

Desarrollo de APIs



Miguel Angel Alvarez
Eduard Tomàs

 desarrolloweb.com

desarrolloweb.com/manuales/manual-desarrollo-api.html

Introducción: Manual de desarrollo de APIs

En este manual pretendemos ofrecer información que tiene que ver con el desarrollo de APIs, una de las claves del desarrollo moderno de aplicaciones, para la web o para dispositivos.

Desarrollar con base a un API tiene muchas ventajas, como explicaremos aquí. Entre ellas nos permite separar la parte del backend y el frontend de las aplicaciones, ofrecer una experiencia de usuario más avanzada y facilita la escalabilidad del proyecto a medida que va creciendo.

El Manual de desarrollo de APIs no pretende ser una descripción de procesos de implementación de APIs, pues se necesitarían cientos o miles de páginas para abarcar todas las posibles situaciones en las que puedes encontrarte, así como decenas de lenguajes, bases de datos, frameworks, etc.

En cambio, nuestro objetivo es reunir artículos que sirvan de guía para las personas que pretenden iniciarse en el desarrollo de APIs, sea cual sea el enfoque o stack de tecnologías a utilizar. O de lectura para las personas que se interesan por estos temas y desean saber cuál es el funcionamiento de las aplicaciones modernas.

Encuentras este manual online en:

<http://desarrolloweb.com/manuales/manual-desarrollo-api.html>

Autores del manual

Las siguientes personas han participado como autores escribiendo artículos de este manual.

Miguel Angel Alvarez

Miguel es fundador de DesarrolloWeb.com y la plataforma de formación online EscuelaIT. Comenzó en el mundo del desarrollo web en el año 1997, transformando su hobby en su trabajo.



Eduard Tomàs

Apasionado de la informática, los videojuegos, rol y... la cerveza. Key Consultant en Pasiona y MVP en #aspnet

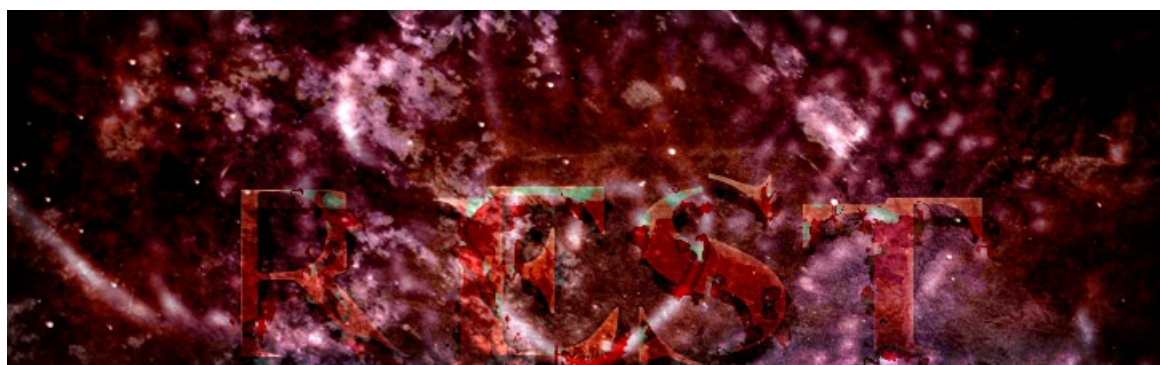


Qué es REST

Puede que te suene por las API REST. Te explicamos por qué se han popularizado tanto en el mundo de Internet y qué características tienen estos sistemas.

Actualmente, las API REST están realmente de moda: parece que cualquier aplicación deba proporcionar su “API REST”. Pero...¿qué significa realmente una API REST?

REST deriva de "REpresentational State Transfer", que traducido vendría a ser “transferencia de representación de estado”, lo que tampoco aclara mucho, pero contiene la clave de lo que significa. Porque la clave de REST es que un servicio REST no tiene estado (es *stateless*), lo que quiere decir que, entre dos llamadas cualesquiera, el servicio pierde todos sus datos. Esto es, que no se puede llamar a un servicio REST y pasarle unos datos (p. ej. un usuario y una contraseña) y esperar que “nos recuerde” en la siguiente petición. De ahí el nombre: el estado lo mantiene el cliente y por lo tanto es el cliente quien debe pasar el estado en cada llamada. Si quiero que un servicio REST me recuerde, debo pasarle quien soy en cada llamada. Eso puede ser un usuario y una contraseña, un *token* o cualquier otro tipo de credenciales, pero debo pasarlas en cada llamada. Y lo mismo aplica para el resto de información.



El no tener estado es una desventaja clara: tener que pasar el estado en cada llamada es, como mínimo, tedioso, pero la contrapartida es clara: escalabilidad. Para mantener un estado se requiere algún sitio (generalmente memoria) donde guardar todos los estados de todos los clientes. A más clientes, más memoria, hasta que al final podemos llegar a no poder admitir más clientes, no por falta de CPU, sino de memoria. Es algo parecido a lo que ocurre con la web tradicional (que también es *stateless*). Sea cual sea la tecnología con la que desarrolles para web, seguro que conoces que puedes crear lo que se llama una “sesión”. Los datos de la sesión se mantienen entre peticiones y son por cada usuario. El clásico ejemplo de sesión puede ser un carrito de la compra, entre las distintas peticiones web, la información del carrito de compra se mantiene (¡joj! hay otras alternativas para implementar carritos de compra).

Por norma general, la sesión se almacena en la memoria RAM del servidor, por lo que es fácil quedarnos sin memoria si introducimos demasiados datos en sesión (o tenemos muchos usuarios simultáneos). Por supuesto, podemos utilizar varios servidores, pero como bien saben los desarrolladores web, mover la sesión contenida en memoria entre servidores es generalmente imposible o altamente ineficiente, lo que

penaliza el balanceo de carga, lo que a su vez redundaría en menor escalabilidad y menor tolerancia a fallos. Hay soluciones intermedias, tales como guardar la sesión en base de datos, pero su impacto en el rendimiento suele ser muy grande. De hecho, el consejo principal para desarrollar aplicaciones web altamente escalables es: no usar la sesión.

Así pues, tenemos que el ser stateless es la característica principal de REST, aunque por supuesto no la única. Así, cualquier servicio REST (si quiere ser merecedor de ese nombre) debería no tener estado, pero no cualquier servicio sin estado es REST. Hay otros factores, pero vamos a destacar el que los ingleses llaman “uniform interface” y es lo que diferencia un servicio web clásico (orientado a RPC) de un servicio REST.

SOAP y REST

SOAP es el acrónimo de “Simple Object Access Protocol” y es el protocolo que se oculta tras la tecnología que comúnmente denominamos “Web Services” o servicios web. SOAP es un protocolo extraordinariamente complejo pensado para dar soluciones a casi cualquier necesidad en lo que a comunicaciones se refiere, incluyendo aspectos avanzados de seguridad, transaccionalidad, mensajería asegurada y demás. Cuando salió SOAP se vivió una época dorada de los servicios web. Aunque las primeras implementaciones eran lo que se llamaban WS1.0 y no soportaban casi ningún escenario avanzado, todo el mundo pagaba el precio de usar SOAP, ya que parecía claro que era el estándar que dominaría el futuro. Con el tiempo salieron las especificaciones WS-* que daban soluciones avanzadas, pero a la vez que crecían las capacidades de SOAP, crecía su complejidad. Al final, los servicios web SOAP terminan siendo un monstruo con muchas capacidades pero que en la mayoría de los casos no necesitamos.

Por su parte REST es simple. REST no quiere dar soluciones para todo y por lo tanto no pagamos con una demasiada complejidad una potencia que quizá no vamos a necesitar.

Uniform interface

Una diferencia fundamental entre un servicio web clásico (SOAP) y un servicio REST es que el primero está orientado a RPC, es decir, a invocar métodos sobre un servicio remoto, mientras que el segundo está orientado a recursos. Es decir, operamos sobre recursos, no sobre servicios.

En una API REST la idea de “servicio” como tal desaparece. Lo que tenemos son recursos, accesibles por identificadores (URIs). Sobre esos recursos podemos realizar acciones, generalmente diferenciadas a través de verbos HTTP distintos.

Así, en un servicio web clásico (SOAP) tendríamos un servicio llamado BeerService que tendría un método llamado GetAll que me devolvería todas las cervezas. La idea, independientemente de la tecnología usada para consumir el servicio web, es que se llama a un método (GetAll) de un servicio remoto (BeerService). Del mismo modo para obtener una cerveza en concreto llamaríamos al método GetById() pasándole el id de la cerveza. De ahí que se diga que están orientados a RPC (Remote Procedure Call – Llamada a método remoto).

Por su parte en un servicio REST la propia idea de servicio se desvanece. En su lugar nos queda la idea de un “recurso”, llamémosle “Colección de cervezas” que tiene una URI que lo identifica, p. ej. /Beers. Así, si invoco dicha URI debo obtener una representación de dicho recurso, es decir, debo obtener el listado de todas las cervezas.

Para obtener datos de una cerveza, habrá otro recurso (cerveza) con una URI asociada. Además, entre los recursos hay relaciones: está claro que una cerveza forma parte de la “Colección de cervezas” así que parece lógico que a partir de la colección entera (/Beers) pueda acceder a uno de sus elementos con una URI tipo /Beers/123, siendo "123" el ID de la cerveza.

Volvamos a nuestro servicio SOAP: para dar de alta una cerveza llamaríamos a un método "AddBeer" de nuestro "BeerService", y le pasaríamos la cerveza a añadir. En cambio, en nuestra API REST añadir una cerveza consiste en usar la URI de dicha cerveza, (p. ej. /Beers/456 si estamos añadiendo la cerveza con ID 456) usando POST o PUT como verbo HTTP y colocando los datos de la cerveza en el cuerpo de la petición. La URI para acceder a la cerveza con ID 456 es siempre /Beers/456 y es el verbo HTTP (GET, POST, PUT, DELETE,...) el que indica cual es la operación que deseamos hacer con esa cerveza.

Este artículo es obra de *Eduard Tomàs*

Fue publicado por primera vez en 25/04/2014

Disponible online en <http://desarrolloweb.com/articulos/que-es-rest-caracteristicas-sistemas.html>

Ventajas e inconvenientes de API REST para el desarrollo

Estudio sobre las ventajas y desventajas del desarrollo de sitios web y aplicaciones de todo tipo usando una **API REST** como modelo de comunicación entre el frontend y el backend.

Estamos ante una corriente que se está extendiendo mucho en el ámbito de la tecnología actual, REST, que soporta un nuevo modelo de desarrollo de software diferente, capaz de aportar varias ventajas en el ciclo de vida de las aplicaciones.

En este artículo os vamos a explicar con detalle a qué nos referimos con el **desarrollo basado en API REST** y vamos a estudiar las ventajas e inconvenientes de este modelo. Revisaremos conceptos y técnicas de desarrollo modernos (estamos en el año 2014) que todo el mundo debería al menos conocer.



Introducir REST

Inevitablemente debemos comenzar por explicar lo que es REST. Esa palabra que todos hemos leído, oído, visto y entendido solo quizás a medias. Y la verdad es que este punto será fácil porque tenemos un artículo anterior que lo explica de manera estupenda, así que me limitaré a enlazarlo: [Qué es REST](#).

Arquitectura de un desarrollo basado en API REST

Una vez que sabemos exactamente las características de REST nos podremos plantear **cómo es una aplicación basada en REST**. Realmente podría haber muchas respuestas, puesto que podemos usar APIs para hacer multitud de sitios web. Por ejemplo, podríamos usar el API REST de Twitter para crear un servicio basada en esa red o el API de Youtube para crear un sitio web que muestra vídeos de distintas maneras. Pero no es eso a lo que me refiero en este artículo.

En este caso quiero proponer un modelo que aprovecha perfectamente sus posibilidades de REST donde tú o tu equipo es el que desarrolla tu propia API y donde tú eres el consumidor de ella.

Nota: Para aclararnos, decir que comúnmente hasta ahora siempre teníamos un cliente (navegador) y un servidor. El cliente hacía una solicitud de una URL y recibía una página HTML, con sus estilos, etc. Esa página tenía un HTML que renderizado en el navegador nos producía la presentación que deseábamos. Esa situación nos ha valido desde mucho tiempo atrás y sigue sirviendo hoy en día, pero ahora existen otras soluciones que nos aportan otras posibilidades. Tienes que mirar todo esto bajo el prisma del momento tecnológico actual. El objetivo sigue siendo hacer aplicaciones, sin embargo, en muchos casos no vamos a pensar solo en hacer aplicaciones web, sino también apps para móviles o cualquier otro tipo de sistema o dispositivo.

Pues bien, ahora en un desarrollo basado en API REST tendremos un esquema como el que puedes ver en esta imagen:



El cliente inicia siempre una solicitud, pero ahora ésta no la produce el usuario en "bruto" hacia el servidor, sino que pasa por Javascript. Nuestro lenguaje del lado del cliente es el que solicitará al servidor un recurso al servidor.

El servidor y el cliente web se comunican en un formato de intercambio de información como puede ser JSON, aunque podría ser otro lenguaje como XML.

Lo importante es que el cliente no recibe HTML para renderizar, sino simplemente los datos que se han generado como respuesta. Es decir, el servidor no escribe HTML, sino únicamente genera los datos para enviarlos al cliente.

¿Qué tecnologías de servidor usas para implementar REST?

Es independiente, podrás tener un servidor que trabaja con PHP, Java, Python, NodeJS o lo que prefieras, o te imponga el proyecto. Tanto el lenguaje como la base de datos son importantes, porque nos sirven para procesar la solicitud y generar la respuesta, pero no importa cómo lo hagas en el servidor. Simplemente que la respuesta la entregues en ese lenguaje de intercambio de información que estés usando, generalmente JSON.

¿Qué librerías JS?

Realmente tampoco estamos obligados a usar ninguna. Lo que estará claro es que si estás produciendo una web en el cliente usarás Javascript, pero la librería que tengas para facilitarte las cosas es indiferente.

Actualmente, por posibilidades, por comunidad y por facilidad de desarrollo los profesionales se están decantando por AngularJS, pero realmente usarás aquella con la que te sientas a gusto.

¿Alguna librería del lado del servidor?

Debe haber cientos, también dependiendo del lenguaje que estés usando en el servidor. Si es con PHP puedes usar Laravel, Symfony o microframeworks como Slim. Si estás usando NodeJS probablemente encuentres útil Express o Sails.js. Son solo por nombrar algunas, puesto que en este caso todo depende mucho de la tecnología que estés usando en el servidor.

¿Qué tipo de aplicaciones son adecuadas para esta arquitectura?

Esta es una buena pregunta, porque realmente el desarrollo basado en API REST no es para cualquier tipo de proyecto. Si estás haciendo un sitio web y planeas que tu sitio va a ser siempre eso "una web", quizás sea innecesario desarrollar en base a un API. Por ejemplo, un sitio centrado en contenido, como un blog o una página de una empresa donde ofrece sus productos, servicios y modos de contacto, no tiene sentido desarrollarla en base a un API. Ahora, si lo que estás es ofreciendo un servicio online o estás realizando una aplicación web de gestión, entonces encaja perfectamente la filosofía de REST.

En general si piensas que tu sistema en el futuro podría ser accedido no solo desde una página web, sino también desde una App para móvil o desde una aplicación de otro tipo, las ventajas de REST serán especialmente útiles. En resumen, si sospechas que los datos o servicios que estás ofreciendo en un futuro puedan llegar a ser consultados desde otros sistemas, te interesa usar REST. Incluso hay profesionales que piensan que, aunque de momento no estés pensando en que esos datos o servicios puedan llegar a ser consultados desde otros sistemas ajenos a tu web, merece la pena usar REST porque es la solución que mayor escalabilidad te va a aportar. Todo esto lo verás mejor enseguida que tratemos las ventajas / inconvenientes.

Ventajas del desarrollo basado en REST

Enumeramos una serie de ventajas que encontrarás en un desarrollo basado en API REST.

1. separación cliente/servidor

Al ser sistemas independientes (solo se comunican con un lenguaje de intercambio como JSON) puedes desarrollarlos proyectos autónomos, equipos autónomos. Al cliente le da igual cómo está hecha la API y al servidor le da igual qué vas a hacer con los datos que te ha proporcionado.

Si necesitas evolucionar/refactorizar uno de los dos, back o front, se puede hacer de manera separada, siempre que se mantenga la interfaz del API.

Puedes hacer varios front con un único backend (frontal no tiene que ser solo web, puede ser un app para Android otro para un App iOS, etc.)

Nota: Ojo con esto. Si son independientes, PERO deben tener mucha comunicación ambos equipos para saber qué hacer y cómo hacerlo; desgraciadamente esta falta de comunicación es un error clásico y muy común que lleva siempre a pérdidas de tiempo y tener que volver a repetir el trabajo.

En este sentido otra ventaja importantísima es la posibilidad de crear tantos frontales como necesites con la misma API. Igual comienzas desarrollando una web, pero con la misma API podrás desarrollar también una aplicación para iOS, Android y si mañana gustas para Windows 8, Windows Phone, el próximo Windows 10, Blackberry, FirefoxOS y tantos como vengan en el futuro a encajar con tu estrategia de negocio.



2. Independencia de tecnologías / lenguajes

Puedes desarrollar en cualquier tipo de tecnología o lenguaje con la que te sientas a gusto o con la que puedas acortar tus tiempos de desarrollo, o encaje con la filosofía o necesidades de tu proyecto. Es indiferente que en el futuro cambies totalmente las tecnologías con las que está implementado tu API REST, siempre y cuando respetes "el contrato", osea, que sigas teniendo las mismas operaciones en el API y hagan las mismas cosas que se supone que deben hacer.

Nota: Como nos sugieren en los comentarios de los artículos, cabe insistir en que JSON no se consume solamente desde Javascript, sino que lo podemos consumir desde cualquier lenguaje / tecnología, desde PHP para desarrollo web del lado del servidor, así como Python, .NET, Objective C o Swift para aplicaciones iOS, Java para aplicaciones Android. Esto son solo unos ejemplos porque hay cientos de lenguajes donde podremos trabajar con JSON.

3. Fiabilidad, escalabilidad, flexibilidad

Al final solo te tienes que preocupar que el nexo cliente / servidor esté correcto. Puedes hacer cambios en tu servidor, lenguajes, bases de datos, etc. y mientras devuelvas los datos que toca todo irá correctamente.

Escalabilidad porque puedes crecer todo lo que necesites en cada momento. Tu API puede responder a otros tipos de operaciones o puede versionarse tanto como desees. También tu programación del lado del cliente puede crecer todo lo necesario con el tiempo, incluso como decíamos, crear otros frontales, no solo web, también de Apps para cualquier dispositivo.

A la hora de ejecutar tu aplicación también tienes una flexibilidad mucho mayor. Las páginas del front las puedes enviar desde unos servidores y las API pueden estar alojadas en servidores independientes, tantos como necesites. Por las características de REST (principalmente no guardar estado) es indiferente qué servidor atienda cada solicitud, pues es el propio cliente el que tiene que mandar el estado al servidor, así que el balanceo de carga es mucho más simple que en aplicaciones tradicionales donde el front está mezclado con el back. También tienes la aproximación de las "micro APIs", en las que puedes dividir los procesos en diferentes servidores que atiendan a diferentes tipos de operaciones del API.

4. Experiencia de usuario

Aunque eso depende más de cómo está hecha la parte del cliente, teóricamente el desarrollo de sitios web basados en un API puede dar mejor desempeño que uno tradicional. Cuando haces una solicitud al servidor lo que tienes como respuesta son datos planos, que requieren tiempos de transferencia menores que si esos mismos datos los recibieras mezclados con el HTML/CSS de la presentación. En este tipo de aplicaciones web no necesitas recargar la página, aunque esto no es una ventaja específica del desarrollo basado en REST, sino del uso de Ajax en general, con el que podemos conseguir aplicaciones web que se asemejan más a aplicaciones de escritorio

5. REST requiere menos recursos del servidor

Esto no es necesariamente cierto aunque en muchos casos sí se pueda deducir. Hay muchas opiniones al rededor de REST. Nosotros basamos esa afirmación en estos motivos:

- No mantener el estado, no requiere memoria, se pueden atender más peticiones
- No requiere escribir el HTML, por lo tanto tienes menos procesamiento en el servidor

Desventajas del desarrollo basado en REST

Tenemos que cambiar el modo de pensar, por lo que los equipos de trabajo se tienen que reciclar. Todos deben de romper con esa idea de que todo está en un servidor y que todo está ahí para desarrollar tus necesidades. En un esquema REST puedes tener varios servidores donde unos no saben que los otros existen. No sabes si un usuario ha iniciado sesión en un servidor y si le has enviado ciertos datos. Tampoco sabes realmente en qué servidor puede caer una solicitud. Romper con ese esquema de todo está en el mismo servidor, tengo todas mis clases y todas las partes de mi aplicación centralizadas es uno de los puntos que pueden resultar más complicados.

En las API REST no mantenemos estado y eso hace que tengas que montar una infraestructura propia para poder conservar el conjunto de la aplicación. Generalmente mandarás un token que indique quien eres al servidor y qué has realizado ya en tu aplicación.

En un principio puedes incurrir en mayores tiempos de desarrollo, porque tienes que montar todo el sistema de tu API. Sin embargo, a futuro y sobretodo cuando ese API la vas a consumir desde otros sistemas, puedes recuperar el tiempo adicional dedicado y ganar mucha velocidad, porque el corazón de los nuevos sistemas (el API) ya lo tienes desarrollado. En cualquier caso, gracias a REST y la separación del código también tendrás menor mantenimiento.

Puede producirse en determinadas circunstancias mayor rigidez en el desarrollo, sobre todo al ser dos

proyectos independientes, tu back basada en REST y el/los frontales, pueden surgir situaciones de des-sincronización. Por ejemplo desde el cliente detectas que necesitas nuevas cosas del API y los encargados de crearlas en la parte del server pueden estar a otro ritmo y tardar en desarrollarlas.

Requiere más conocimientos, te obliga a salir de la zona de confort a la que quizás estemos. Aparte de conocer tus lenguajes, bases de datos, librerías, necesitarás aprender acerca del protocolo HTTP.

Conclusión

En definitiva observarás que las ventajas e inconvenientes del desarrolla basado en API REST están muy interligados. Lo que puede comenzar como una ventaja puede suponer una desventaja en otro punto de tu aplicación. Sin embargo, como hemos dicho, existen muchas situaciones en las que desarrollar basados en un API nos puede aportar un gran adelanto.

Esperamos que este artículo te haya parecido útil.

Nota sobre la autoría: Este artículo es el resultado de una investigación por parte del autor, en base a la experiencia y a la conversación en un #programadorIO en el que explicamos justamente las características y posibilidades del desarrollo basado en API. Por tanto, debemos compartir el crédito con nuestros compañeros:

Carlos Ruiz Ruso [@micromante](#) Erick Ruiz de Chavez [@erickrdch](#) Roberto Segura [@phproberto](#)

Este artículo es obra de *Miguel Angel Alvarez*

Fue publicado por primera vez en 19/12/2014

Disponible online en <http://desarrolloweb.com/articulos/ventajas-inconvenientes-api-rest-desarrollo.html>

Autenticación por token

Qué es la autenticación por token, procedimiento habitual para loguear a los usuarios en APIs REST y recordarles para los futuros accesos al servicio web.

En este artículo vamos a abordar, de manera conceptual, una práctica recurrente a la hora de mantener la seguridad de un API REST, como es la autenticación por token. Como decimos, es habitual en arquitecturas REST, aunque también puede ser usado para cualquier otra, por ejemplo GraphQL u otros mecanismos para implementar servicios web.

La autenticación por token es algo que se tiene que hacer del lado del servidor y por tanto su implementación depende en gran medida de las tecnologías que estemos usando en el backend. Sin embargo, siempre se trabaja con el mismo flujo de aplicación y los conceptos que vamos a describir a continuación son perfectamente válidos para cualquier lenguaje, base de datos o tipo de servidor que podamos estar usando.



Qué es la autenticación por token

Para el que no tenga nociones explicaremos, de manera general, cómo sería un modelo de autenticación adecuado para las APIs.

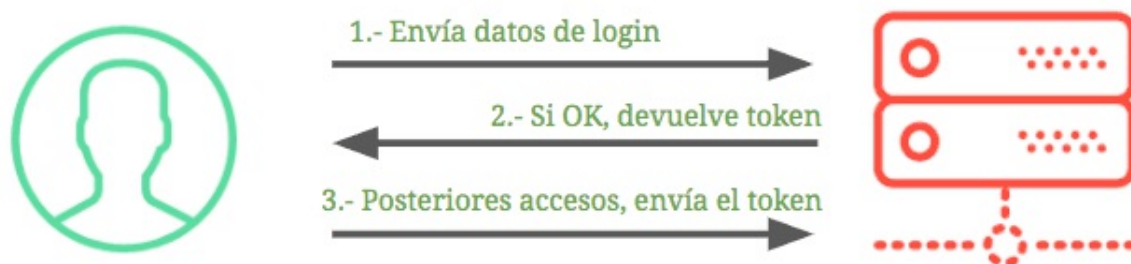
Si hay algo característico de las API, REST o cualquier otra arquitectura, es que no manejan estado y por lo tanto no tienen sesiones. En el artículo de [Qué es REST](#) ya tuvimos ocasión de dar más detalles sobre esto, de modo que no vamos a repetirnos. Lo destacable ahora es que, como consecuencia de no tener estado y no tener sesiones en el servidor para cada usuario, el backend es incapaz de recordar al usuario entre llamada y llamada al API.

Para evitar que en cada llamada el usuario deba entregar su clave y password se usa un token, que no es más que una cadena de texto bien larga, encriptada con una clave. Según se hace el login, el servidor devuelve al cliente un token y el cliente en futuros accesos entrega ese token para certificar que está autenticado.

Nota: Desarrollar basado en API tiene diversas ventajas, como aportar la posibilidad que otros trabajen con tus servicios web desde sus aplicaciones. Pero aunque solo fueses tú quien va a usar un API y no la liberes públicamente, existen diversas ventajas también. Si quieres agregar algo de más información puedes leer también el artículo [Un mismo backend, diferentes frontales](#).

Flujo de autenticación y autorización

En resumen, el flujo de trabajo para la autenticación por token, en el marco del desarrollo frontend, se puede apreciar en la siguiente imagen:



A.- El cliente envía los datos de login

A través de un formulario en la página se recaba el usuario y la clave de acceso. Estos datos se enviarán al servidor. En este punto pueden pasar dos cosas:

1. Si el login es correcto, se da al cliente por autenticado. Entonces se genera un token en el lado del servidor, que se enviará al cliente de vuelta.
2. Si el login no fue correcto, entonces se le manda un código de error al cliente.

B.- Almacenar el token

En el caso que se realizase un intento correcto de autenticación, y por tanto el servidor nos devolviese el token, en el lado del cliente se deberá almacenar el token, para poder usarlo más tarde.

Lo ideal será almacenar el token en el navegador, en un lugar donde se pueda acceder a él en futuras consultas. Es decir, sería interesante proveer algún sistema de persistencia, para que por ejemplo, si el usuario refresca la página o accede a la misma web pasados unos minutos, se disponga del token generado anteriormente y no se tenga que obligar al usuario a pasar de nuevo por el proceso de login. Lo más normal para almacenar el token sería el LocalStorage del navegador.

C.- Envío del token en posteriores accesos

Cada vez que el cliente, una vez autenticado, desee acceder de nuevo a un recurso de ese API, entonces tendrá que enviar el token al servidor para informarle que es él mismo y que ya se autenticó correctamente en un paso anterior.

Al recibir el token, el servidor lo tendrá que verificar y, si lo encuentra válido, sabrá que corresponde a un

usuario de su aplicación, por lo que podrá conceder el acceso al recurso consultado.

Otras situaciones importantes del flujo de control de usuario

Además del proceso de autenticación de usuarios y posterior autorización en los accesos al API, existen un par de casos adicionales que debemos implementar para el flujo del control de usuario en la aplicación.

Al iniciarse la aplicación

Al arrancar todo el proceso de inicio de la aplicación, sea cual sea el framework Javascript que se esté usando, puede ocurrir que se disponga de un token ya almacenado en un espacio de persistencia en el navegador.

Por lo tanto, también en el lado del frontend, el cliente deberá comprobar si tiene el token en el almacenamiento local.

1. Si lo tiene, debe verificar que es correcto. Para ello lanzará una request contra el servidor, que es quien puede verificar ese token. a) Si es correcto entonces está autenticado. Con lo que el cliente debería modificar el estado de la aplicación para mostrar que el usuario está logueado. b) Si no es correcto, entonces se entiende que el token no es válido y por tanto lo lógico será borrarlo del almacenamiento local y mantener el estado de aplicación, mostrando que no está autenticado.
2. b) Si el cliente no tiene el token en un espacio de almacenamiento, entonces se entiende que no está autenticado y no necesitamos verificar nada. En este caso el estado de la aplicación debería demostrar que no se está logueado.

Al hacer logout

La operativa para hacer logout consiste en borrar el token del almacenamiento local del navegador. Además en la aplicación se debería modificar el estado para indicar al usuario que no se está logueado.

Caducidad del token y refresh token

Otra cosa que no hemos mencionado en este flujo se deriva de la caducidad del token. El token generalmente tiene un tiempo de validez, que se establece en el servidor, al momento de generar el token.

Durante el tiempo de validez del token podrá ser utilizado para autorizar el acceso a ciertos recursos por parte del cliente autenticado. Sin embargo, si el token ha caducado, entonces el servidor lo rechazará y el cliente no podrá acceder a aquella información u operación solicitada.

Para establecer un flujo mediante el cual se mitiguen posibles problemas por caducidad del token, se puede proporcionar también un token de refresco. Ese token servirá para aumentar el tiempo de validez del token o generar uno nuevo más adelante, sin necesidad de obligar al usuario a volver a enviar sus datos de inicio de sesión.

El token de refresco se utiliza opcionalmente, por lo que no todas las implementaciones deben controlarlo necesariamente.

JSON Web Token

Uno de los estándares para producir los token del lado del servidor es JSON Web Token. Básicamente especifica toda una operativa, que se debe implementar del lado del servidor, para producir tales token y verificar su validez cuando sea necesario.

El estándar JSON Web Token indica que los token del lado del servidor se crean mediante una cadena de texto encriptada por una clave, que se mantiene en secreto del lado del servidor. Esa misma clave se debe de usar para descryptar el token y verificar su validez.

El token en sí, una vez descryptado, permite conocer cosas sobre el usuario que ha realizado la solicitud, es decir, es una manera de mantener los datos de sesión y saber cosas del usuario, como su email, nombre o cualquier otra información que la aplicación juzgue necesaria.

Nota: La sesión, que es una parte fundamental de las aplicaciones del lado del servidor, es problemática para su almacenamiento en sistemas distribuidos (por ejemplo en servicios que funcionan mediante balanceo de carga). Por ese motivo, en JSON Web Token se espera que la sesión se almacene en el cliente, como se describió anteriormente. Para aclarar posibles dudas para aquellos que vienen de la programación backend tradicional, en este modelo de autenticación por token también se dispone de unos datos de sesión, solo que la sesión en este caso se almacena en el cliente y no en el servidor. El cliente es el que le manda los datos de sesión al servidor en cada acceso que necesite autorización.

Librerías para JSON Web Token

Dependiendo de la tecnología de backend que se pretenda usar podemos encontrar diferentes librerías para implementar de una manera cómoda la generación y verificación de tokens. Existen librerías para todos los lenguajes de backend y habitualmente los frameworks ofrecen alguna alternativa.

En el caso concreto de Javascript (relevante por ser NodeJS una plataforma habitual para la implementación de APIs REST), la creación y verificación del token se puede hacer con la librería "jsonwebtoken", que es bastante sencilla de utilizar.

Implementación de un proceso de autenticación y autorización por JSON Web Token

En este artículo lo que nos interesaba era dejar claros una serie de conceptos, ya que de por sí este proceso de autenticación y autorización no es trivial. Pero seguro que te interesa contar con alguna idea extra que te permita implementar este esquema habitual de funcionamiento de las aplicaciones modernas.

El proceso tiene dos partes fundamentales para su implementación, que son totalmente independientes entre sí, ya que se deben encarar como dos desarrollos distintos, la parte del cliente y la parte del servidor.

JWT del lado del servidor

Como hemos dicho, la implementación depende mayoritariamente de las tecnologías que usemos del lado del servidor, por lo que sería complicado verlas todas, no obstante, si quieres algo de código puedes echar un vistazo a este artículo, donde se explica un [software genérico para creación de un API REST a partir de un JSON, con autorización por JWT](#).

Otras implementaciones, explicadas paso a paso y con mucho detalle las vas a encontrar en cursos de EscuelaIT. Te recomiendo el [Curso de API RESTful con Laravel](#) o el [curso de Desarrollo con NodeJS y MongoDB](#).

Implementación del lado del cliente de la autenticación y autorización en un API

Para implementar el proceso de autenticación del lado del cliente te recomendamos usar un framework Javascript. Puedes aprender aquí mismo con el [Manual de Angular](#) o el [Manual de Polymer 2](#), por ejemplo.

Estos dos frameworks/librerías se explican también en cursos de EscuelaIT donde se ve paso a paso el proceso del cliente para autenticación y autorización. Por ejemplo te recomendamos el [curso de Angular \(este es el más reciente en el momento de escribir este texto\)](#), o el [Curso de Desarrollo de aplicaciones con Polymer 2](#).

Este artículo es obra de *Miguel Angel Alvarez*
Fue publicado por primera vez en 11/04/2018
Disponible online en <http://desarrolloweb.com/articulos/autenticacion-token.html>

Crear un API REST, 5 minutos con json-server

Crear en 5 minutos un API REST, con fines didácticos, ideal para aprender desarrollo frontend con un framework Javascript del lado del cliente, usando json-server.

En este artículo vamos a ver cómo podemos crear un API REST en menos de 5 minutos, que podremos usar con fines didácticos o para prototipado, ideal cuando estamos aprendiendo a desarrollar con un framework Javascript del lado del cliente.

La idea es conseguir un API perfectamente funcional, que admita las operaciones típicas de cualquier API REST estándar, ofreciendo las típicas operaciones través de los distintos verbos del HTTP, pero sin tener que invertir horas de nuestro tiempo para aprovisionarlo.

Para poder ser tan rápidos usaremos un sencillo archivo JSON como fuente de datos, donde podremos generar nuestro modelo de datos, con distintas entidades, y datos de prueba. Como servidor REST usaremos un programa llamado "json-server", desarrollado con NodeJS que nos ofrece la posibilidad de tener un servidor web que responde a las operaciones típicas de las API.

El API REST que vamos a crear será perfectamente funcional, es decir, realizará todas las operaciones posibles sobre nuestros datos (lecturas, modificaciones, inserciones y borrados), almacenando los datos en el archivo JSON. La persistencia de los datos se realiza en el propio archivo de texto JSON, de este modo, en futuros accesos a la aplicación web, el API recordará su estado manteniendo toda la actividad que se haya realizado a lo largo del tiempo.



Al final del artículo verás además un vídeo donde se explica todo el proceso y se muestra cómo realizarlo en pocos minutos.

Nota: Si no tienes claro lo que es REST te recomiendo que te leas el artículo [Qué es REST](#) de Eduard Tomàs que lo explica muy bien y muy rápido. Incluso si tienes una idea de lo que es un API REST y quieres completar la información y saber más detalles de este tipo de recursos, la lectura te vendrá muy bien.

Descargar json-server

El primer paso para tener nuestra API es descargar e instalar json-server. Este programa es un paquete de NodeJS que instalaremos vía npm.

Nota: Si no tienes NodeJS deberás instalarlo en tu ordenador. Es muy sencillo entrando en <https://nodejs.org> y siguiendo las instrucciones para instalación. También te recomendamos el [Manual de NodeJS de Desarrolloweb.com](#), donde encontrarás información útil sobre "Node". El gestor de paquetes "npm" se instala automáticamente durante la instalación de NodeJS en tu máquina.

Realizaremos el siguiente comando en nuestro terminal:

```
npm install -g json-server
```

El gestor de paquetes npm nos instalará json-server con todas sus dependencias. Mientras tanto, podemos ir realizando el siguiente paso.

Crear un archivo JSON con los datos de nuestra API

En cualquier carpeta de nuestro ordenador debemos crear el archivo JSON que va a servir de origen de datos para nuestra API. Los datos serán aquellos que necesitemos en nuestra aplicación y en el propio JSON se puede introducir de inicio un conjunto de datos de prueba.

Nota: Puedes [saber algo más de JSON en este artículo](#). En realidad no es más un archivo de texto plano, que guardarás con codificación UTF-8. La notación para definir los datos la realizamos como si fuera un objeto Javascript. Al trabajar con JSON no se realiza una definición del modelo de datos como se hace en una base de datos relacional, ya que este modelo de datos se define a través de los propios datos.

Podemos ver un juego de datos de muestra, en el siguiente código JSON, que almacena películas y un conjunto de clasificaciones de películas.

```
{
  "películas": [
    {
      "id": 1,
      "nombre": "El sexto sentido",
      "director": "M. Night Shyamalan",
      "clasificacion": "Drama"
    },
    {
```

```
{
  "id": 2,
  "nombre": "Pulp Fiction",
  "director": "Tarantino",
  "clasificacion": "Acción"
},
{
  "id": 3,
  "nombre": "Todo Sobre Mi Madre",
  "director": "Almodobar",
  "clasificacion": "Drama"
},
{
  "id": 4,
  "nombre": "300",
  "director": "Zack Snyder",
  "clasificacion": "Acción"
},
{
  "id": 5,
  "nombre": "El silencio de los corderos",
  "director": "Jonathan Demme",
  "clasificacion": "Drama"
},
{
  "id": 6,
  "nombre": "Forrest Gump",
  "director": "Robert Zemeckis",
  "clasificacion": "Comedia"
},
{
  "id": 7,
  "nombre": "Las Hurdes",
  "director": "Luis Buñuel",
  "clasificacion": "Documental"
}
],
"clasificaciones": [
  {
    "nombre": "Drama",
    "id": 1
  },
  {
    "nombre": "Comedia",
    "id": 2
  },
  {
    "nombre": "Documental",
    "id": 3
  },
  {
    "nombre": "Acción",
    "id": 4
  }
]
}
```

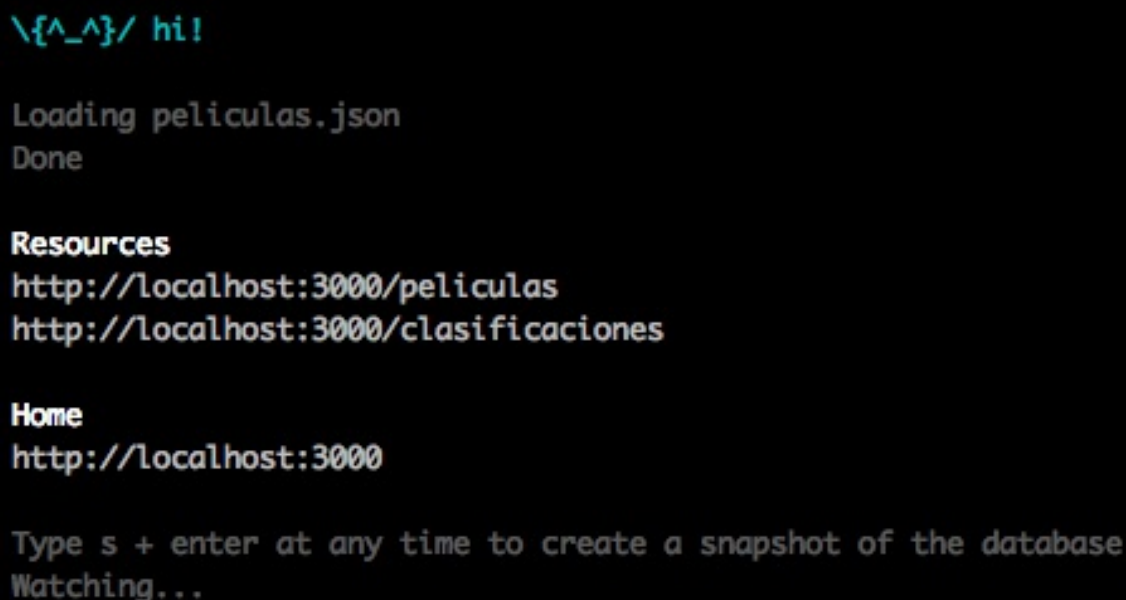

Este archivo JSON tendrá típicamente una extensión .json. En nuestro caso lo podríamos guardar como "películas.json".

Arrancar el servidor del API REST

Como tercer y último paso para tener nuestro API funcionando debemos arrancar el servidor del API, es decir, nuestro recién instalado json-server. Esto también lo hacemos desde el terminal, con el siguiente comando.

```
json-server --watch películas.json
```

Enseguida veremos que nos aparece una serie de mensajes indicando cómo podríamos acceder a nuestro origen de datos, a través de las URL del servidor, o las URL de los recursos o entidades generadas en el JSON.



```
\{^_^}/ hi!  
  
Loading películas.json  
Done  
  
Resources  
http://localhost:3000/películas  
http://localhost:3000/clasificaciones  
  
Home  
http://localhost:3000  
  
Type s + enter at any time to create a snapshot of the database  
Watching...
```

Acceder al API

Ahora ya solo nos queda acceder al API. Realmente podrás entrar en la "home" del servidor json-server para tu API a través de una URL como esta:

```
http://localhost:3000
```

En esa página encontrarás enlaces a los distintos recursos de tu API. Si los abres verás que te devuelve datos JSON como la mayoría de las API REST. Las operaciones que podrás realizar sobre el API son las típicas y se opera básicamente mediante los verbos del HTTP. Realmente aquí no difiere en nada de cualquier otro API que puedas usar.

Además, el API está perfectamente habilitada para invocarla desde otros dominios gracias a tener habilitado CORS, o si se desea usando [JSONP](#).

En resumen, en menos de 5 minutos tenemos un API funcionando que nos vendrá perfecto para realizar nuestra programación Javascript del lado del cliente con acceso a datos de un API. Como hemos dicho, este servidor de JSON está más enfocado a fines didácticos o también al prototipado de un proyecto web o App para móviles.

Objeto de estudio aparte sería el acceso a esos datos, que generalmente harás con Ajax, usando un framework como [AngularJS](#) o una librería como [BackboneJS](#), o al menos con [jQuery](#).

Vídeo de la creación del API

A continuación puedes ver todo el proceso de creación del API REST en vídeo.

Para ver este vídeo es necesario visitar el artículo original en: <http://desarrolloweb.com/articulos/crear-api-rest-json-server.html>

Este artículo es obra de *Miguel Angel Alvarez*

Fue publicado por primera vez en 27/11/2015

Disponible online en <http://desarrolloweb.com/articulos/crear-api-rest-json-server.html>

API REST con Autenticación JWT con JSON Server y jsonwebtoken

Cómo crear un API REST en 30 segundos, con funciones de autenticación basadas en JWT (JSON Web Token) usando JSON Server y jsonwebtoken.

En este artículo te voy a enseñar cómo puedes crear en menos de 10 minutos un API REST con autenticación basada en JSON Web Token (JWT), con prácticamente cero configuración, ideal para prototipar aplicaciones frontend SPA.

La base de este proyecto es JSON Server, del que ya hemos tenido ocasión de hablar. JSON Server permite crear un API REST a partir de un archivo JSON que tengas en tu disco duro, aportando un servidor capaz de recibir operaciones GET, PUT, DELETE, etc. Como ya hemos presentado este software en otra ocasión, lo vamos a dar por entendido. Puedes acceder al artículo anterior donde se explica JSON Server con mayor detalle.

El extra que venimos a relatar en esta ocasión es el sistema de autenticación para API REST basado en token. JSON Server no tiene implementada una autenticación, por lo que sólo con él no era posible prototipar aplicaciones con partes restringidas por login (usuario y contraseña). Disponer de ello es interesante para poder desarrollar una parte tan importante en un proyecto como es la autenticación.

Nota: Para saber más sobre la autenticación por token te recomendamos la lectura del artículo [Autenticación por token](#) que aborda muchos conceptos que te aclararán este modelo popular para la seguridad de las aplicaciones.

Pero lo cierto es que, tal como está construido JSON Server es bien sencillo crear un sistema que, apoyándose en él, pueda restringir ciertos recursos del API, de modo que sólo estén disponibles para usuarios correctamente autenticados.



"JSON Server" y JWT Auth

En este proyecto de Github encontramos todo lo que necesitamos para desplegar tu API REST en 30 segundos (lo que tardas en escribir un comando de consola), con la autenticación por JWT incorporada de casa. Se trata de un proyecto en el que tengo varias ideas para mejorarlo, pero que se puede usar perfectamente ya.

Puedes encontrar el repositorio en GitHub en este enlace: <https://github.com/deswebcom/json-server-with-jwt-auth>

Como había mencionado, básicamente implementa otro software "open source" bastante popular, llamado JSON Server, al que hemos agregado una autenticación básica basada en JSON Web Token.

A continuación dejo una traducción al español del archivo del README.md, en el que doy los detalles para usar el software y para poder configurarlo con nuestro propio juego de datos.

Instrucciones específicas para montar y configurar el servidor REST con autenticación JWT

Una vez clonado o descargado el repositorio en tu ordenador local, lo único que debes hacer para poner en marcha tu API REST con autenticación es lanzar la ejecución del archivo de arranque, llamado "index.js".

```
node index
```

Cuando invocamos así la ejecución del archivo index.js, el software arranca con las configuraciones definidas como predeterminadas.

- JSON server arranca en el puerto 3000
- Como juego de datos usa el archivo que se encuentra en la ruta: "json-samples/default.json".
- Retardo de 1500 milisegundos en la respuesta

Argumentos de arranque

Puedes configurar el servidor de tu API con autenticación a través de argumentos en la llamada a node index.

- --file sirve para indicar la ruta a otro fichero json con los datos del API.
- --port sirve para indicar el puerto de arranque
- --authentication sirve para poder definir si queremos autenticación o no.
- --delay: indicar el tiempo de retraso en la respuesta.

Por ejemplo, este comando arranca otro archivo JSON como modelo de datos y escucha en el puerto 3000. Además arranca con el sistema de autenticación funcionando. (Ver más adelante las rutas que están protegidas y las rutas que están libres, así como las rutas para loguear y crear nuevos usuarios).

```
node index --file=./json-samples/series.json --port=3333
```

Este segundo comando, aparte de configurar la ruta y el puerto, indica que no se desea autenticación.

```
node index --file=./json-samples/marcadores.json --port=3333 --authentication=no
```

Otra modificación que he realizado a JSON Server y que me viene muy bien para simular mejor cómo funcionan las API REST es la creación de un tiempo de retardo en la respuesta. Así, aunque tengas funcionando el API en local, el servidor tardará en responder como si fuera un servidor en remoto, pudiendo percibir efectos como la típica ruedita del ajax rodando mientras se espera la respuesta.

Si quieres, puedes quitar el retardo en la respuesta, o configurarlo de otra manera, puedes hacerlo con el argumento "delay", tal como sigue:

```
node index --file=./json-samples/series.json --delay=0
```

El retardo se mide en milisegundos. De modo que, si quieres indicar el tiempo de retardo a medio segundo podrías arrancar con:

```
node index --delay=500
```

Rutas para el sistema de autenticación

Las siguientes rutas te sirven para implementar el flujo para autenticación.

- Login: /login
- Registro: /sign-in
- Verificar un usuario: /verify

Para usar las rutas del login y registro se deben enviar solicitudes "POST" a esas rutas, indicando el usuario a registrar o autenticar mediante un objeto JSON que colocaremos en el cuerpo de la solicitud. Esa parte la tendrás que hacer con tu librería o framework Javascript preferido.

El objeto que colocarás en el body del request será algo como esto:

```
{email: "desarrolloweb@example.com", password: "1234"}
```

La ruta de verificación de un usuario "/verify" sirve para comprobar si el token que tenemos sobre un usuario supuestamente autenticado es válido. Para esa comprobación tendrás que enviar el token desde el cliente, a través de las cabeceras de la solicitud HTTP.

```
{  
  "Content-Type": "application/json",  
  "token": "Aquí_colocas_el_token"  
}
```

Nota: en esta versión comentada de este sistema de autenticación los datos de los usuarios se almacenan simplemente en la memoria de Node. Por ello, con cualquier reinicio del servidor, todos los usuarios se pierden.

Para facilitar el flujo de trabajo, cuando el servidor del API REST se inicia, se crea directamente un usuario en la memoria, que puedes usar para tus pruebas, sin que tengas que registrarte.

- email: user@example.com
- password: 1234

Además, si tienes un mínimo conocimiento de Javascript y NodeJS, podrías editar el archivo index.js para configurar este usuario con otros datos, o añadir al setup otros usuarios que se creen automáticamente cuando el servidor se reinicia.

Rutas protegidas y rutas públicas

De manera que en tus aplicaciones puedas jugar con distintos tipos de rutas, tanto públicas como restringidas, hemos realizado este programa colocando rutas que no requieren autorización. El criterio que se ha seguido es el siguiente:

- Todas las rutas de tipo GET al API REST se consideran rutas públicas, en las cuales no se verifica que haya un usuario correctamente autenticado.
- Todos los demás métodos del HTTP (POST, PUT, DELETE...) están detrás de rutas autenticadas, para las que se realiza el proceso de verificación del usuario.

Para verificar el usuario en una ruta autenticada se hace el mismo proceso que se describió en la ruta "/verify". Es decir, tienes que enviar en las cabeceras del HTTP el token de autenticación válido. En caso que no se envíe el token, o el mismo sea incorrecto o haya caducado, el servidor responderá con un error 401.

El token se envía en las cabeceras, de una manera como esta:

```
{
  "Content-Type": "application/json",
  "token": "Put_here_the_token"
}
```

Espero que con estas notas puedas crear tu propio sistema de API REST con autenticación JWT de una manera ágil y puedas desarrollar tu proyecto frontend con un servidor estándar y extremadamente fácil de configurar.

Este artículo es obra de *Miguel Angel Alvarez*
Fue publicado por primera vez en 20/03/2018
Disponible online en <http://desarrolloweb.com/articulos/api-rest-autenticacion-jwt.html>

