

BackEnd - Node.JS

- 1 require HTTP
- 2 writeHead – Crear / Modificar el encabezado HTTP
- 3 end() para el envío de cadenas en HTML
- 4 console.log desde Node.
- 5 Creando un servidor. El módulo HTTP
- 6 Configuración del puerto de escucha con listen()
- 7 Usando http.createServer()
- 8 Análisis de su funcionamiento y detalle de librerías activas.
- 9 Instalando librerías a Visual Studio
- 10 Creando un proyecto en Node con NPM
- 11 Leer Cadenas en consultas
- 12 El error 404 y el código 200
- 13 Módulo de archivos.

Web services: servicios de máquina a máquina

A día de hoy, usar diferentes servicios a través de la web es una actividad habitual. Comprar online, leer el periódico, reservar una mesa en un restaurante o ver películas son solo algunos ejemplos de las muchas interacciones que se producen a diario entre el usuario y la máquina. Pero, además, y aunque pueda pasar desapercibido, estas interacciones también tienen lugar entre máquinas: el cliente y el servidor se están enviando continuamente solicitudes y respuestas, transmisión que se produce gracias a los web services o servicios web.

¿Qué es un web service?

Un web service facilita un servicio a través de Internet: se trata de una interfaz mediante la que dos máquinas (o aplicaciones) se comunican entre sí. Esta tecnología se caracteriza por estos dos rasgos:

Multiplataforma: cliente y servidor no tienen por qué contar con la misma configuración para comunicarse. El servicio web se encarga de hacerlo posible.

Distribuida: por lo general, un servicio web no está disponible para un único cliente, sino que son diferentes los que acceden a él a través de Internet.

Cuando se utiliza un web service, un cliente manda una solicitud a un servidor, desencadenando una acción por parte de este. A continuación, el servidor devuelve una respuesta al cliente.

La tecnología detrás de un servicio web con ejemplo

Todos los web services cuentan con un Uniform Resource Identifier (URI) unívoco, esto es, la dirección del servicio web. Es similar al Uniform Resource Locator (URL) que permite acceder a páginas web. El catálogo UDDI debía desempeñar también un papel importante, pues permitía encontrar los servicios web, pero este servicio nunca logró imponerse y sus mayores partidarios terminaron retirándose del proyecto.

Más importancia tiene el lenguaje Web Service Description Language (WSDL). Un servicio web contiene un archivo en WSDL en el que se describe el servicio de forma detallada. Con esta información, el cliente puede comprender qué funciones puede ejecutar en el servidor a través del servicio web. La comunicación funciona exclusivamente mediante diferentes protocolos y arquitecturas. Entre ellos, son muy populares el protocolo de red SOAP en combinación con el estándar de Internet HTTP o los servicios web basados en una arquitectura REST.

Con estas tecnologías se posibilita el intercambio de peticiones y respuestas a menudo utilizando el lenguaje de marcado extensible (XML). Este lenguaje, relativamente simple, puede ser interpretado en igual medida por personas y ordenadores, y además es adecuado para unir sistemas con requisitos diferentes. Con todo, REST también admite otros formatos, como JSON.

Veamos cómo funciona la mecánica de esta tecnología con un ejemplo de web service. Partamos de un software escrito en Visual Basic que se ejecuta en una máquina con sistema Windows. El programa necesita el servicio de un servidor web Apache. Para ello, el cliente envía una solicitud SOAP en forma de mensaje HTTP al servidor. El web service interpreta el contenido de la solicitud y se encarga de que el servidor lleve a cabo una acción. Finalmente, el servicio web formula una respuesta y la envía de vuelta al cliente (de nuevo con SOAP y HTTP), que vuelve a interpretarla. La información se envía entonces al software, donde será procesada.

Ventajas y desventajas de los servicios web

La ventaja principal de los servicios web es que la comunicación no depende de una plataforma determinada, por lo que el cliente y el servidor apenas han de presentar rasgos en común para poder comunicarse. Para ello, la tecnología web service recurre a formatos estandarizados que interpretan todos los sistemas.

Pero en estos formatos es donde encontramos una de las desventajas. Precisamente, XML es un formato más bien voluminoso que genera grandes paquetes de datos, lo que puede crear problemas en las conexiones de red lentas. Otra posibilidad que permite conectar a dos sistemas a través de Internet son las API web. Aunque, por lo general, son más rápidas, someten a cliente y servidor a especificaciones más concretas, con lo que la interoperabilidad se ve limitada.

Web Services

Funcionamiento y Objetivo

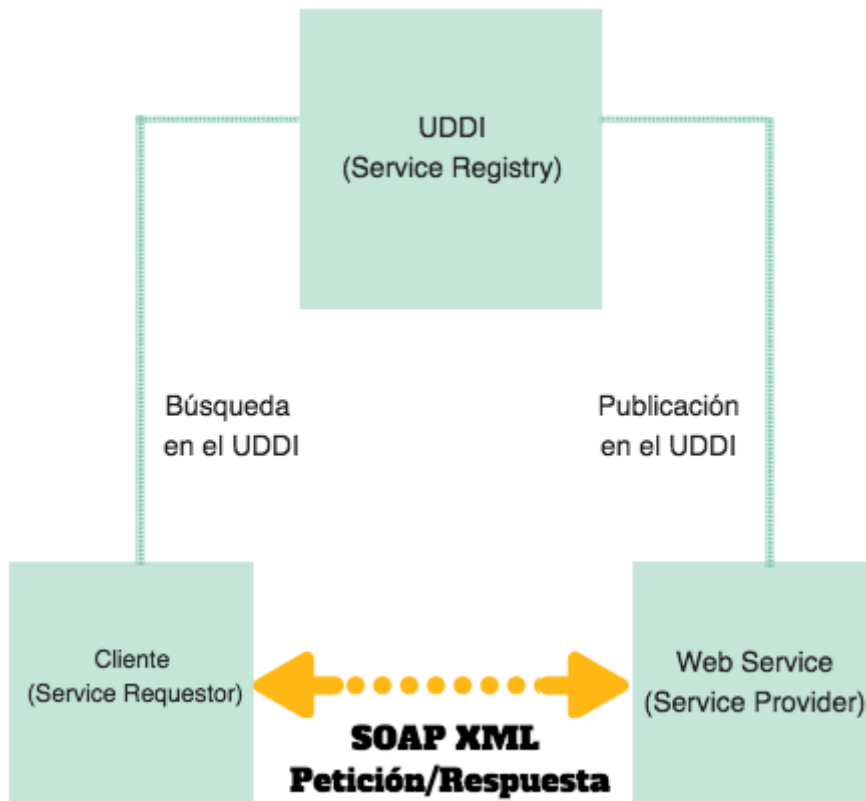
1. Introducción

Un Web Service, o Servicio Web, es un método de comunicación entre dos aparatos electrónicos en una red. Es una colección de protocolos abiertos y estándares usados para intercambiar datos entre aplicaciones o sistemas. Las aplicaciones escritas en varios lenguajes de programación que funcionan en plataformas diferentes pueden utilizar web services para intercambiar información a través de una red. La interoperatividad, por ejemplo entre Java y Python o Windows y Linux se debe al uso de estándares abiertos.

Como sistema de mensajes se utiliza XML estandarizado. El protocolo más simple para el intercambio de información entre ordenadores es XML-RPC, que emplea XML para llevar a cabo RPCs. RPC, Remote Procedure Call, es un protocolo de red que permite a un programa a ejecutar código en una máquina remota. Los XML-RPC requests son una combinación entre contenido XML y headers HTTP. La simpleza de los XML-RPC hizo que el estándar evolucionase a SOAP, uno de los componentes básicos de los Web Services.

La base de comunicación entre web services es por tanto el lenguaje XML y el protocolo HTTP.

2. Componentes de los Web Services



Los web services estandarizados funcionan con los siguientes componentes:

- SOAP – Simple Object Access Protocol

SOAP es un protocolo escrito en XML para el intercambio de información entre aplicaciones. Es un formato para enviar mensajes, diseñado especialmente para servir de comunicación en Internet, pudiendo extender los HTTP headers. Es una forma de definir qué información se envía y cómo mediante XML. Básicamente es un protocolo para acceder a un Web Service.

- WSDL – Web Services Description Language

WSDL es un lenguaje basado en XML para describir los servicios web y cómo acceder a ellos. Es el formato estándar para describir un web service, y fue diseñado por Microsoft e IBM. WSDL es una parte integral del estándar UDDI, y es el lenguaje que éste utiliza.

- UDDI – Universal Description, Discovery and Integration

UDDI es un estándar XML para describir, publicar y encontrar servicios web. Es un directorio donde las compañías pueden registrar y buscar servicios web. Es un

directorio de interfaces de servicios web descritos en WSDL que se comunican mediante SOAP.

3. Arquitectura de los Web Services

- Service Discovery. Responsable de centralizar servicios web en un directorio común de registro y proveer una funcionalidad sencilla para publicar y buscar. UDDI se encarga del Service Discovery.

- Service Description. Uno de los aspectos más característicos de los web services es que se autodescriben. Esto significa que una vez que se ha localizado un Web Service nos proporcionará información sobre que operaciones soporta y cómo activarlo. Esto se realiza a través del Web Services Description Language (WSDL).

- Service Invocation. Invocar a un Web Service implica pasar mensajes entre el cliente y el servidor. SOAP (Simple Object Access Protocol) especifica cómo deberíamos formatear los mensajes request para el servidor, y cómo el servidor debería formatear sus mensajes de respuesta.

- Transport. Todos estos mensajes han de ser transmitidos de alguna forma entre el servidor y el cliente. El protocolo elegido para ello es HTTP (HyperText Transfer Protocol). Se pueden utilizar otros protocolos pero HTTP es actualmente el más usado.

4. Como funciona un Web Service

1. El Service Provider genera el WSDL describiendo el Web Service y registra el WSDL en el directorio UDDI o Service Registry.

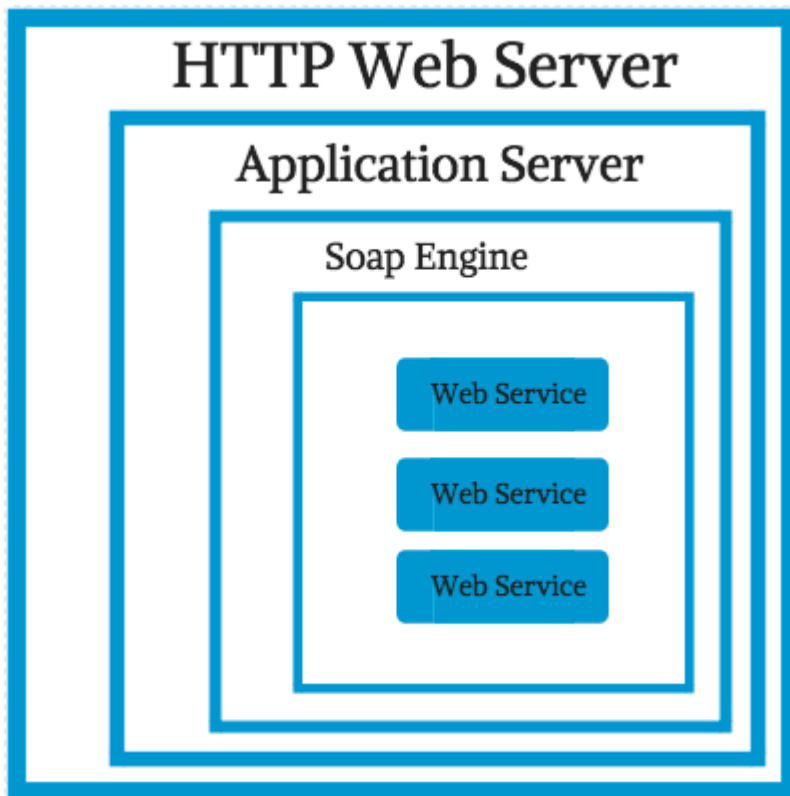
2. El Service Requestor o la aplicación del cliente requiere un Web Service y se pone en contacto con el UDDI para localizar el Web Service.

3. El cliente, basándose en la descripción descrita por el WSDL, envía un request para un servicio particular al Web Service Listener, que se encarga de recibir y enviar los mensajes en formato SOAP.

4. El Web Service analiza el mensaje SOAP del request e invoca una operación particular en la aplicación para procesar el request. El resultado se escribe de nuevo en SOAP en forma de respuesta y se envía al cliente.

5. El cliente analiza el mensaje de respuesta SOAP y lo interpreta o genera un error si ha habido alguno.

5. Componentes de los servidores en una aplicación Web Service



⑩ Web Service. Es el software o componente que realiza las operaciones. Si está escrito en Java, estas operaciones se realizarán en lenguaje Java. Los clientes invocarán estas operaciones enviando mensajes SOAP.

⑩ SOAP Engine. El Web Service no sabe interpretar SOAP requests y crear SOAP responses. Para hacer esto hace falta un SOAP engine, un software que se encarga del manejo de estos mensajes. Apache Axis es un ejemplo.

• Application Server. Para funcionar como un servidor que puede recibir requests desde diferentes clientes, el SOAP engine normalmente funciona dentro de un application server. Este es otro software que proporciona un espacio libre para aplicaciones que han de ser accedidas por múltiples clientes. El SOAP engine funciona como una aplicación dentro del application server. Ejemplos son Apache Tomcat server, Java Servlet y JSP container.

⑩ HTTP Server. Algunos application servers incluyen funcionalidades HTTP, por lo que se pueden tener Web Services funcionando instalando simplemente un SOAP engine y un application server. Sin embargo cuando un application server carece de funcionalidad HTTP es necesario también un

HTTP server, más comúnmente llamado Web Server. Es un software que sabe cómo manejar mensajes HTTP. Los dos más populares en la actualidad son Apache HTTP Server y Nginx.

Api's y servicios. Concepto y uso.

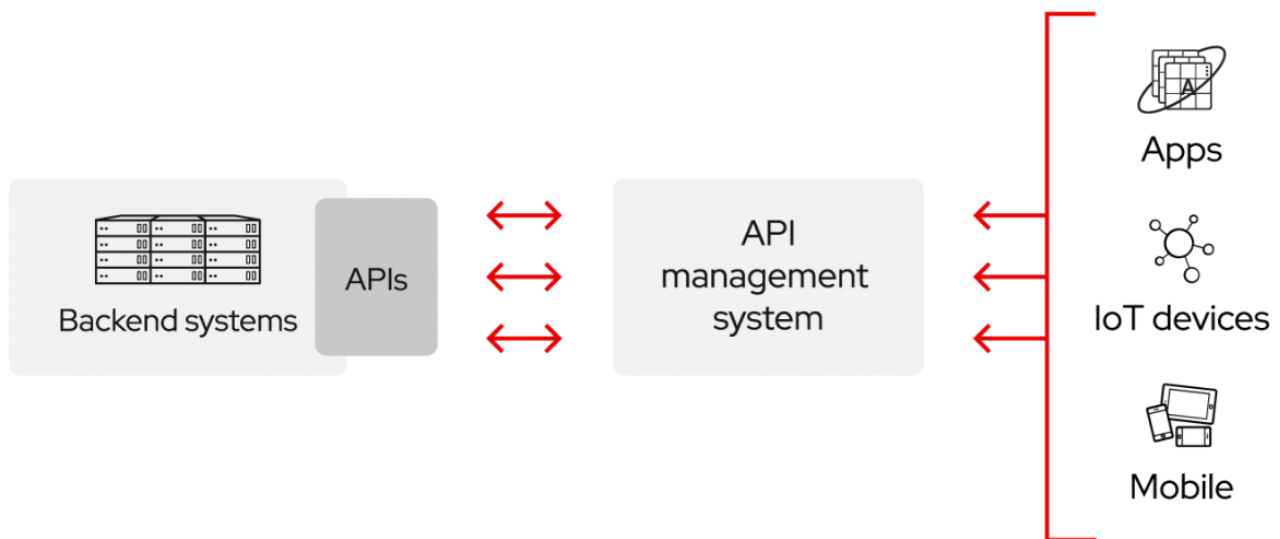
Una API es un conjunto de definiciones y protocolos que se utiliza para desarrollar e integrar el software de las aplicaciones. API significa interfaz de programación de aplicaciones.

Las API permiten que sus productos y servicios se comuniquen con otros, sin necesidad de saber cómo están implementados. Esto simplifica el desarrollo de las aplicaciones y permite ahorrar tiempo y dinero. Las API le otorgan flexibilidad; simplifican el diseño, la administración y el uso de las aplicaciones, y proporcionan oportunidades de innovación, lo cual es ideal al momento de diseñar herramientas y productos nuevos (o de gestionar los actuales).

A veces, las API se consideran como contratos, con documentación que representa un acuerdo entre las partes: si una de las partes envía una solicitud remota con cierta estructura en particular, esa misma estructura determinará cómo responderá el software de la otra parte.

Debido a que simplifican la forma en que los desarrolladores integran los elementos de las aplicaciones nuevas en una arquitectura actual, las API permiten la colaboración entre el equipo comercial y el de TI. Las necesidades comerciales suelen cambiar rápidamente en respuesta a los mercados digitales en constante cambio, donde la competencia puede modificar un sector entero con una aplicación nueva. Para seguir siendo competitivos, es importante admitir la implementación y el desarrollo rápidos de servicios innovadores. El desarrollo de aplicaciones nativas de la nube es una forma identificable de aumentar la velocidad de desarrollo y se basa en la conexión de una arquitectura de aplicaciones de microservicios a través de las API.

Las API son un medio simplificado para conectar su propia infraestructura a través del desarrollo de aplicaciones nativas de la nube, pero también le permiten compartir sus datos con clientes y otros usuarios externos. Las API públicas representan un valor comercial único porque simplifican y amplían la forma en que se conecta con sus partners y, además, pueden rentabilizar sus datos (un ejemplo conocido es la API de Google Maps).



Por ejemplo, imagínese una empresa distribuidora de libros. Podría ofrecer a los clientes una aplicación que les permita a los empleados de la librería verificar la disponibilidad de los libros con el distribuidor. El desarrollo de esta aplicación podría ser costoso, estar limitado por la plataforma y requerir mucho tiempo de desarrollo y mantenimiento continuo.

Otra opción es que la distribuidora de libros proporciona una API para verificar la disponibilidad en inventario. Existen varios beneficios de este enfoque:

- Permite que los clientes accedan a los datos con una API que les ayude a añadir información sobre su inventario en un solo lugar.
- La distribuidora de libros podría realizar cambios en sus sistemas internos sin afectar a los clientes, siempre y cuando el comportamiento de la API fuera el mismo.
- Con una API disponible de forma pública, los desarrolladores que trabajan para la distribuidora de libros, los vendedores o los terceros podrían desarrollar una aplicación para ayudar a los clientes a encontrar los libros que necesitan. Esto podría dar como resultado mayores ventas u otras oportunidades comerciales.

En resumen, las API le permiten habilitar el acceso a sus recursos y, al mismo tiempo, mantener la seguridad y el control. Cómo habilitar el acceso y a quiénes depende de usted. Seguridad de las API. Para conectarse a las API y crear aplicaciones que utilicen los datos o las funciones que estas ofrecen, se puede utilizar una plataforma de integración distribuida que conecte todos los elementos, incluidos los sistemas heredados y el Internet de las cosas (IoT).

Existen tres enfoques respecto a las políticas de las versiones de las API.

Partners

Las API son como parte de las API más importantes sin precedentes, los estándares API que influyen en las. En el caso de las API, se trata de un negocio de la vida. Este tipo de API proporciona flujos de ingresos a las aplicaciones, que interpretan como API, y puede ser un recurso para innovar.

Innovaciones con las API

El acceso de los partners o el público a sus API permite:

- ⑩ Crear nuevos canales de ingresos o ampliar los existentes.
- ⑩ Expandir el alcance de su marca.
- ⑩ Facilitar la innovación abierta o lograr mayor eficiencia, gracias al desarrollo y la colaboración externos.

Suena maravilloso, ¿no? ¿Pero cómo lo logran las API?

Volvamos al ejemplo de la empresa distribuidora de libros.

Supongamos que uno de los partners de la empresa desarrolla una aplicación que ayuda a las personas a encontrar libros en los estantes de cierta librería. Esta experiencia mejorada atrae más compradores a la librería (que es cliente de la distribuidora) y expande un canal de ingresos existente.

Es posible que un tercero use una API pública para desarrollar una aplicación que permita a las personas comprar libros directamente de la distribuidora, en lugar de hacerlo en una tienda, lo cual abre un nuevo canal de ingresos para la distribuidora de libros. Esto abre un nuevo canal de ingresos para la distribuidora de libros.

Las API compartidas, ya sea con los partners elegidos o con todo el mundo, tienen efectos positivos. Mientras más se asocie con otros gracias a las API, mayor difusión obtendrá su marca, independientemente de los esfuerzos publicitarios de la empresa. Si usted utiliza una API pública, por ejemplo, para dar acceso a la tecnología a todo el mundo, alienta a los desarrolladores a crear un ecosistema de aplicaciones en torno a su API. Mientras más personas usen su tecnología, más personas estarán dispuestas a hacer negocios con usted. Hacer pública la tecnología da resultados novedosos e inesperados que a veces alteran sectores completos. Estos resultados, a veces, alteran sectores completos. En el caso de nuestra distribuidora de libros, las nuevas empresas (un servicio de préstamo de libros, por ejemplo) pueden cambiar la manera en que comercializa sus servicios. Los partners y las API públicas le permiten aprovechar los esfuerzos creativos de una comunidad más grande que su equipo de desarrolladores internos. Las nuevas ideas surgen de todas partes, y las empresas deben ser conscientes de los cambios en su mercado y estar listas para actuar en consecuencia. Las API son de gran ayuda.

Breve historia de las API

Las API surgieron los primeros días de la informática, mucho antes que la computadora personal. En esa época, una API normalmente se usaba como biblioteca para los sistemas operativos. Casi siempre estaban habilitadas localmente en los sistemas en los que operaban, aunque a veces pasaban mensajes entre las computadoras centrales. Después de

casi 30 años, las API se expandieron más allá de los entornos locales. A principios del año 2000, ya eran una tecnología importante para la integración remota de datos.

Las API remotas

Las API remotas están diseñadas para interactuar en una red de comunicaciones. Por "remoto" nos referimos a que los recursos que administran las API están, de alguna manera, fuera de la computadora que solicita alguno de dichos recursos. Debido a que la red de comunicaciones más usada es Internet, la mayoría de las API están diseñadas de acuerdo con los estándares web. No todas las API remotas son API web, pero se puede suponer que las API web son remotas.

Las API web normalmente usan HTTP para solicitar mensajes y proporcionar una definición de la estructura de los mensajes de respuesta. Por lo general, estos mensajes de respuesta toman la forma de un archivo XML o JSON, que son los formatos preferidos porque presentan los datos en una manera fácil de manejar para otras aplicaciones.

¿Qué se ha hecho para mejorar las API?

A medida que las API han evolucionado en las ahora generalizadas API web, se han realizado muchos esfuerzos para simplificar su diseño y facilitar su implementación.

Un poco de SOAP, mucho de REST

A medida que se han difundido las API, se desarrolló una especificación de protocolo para permitir la estandarización del intercambio de información; se llama Protocolo de Acceso a Objetos Simples, más conocido como SOAP. Las API diseñadas con SOAP usan XML para el formato de sus mensajes y reciben solicitudes a través de HTTP o SMTP. Con SOAP, es más fácil que las aplicaciones que funcionan en entornos distintos o están escritas en diferentes lenguajes compartan información.

Otra especificación es la Transferencia de Estado Representacional (REST). Las API web que funcionan con las limitaciones de arquitectura REST se llaman API de RESTful. La diferencia entre REST y SOAP es básica: SOAP es un protocolo, mientras que REST es un estilo de arquitectura. Esto significa que no hay ningún estándar oficial para las API web de RESTful. Las API son RESTful siempre que cumplan con las 6 limitaciones principales de un sistema RESTful:

- ⑩ **Arquitectura cliente-servidor:** la arquitectura REST está compuesta por clientes, servidores y recursos, y administra las solicitudes con HTTP.
- ⑩ **Sin estado:** el contenido de los clientes no se almacena en el servidor entre las solicitudes, sino que la información sobre el estado de la sesión se queda en el cliente. En su lugar, la información sobre el estado de la sesión está en posesión del cliente.
- ⑩ **Capacidad de caché:** el almacenamiento en caché puede eliminar la necesidad de algunas interacciones cliente-servidor.

⑩ **Sistema en capas:** las interacciones cliente-servidor pueden estar mediadas por capas adicionales, que pueden ofrecer otras funciones, como el equilibrio de carga, los cachés compartidos o la seguridad. Estas capas pueden ofrecer funcionalidades adicionales, como equilibrio de carga, cachés compartidos o seguridad.

⑩ **Código de demanda (opcional):** los servidores pueden extender las funciones de un cliente transfiriendo código ejecutable.

⑩ **Interfaz uniforme:** esta limitación es fundamental para el diseño de las API de RESTful e incluye 4 aspectos:

⑩ **Identificación de recursos en las solicitudes:** los recursos se identifican en las solicitudes y se separan de las representaciones que se devuelven al cliente.

⑩ **Administración de recursos mediante representaciones:** los clientes reciben archivos que representan los recursos. Estas representaciones deben tener la información suficiente como para poder ser modificadas o eliminadas.

⑩ **Mensajes autodescriptivos:** cada mensaje que se devuelve al cliente contiene la información suficiente para describir cómo debe procesar la información.

⑩ **Hipermédios es el motor del estado de la aplicación:** después de acceder a un recurso, el cliente REST debe ser capaz de descubrir mediante hipervínculos todas las otras acciones que están disponibles actualmente.

Estas limitaciones pueden parecer demasiadas, pero son mucho más sencillas que un protocolo definido previamente. Por eso, las API de RESTful son cada vez más frecuentes que las de SOAP.

En los últimos años, la especificación de OpenAPI se ha convertido en un estándar común para definir las API de REST. OpenAPI establece una forma independiente del lenguaje para que los desarrolladores diseñen interfaces API de REST, que permite a los usuarios entenderlas con el mínimo esfuerzo.

SOA frente a la arquitectura de microservicios

Los dos enfoques de arquitectura que más se utilizan para las API remotas son la arquitectura orientada al servicio (SOA) y la arquitectura de microservicios. La SOA es el más antiguo de los dos, y comenzó como una mejora de las aplicaciones monolíticas. En lugar de usar una sola aplicación que haga todo, se pueden usar varias aplicaciones que proporcionen diferentes funciones y que no tengan conexión directa, todo gracias a un patrón de integración, como un bus de servicios empresariales (ESB).

Aunque en muchos aspectos la SOA es más sencilla que una arquitectura monolítica, conlleva un riesgo de cambios en cascada en todo el entorno si las interacciones de los componentes no se comprenden claramente. Esta complejidad adicional vuelve a presentar algunos de los problemas que la SOA pretendía solucionar.

Las arquitecturas de microservicios se parecen a los patrones SOA en que los servicios son especializados y no tienen conexión directa. Pero además, descomponen las arquitecturas tradicionales en partes más pequeñas. Los servicios de la arquitectura de microservicios usan un marco de mensajería común, como las API de RESTful. Utilizan

API de RESTful para comunicarse entre sí, sin necesidad de operaciones complejas de conversión de datos ni capas de integración adicionales. Usar las API de RESTful permite e incluso fomenta una distribución más rápida de nuevas funciones y actualizaciones. Cada servicio es independiente. Un servicio se puede reemplazar, mejorar o abandonar, sin afectar los demás servicios de la arquitectura. Esta arquitectura liviana optimiza los recursos distribuidos o en la nube y admite la escalabilidad dinámica de los servicios individuales.

Te mostramos de forma práctica cómo instalar Postman y cómo enviar una primera petición a una API REST.

Descarga e instalación de Postman

Para descargar Postman accedemos a su **página oficial**: <https://www.postman.com/downloads/>

Una vez hemos accedido a la web elegimos el sistema operativo en el que queremos instalar Postman, ya que se puede utilizar como un plugin de Google Chrome o se puede utilizar en Linux, MacOS o en Windows.

Cuando hayamos realizado la descarga ejecutamos el instalador, que va a realizar una instalación que resulta bastante automática. No es la típica instalación en la que se nos pregunta dónde queremos instalar la aplicación, o en la que hay que aceptar los términos y condiciones, sino que automáticamente se va a instalar, en el directorio AppData de nuestro usuario.

Tras la instalación se nos solicita introducir una cuenta que tengamos ya registrada o bien se nos ofrece la posibilidad de crear una nueva cuenta.

Esto se debe a que Postman tiene muchas funciones que interactúan con una nube, por lo que para poder almacenar un registro de las peticiones y de nuestro trabajo en la nube, y poder trabajar compartiendo un workspace o espacio de trabajo con otros compañeros de equipo, necesitamos identificarnos con una cuenta de Postman.

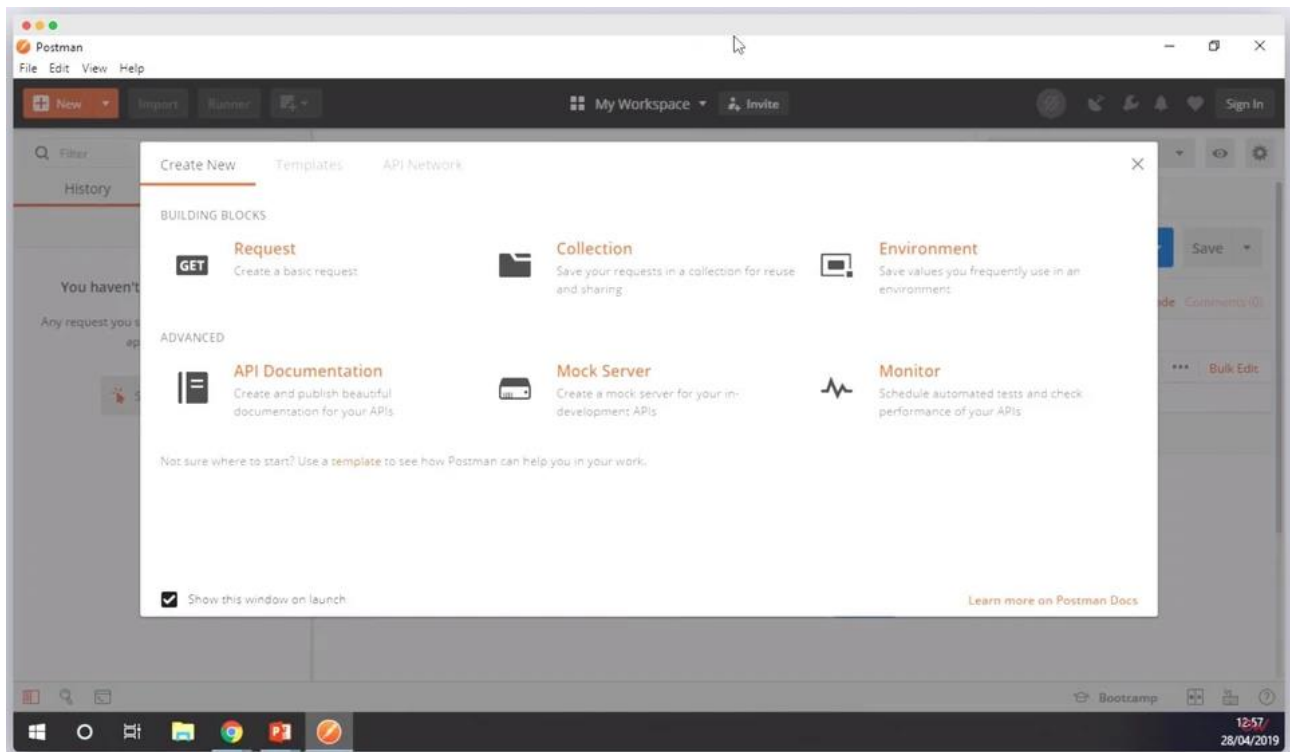
Para esta introducción no vamos a hacer uso de estos, sino que **vamos a continuar pulsando en “Skip signing in and take me straight to the app”**, para entrar en la aplicación sin tener que registrarse o iniciar sesión en Postman.

Envío de una petición con Postman

Esta es la vista inicial de Postman una vez ejecutado.

En esta pantalla tenemos diversas funcionalidades, características y elementos con los que podemos interactuar, como son:

- ⑩ **Request:** Crear una petición REST.
- ⑩ **Collection:** Crear una colección.
- ⑩ **Environment:** Crear un entorno.



- ⑩ **API Documentation:** Crear la documentación de una API y publicarla.
- ⑩ **Mock Server:** Crear un Mock Server.
- ⑩ **Monitor:** Crear un monitor.

Para el ejemplo vamos a crear una petición, para lo que tenemos que encontrar una API a la que queramos llamar. Usaremos la API que encontraremos en esta web: <https://rickandmortyapi.com/>.

Al ser una API pública no necesitamos autenticación y tiene un uso muy sencillo.

En la documentación de esta API podemos, vamos a obtener la lista de episodios de la serie.

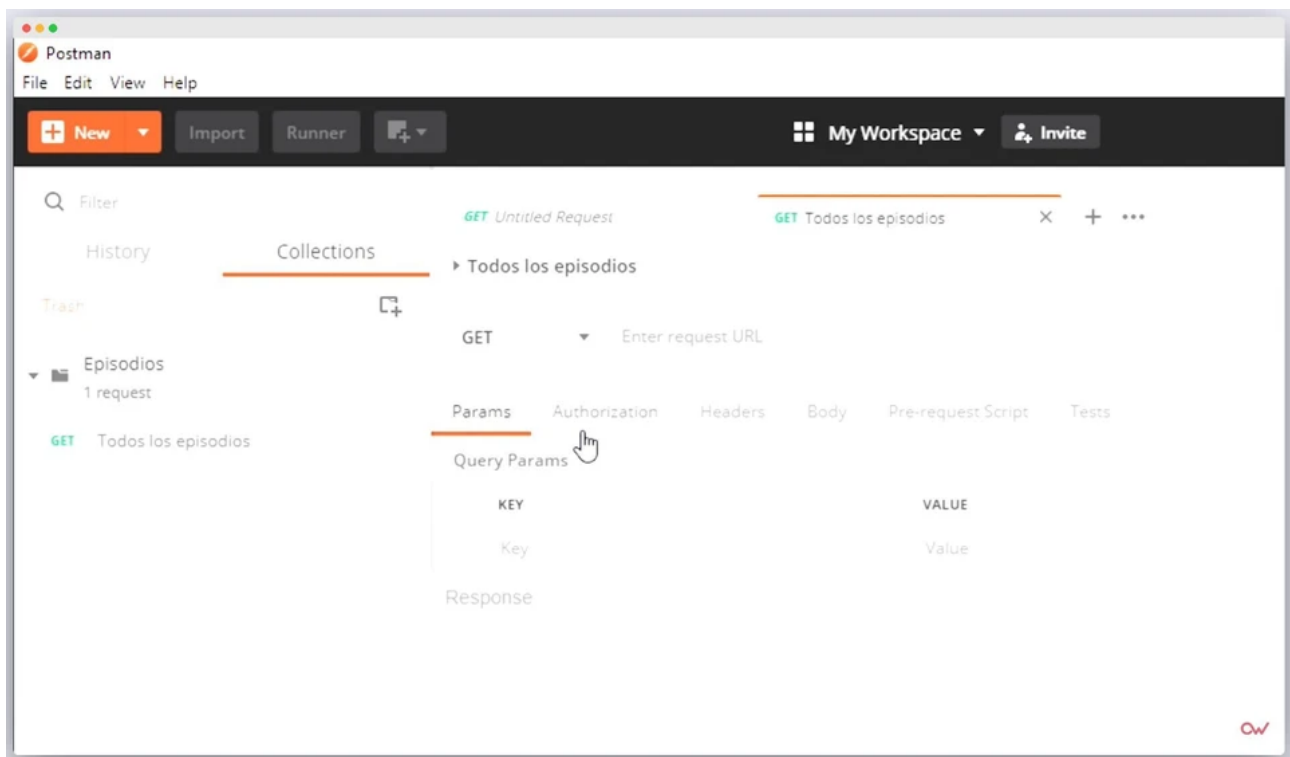
Podemos ver los esquemas de los episodios en [esta página](#).

Vamos a ir al primer endpoint, al que se accede a través de [esta ruta](#).

Volvemos a Postman a la creación de una petición REST, y a la misma le damos el nombre “Todos los episodios”, y en la descripción indicamos **“Petición para obtener una lista de episodios de Rick y Morty.”**, por ejemplo.

Después **pulsamos en “Create Collection”** para crear una colección, que simplemente una forma de almacenar peticiones, **la llamamos “Episodios” y guardamos nuestra petición.**

De esta forma hemos creado una petición que se va a llamar “Todos los episodios”, que va a servir para obtener una lista de episodios de esa API, y que la vamos a guardar en la colección “Episodios”. Así tenemos creada una colección en la que vamos a almacenar todas las peticiones que queramos almacenar en esa colección.



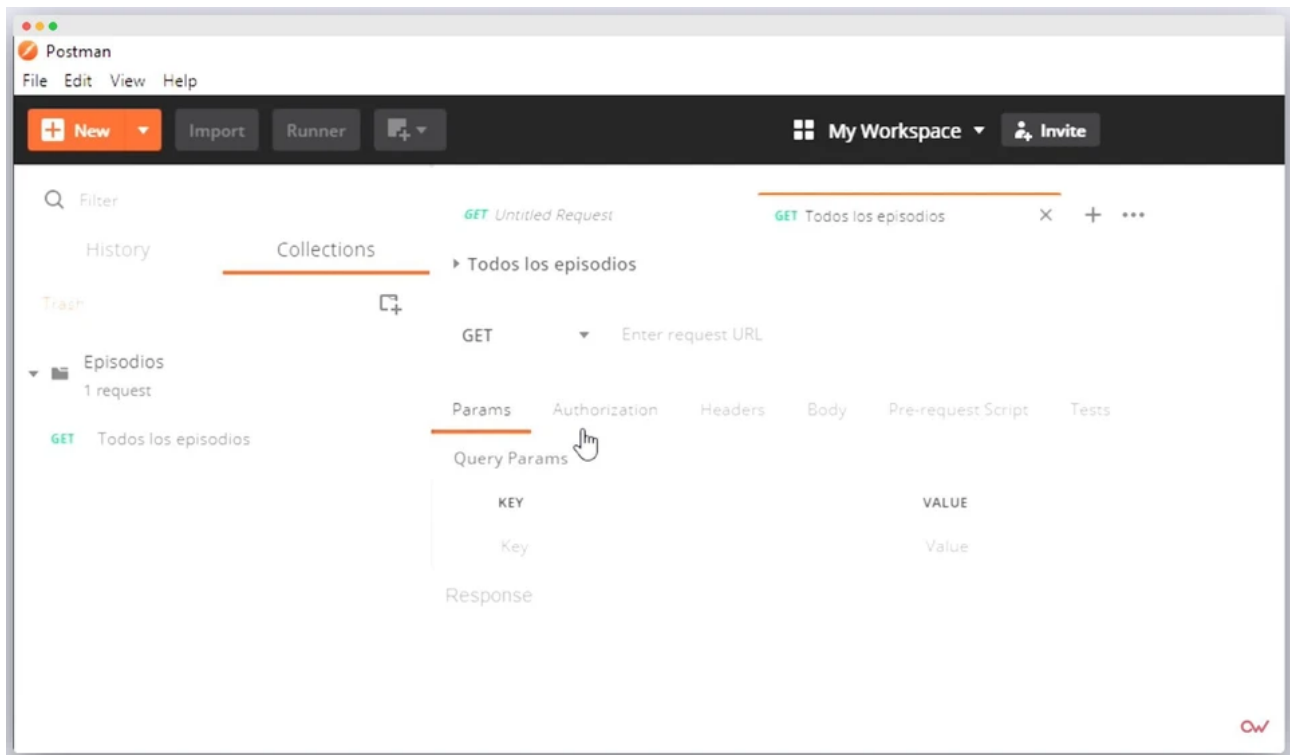
Podemos especificar el método HTTP que queremos utilizar, y vamos a definir la URL para acceder a ese endpoint.

También podemos modificar parámetros de query, métodos de autenticación, añadir headers y también añadir algún tipo de body, por ejemplo si quisiéramos hacer POST en lugar de GET, tendríamos que elegir raw y en Text añadir JSON.

Además se pueden introducir scripts que se ejecute antes de la petición o después de la petición, como por ejemplo, para poder validar parámetros, es decir, para poder hacer pruebas.

A pesar de que los scripts se definen en JavaScript, no hace falta un conocimiento extensivo de JavaScript para poder crear pruebas automatizadas en Postman, ya que se incluye la generación de diversos snippets de código para validar o hacer ciertas operaciones, como por ejemplo comprobar que el código de estado es 200.

Vamos a añadir nuestra petición, que quedaría de esta forma:

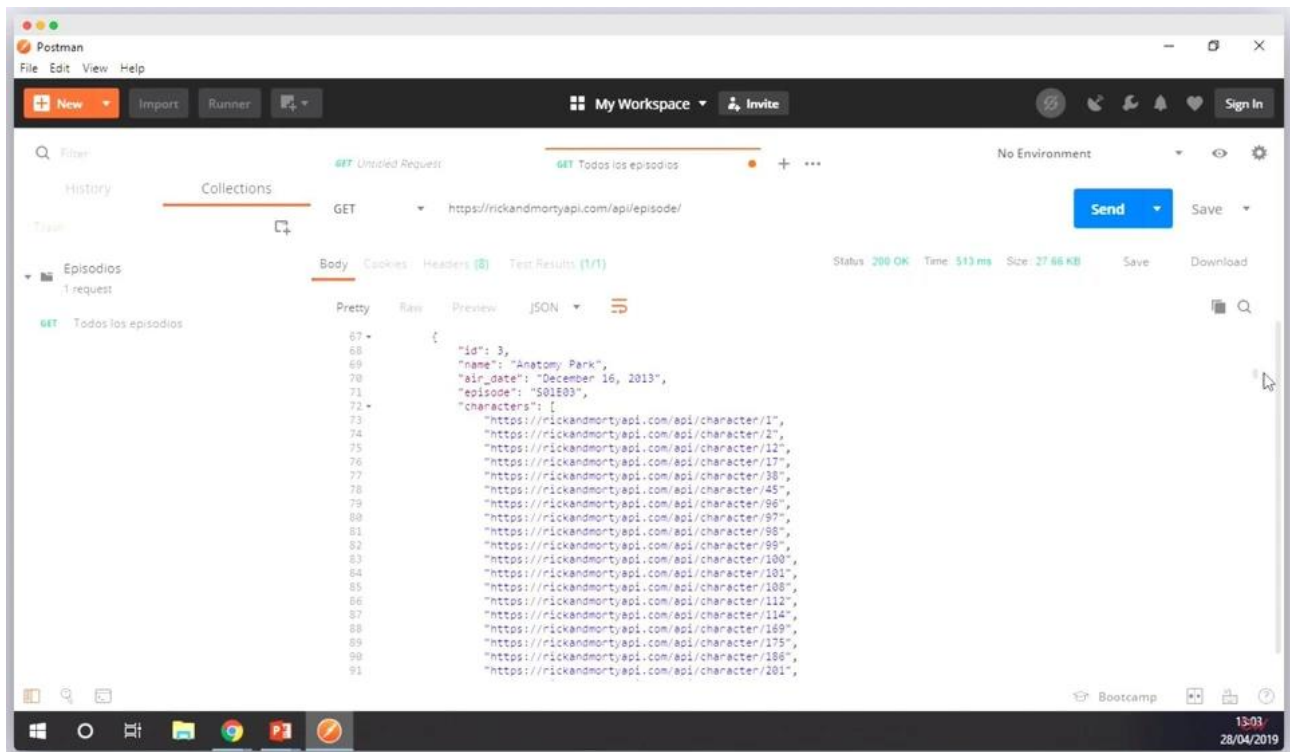


De esta forma enviaríamos una petición de tipo GET a ese PATH, bajo el protocolo HTTP, con el host rickyandmortyapi.com. Además añadimos unos parámetros adicionales que añade Postman y que se pueden configurar las opciones de la aplicación.

Pulsamos SEND para enviar la petición, y comprobamos que todo ha ido bien, porque el servidor nos ha devuelto un código 200, y podemos obtener la lista de los episodios, en base al modelo que podemos encontrar en la documentación de la API.

Podemos ver cada episodio con su título, la fecha en la que se emitió o los personajes que intervienen en el mismo. En esta lista de personajes, los mismos aparecen con el identificador con el que podemos consultarlos en la misma API en la sección de personajes.

También podemos ver los headers que nos devuelve el servidor, podemos comprobar el tiempo de respuesta, el tamaño de la petición del body o de los headers, además de una lista de los tests que hayamos definido a través de los scripts, que en nuestro caso muestra que ha pasado el código 200.



Zona de trabajo de Postman

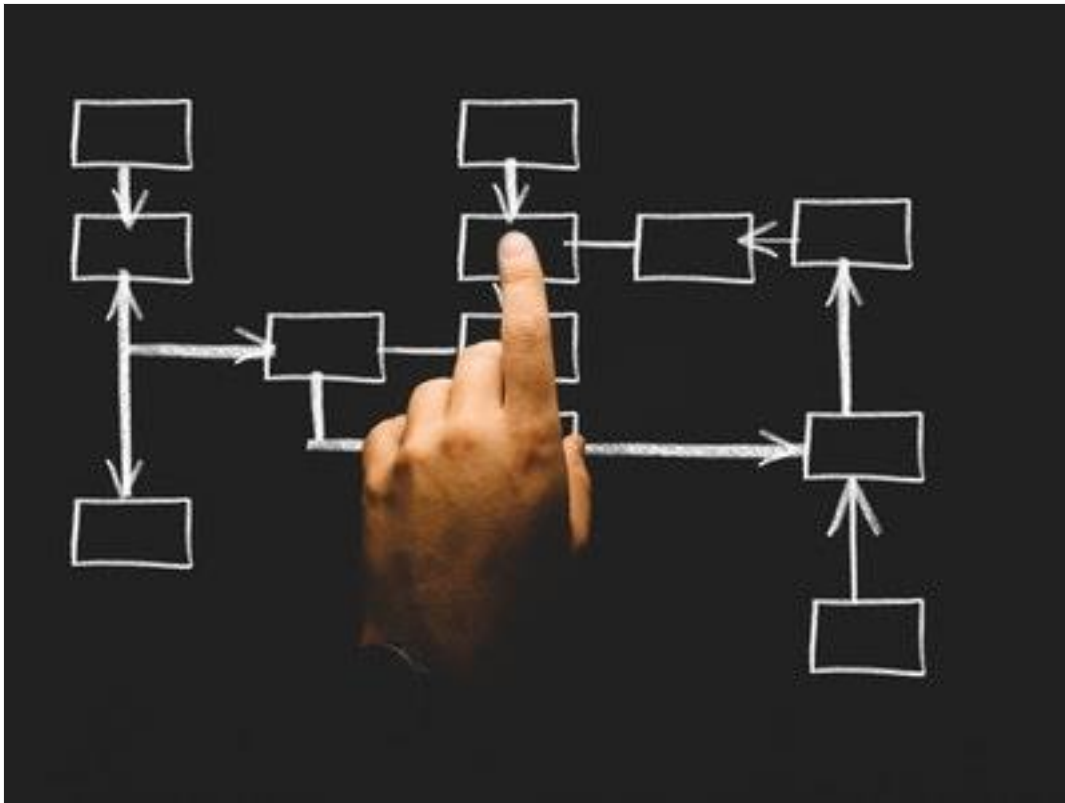
En la parte superior de Postman tenemos lo que sería la barra de tareas, **donde podemos crear nuevo contenido, nuevas colecciones y nuevas peticiones**. También podemos buscar las opciones para configurar la aplicación.

La zona izquierda es muy importante, porque **es donde vamos a definir las colecciones y las peticiones que se incluirán en esas colecciones**, que como dijimos antes, una colección es un elemento de Postman que contiene peticiones.

También podemos encontrar la colección History, que es el historial de todas las peticiones que hemos enviado, no importa a qué colección pertenezcan.

Y finalmente, en la sección central de Postman es dónde podremos interactuar con las peticiones. Aprende las bases del testing y cómo aplicarlas para probar APIs REST con **Postman**, uno de los clientes más utilizados actualmente con el que podrás consumir, probar, documentar e incluso simular APIs REST.

5 Herramientas de testing de servicios web



Los servicios web son un estándar en lo que se refiere al diseño y desarrollo de interfaces API entre aplicaciones de software empresarial, es por ello que existen diversas herramientas de testing de servicios web que brindan la capacidad a los testers de diseñar y ejecutar pruebas de software.

En este artículo te presentamos 5 herramientas para probar servicios web, específicamente SoapUI, JMeter, Postman, SoapSonar y Wizdler.

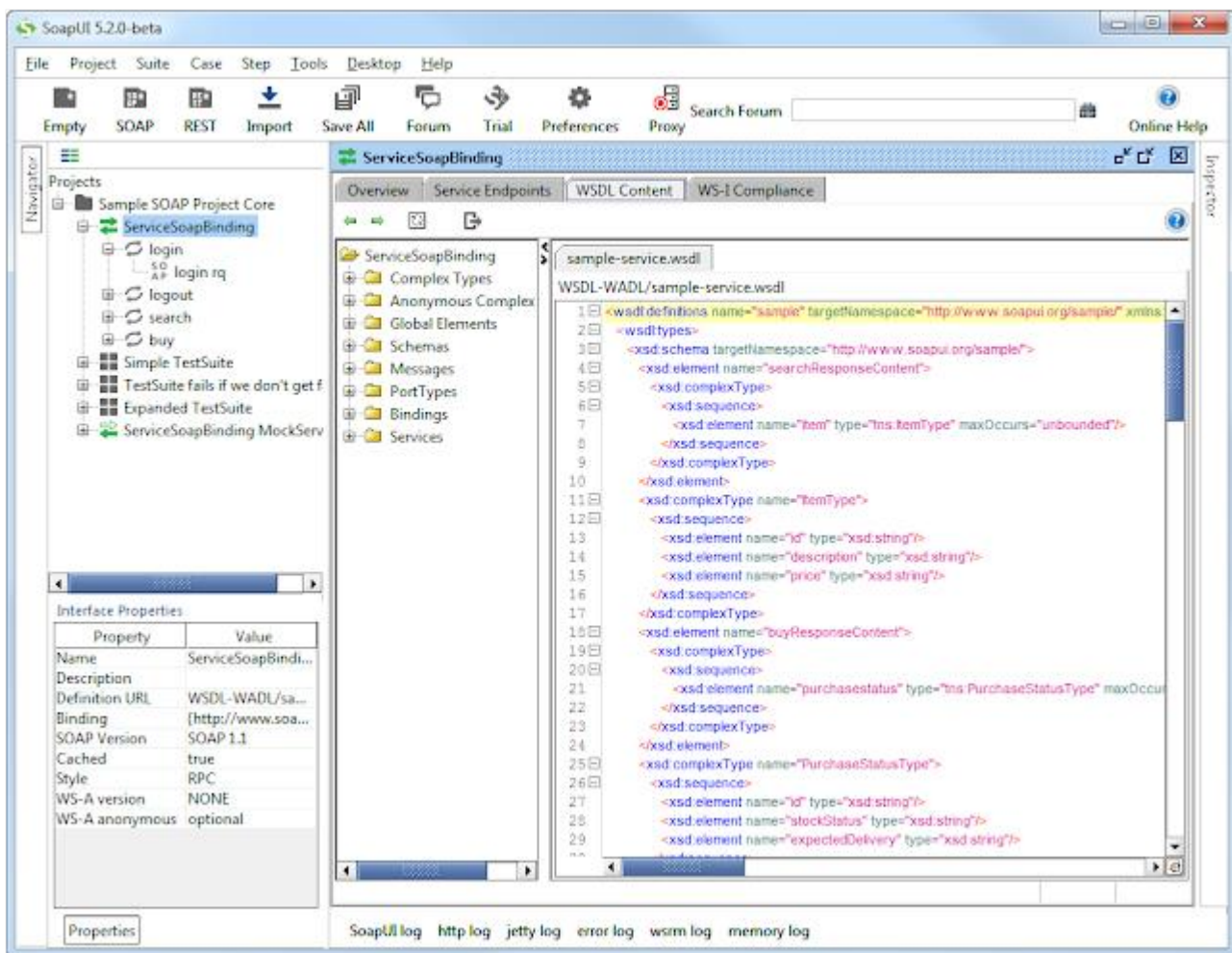
Estas herramientas proporcionan diversas capacidades para abarcar todos los tipos de pruebas de servicios web que se pueden realizar, tales como las pruebas funcionales, simulación de servicios (Mocking), evaluación de seguridad, pruebas de carga y de rendimiento.

SoapUI

SoapUI es una de las herramientas para probar servicios web que permite diseñar pruebas de web services de Arquitecturas SOA y REST. Está en el mercado desde 2005.

Con SoapUI puedes probar web services en los protocolos SOAP, REST, JMS y AMF, además, puedes realizar llamadas HTTP(S) para aplicaciones web y JDBC para bases de datos.

Posee una versión gratuita y una versión profesional (esta última tiene un costo).



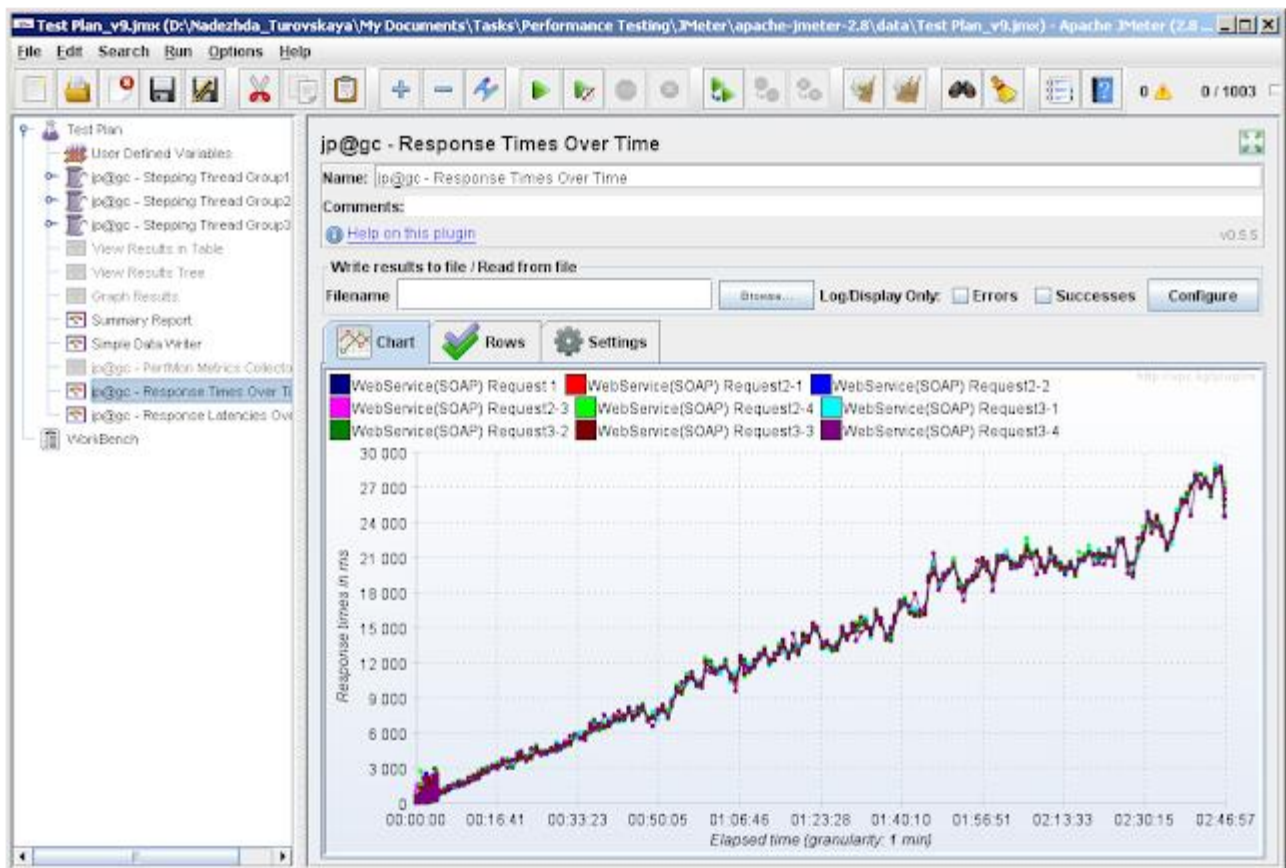
SoapUI permite la creación y ejecución automatizada de distintos **tipos de pruebas de software** sobre servicios web, incluyendo:

- ⑩ Testing funcional.
- ⑩ Pruebas de regresión.
- ⑩ Simulación de servicios (Mocking).
- ⑩ Pruebas de carga y rendimiento.
- ⑩ Evaluación de seguridad.

JMeter

Apache JMeter es una aplicación de código abierto diseñada para la ejecución de pruebas de carga y mediciones de desempeño en aplicaciones.

Es una herramienta desarrollada por la fundación de software Apache 100% en Java. Fue diseñada originalmente para probar aplicaciones web pero desde entonces se ha expandido para abarcar otras funcionalidades.



JMeter posee capacidades para realizar pruebas de carga y desempeño en diferentes aplicaciones, servidores y protocolos, incluyendo:

- ⑩ Web - HTTP, HTTPS (Java, NodeJS, PHP, ASP.NET, ...).
- ⑩ Servicios web SOAP / REST.
- ⑩ Servicios FTP.
- ⑩ Bases de datos, por medio de JDBC.
- ⑩ Servicios de directorio LDAP.
- ⑩ Middleware orientado a mensajes, por medio de JMS.
- ⑩ Correo electrónico - SMTP(S), POP3(S) e IMAP(S).
- ⑩ Comandos nativos o Shell Scripts.
- ⑩ Native commands or shell scripts.
- ⑩ Transmission Control Protocol (TCP).
- ⑩ Objetos Java.

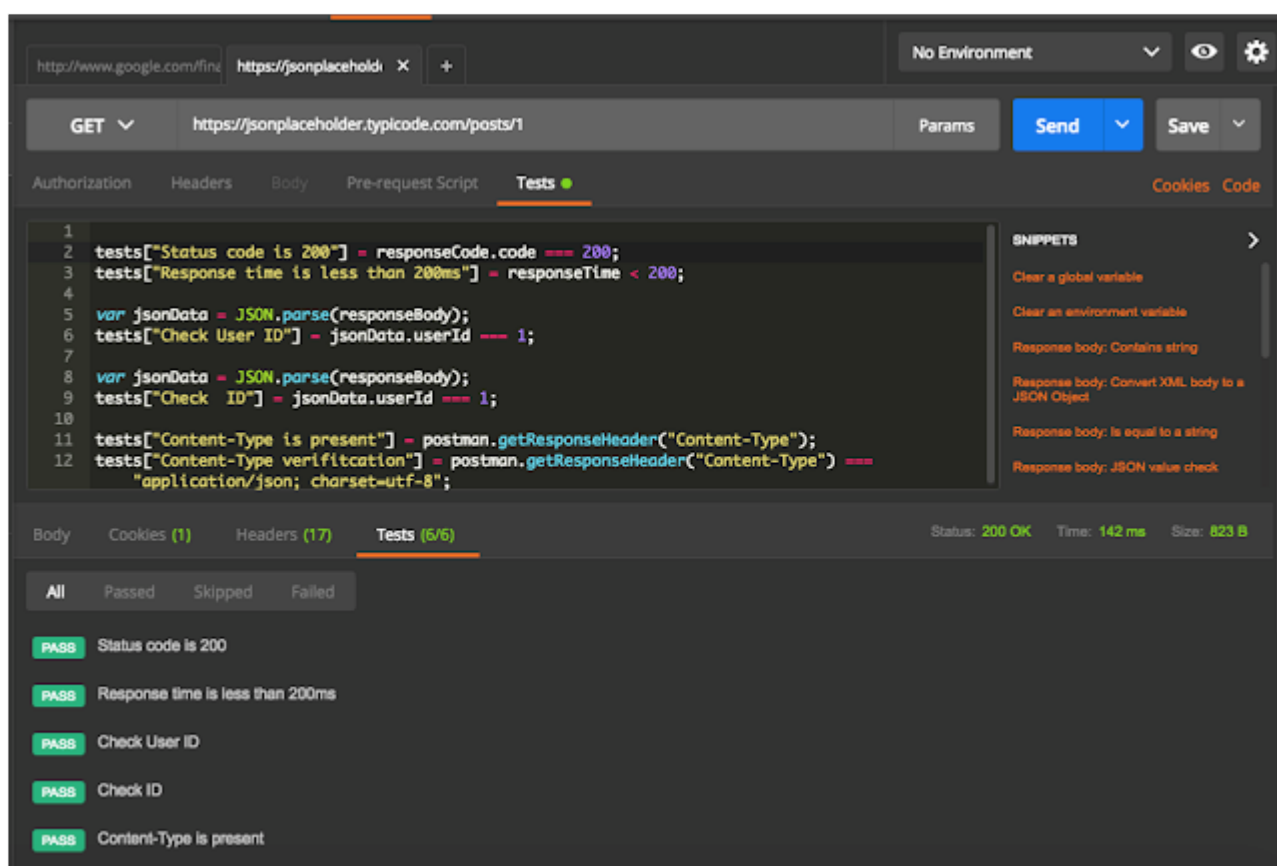


Postman

Postman es una herramienta para probar web services e interfaces de aplicación (API) en arquitectura REST. En términos generales, te permite enviar solicitudes post a cualquier servidor web y te proporciona y cataloga la respuesta.

Comenzó como un plugin del navegador Chrome, pero luego fue lanzada como versión nativa tanto para Windows como Mac.

Posee una versión gratuita, versión profesional y versión empresarial a diferentes costos.



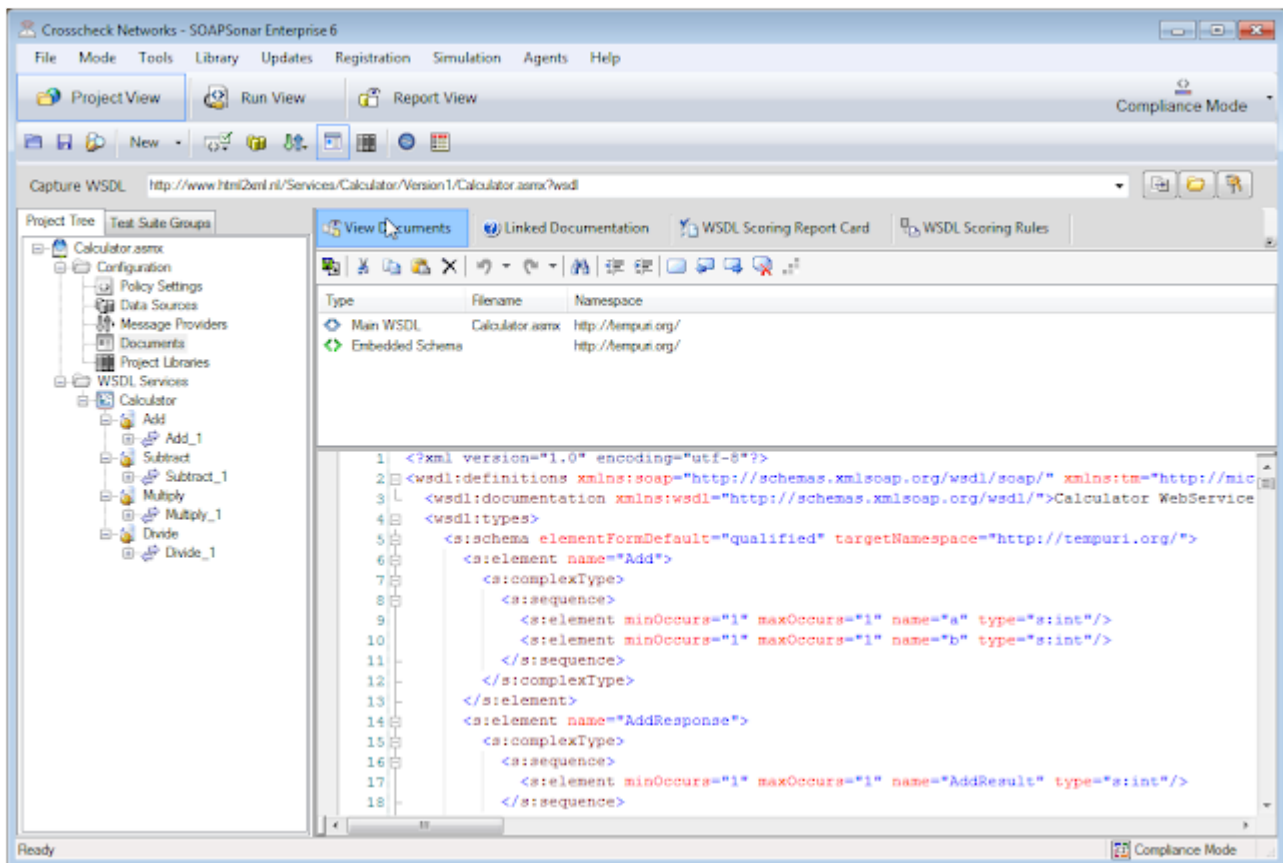
Entre sus funcionalidades se encuentran:

- ⑩ Se puede usar tanto para pruebas exploratorias como pruebas automatizadas.
- ⑩ Puede utilizarse en Windows, Mac, Linux y Apps del navegador Chrome.
- ⑩ Posee un conjunto de integraciones con formatos Swagger y RAML.
- ⑩ Funcionalidades para ejecutar, probar, documentar y monitorear las pruebas.
- ⑩ No requiere aprender nuevos lenguajes de programación.

SoapSonar

SOAPSonar es una herramienta que ofrece facilidades para pruebas web services para HTML, XML, SOAP, REST y JSON. Permite realizar pruebas funcionales, de desempeño, cumplimiento, interoperabilidad y de seguridad. Soporta los estándares OASIS y W3C.

Desarrollada por Cross Check Networks, está disponible en una edición personal (gratuita), edición profesional y edición de servidor, a distintos costos.



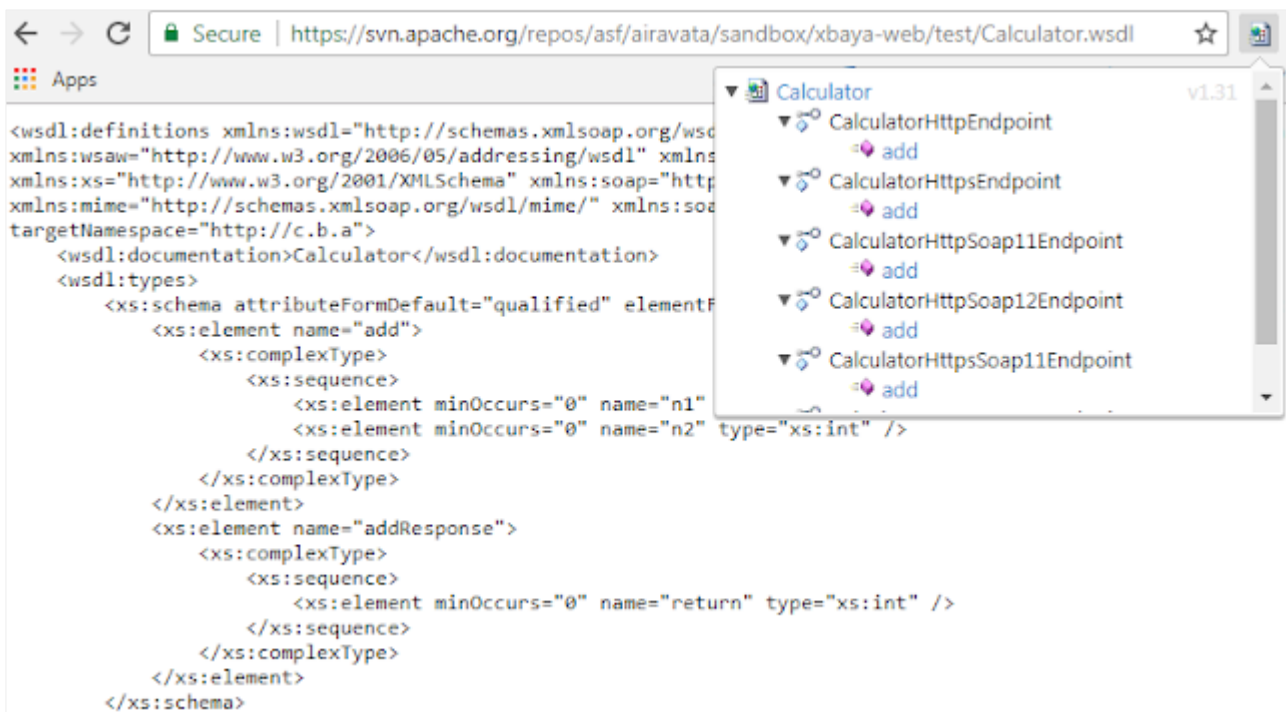
Entre sus funcionalidades están las siguientes:

- ⑩ Puede realizar pruebas de vulnerabilidad con mutación XSD.
- ⑩ Funcionalidades para leer WSDL y Schema.
- ⑩ Pruebas no funcionales como son las de carga con modelamiento de comportamiento.
- ⑩ Carga de transacciones simultaneas (Para pruebas de carga).
- ⑩ Reportes en formato XML, DOC, XLS, PDF, RTF y RPT.
- ⑩ Integración con HP Quality Center.

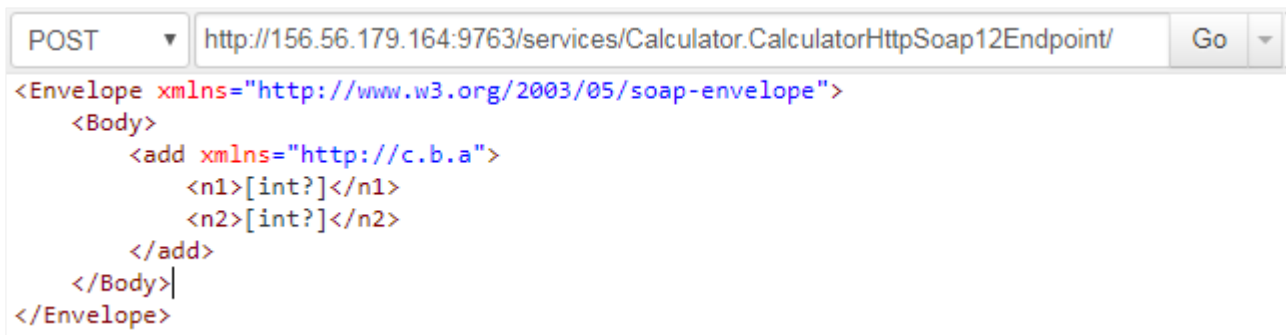
Wizdler

Si no necesitas todas las funcionalidades avanzadas, y más bien buscas algo sencillo, puedes probar Wizdler, una herramienta de testing de web services que soporta únicamente arquitectura SOA, presentada como extensión de navegador y disponible para Google Chrome y Mozilla Firefox.

Usarla es muy sencillo, luego de instalado el plugin, ingresa la dirección WSDL en la dirección de página del navegador, Wizdler la reconoce y lista todas las operaciones y servicios.



Luego haces click sobre cualquier operación y el XML de request se mostrará en el editor SOAP.



Modifica para pasar los parámetros y haz click en “Go”, para ver el Response en el editor. Wizzler puede de esta forma generar el XML de Request Soap.

Métodos de petición HTTP

HTTP define un conjunto de métodos de petición para indicar la acción que se desea realizar para un recurso determinado. Aunque estos también pueden ser sustantivos, estos métodos de solicitud a veces son llamados HTTP verbs. Cada uno de ellos implementan una semántica diferente, pero algunas características similares son compartidas por un grupo de ellos: ej. un request method puede ser [safe](#), [idempotent \(en-US\)](#), o [cacheable](#).

GET

El método `GET` solicita una representación de un recurso específico. Las peticiones que usan el método `GET` sólo deben recuperar datos.

HEAD

El método `HEAD` pide una respuesta idéntica a la de una petición `GET`, pero sin el cuerpo de la respuesta.

POST

El método `POST` se utiliza para enviar una entidad a un recurso en específico, causando a menudo un cambio en el estado o efectos secundarios en el servidor.

PUT

El modo `PUT` reemplaza todas las representaciones actuales del recurso de destino con la carga útil de la petición.

DELETE

El método `DELETE` borra un recurso en específico.

CONNECT

El método `CONNECT` establece un túnel hacia el servidor identificado por el recurso.

OPTIONS

El método `OPTIONS` es utilizado para describir las opciones de comunicación para el recurso de destino.

TRACE

El método `TRACE` realiza una prueba de bucle de retorno de mensaje a lo largo de la ruta al recurso de destino.

PATCH

El método `PATCH` es utilizado para aplicar modificaciones parciales a un recurso.

Configurando Nodemon

En los siguientes pasos veremos como instalar y configurar Nodemon en tu proyecto y como poner a correr un servidor Node.js.

Paso 1

```
{
  "name": "nodemon",
  "version": "1.0.0",
  "description": "",
  "main": "server.js",
  "scripts": {
    "start": "node src/server.js"
  },
  "keywords": [],
  "author": "Francisco Suarez <imanolsuarez98@gmail.com>",
  "license": "MIT"
}
```

Organizar el directorio de origen src y iniciar el servidor en un archivo server.js, el archivo puede llevar cualquier convención que se utilice para arrancar un servidor Node.js (index.js o app.js)

Actualizar el package.json agregando un script start

Paso 2

```
'use strict';

const express = require('express');
const app = express();

app.use('/', (req, res) => {
  res.status(200).send('La API funciona correctamente');
});

app.listen(3000);
```

Agregar express el cual nos va a permitir arrancar un mínimo servidor para realizar esta prueba server.js

Iniciá una terminal nueva en la cual iniciaremos el servidor corriendo el script npm start luego de ejecutarlo nos retornara un mensaje como el siguiente node src/index.js


Abrir una nueva terminal y ejecutaremos el siguiente código curl -X GET http://localhost:3000/ el cual nos permitirá comprobar que la API este funcionando de manera correcta.

A terminal window with a dark background and three colored window control buttons (red, yellow, green) in the top left corner. It displays the command `$ curl -X GET http://localhost:3000/` and the output `La API funciona correctamente!`.

```
$ curl -X GET http://localhost:3000/  
La API funciona correctamente!
```

Si nos retorna el mensaje La API funciona correctamente quiere decir que vamos bien!

Ahora, si cambiamos el mensaje de respuesta en el archivo server.js, debo reiniciar el servidor para obtener el resultado deseado:

A terminal window with a dark background and three colored window control buttons (red, yellow, green) in the top left corner. It displays the following JavaScript code snippet:

```
app.use('/', (req, res) => {  
  res.status(200).send('Lorem Ipsum');  
});
```

Utilizar Ctrl + C para detener el servidor que se está ejecutando actualmente y volver a iniciarlo usando el mismo comando antes: npm run start.

Usando el comando curl nuevamente desde la ventana de terminal obtenemos el resultado deseado:



```
curl -X GET http://localhost:3000/  
Lorem Ipsum
```

Paso 3

Agrega nodemon como devDependency:



```
$ npm i -D nodemon
```

Revisaremos el package.json

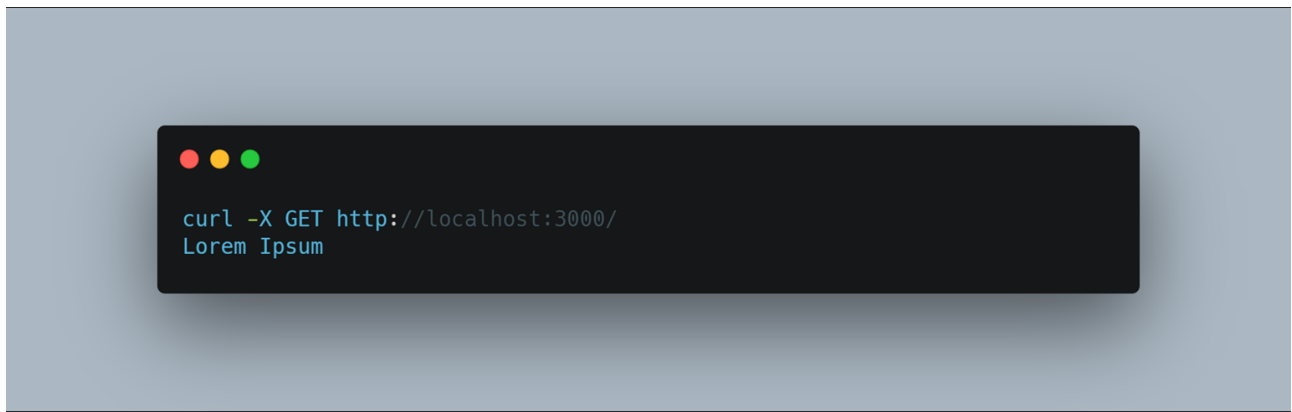
Paso 4

Agregar el comando dev en el archivo package.json

```
"name": "nodemon",
"version": "1.0.0",
"description": "",
"main": "server.js",
"scripts": {
  "start": "node src/server.js"
},
"keywords": [],
"author": "Francisco Suarez <imanolsuarez98@gmail.com>",
"license": "MIT",
"dependencies": {
  "express": "4.15.3"
},
"devDependencies": {
  "nodemon": "1.11.0"
}
}
```

```
"scripts": {
  "start": "node src/server.js",
  "dev": "nodemon --watch src src/server.js"
}
}
```

Ahora ejecute `npm run dev` y solicite el uso nuevamente del comando `curl`, y veremos que el mensaje es el mismo que teníamos antes:



Si cambio nuevamente el mensaje en el archivo `server.js` por cualquier otro, ya esta vez no tendré que reiniciar el servidor dado que Nodemon esta observando los cambios usando el directorio `src`, mediante su parámetro `--watch`.

Vera que se actualiza solo sin tener que reiniciar el servidor, para cortarlo presionar `CTRL + C`