky": "0",
    "tnid": "0",
    "translate": "0",
    "uri": "http://10.11.12.201/api/demo/v1/node/48"
  },
  {
    "nid": "34",
    "vid": "34",
    "type": "page",
    "language": "und",
    "title": "Exerci Oppeto Sit",
    "uid": "0",
    "status": "1",
    "created": "1451701672",
    "changed": "1451784579",
    "comment": "0",
    "promote": "0",
    "sticky": "0",
    "tnid": "0",
    "translate": "0",
    "uri": "http://10.11.12.201/api/demo/v1/node/34"
  },
  {
    "nid": "36",
    "vid": "36",
    "type": "article",
    "language": "und",
    "title": "Decet Nunc Tation",
    "uid": "0",
    "status": "1",
    "created": "1451698461",
    "changed": "1451784579",
    "comment": "0",
    "promote": "1",
    "sticky": "0",
    "tnid": "0",
    "translate": "0",
    "uri": "http://10.11.12.201/api/demo/v1/node/36"
  },
]
```


La respuesta variará en función de sus content types, sus fields, etc. Para el caso de retrieve, para traer solo un node, será

GET <http://10.11.12.201/api/demo/v1/node/48>, Eso traerá el nodo 48.

```
{
  "vid": "48",
  "uid": "0",
  "title": "Pneum",
  "log": "",
  "status": "1",
  "comment": "0",
  "promote": "0",
  "sticky": "0",
  "nid": "48",
  "type": "article",
  "language": "und",
  "created": "1451724514",
  "changed": "1451784579",
  "tnid": "0",
  "translate": "0",
  "revision_timestamp": "1451784579",
  "revision_uid": "1",
  "body": {
    "und": [
      {
        "value": "Augue comis ...nim sino utrum virtus.\n\n",
        "summary": null,
        "format": "filtered_html",
        "safe_value": "<p>Augu....aecus .</p>\n",
        "safe_summary": ""
      }
    ]
  },
  "field_tags": [],
  "field_image": {
    "und": [
      {
        "fid": "2", "uid": "1", "filesize": "1933", "status": "1",
        "filename": "imagefield_TNF3R8.png",
        "uri": "public://field/image/imagefield_TNF3R8.png",
        "filemime": "image/png",
        "timestamp": "1451784579",
        "alt": "Abdo abicius refoveo saluto.",
        "title": "At camur commodtus quidne ut.",
        "width": "322", "height": "561"
      }
    ]
  },
  "name": "",
  "picture": "0",
  "data": null,
  "path": "http://10.11.12.201/node/48"
}
```

Como ven es así de sencillo, en unos cuantos clicks expusimos nuestra información a cualquier sistema que desee consumirla.

Asi podemos ir deduciendo como será para los otros casos. Si quieren más información revisen este post

<https://www.drupal.org/node/783254>

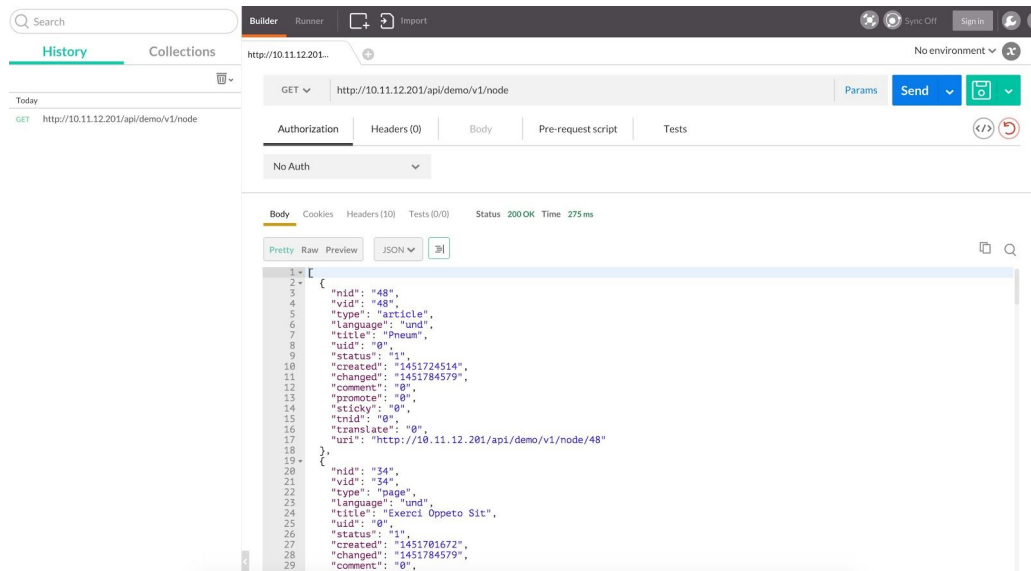
Testeando los endpoints

Para testear los endpoints podemos utilizar cualquier herramienta que nos permita armar requests http, por ejemplo Curl, el tema es que al requerir modificar headers, armar payloads y poder manipular los parametros de la request es práctico contar con algo más visual. Personalmente mi preferida es postman, esta es una extensión de chrome que nos permite crear request, guarda un histórico de las peticiones realizadas, permite exportar las peticiones e importarlas nuevamente, es ideal para el desarrollo de APIs.

Su uso es muy sencillo, instalen la extensión de chrome y tendrá la chrome app de postman lista para usarse.

En la parte superior pueden elegir el verbo HTTP, setear el endpoint, si el endpoint tiene parametros pueden setear los parametros, si requiere un header especial pueden setearlo, si requiere un header especial tambien, en caso de ser un POST o PUT puede cargar un payload en diferentes formatos como form-data, x-www-form-urlencoded, raw o binary, dependiendo lo que requiere el endpoint.

Al darle click en send podrán obtener las respuestas, headers de la respuesta, codigos de respuesta etc.



Codeando nuestros propios resources

Implementar nuestros propios resources es verdaderamente sencillo y muy similar a implementar un hook menu y llamar a un callback para procesar los parametros de entrada y generar la salida.

El hook a implementar para definir nuevos resources es `hook_services_resources`, en hook debemos crear un array donde cada elemento será un resource, dentro del resource podremos definir nuevas entradas al array que podrán ser

- operations
- actions
- targeted_actions
- relationships

Y opérations puede tener los elementos de CRUD

- retrieve
- create
- update
- delete
- index

Dependiendo de lo que necesitemos implementar podemos ir definiendo las entradas al resource.

Por cada uno de estas entradas del resources debemos ir definiendo

- help : Título para el endpoint
- access callback : Función para validar si el usuario puede o no usar el callback
- access arguments : Argumentos para la función de callback
- callback : Función de callback que será invocada
- args : Array que define cómo serán los parámetros

Aquí es donde veo la similitud con el hook menú dado que las entradas son muy parecidas. Args es la entrada más complicada, esta define los parámetros de entrada y de dónde vienen. Nuevamente es un array con varios ítems, uno por cada parámetro.

Cada arg tendrá

- name: El nombre del parámetro
- type: El tipo (int, string, array)
- description: Descripción del argumento
- optional: TRUE/FALSE que determina si es o no opcional
- source: De donde viene?. Puede ser
 - data : POST/PUT payload
 - param : parámetro en el query string
 - path: parámetro que viene en el path, debe indicarse el número de índice en el path.
 - default value: valor por defecto para parámetros opcionales.

Para dejar bien claro esto deberíamos ver un ejemplo.

Primero crearemos un módulo llamado demo_services (no explicaré como hacerlo).

En el módulo implementaremos el hook_services_resources, empezaremos con el hook vacío.

```
<?php
/**
 * Implements hook_services_resources.
 */
function demo_services_services_resources() {
  $resources = array();
  //Resources goes here.
  return $resources;
}
```

Ahora, definiremos un resource news que trabajará sobre un content type news, los métodos trabajarán solo sobre este tipo de content types.

Definamos lo básico para el resource news.

```
$resources['news'] = array(
  'operations' => array(
    'create' => array(),
    'retrieve' => array(),
    'update' => array(),
    'delete' => array(),
    'index' => array(),
  ),
  'actions' => array(),
  'relationships' => array(),
  'targeted_actions' => array(),
);
```

Como ven, el index se llama news, osea que nuestro resource será accesible por dicho index (api/demo/v1/news), el string que utilicemos aquí se mapea en el path al endpoint.

Todas las operaciones disponibles están vacías (los arrays vacíos), aquí es donde deberemos empezar a definirlos.

Creando CRUD Resources

Create

Create se define mediante un array como el siguiente

```
'create' => array(
  'help' => 'Create a Single News',
  'callback' => '_demo_services_news_create',
  'access callback' => '_demo_services_access_callback',
  'args' => array(
    array(
      'name' => 'data',
      'type' => 'array',
      'optional' => FALSE,
      'source' => 'data',
    )
  )
)
```

Como aquí pueden ver, callback dice que la función que atenderá la request será **_demo_services_news_create**, debemos tener esa función definida.

Access callback dice que la función que validará si el usuario puede o no acceder al recurso será **_demo_services_access_callback**, debemos tener esa función definida. Funciona igual que los access callbacks de un menú. Por ahora solo para no profundizar en esto la función será así.

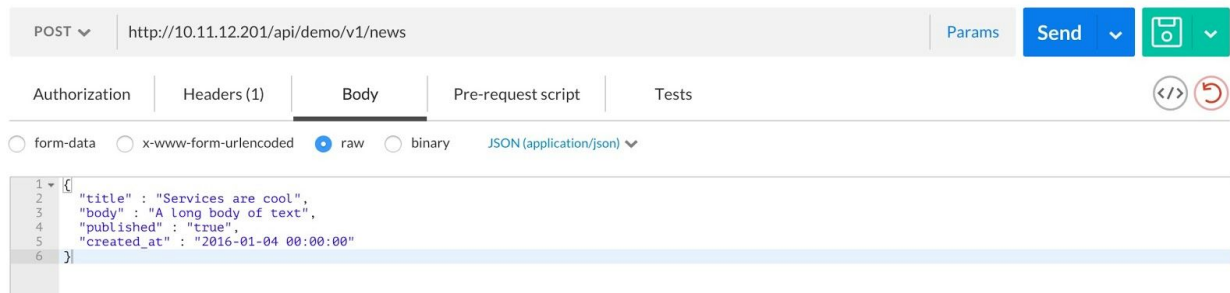
```
function _demo_services_access_callback() {
  //@todo : Write something secure
  return TRUE;
}
```

La misma tendrá como parámetros una variable llamada \$data que será un array con lo que sea que se pasa en el payload de la request.

En endpoint será POST api/demo/v1/news, POST porque es create y el path es el prefijo más el nombre del resource, news.

En el payload deberemos enviar los datos necesarios para crear la news, los mismos deberán estar definidos en un json. Al enviar como json el header de la request debe tener la directiva Content-type application/json para que el REST server realice el parseo del payload.

Si miramos cómo sería en postman nos quedaría una cosa así



Podemos ver la url, el verbo post, el payload (o body) que esta como raw, el json que enviaremos con los parámetros y el header que esta seteado como application/json.

La función **_demo_services_news_create** será algo así

```
function _demo_services_news_create($data) {
  //Data will have all the items as an array, for example $data['title'];

  //@todo, do something to create the node

  return $status; //What do you want to send back ?
}
```

No voy a entrar a explicar la lógica para crear un nodo programáticamente para no irnos del tema, pero si googlean un poco podrán encontrar mucha documentación acerca de node api, sino vean el fuente en el repo.

La función debe devolver algo que pueda ser convertido como json para enviar como respuesta al cliente. Aquí puede ser un objeto complejo, un array, o un simple string con el estado.

Para devolver errores podemos usar la función **services_error**, esta recibe como parámetros el mensaje a enviar (un string), código de respuesta (400,500, 504, etc) y opcionalmente data que es información adicional para el error (puede ser objeto o array o string)

Esta forma que vimos, definiendo el tipo como array y como source data, inyecta directamente todo lo que venga en el json dentro del array \$data. Si quieren hacerlo más granular puede ir levantando una por una las fields del json y definir que se inyectará dentro de \$data que se pasará a la función, una razón para hacer esto puede ser seguridad, otra claridad en los parámetros de entrada. En dicho caso sería así

```

'create' => array(
  'help' => 'Create a Single News',
  'callback' => '_demo_services_news_create',
  'access callback' => '_demo_services_access_callback',
  'args' => array(
    array(
      'name' => 'title',
      'type' => 'string',
      'optional' => FALSE,
      'source' => array('data' => 'title'),
    ),
    array(
      'name' => 'body',
      'type' => 'string',
      'optional' => FALSE,
      'source' => array('data' => 'body'),
    )
  )
),

```

y el callback quedaría así

```

function _demo_services_news_create($title, $body) {
  //Code here.
}

```

Observe que aquí el type de los args ya no es array, string o int y por último todos los args son pasados a la función de callback.

Nota : Cualquier cambio en la definición de recursos necesita una limpieza de la cache.

Update

Los updates se realizan con el verbo PUT, tambien se realizan contra un recurso individual asi que deberíamos pasar el ID al recurso. El path sería algo asi

PUT api/demo/v1/news/ID

Y en el payload enviaremos la data a actualizar.

Para definir una operación update crearemos una entrada como esta

```
'update' => array(
  'help' => 'Update a News',
  'callback' => '_demo_services_news_update',
  'access callback' => '_demo_services_access_callback',
  'args' => array(
    array(
      'name' => 'nid',
      'type' => 'int',
      'optional' => FALSE,
      'source' => array('path' => 0),
    ),
    array(
      'name' => 'data',
      'type' => 'array',
      'optional' => FALSE,
      'source' => 'data',
    )
  )
),
```

Si analizamos el array veremos

_demo_services_news_update es la función de callback

El primer parámetro será tomado del path y sería el ID el cual está en la posición 0 del path y es un int.

El segundo parámetro viene del payload (como en create).

Sabemos que nuestro path es demo/api/v1/news, de ahí en más pueden venir partes del path adicionales que nosotros definamos, en este ejemplo ID es una de ellas pero puede haber más, para poder tomarlas en nuestra definición de la operación en el recurso usamos source asignándole un array cuya clave es path y el valor es el índice del parámetro en el path, el índice empieza a contarse después del recurso, en este caso sería

demo/api/v1/news/0/1/2/3/..../N

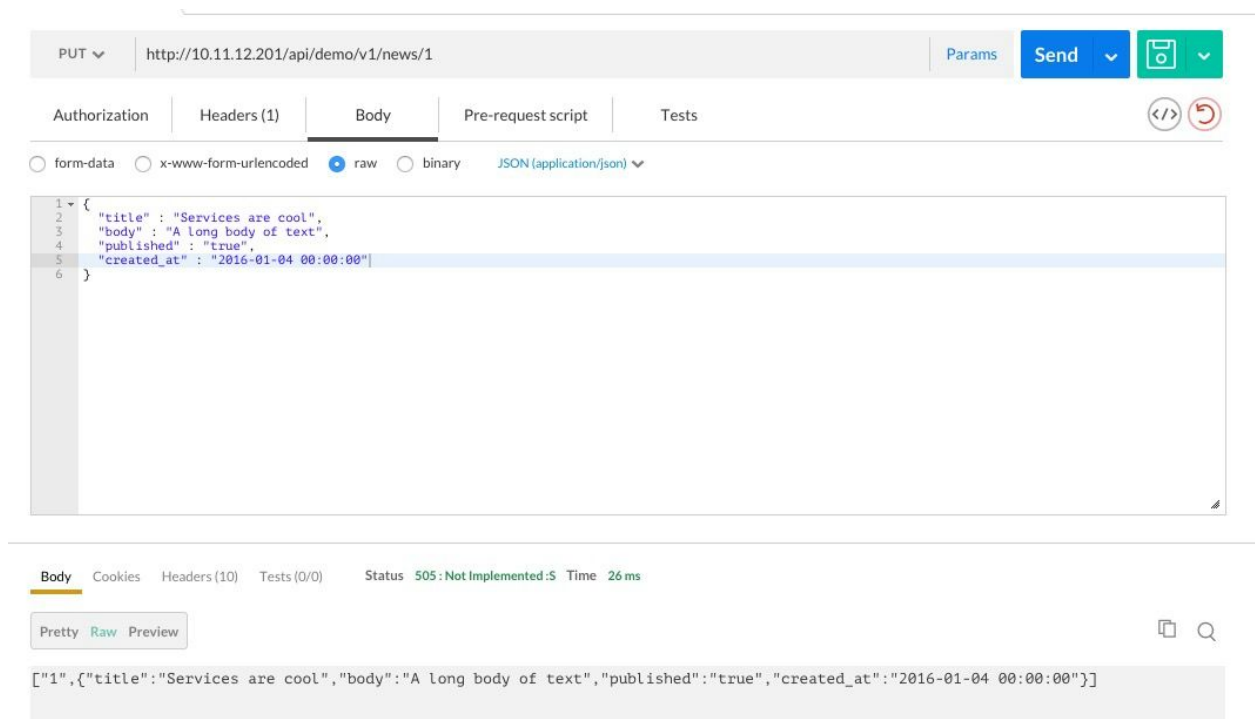
Depende de cuantos parámetros queramos pasar por path.

Si lo piensan, es parecido a `arg()`, solo que el índice no comienza en el primer elemento del path sino que comienza después del resource, recuerden que los prefijos pueden variar dependiendo de la configuración, este concepto de service se abstrae específicamente al recurso.

El callback nos quedaría así

```
function _demo_services_news_update($nid, $data) {  
  return services_error('Not Implemented', 501, array(  
    $nid, $data  
  ));  
}
```

En postman podemos probarlo



Delete

Para hacer deletes debemos usar el verbo de http delete, la entrada en operations será de la siguiente forma

```
'delete' => array(
  'help' => 'Delete a Single News',
  'callback' => '_demo_services_news_delete',
  'access callback' => '_demo_services_access_callback',
  'args' => array(
    array(
      'name' => 'nid',
      'type' => 'int',
      'optional' => FALSE,
      'source' => array('path' => 0),
    ),
  )
),
```

Sencillo de entender por lo visto previamente. El callback también obvio a estas alturas

```
function _demo_services_news_delete($nid) {
  return services_error('Not Implemented', 501, array($nid));
}
```

Un solo parámetro dado que tenemos un solo argumento que viene por el path

Index

Index nos permite traer colecciones del recurso, es ideal para armar listados, tablas, etc. Para usarlo debemos utilizar el verbo GET el path sería así

GET demo/api/v1/news

Esto debería traernos una colección de news. Ahora que pasaría si tenemos 100.000 entradas en la base de datos para news, no conviene que devuelva todo de una sola vez dado el tiempo de transferencia de tanta información no ayudaría. Es aquí donde debemos comenzar a pensar en parámetros adicionales para manejar lo que se devolverá.

Lo primero que se viene a la mente es parámetros para manejar un paginador y traer resultados por lotes, otro puede ser filtros de búsqueda por título, por fecha, etc.

Al ser una petición GET la misma no tiene payload, solo podemos meter estos parámetros por el path o por el query string. El path no debería usarse para estos tipos de parámetros dado que no son representativos al recurso, son parámetros de control, por ende conviene usarlos en el query string.

Para reflejar el ejemplo, implementaremos parámetros para paginar los resultados. La entrada en operations nos quedaría así

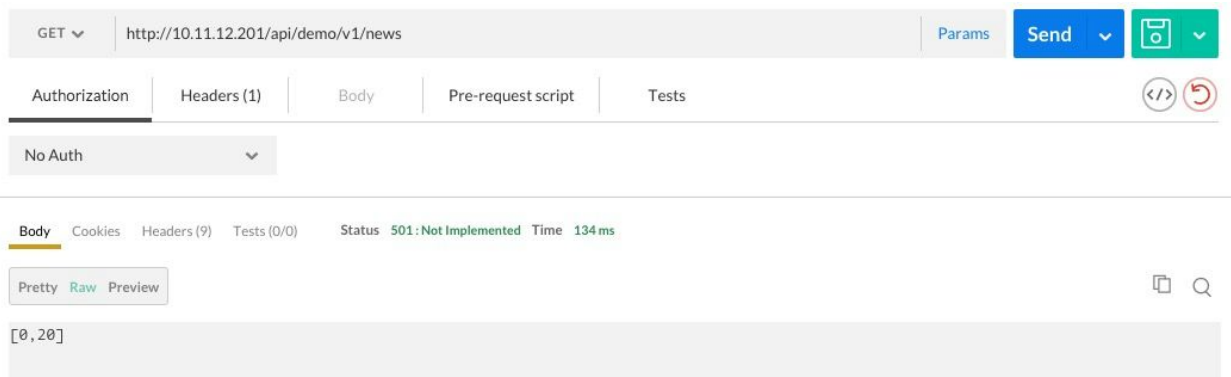
```
'index' => array(
  'help' => 'Get all News',
  'callback' => '_demo_services_news_index',
  'access callback' => '_demo_services_access_callback',
  'args' => array(
    array(
      'name' => 'page',
      'type' => 'int',
      'optional' => TRUE,
      'source' => array('param' => 'page'),
      'default value' => 0,
    ),
    array(
      'name' => 'limit',
      'type' => 'int',
      'optional' => TRUE,
      'source' => array('param' => 'limit'),
      'default value' => 20,
    )
  )
),
),
```

Lo único no visto hasta este punto es cómo levantamos los parámetros del query string, como puede verse lo hacemos por medio de un array donde la key es param y el value es el nombre de la variable en el query string. Tampoco vimos como hacer cuando los parámetros son opcionales, en este caso a lo mejor queremos la primer página, no es necesario que el cliente indique los parámetros para la página inicial, por ende podemos especificar un default value.

Con un callback similar al que veníamos haciendo para index tendríamos algo así

```
function _demo_services_news_index($page, $limit) {
  return services_error('Not Implemented', 501, array($page, $limit));
}
```

Probando con Postman



Retrieve

Nos permite obtener un recurso particular mediante el ID, a este punto si entendieron las operaciones anteriores estas les va a parecer super sencilla.

Para usar retrieve debemos usar GET, el path sería demo/api/v1/news/ID.

La entrada en operations será

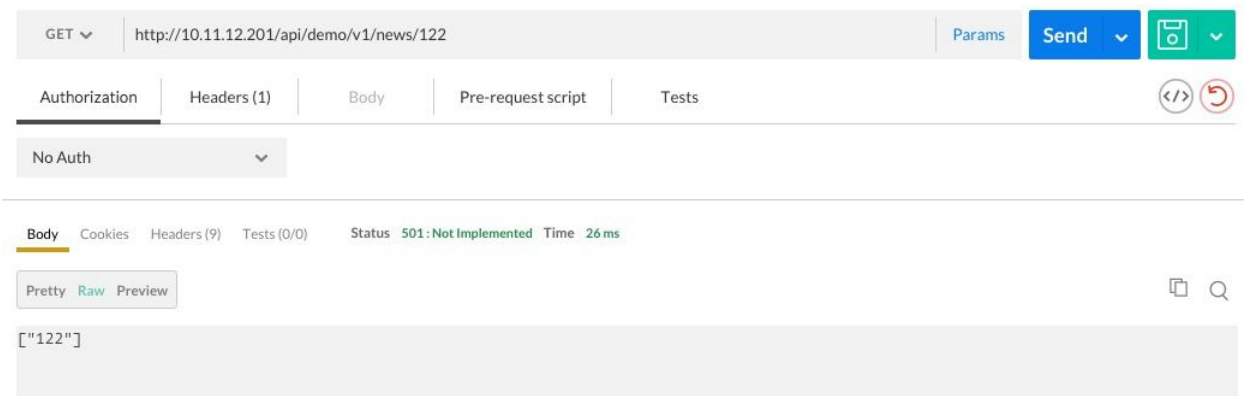
```
'retrieve' => array(
  'help' => 'Get a Single News',
  'access callback' => '_demo_services_access_callback',
  'args' => array(
    array(
      'name' => 'nid',
      'type' => 'int',
      'optional' => FALSE,
      'source' => array('path' => 0),
    )
  )
)
```

Leemos el ID del path con source path = 0, el type es int dado que será un nid y en drupal los ids son numéricos por defecto.

El callback idem a los anteriores

```
function _demo_services_news_retrieve($nid) {
  return services_error('Not Implemented', 501, array($nid));
}
```

En Postman



NOTA : Todo lo escrito hasta el momento puede encontrarse en [Github](#)

Creando Actions

Las acciones son para realizar procesos y lógica particular sobre varios, o todos, los recursos individuales.

Un ejemplo que planteo para nuestro escenario es traer las noticias más relevante, para ello usaremos el criterio de drupal basado en promote to the front and sticky at the top, para considerar la relevantes.

En la entrada del hook resources tendremos una entrada actions con cada una de las acciones a realizar, estas siguen las mismas reglas que en operations. Para nuestro caso será

```
'actions' => array(
  'most-relevant' => array(
    'help' => 'Get Most Relevant News',
    'callback' => '_demo_services_actions_most_relevant',
    'access callback' => '_demo_services_access_callback',
    'args' => array()
  )
),
```

No tiene ningún argumento.

Dado que este es una nueva acción en nuestro recurso tendremos que limpiar las caches e ir a el administrador de recursos del servicio en la UI del admin para habilitarlo, caso contrario no seremos capaces de utilizarlo.

El callback de esta acción puede ser algo así

```
function _demo_services_actions_most_relevant() {
  $select = db_select('node', 'n')
    ->fields('n', array('nid'))
    ->condition('n.promote', 1)
    ->condition('n.status', 1)
    ->condition('n.type', 'news')
    ->orderBy('n.sticky', 'DESC')
    ->orderBy('n.created', 'DESC')
    ->range(0, 5);

  $nids = $select->execute()->fetchAllKeyed(0, 0 );

  $nodes = node_load_multiple($nids);

  $nodes = _demo_services_reindex_for_js($nodes);

  return $nodes;
}
```

Noten la función **_demo_services_reindex_for_js**, entity load indexa el array de resultado usando el id de la entidad (nido, nid, tic, etc). Si nosotros exportamos como json ese array cuando sea parseado por javascript usará esos ids como nombres de atributos del objeto, por ende perderá el orden de las entidades (nodos en este caso) el cual es importante, por ello escribí esta sencilla función que realiza esa reindexación.

```
function _demo_services_reindex_for_js($nodes) {  
  $clean_nodes = array();  
  foreach ($nodes as $node) {  
    $clean_nodes[] = $node;  
  }  
  
  return $clean_nodes;  
}
```

Ténganse en cuenta que puede ahorrarle unos dolores de cabeza.

Para invocar a la acción usaremos POST y el path será así `api/demo/v1/news/most-relevant`

El resultado será un array de nodes (objetos).

Creando Targeted Actions

Las targeted actions son cosas que podemos hacer sobre una instancia en particular de un recurso, por ejemplo podemos trackear likes sobre una noticia.

La entrada en el hook sería así

```
'targeted_actions' => array(  
  'like' => array(  
    'help' => 'Set a Like for the News',  
    'callback' => '_demo_services_target_actions_link',  
    'access callback' => '_demo_services_access_callback',  
    'args' => array(  
      array(  
        'name' => 'nid',  
        'type' => 'int',  
        'optional' => FALSE,  
        'source' => array('path' => 0),  
      )  
    )  
  )  
,
```

Para invocar a este resource sería

```
api/demo/v1/news/1133343/like
```

El callback podría tomar el nid y guardar en una tabla el contador por cada nid de todas las veces que le dieron me gusta (no lo implementaremos en este documento).

Creando Relationships Actions

Lo último que nos queda por ver son las relationships, estas sirven para obtener instancias de recursos relacionados a una instancia particular del recurso primario. Por ejemplo, una noticia tiene comentarios, podríamos usar esto para traer los comentarios de una noticia determinada.

La entrada en el hook sería

```
'relationships' => array(
  'comments' => array(
    'help' => 'Get comments of a single news',
    'callback' => '_demo_services_relationship_comments',
    'access callback' => '_demo_services_access_callback',
    'args' => array(
      array(
        'name' => 'nid',
        'type' => 'int',
        'optional' => FALSE,
        'source' => array('path' => 0),
      )
    )
  )
)
```

Para poder acceder al recurso tendremos que utilizar GET

GET `api/demo/v1/news/99/comments`

El callback puede ser algo así como esto, basado en el resource node que viene en services por defecto


```

function _demo_services_relationship_comments($nid, $page = 0, $limit = 100) {
  if (module_exists('comment')) {
    $node = node_load($nid);

    if (empty($node) || $node->type != 'news') {
      return services_error('Not Found', 404, array($nid));
    }

    $query = db_select('comment', 'c');
    $query->innerJoin('node', 'n', 'n.nid = c.nid');
    $query->addTag('node_access');
    $query->fields('c', array('cid'))
      ->condition('c.nid', $nid);

    if ($limit) {
      $query->range($page * $limit, $limit);
    }

    $result = $query->execute()
      ->fetchAll();

    foreach ($result as $record) {
      $cids[] = $record->cid;
    }

    !empty($cids) ? $comments = comment_load_multiple($cids) : array();

    $return_comments = array();

    foreach ($comments as $comment) {
      $return_comments[] = services_field_permissions_clean('view', 'comment', $comment);
    }

    return $return_comments;
  }
  else {
    return services_error('Comments is not Enabled.', 500);
  }
}

```

El resultado será un json que tendrá un array con los comentarios.

Autenticación

Services reutiliza los conceptos de usuarios, roles y permisos que posee Drupal, no reinventa la rueda en estos aspectos, por ende deben entender muy bien como trabajan los usuarios, los roles, los permisos, como definir nuevos permisos, como validar permisos o roles. Asumimos que ustedes está familiarizado con estos conceptos si trabaja con Drupal por ende no entraremos en detalles finos.

El verdadero desafío aquí es poder autenticar el usuario a fin de que la variable global `$user` contenga los datos del usuario logueado, de este modo drupal internamente puede aplicar todas sus reglas relacionadas a permisos y roles, nosotros construiremos nuestras reglas en nuestros callbacks para implementar la seguridad para nuestra API.

En este documento presentaremos dos formas de lograr la autenticación del usuario en cada request a la API.

Una de ellas estará basada en la session de PHP, para esto no hay nada raro, si entiende cómo trabaja el sistema de sesiones de PHP basados en la cookie de session, verá que lo único que debe hacer es incluir la cookie en las requests.

La otra será basada en tokens, para esto debemos hacer unas llamadas al servidor para obtener un token y utilizaremos el token (que será un parámetro más) en cada request para autenticar quienes somos.

La forma más aceptada últimamente es la basada en tokens, otorga más confiabilidad y escalabilidad, pero veremos ambas.

Basada en Sesión

En esencia la autenticación basada en session es exactamente igual a como funciona cualquier website. Primero nos conectamos al site y este nos entrega una cookie de session, dicha cookie es tomada por PHP para levantar los datos de session en el servidor.

Lo que debemos hacer es autenticar al usuario para que drupal guarde los datos de usuario en la session del servidor así futuras requests realizadas por el cliente al server utilizando la cookie serán realizadas por el usuario que fue autenticado.

Por cada servicio, en el código, usted deberá aplicar las reglas de permisos y roles necesarios para el uso de su servicio. Ahí usted deberá explicitar mediante un access callback si el usuario autenticado puede o no realizar la actividad que expone el servicio.

Habilitar la autenticación basada en sesiones

Para habilitar debemos ir a Structure > Services, aquí en operations debe buscar el servicio que desea autenticar y editar los recursos.

Home > Administration > Structure

Services

Services are collections of methods available to remote applications. They are defined in modules, and may be accessed in a number of ways through server modules. Visit the [Services Handbook](#) for help and information.

All enabled services and methods are shown. Click on any method to view information or test.

+ Add + Import

Storage

Enabled

Search

- All -

- All -

Sort by

Order

Enabled, name

Up

Apply

Reset

NAME	STORAGE	OPERATIONS
demo	Normal	<div>Edit Resources</div>

Una vez en la pantalla de recursos hay que dar click en Edit y buscar la opcion de Session Authentication, checkearla y guardar.

Home

Edit endpoint demo

EDIT

SERVER

AUTHENTICATION

RESOURCES

EXPORT

Machine-readable name of the endpoint *

demo

The endpoint name can only consist of lowercase letters, underscores, and numbers.

Server *

REST

Select a the server that should be used to handle requests to this endpoint.

Path to endpoint *

api/demo/v1

☐ Debug mode enabled

Useful for developers. Do not enable on production environments

Authentication

☐ OAuth2 authentication

☒ Session authentication

Choose which authentication schemes that should be used with your endpoint. If no authentication method is selected all requests will be done by an anonymous user.

Save

Delete

El último paso es habilitar, en resources, ciertas actions que nos permitirán loguear al usuario, desloguearse y pedir un token CSRF.

Actions	
<input checked="" type="checkbox"/> login Login a user for a new session	Services Resource API Version 1.0
<input checked="" type="checkbox"/> logout Logout a user session	Services Resource API Version 1.0
<input checked="" type="checkbox"/> token Returns the CSRF token.	

Realizando una Autenticación

Para realizar la autenticación se debe realizar una request POST al siguiente path `api/demo/v1/user/login` (`api/demo/v1` es el prefix del ejemplo, cambiar por el suyo). El payload de la request debe ser el username y la password.

```
{
  "username": "root",
  "password": "root"
}
```

El resultado de dicha request será un json con datos como el `sessid` y el `session_name`.

```
1 {
2   "sessid": "eSDHgYIHffRUS3_R6a12o-RJMTvTjIRR1eRGBg0sgdI",
3   "session_name": "SESS79271fade897d977adce2723070b24f6",
4   "token": "ZjHnLMmPEwR0NdQuTeroza8Aa_leY4qDnuVKShdFyB4",
5   "user": {
6     "uid": "1",
7     "name": "root",
8     "mail": "agusting42mate.com",
9     "theme": "",
10    "signature": "",
11    "signature_format": null,
12    "created": "1451781258",
13    "access": "1457101258",
14    "login": 1457101321,
15    "status": "1",
16    "timezone": "Europe/Berlin",
17    "language": "",
18    "picture": null,
19    "init": "agusting42mate.com",
20    "data": false,
21    "roles": {
22      "2": "authenticated user",
23      "3": "administrator"
24    }
25  }
26 }
```

El `session_name` and `sessid` serán utilizados en las futuras request, como parte del header de la misma (Cookies) para autenticar la request.

CRSF Tokens

Una de las medidas de seguridad que tiene Services es el uso de CSRF tokens en las requests de actualización, creado o borrado. Para poder realizar este tipo de requests primero deberemos obtener un CSRF token.

Para obtener un token debemos hacer una petición a la API con post al siguiente path `api/demo/v1/user/token` lo que nos devolverá un token.

```
{
  "token": "ZjHnLMmPEwR0NdQuTeroza8Aa_leY4qDnuVKShdFyB4"
}
```


Dicho token lo usaremos para la petición de actualización, borrado o creación en el header de la misma bajo la entrada X-CSRF-Token. Debería quedar así

X-CSRF-Token: ZjHnLMmPEwR0NdQuTeroza8Aa_leY4qDnuVKShdFyB4

Una request autenticada con las cookies

Para ver cómo hacer una request basada en la autenticación intentaremos actualizar un nodo usando el service de node por defecto. Asumiendo que tenemos un nodo con el nid 53, deberemos realizar una petición PUT al path `api/demo/v1/node/53`.

El header de la petición deberá contener, el content type, la Cookie de session y el CSRF token.

Authorization	Headers (3)	Body	Pre-request script	Tests
<input checked="" type="checkbox"/>	Content-Type		application/json	
<input checked="" type="checkbox"/>	Cookie		SESS79271fade897d977adce2723070b24f6=eSDHgYIHffRUS3_R6a12o-RJM	
<input checked="" type="checkbox"/>	X-CSRF-Token		ZjHnLMmPEwR0NdQuTeroza8Aa_leY4qDnuVKShdFyB4	
<input checked="" type="checkbox"/>	Header		Value	
	Header		Value	

El payload puede ser como este

Authorization	Headers (3)	Body	Pre-request script	Tests
<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>	JSON (application/json) ▼
		<pre>1 { 2 "title": "Update" 3 }</pre>		

Y el resultado en caso de ser exitoso será algo así.

```
1 {  
2   "nid": "53",  
3   "uri": "http://192.168.1.131/api/demo/v1/node/53"  
4 }
```

Si el token no es válido tendrá como resultado

"CSRF validation failed"

Si la cookie no es válida tendrá

"Access denied for user anonymous"

NOTA : Si utiliza postman y realiza el login, postman automáticamente guardará las cookies de session y las futuras requests utilizará estas cookies internamente sin que usted lo sepa. Por esto puede que funcione sin que usted declare explícitamente el header cookie, cuando trabaje en un cliente diferente a postman deberá hacerlo.

Oauth2 (Experimental)

OAuth2 es un protocolo para autenticación basada en tokens, es la evolución de OAuth y en el mismo se implementaron bastantes funcionalidades nuevas como independencia en el formato de los tokens y soporte para diferentes tipos de clientes/arquitecturas. Al ser un protocolo genérico define ciertas reglas que deberían cumplirse para la interacción entre sistemas totalmente independientes, define interfaces, pero deja abierto el tema de cómo lo implementaremos en nuestro sistema internamente.

Para Drupal existe un módulo que se integra a services llamado `oauth2_server`, funciona bastante bien e integra gran parte de OAuth2 de una manera muy sencilla. Lamentablemente no hay mucha documentación al respecto de su integración y uso así que cuando tuve que implementarlo me arremangue e hice lo mejor que pude. No estoy seguro si es la mejor manera, pero funcionó y fue bastante eficiente su implementación.

El requerimiento que tenía para implementar OAuth2 era que funcione bajo JWT tokens. La idea era que el cliente entregue el username y password al server mediante un servicio y luego este devuelva un token JWT.

Luego de investigar el módulo `oauth2_server` puede ver que tiene soporte para JWT pero hasta donde entendí la implementación confía en claves públicas y privadas para realizar la autenticación y devuelve un token al cliente basado en las claves.

En mi escenario no podía implementar esa solución porque nunca sabíamos quién o qué sería el cliente, lo único que teníamos confiable para dar el acceso era el challenge de username y password. Por ello realicé un módulo que trabaja sobre OAuth2 server a fin de poder autenticar usuarios y darles un token con su username y password, esto es lo que les mostraré.

Aquí la explicación es 100% experimental pero lo que hice me funciona y se encuentra en producción. Ante la falta de documentación existente sobre el tema me animo a escribir al respecto por que estoy convencido que de todas maneras les servirá a alguien.

Instalar OAuth2 Server

Lo primero a realizar es instalar y configurar el módulo OAuth2 Server, la instalación es la de siempre, bajan el módulo y lo habilitan.

Una vez habilitado tendrán en Structure > OAuth2 Servers lo necesario para administrar el módulo.

El módulo OAuth2 permite levantar varios servers, cada server a su vez puede tener varios clientes y los clientes pueden tener diferentes scopes. Para no entrar en complicaciones vamos a hacerlo sencillo, Un server, Un cliente y Un Scope.

Primero crearemos un Server, vamos a Structure > OAuth2 Servers y le damos en Add Server.



Luego le damos un nombre, en este caso API, y tildaremos "Use JWT Access Tokens", vamos a habilitar el grant type JWT Bearer y Refresh tokens. Pondremos que siempre genere un refresh token y que borre el token después de usar el refresh token. Por último en Avanzado pueden setear los tiempos de vida de los tokens.

Label *
api Machine name: api
The human-readable name of this server.

SETTINGS

- ☐ Allow the implicit flow
Allows clients to receive an access token without the need for an authorization request token.
- ☐ Use OpenID Connect
Strongly recommended for login providers.
- ☒ Use JWT Access Tokens
Sends encrypted JWT access tokens that aren't stored in the database. See the [documentation](#) for more details.

Enabled grant types

- ☐ Authorization code
- ☐ Client credentials
- ☒ JWT bearer
- ☒ Refresh token
- ☐ User credentials

- ☒ Always issue a new refresh token after the existing one has been used
- ☒ Unset (delete) the refresh token after it has been used

ADVANCED SETTINGS

Access token lifetime
3600
The number of seconds the access token will be valid for.

Refresh token lifetime
1209600
The number of seconds the refresh token will be valid for. 0 for forever.

- ☒ Require exact redirect uri
Require the redirect uri to be an exact match of the client's redirect uri if not enabled, the redirect uri in the request can contain additional segments, such as a query string.

[Save server](#) [Delete server](#)

Les quedará así, el próximo paso será crear un cliente y un scope.



Para añadir entren a clients y den click a añadir cliente. Ahí deberán poner el nombre del cliente, api nuevamente, el Client ID un identificador para el cliente y la public key.

La public key se usará para encriptar y desencriptar el token, para tener una debemos poder generarla, para eso usamos el comando openssl.

→ ~ openssl rsa -in privkey.pem -pubout -out pubkey.pem

writing RSA key

→ ~ cat pubkey.pem

-----BEGIN PUBLIC KEY-----

```
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAIgH2ed+4KVc6WVHHP6A0
sQn8+k+sN/sxPIGXUn6ysVLHPc8Zlg063S9ku5HMEigfys3li2OxJDj5we05dTlr
Vv7pmY2KD5uShUZkTqsYvgfXUjlx/26ygOOLc9rhQII+djT2ZpjoNnpPlthmmrxW
DD0hA9dXBXR+sd896rYNdxNsRY3/qSZzRuz7PGyxzmkw9B3snMV0DP4OoLNCptRW
vvhHtztzTlhbPSZbedxrY0/wdRGiGbW0i6S+0Ui7QpGY2DNDfqufQAIHdr3IAIX
4l2GhB4gc3b2bmQUhZxqhrGBhmZJCsn4jl8sqSYEAP/BNkeLigrBVAjDIQ/5UZiy
6wIDAQAB
```

-----END PUBLIC KEY-----

→ ~

Asi les quedaría el form. En redirect pongan none pero no lo usaremos.

Label *
api
The human-readable name of this client.

Client ID *
api

☐ Require a client secret

Public key *
-----BEGIN PUBLIC KEY-----
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAIgH2ed+4KVc6WVHHP6A0
sQn8+k+sN/sxPIGXUn6ysVLHPc8Zlg063S9ku5HMEigfys3li2OxJDj5we05dTlr
Vv7pmY2KD5uShUZkTqsYvgfXUjlx/26ygOOLc9rhQII+djT2ZpjoNnpPlthmmrxW
DD0hA9dXBXR+sd896rYNdxNsRY3/qSZzRuz7PGyxzmkw9B3snMV0DP4OoLNCptRW
vvhHtztzTlhbPSZbedxrY0/wdRGiGbW0i6S+0Ui7QpGY2DNDfqufQAIHdr3IAIX
4l2GhB4gc3b2bmQUhZxqhrGBhmZJCsn4jl8sqSYEAP/BNkeLigrBVAjDIQ/5UZiy
6wIDAQAB
-----END PUBLIC KEY-----

Used to decode the JWT when the JWT bearer grant type is used.

Redirect URIs *
none

The absolute URIs to validate against. Enter one value per line.

☐ Automatically authorize this client
This will cause the authorization form to be skipped. Warning: Give to trusted clients only!

ADVANCED SETTINGS

☐ Override available grant types

Save client Delete client

El último paso será el scope. Diríjase a scope y creen uno usando uno, el form es sencillo.

Home » Administration » Structure » OAuth2 Servers » api

api

EDIT SCOPES CLIENTS

+ Add scope

SCOPE	OPERATIONS
api (Default)	edit delete

El último paso es darle permisos al usuario anónimo, al rol, de utilizar el oauth2 server. Vayan a People > Permissions busquen Usage OAuth2 y denle permisos.



Instalar el Módulo JWT Auth

El módulo jwt auth trabaja sobre OAuth2 Server y lo que hace es generar el token JWT basándose en el username y password.

Bajen el módulo y enciéndalo (<https://github.com/casivaagustin/jwt-auth>), paso siguiente será configurarlo; para ello vayan a **/admin/config/oauth2jwt** y ahí deberán setear la private key pero antes deben generarla. Para generala usen este comando

```
➔ ~ openssl genrsa -out privkey.pem 2048
Generating RSA private key, 2048 bit long modulus
~ cat privkey.pem
-----BEGIN RSA PRIVATE KEY-----
MIIEpAIBAAKCAQEAIgH2ed+4KVc6WVHHP6A0sQn8+k+sN/sxPIGXUn6ysVLHPc8Z
lg063S9ku5HMEigfys3li2OxJDj5we05dTlrVv7pmY2KD5uShUZkTqsYvgfXUjlx
/26ygO0Lc9rhQII+djT2ZpjoNnpPlthmmrxWDD0hA9dXBXR+sd896rYndxNsRY3/
qSZzRuz7PGyxzmkw9B3snMVoDP4OoLNCptRWvvhHtztDTlhbPSZbedxrY0/wdRGi
GbW0i6S+0Ui7QpGY2DNDfqufQAIHDr3IAIX4I2GhB4gc3b2bmquHxXqhrGBhmZJ
CsN4jl8sqSYEAP/BNkeLigrBVAjDIQ/5UZiy6wIDAQABAoIBADNceXGUM+3zaJoU
SZDHhpWlBhDEdUtnYpKcFOIUQwxIJ/IIH13/EFoRlp+qhuXeB6sucfV1y5JLOoCa
+aGO/428bOWe7aQbaKtDQ2NLRcptGQA4LDKlGbRI2QcAQ6oNq0tFZ+j5kp+Q1M3V
GCcFFa9+dVgcCN41hclSCd+dpqmJg6nuQbW90nelYJg0I05RvlwECmGmRKsUjRC
GRaPi0apLba1vG1FutiAv+i0T3IKniorTZX2aYSQ6/DjZBuqO6XiY1ZhHc7vVhfe
VI19HdCWej/D7dF4M5q6zt2eEfVn/OmezLuEeYoSsfaytqo5MK02CyB3qwpNYph2
2zM6c1ECgYEAw+AEqzbvLeYVuleVlp7xkmrykvk5cB2FS9Ifz4cWgKPwVyAksgmF
h5Lq5Ylrb7FzYRXndHUy+6QrGkFmvP7MRaFcE1u3jZwdpCFVrzCR3C39/uF3z6X
LgTBAP7SYxgonL6O8eG0psPA5x0hDMPMLgjJpUYI9M1h+IXLcaol3tMCgYEAwe32
cfjhSZEhHAA6JGroy1FgPHMAXc2YsE+dzi+5qam4/nUDoVUSeQs7OBTBA6q3LavR
g8/SjABgTWM8UX2rIJkfA7XMwzDsiRPun6FIWx8WOLsmRQzbe4GdpC9CunUGMtfk
ROlRwaJOPPRI+hhKNrcap8wmxixBAGjwMI2ZPIkCgYEAionao4KSPsN64LZri/Yh
ZG7yLRPLFUo76jmvOCgSuV+6cNUSPz5OzGoUa877Rx+Ilpq0fwEZ0zP1LHfPg6Mv
tZwANMztEWK28EyHfw8ZVIAaPnlO6KwrX2gLXnndMit3Jg39qoT6Me1Ide57Xrzp
oDGk9ZmIYiUVxGqD1c0bRfMCgYAR01W71CyhRLrOYK/reQg86BX4eHmoJaXLTwYq
VaizN66RdSAITFGOXsHqvb2gIKkFwa1YeFnaFI7FjkGVHAARgKGhLazMpEqZrwNY
GPXAEnNwc4NpBW2HRLrAxLdVfDw9oCqTSmhnrmuUPsJ6SpFHxjzk/5bw0FCJZ1js
nZdgQQKBgQCjY1bXuEA1AWH09Pszzs2PYEM95Le+WMZmnTgKBMc8bZf0fCUMHs2/
h91xH0zydQWhcYuuTFLpa8U2W9+jgB80iuUZRWnDXREhNjoA67sFyXQu+7/locy
WUH6gpmRln9rnngEPBjwq6l5aE3UgHDhXCnirapHZJgT1MoSNzrQsQ==
-----END RSA PRIVATE KEY-----
```

Una vez generada pongala en el form de config y debe guardarlo.

Algunos Hacks

Por defecto el módulo oauth2_server tiene dos cosas que me trajeron problema, la primera es que el campo token de la tabla oauth2_server_token me quedo chico, con varchar 255 no entraba el JWT token. Tuve que agrandarlo.

```
--- a/src/html/sites/all/modules/oauth2_server/oauth2_server.install
+++ b/src/html/sites/all/modules/oauth2_server/oauth2_server.install
@@ -284,8 +284,8 @@ function oauth2_server_schema() {
    ),
    'token' => array(
      'description' => 'The token.',
-    'type' => 'varchar',
-    'length' => 255,
+    'type' => 'text',
+    'length' => 'normal',
      'not null' => TRUE,
    ),
@@ -499,3 +499,21 @@ function oauth2_server_update_7107() {
  );
  db_add_field('oauth2_server_token', 'created', $spec);
}
+
+
+/**
+ * Fix for JWT tokens.
+ */
+function oauth2_server_update_7108() {
+  db_drop_index('oauth2_server_token', 'token');
+
+  $spec = array(
+    'description' => 'The token.',
+    'type' => 'text',
+    'size' => 'normal',
+    'not null' => TRUE,
+  );
+
+  db_change_field('oauth2_server_token', 'token', 'token', $spec);
+  db_add_index('oauth2_server_token', 'token', array(array('0' => 'token', '1' => '255')));
+}
\ No newline at end of file
```

También tuve que hacer este cambio porque el token nunca contaba con información del server por ende siempre fallaba, con este fix puede saltar ese problema.

```
\ No newline at end of file
diff --git a/src/html/sites/all/modules/oauth2_server/oauth2_server.module
b/src/html/sites/all/modules/oauth2_server/oauth2_server.module
index ff456bc..743274a 100755
--- a/src/html/sites/all/modules/oauth2_server/oauth2_server.module
+++ b/src/html/sites/all/modules/oauth2_server/oauth2_server.module
@@ -1040,10 +1040,12 @@ function oauth2_server_check_access($server_name, $scope = NULL) {
  }
}
```

```

// Make sure that the token we have matches our server.
- if ($token['server'] != $server->name) {
-   $response->setError(401, 'invalid_grant', 'The access token provided is invalid');
-   $response->addHttpHeaders(array('WWW-Authenticate' => sprintf('%s, realm="%s", scope="%s"', 'bearer', 'Service', $scope)));
-   return $response;
+ if (!empty($token['server'])) {
+   if ($token['server'] != $server->name) {
+     $response->setError(401, 'invalid_grant', 'The access token provided is invalid');
+     $response->addHttpHeaders(array('WWW-Authenticate' => sprintf('%s, realm="%s", scope="%s"', 'bearer', 'Service', $scope)));
+     return $response;
+   }
+ }
}

```

El otro es que la librería de oauth que usa el oauth2_server por alguna razón no genera los refresh tokens en JWT y como yo quería usar refresh tokens tuve que cambiar el código para generarlos.

```

--- a/src/html/sites/all/libraries/oauth2-server-php/src/OAuth2/GrantType/JwtBearer.php
+++ b/src/html/sites/all/libraries/oauth2-server-php/src/OAuth2/GrantType/JwtBearer.php
@@ -219,7 +219,7 @@ class JwtBearer implements GrantTypeInterface, ClientAssertionTypeInterface
 */
public function createAccessToken(AccessTokenInterface $accessToken, $client_id, $user_id, $scope)
{
-   $includeRefreshToken = false;
+   $includeRefreshToken = true;

    return $accessToken->createAccessToken($client_id, $user_id, $scope, $includeRefreshToken);
}

```

Habilitando OAuth2 en lo servicios

Hasta este punto tenemos el OAuth2 Server configurado pero los servicios todavía no lo están usando, debemos configurar para que lo usen.

Lo primero es tildar que la autenticación que usaremos será basada en OAuth2

Home

Edit endpoint demo

EDIT SERVER AUTHENTICATION RESOURCES EXPORT

Machine-readable name of the endpoint *

demo

The endpoint name can only consist of lowercase letters, underscores, and numbers.

Server *

RIBT

Select a server that should be used to handle requests to this endpoint.

Path to endpoint *

api/endpoint

☐ Debug mode enabled

Useful for developers. Do not enable on production environments.

Authentication

☒ OAuth2 authentication

☐ Session authentication

Choose which authentication schemes that should be used with your endpoint. If no authentication method is selected all requests will be done by an anonymous user.

Save Delete

Lo segundo es configurar que server de OAuth2 vamos a utilizar

Home » Administration » Structure » Services

Services

EDIT SERVER AUTHENTICATION RESOURCES EXPORT

OAUTH2 AUTHENTICATION

OAuth2 server *

api

Save

Lo tercero es por cada servicio, tildar que requiere autenticación OAuth2.

node

CRUD operations

<input checked="" type="checkbox"/> retrieve	Retrieve a node	OAuth2 authentication	<input checked="" type="checkbox"/> Require authentication	Scope	
A space-separated list of required scopes. Leave empty to ignore the check.					
<input checked="" type="checkbox"/> create	Create a node	OAuth2 authentication	<input checked="" type="checkbox"/> Require authentication	Scope	
A space-separated list of required scopes. Leave empty to ignore the check.					
<input checked="" type="checkbox"/> update	Update a node	OAuth2 authentication	<input checked="" type="checkbox"/> Require authentication	Scope	
A space-separated list of required scopes. Leave empty to ignore the check.					
<input checked="" type="checkbox"/> delete	Delete a node	OAuth2 authentication	<input checked="" type="checkbox"/> Require authentication	Scope	
A space-separated list of required scopes. Leave empty to ignore the check.					
<input checked="" type="checkbox"/> index	List all nodes	OAuth2 authentication	<input type="checkbox"/> Require authentication	Scope	
A space-separated list of required scopes. Leave empty to ignore the check.					

Luego de todo esto el servidor va a requerir el token y va a validarlo.

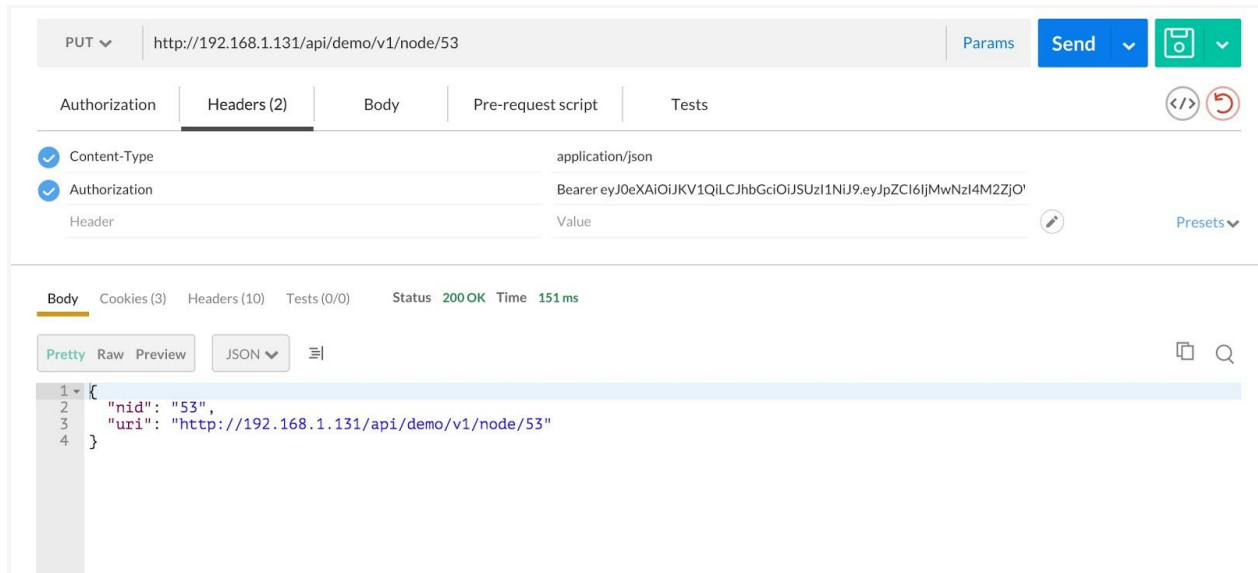
Desde el punto de vista de la programación, en drupal, sigue todo como siempre. La variable global \$user tendrá información del usuario en curso, mediante esto podremos validar roles y permisos para habilitar o denegar la acción. Muy transparente y elegante la implementación.

Para obtener un token tendremos que usar una petición POST al path `/oauth2/auth`. El content type será `application/json` y el payload tendrá que tener

El resultado, si fue todo bien, será

El access token lo tendremos que guardar para futuras request, por ese token el server sabrá que nosotros somos el usuario root. Este token tiene una duración de 3600 segundos, luego no será más válido, para esto entra en juego el refresh token.

Un ejemplo



Obteniendo un nuevo Token con un refresh token

Para usar el refresh token debemos usar el recurso `oauth2/refresh` con el método POST. En el payload debemos tener

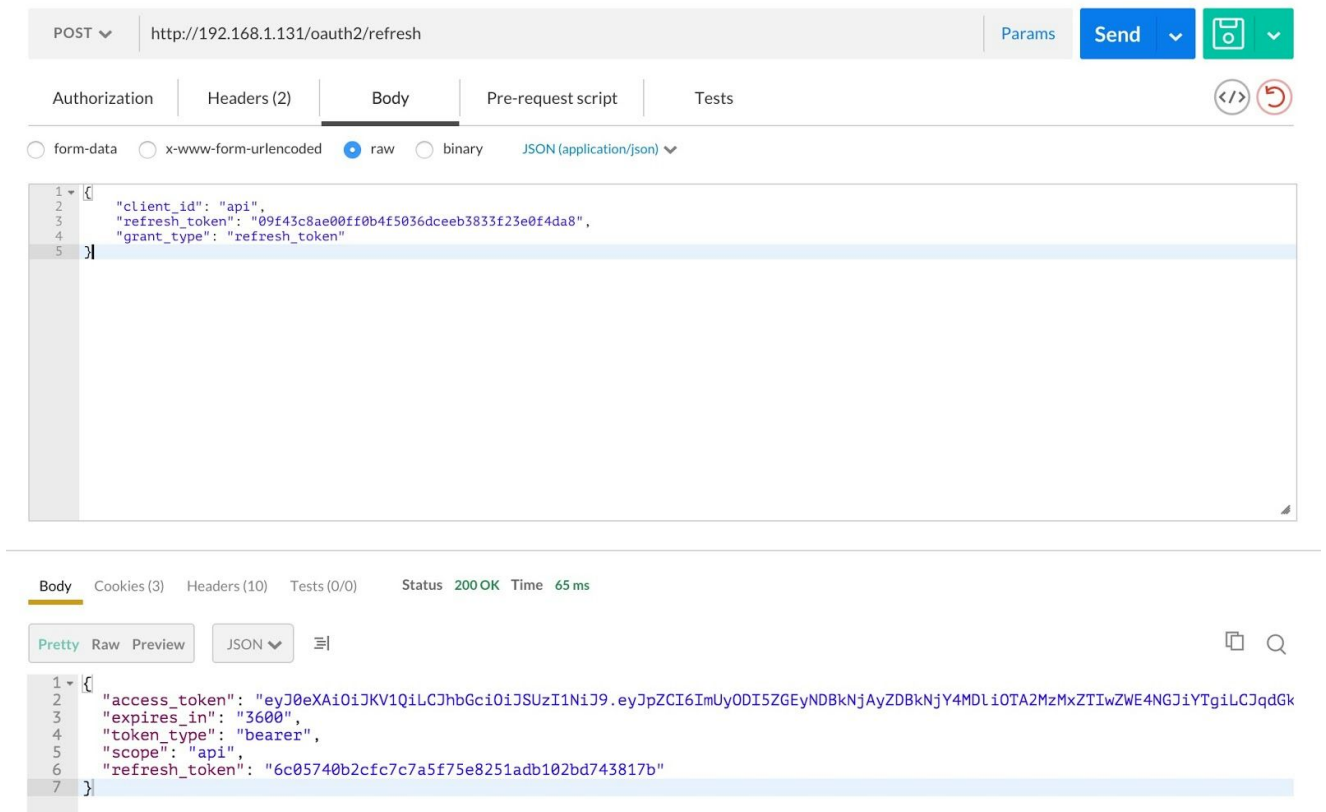
```
{
  "client_id": "api",
  "refresh_token": "09f43c8ae00ff0b4f5036dceeb3833f23e0f4da8",
  "grant_type": "refresh_token"
}
```

La respuesta será como si hubiésemos pedido un token, tendremos un nuevo token y un nuevo refresh token.

```
{
  "access_token":
    "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJpZCI6ImUyODI5ZGEyNDBkNjAyZDBkNjY4MDIiOTA2MzMxZWlWZWWE4NGJiTGtLCjQdGkiOiJIMjgyOWRhMjQwZDYyZWVmMmQwZDZDOA5YjkwNjMzMWUyMGVhODRiYmE4IiwiaXNzIjoiaHR0cDpcL1wwMTkyLjE2OC4xLjEzMVvviwiYXVkljoiYXBpIiwic3ViOiJMSlsmV4cCl6MTQ1NzEyNDcxMCwiaWF0IjoxNDU3MTIxMTEwLCJ0b2tlbi90eXBlljoiYmVhcmVyIiwic2NvcGUiOiJhcGkifQ.E3MtYGWqZ8tkLillAPX7KmvPGDU_b1lvYh7RNng803I8JVx7_Q1FS1_0JL8xlqC5irX1GI5yvvlupV6vpB44ihR-AjewbX-n9shBOlyqwqOUyhUpmY2znI8kd rPBLk_iR7RbxyDfeaEwAHvLxCrjiAKClmOXog03AWXtykokBN2ZE99NwsEEw1cwFQKKWqG_6BxAqRtBp5kuuJcdGWtiL2fve1IWSRkDWYrOY OegwtgqOejw2QVaQHzyMfURTUSeCYsMMN1iQVwuyKdp-looGd_WvLOlh7Yts5mM9KnjXoeC8PhIJf9SI436q6fyJXpLCBBffkGrc7tI4Bda",
  "expires_in": "3600",
  "token_type": "bearer",
  "scope": "api",
  "refresh_token": "6c05740b2cfc7c7a5f75e8251adb102bd743817b"
}
```

El refresh token utilizado no servirá más luego de haber sido utilizado.

Ejemplo



Referencias

Algunos links que me ayudaron a aprender services y a escribir este artículo

<https://www.drupal.org/documentation/modules/services>

<https://www.drupal.org/node/736522> - Service 3x

<https://www.drupal.org/node/783460> - Creating a resource for Services

<https://www.drupal.org/node/2424977> - TOKEN SESSION

<https://www.drupal.org/node/113697> - Service tutorial examples

<https://www.drupal.org/node/1975670> - Tools for testing the server

<https://www.drupal.org/node/783254> - Working with REST

Por último, todos los ejemplos y el código que use en este documento puede encontrarse en [github](#)

<https://github.com/casivaagustin/drupal-services>

JWT Module

<https://github.com/casivaagustin/jwt-auth>