



Postman



<https://www.postman.com>

© JMA 2020. All rights reserved

INTRODUCCIÓN

© JMA 2020. All rights reserved

Introducción

- El software generado en la fase de implementación no puede ser "entregado" al cliente para que lo utilice, sin practicarle antes una serie de pruebas.
- La fase de pruebas tienen como objetivo encontrar defectos en el sistema final debido a la omisión o mala interpretación de alguna parte del análisis o el diseño. Los defectos deberán entonces detectarse y corregirse en esta fase del proyecto.
- En ocasiones los defectos pueden deberse a errores en la implementación de código (errores propios del lenguaje o sistema de implementación), aunque en esta etapa es posible realizar una efectiva detección de los mismos, estos deben ser detectados y corregidos en la fase de implementación.
- La prueba puede ser llevada a cabo durante la implementación, para verificar que el software se comporta como su diseñador pretendía, y después de que la implementación esté completa.

© JMA 2020. All rights reserved

Principios fundamentales

- Hay 6 principios fundamentales respecto a las metodologías de pruebas que deben quedar claros desde el primer momento aunque volveremos a ellos continuamente:
 - Las pruebas exhaustivas no son viables
 - Ejecución de pruebas bajo diferentes condiciones
 - El proceso de pruebas no puede demostrar la ausencia de defectos
 - Las pruebas no garantizan ni mejoran la calidad del software
 - Las pruebas tienen un coste
 - Inicio temprano de pruebas

© JMA 2020. All rights reserved

V & V

- Validación: ¿Estamos construyendo el sistema correcto?
 - Proceso de evaluación de un sistema o componente durante o al final del proceso de desarrollo para comprobar si se satisfacen los requisitos especificados (IEEE Std610.12-1990)
- Verificación: ¿Estamos construyendo correctamente el sistema?
 - Proceso de evaluar un sistema o componente para determinar si los productos obtenidos en una determinada fase de desarrollo satisfacen las condiciones impuestas al comienzo de dicha fase (IEEE Std610.12-1990)

© JMA 2020. All rights reserved

Error, defecto o fallo

- En el área del aseguramiento de la calidad del software, debemos tener claros los conceptos de Error, Defecto y Fallo. En muchos casos se utilizan indistintamente pero representan conceptos diferentes:
 - Error: Es una acción humana, una idea equivocada de algo, que produce un resultado incorrecto. Es una equivocación por parte del desarrollador o analista.
 - Defecto: Es una imperfección de un componente causado por un error. El defecto se encuentra en algún componente del sistema. El analista de pruebas es quien debe encontrar el defecto ya que es el encargado de elaborar y ejecutar los casos de prueba.
 - Fallo: Es la manifestación visible de un defecto. Si un defecto es encontrado durante la ejecución de una aplicación entonces va a producir un fallo.
- Un error puede generar uno o más defectos y un defecto puede causar un fallo.

© JMA 2020. All rights reserved

Depuración y Pruebas

- Cuando se han encontrado fallos en un programa, éstos deben ser localizados y eliminados. A este proceso se le denomina depuración.
- La prueba de defectos y la depuración son consideradas a veces como parte del mismo proceso. En realidad, son muy diferentes, puesto que la prueba establece la existencia de fallos, mientras que la depuración se refiere a la localización los defectos/errores que se han manifestado en los fallos y corrección de los mismos.
- El proceso de depuración suele requerir los siguientes pasos:
 - Identificación de errores
 - Análisis de errores
 - Corrección y validación

© JMA 2020. All rights reserved

Clasificación de Pruebas

- Las actividades de las pruebas pueden centrarse en comprobar el sistema en base a un objetivo o motivo específico:
 - Una función a realizar por el software.
 - Una característica no funcional como el rendimiento o la fiabilidad.
 - La estructura o arquitectura del sistema o el software.
 - Los cambios para confirmar que se han solucionado los defectos o localizar los no intencionados.
- Las pruebas se pueden clasificar como:
 - Pruebas funcionales
 - Pruebas no funcionales
 - Pruebas estructurales
 - Pruebas de mantenimiento
- Las distintas clases de prueba utilizan clases de datos de prueba diferentes:
 - Prueba estadística
 - Prueba de defectos

© JMA 2020. All rights reserved

Prueba estadística

- La **prueba estadística** se puede utilizar para probar el rendimiento del programa y su confiabilidad.
- Las pruebas se diseñan para reflejar la frecuencia de entradas reales de usuario.
- Después de ejecutar las pruebas, se puede hacer una estimación de la confiabilidad operacional del sistema.
- El rendimiento del programa se puede juzgar midiendo la ejecución de las pruebas estadísticas.

© JMA 2020. All rights reserved

Prueba de defectos

- La **prueba de defectos** intenta incluir áreas donde el programa no está de acuerdo con sus especificaciones.
- Las pruebas se diseñan para revelar la presencia de defectos en el sistema que se manifiestan en forma de fallos.
- Los fallos demuestran la existencia de defectos que han sido ocasionados por errores.
- Cuando se han encontrado defectos en un programa, éstos deben ser localizados y eliminados. A este proceso se le denomina **depuración**.

© JMA 2020. All rights reserved

Pruebas de regresión

- Durante la depuración se debe generar hipótesis sobre el comportamiento observable del programa para probar entonces estas hipótesis esperando provocar un fallo y encontrar el defecto que causó la anomalía en la salida.
- Después de descubrir un error en el programa, debe corregirse y volver a probar el sistema.
- A esta forma de prueba se le denomina **prueba de regresión**.
- La prueba de regresión se utiliza para comprobar que los cambios hechos a un programa no han generado nuevos fallos en el sistema.

© JMA 2020. All rights reserved

Niveles de pruebas

- **Pruebas Unitarias o de Componentes:** verifican la funcionalidad y estructura de cada componente individualmente, una vez que ha sido codificado.
- **Pruebas de Integración:** verifican el correcto ensamblaje entre los distintos componentes una vez que han sido probados unitariamente, con el fin de comprobar que interactúan correctamente a través de sus interfaces, cubren la funcionalidad establecida y se ajustan a los requisitos no funcionales especificados en las verificaciones correspondientes.
- **Pruebas de Regresión:** verifican que los cambios sobre un componente de un sistema de información no introducen un comportamiento no deseado o errores adicionales en otros componentes no modificados.
- **Pruebas del Sistema:** ejercitan profundamente el sistema comprobando la integración del sistema de información globalmente, verificando el funcionamiento correcto de las interfaces entre los distintos subsistemas que lo componen y con el resto de sistemas de información con los que se comunica.
- **Pruebas de Aceptación:** validan que un sistema cumple con el funcionamiento esperado y permitir al usuario de dicho sistema, que determine su aceptación desde el punto de vista de su funcionalidad y rendimiento.

© JMA 2020. All rights reserved

Pruebas del Sistema

- Las pruebas del sistema tienen como objetivo ejercitar profundamente el sistema comprobando la integración del sistema de información globalmente, verificando el funcionamiento correcto de las interfaces entre los distintos subsistemas que lo componen y con el resto de sistemas de información con los que se comunica.
- Son pruebas de integración del sistema de información completo, y permiten probar el sistema en su conjunto y con otros sistemas con los que se relaciona para verificar que las especificaciones funcionales y técnicas se cumplen. Dan una visión muy similar a su comportamiento en el entorno de producción.
- Una vez que se han probado los componentes individuales y se han integrado, se prueba el sistema de forma global. En esta etapa pueden distinguirse diferentes tipos de pruebas, cada uno con un objetivo claramente diferenciado.

© JMA 2020. All rights reserved

Pruebas del Sistema

- **Pruebas funcionales:** dirigidas a asegurar que el sistema de información realiza correctamente todas las funciones que se han detallado en las especificaciones dadas por el usuario del sistema.
- **Pruebas de humo:** son un conjunto de pruebas aplicadas a cada nueva versión, su objetivo es validar que las funcionalidades básicas de la versión se cumplen según lo especificado. Impiden la ejecución el plan de pruebas si detectan grandes inestabilidades o si elementos clave faltan o son defectuosos.
- **Pruebas de comunicaciones:** determinan que las interfaces entre los componentes del sistema funcionan adecuadamente, tanto a través de dispositivos remotos, como locales. Asimismo, se han de probar las interfaces hombre-máquina.
- **Pruebas de rendimiento:** consisten en determinar que los tiempos de respuesta están dentro de los intervalos establecidos en las especificaciones del sistema.
- **Pruebas de volumen:** consisten en examinar el funcionamiento del sistema cuando está trabajando con grandes volúmenes de datos, simulando las cargas de trabajo esperadas.
- **Pruebas de sobrecarga o estrés:** consisten en comprobar el funcionamiento del sistema en el umbral límite de los recursos, sometiéndole a cargas masivas. El objetivo es establecer los puntos extremos en los cuales el sistema empieza a operar por debajo de los requisitos establecidos.

© JMA 2020. All rights reserved

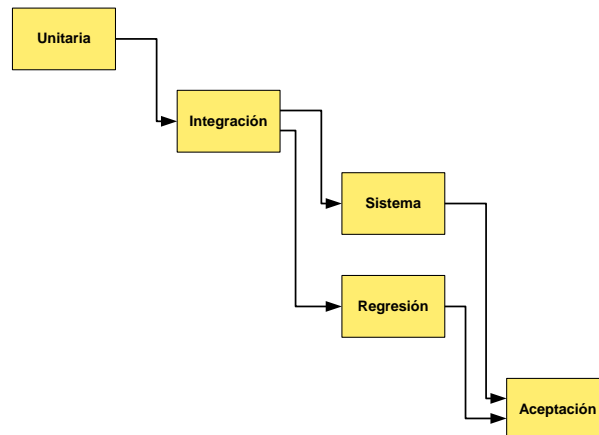
Pruebas del Sistema

- **Pruebas de disponibilidad de datos:** consisten en demostrar que el sistema puede recuperarse ante fallos, tanto de equipo físico como lógico, sin comprometer la integridad de los datos.
- **Pruebas de usabilidad:** consisten en comprobar la adaptabilidad del sistema a las necesidades de los usuarios, tanto para asegurar que se acomoda a su modo habitual de trabajo, como para determinar las facilidades que aporta al introducir datos en el sistema y obtener los resultados.
- **Pruebas extremo a extremo (e2e):** consisten en interactuar con la aplicación como un usuario regular lo haría, cliente-servidor, y evaluando las respuestas para el comportamiento esperado.
- **Pruebas de configuración:** consisten en comprobar todos y cada uno de los dispositivos, en sus propiedades mínimo y máximo posibles.
- **Pruebas de operación:** consisten en comprobar la correcta implementación de los procedimientos de operación, incluyendo la planificación y control de trabajos, arranque y re-arranque del sistema, etc.
- **Pruebas de entorno:** consisten en verificar las interacciones del sistema con otros sistemas dentro del mismo entorno.
- **Pruebas de seguridad:** consisten en verificar los mecanismos de control de acceso al sistema para evitar alteraciones indebidas en los datos.

© JMA 2020. All rights reserved

Niveles de pruebas y orden de ejecución.

- De tal forma que la secuencia de pruebas es:



© JMA 2020. All rights reserved

Prueba exploratoria

- Incluso los esfuerzos de automatización de pruebas más diligentes no son perfectos. A veces se pierden ciertos casos extremos en sus pruebas automatizadas. A veces es casi imposible detectar un error en particular escribiendo una prueba unitaria. Ciertos problemas de calidad ni siquiera se hacen evidentes en las pruebas automatizadas (como en el diseño o la usabilidad).
- Las pruebas exploratorias es un enfoque de prueba manual que enfatiza la libertad y creatividad del probador para detectar problemas de calidad en un sistema en ejecución.
 - Simplemente tome un tiempo en un horario regular, arremángate e intenta romper la aplicación.
 - Usa una mentalidad destructiva y encuentra formas de provocar problemas y errores en la aplicación.
 - Ten en cuenta los errores, los problemas de diseño, los tiempos de respuesta lentos, los mensajes de error faltantes o engañosos y, en general, todo lo que pueda molestarte como usuario de una aplicación.
 - Documenta todo lo que encuentre para más adelante.
- La buena noticia es que se puede automatizar la mayoría de los hallazgos con pruebas automatizadas.
- Escribir pruebas automatizadas para los errores que se detectan asegura que no habrá regresiones a ese error en el futuro. Además, ayuda a reducir la causa raíz de ese problema durante la corrección de errores.

© JMA 2020. All rights reserved

Análisis estático con herramientas

- El objetivo del análisis estático es detectar defectos en el código fuente y en los modelos de software.
- El análisis estático se realiza sin que la herramienta llegue a ejecutar el software objeto de la revisión, como ocurre en las pruebas dinámicas, centrándose mas en como está escrito el código que en como se ejecuta el código.
- El análisis estático permite identificar defectos difíciles de encontrar mediante pruebas dinámicas.
- Al igual que con las revisiones, el análisis estático encuentra defectos en lugar de fallos.
- Las herramientas de análisis estático analizan el código del programa (por ejemplo, el flujo de control y flujo de datos) y las salidas generadas (tales como HTML o XML).
- Algunos de los posibles aspectos que pueden ser comprobados con análisis estático:
 - Reglas, estándares de programación y buenas practicas.
 - Diseño de un programa (análisis de flujo de control).
 - Uso de datos (análisis del flujo de datos).
 - Complejidad de la estructura de un programa (métricas, por ejemplo valor ciclomático).

© JMA 2020. All rights reserved

Pirámide de pruebas



© JMA 2020. All rights reserved

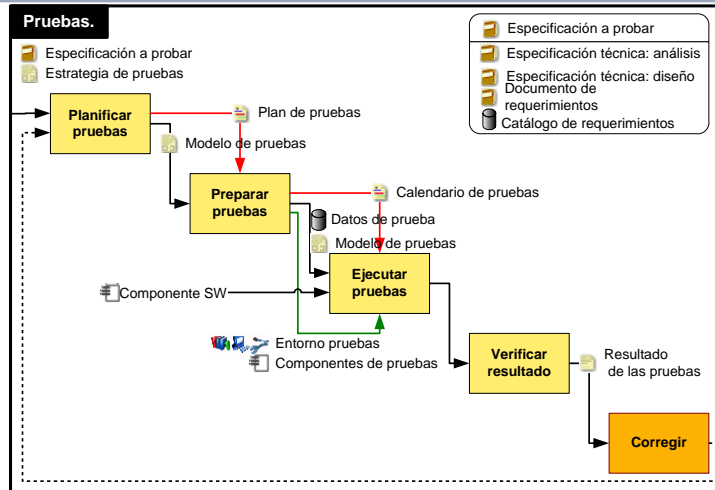
<https://martinfowler.com/bliki/TestPyramid.html>

Pruebas de mutaciones

- Las pruebas de mutaciones son las pruebas de las pruebas unitarias y el objetivo es tener una idea de la calidad de las pruebas en cuanto a fiabilidad.
- Su funcionamiento es relativamente sencillo: la herramienta que se utilice debe generar pequeños cambios en el código fuente. A estos pequeños cambios se les conoce como mutaciones y crean mutantes.
- Una vez creados los mutantes, se lanzan todos los tests:
 - Si los test unitarios fallan, es que han sido capaces de detectar ese cambio de código. En este caso el mutante se considera eliminado.
 - Si, por el contrario, los test unitarios pasan, el mutante sobrevive y la fiabilidad (y calidad) de los tests unitarios queda en entredicho.
- Los test de mutaciones presentan informes del porcentaje de mutantes detectados: cuanto más se acerque este porcentaje al 100%, mayor será la calidad de nuestros test unitarios.

© JMA 2020. All rights reserved

Metodología



Principios F.I.R.S.T.

- El principio FIRST fue definido por Robert Cecil Martin en su libro Clean Code. Este autor, entre otras muchas cosas, es conocido por ser uno de los escritores del Agile Manifesto, escrito hace más de 15 años y que a día de hoy se sigue teniendo muy en cuenta a la hora de desarrollar software.
 - Fast: Los tests deben ser rápidos, del orden de milisegundos, hay cientos de tests en un proyecto.
 - Isolated/Independent (Aislado/Independiente). Los tests no deben depender del entorno ni de ejecuciones de tests anteriores.
 - Repeatable. Los tests deben ser repetibles y ante la misma entrada de datos, los mismos resultados.
 - Self-Validating. Los tests tienen que ser autovalidados, es decir, NO debe de existir la intervención humana en la validación
 - Thorough and Timely (Completo y oportuno). Los tests deben de cubrir el escenario propuesto, no el 100% del código, y se han de realizar en el momento oportuno

Diseñar la prueba

- Para diseñar la prueba empiezas por identificar y describir los casos de prueba de cada componente.
- La selección de las técnicas de pruebas depende factores adicionales como pueden ser requisitos contractuales o normativos, documentación disponible, tiempo, presupuesto, conocimientos, experiencia, ...
- Cuando dispongas de los casos de prueba, identificas y estructuras los procedimientos de prueba describiendo cómo ejecutar los casos de prueba.

© JMA 2020. All rights reserved

Patrones

- Los casos de prueba se pueden estructurar siguiendo diferentes patrones:
 - ARRANGE-ACT-ASSERT: Preparar, Actuar, Afirmar
 - GIVEN-WHEN-THEN: Dado, Cuando, Entonces
 - BUILD-OPERATE-CHECK: Generar, Operar, Comprobar
- Aunque con diferencias conceptuales, todos dividen el proceso en tres fases:
 - Una fase inicial donde montar el escenario de pruebas que hace que el resultado sea predecible.
 - Una fase intermedia donde se realizan las acciones que son el objetivo de la prueba.
 - Una fase final donde se comparan los resultados con el escenario previsto. Pueden tomar la forma de:
 - Aserción: Es una afirmación sobre el resultado que puede ser cierta o no.
 - Expectativa: Es la expresión del resultado esperado que puede cumplirse o no.

© JMA 2020. All rights reserved

Preparación mínima

- La sección de preparación, con la entrada del caso de prueba, debe ser lo más sencilla posible, lo imprescindible para comprobar el comportamiento que se está probando.
- Las pruebas se hacen más resistentes a los cambios futuros en el código base y más cercano al comportamiento de prueba que a la implementación.
- Las pruebas que incluyen más información de la necesaria tienen una mayor posibilidad de incorporar errores en la prueba y pueden hacer confusa su intención. Al escribir pruebas, el usuario quiere centrarse en el comportamiento. El establecimiento de propiedades adicionales en los modelos o el empleo de valores distintos de cero cuando no es necesario solo resta de lo que se quiere probar.

© JMA 2020. All rights reserved

Actuación mínima

- Al escribir las pruebas hay que evitar introducir condiciones lógicas como if, switch, while, for, etc.
- Minimiza la posibilidad de incorporar un error a las pruebas.
- El foco está en el resultado final, en lugar de en los detalles de implementación.
- Al incorporar lógica al conjunto de pruebas, aumenta considerablemente la posibilidad de agregar un error. Cuando se produce un error en una prueba, se quiere saber realmente que algo va mal con el código probado y no en el código que prueba. En caso contrario, restan confianza y las pruebas en las que no se confía no aportan ningún valor.
- El objetivo de la prueba debe ser único, si la lógica en la prueba parece inevitable, denota que el objetivo es múltiple y hay que considerar la posibilidad de dividirla en dos o más pruebas diferentes.

© JMA 2020. All rights reserved

Comprobación mínima

- Al escribir las pruebas, hay que intentar comprobar una única cosa, es decir, incluir solo una aserción por prueba.
 - Si se produce un error en una aserción, no se evalúan las aserciones posteriores.
 - Garantiza que no se estén declarando varios casos en las pruebas.
 - Proporciona la imagen exacta de por qué se producen errores en las pruebas.
- Al incorporar varias aserciones en un caso de prueba, no se garantiza que se ejecuten todas. Es un todas o ninguna, se sabe por cual fallo pero no si el resto también falla o es correcto, proporcionando la imagen parcial.
- Una excepción común a esta regla es cuando la validación cubre varios aspectos. En este caso, suele ser aceptable que haya varias aserciones para asegurarse de que el resultado está en el estado que se espera que esté.
- Los enfoques comunes para usar solo una aserción incluyen:
 - Crear una prueba independiente para cada aserción.
 - Usar pruebas con parámetros.

© JMA 2020. All rights reserved

HERRAMIENTAS PARA PRUEBAS

© JMA 2020. All rights reserved

Ecosistema

- Alrededor de las pruebas y el aseguramiento de la calidad existe todo un ecosistema de herramientas que se utilizan en una o más actividades de soporte de prueba, entre las que se encuentran:
 - Las herramientas que se utilizan directamente en las pruebas, como las herramientas de ejecución de pruebas, las herramientas de generación de datos de prueba y las herramientas de comparación de resultados.
 - Las herramientas que ayudan a gestionar el proceso de pruebas, como las que sirven para gestionar pruebas, resultados de pruebas, datos requisitos, incidencias, defectos, etc., así como para elaborar informes y monitorizar la ejecución de pruebas.
 - Las herramientas que se utilizan en la fase de reconocimiento, como las herramientas de estrés y las herramientas que monitorizan y supervisan la actividad del sistema.
 - Cualquier otra herramienta que contribuya al proceso de pruebas sin ser específicas del mismo, como las hojas de cálculo, procesadores de texto, diagramadores, ...
- Las herramientas pueden clasificarse en base a distintos criterios, tales como el objetivo, comercial/código abierto, específicas/integradas, tecnología utilizada ...

© JMA 2020. All rights reserved

Objetivos

- Mejorar la eficiencia de las tareas de pruebas automatizando tareas repetitivas o dando soporte a las actividades de pruebas manuales, como la planificación, el diseño, la elaboración de informes y la monitorización de pruebas.
- Automatizar aquellas actividades que requieren muchos recursos si se hacen de forma manuales (como por ejemplo, las pruebas estáticas, pruebas de GUI).
- Automatizar aquellas actividades que no pueden ejecutarse de forma manual (como por ejemplo, pruebas de rendimiento a gran escala de aplicaciones cliente-servidor).
- Aumentar la fiabilidad de las pruebas (por ejemplo, automatizando las comparaciones de grandes ficheros de datos y simulando comportamientos).

© JMA 2020. All rights reserved

Ventajas

- Reducción del trabajo repetitivo (por ejemplo, la ejecución de pruebas de regresión, la reintroducción de los mismos datos de prueba y la comprobación contra estándares de codificación).
- Mayor consistencia y respetabilidad (por ejemplo las pruebas ejecutadas por una herramienta en el mismo orden y con la misma frecuencia, y pruebas derivadas de los requisitos).
- Evaluación de los objetivos (por ejemplo, medidas estáticas, cobertura).
- Facilidad de acceso a la información sobre las pruebas (por ejemplo, estadísticas y gráficos sobre el avance de las pruebas, la frecuencia de incidencias y el rendimiento).

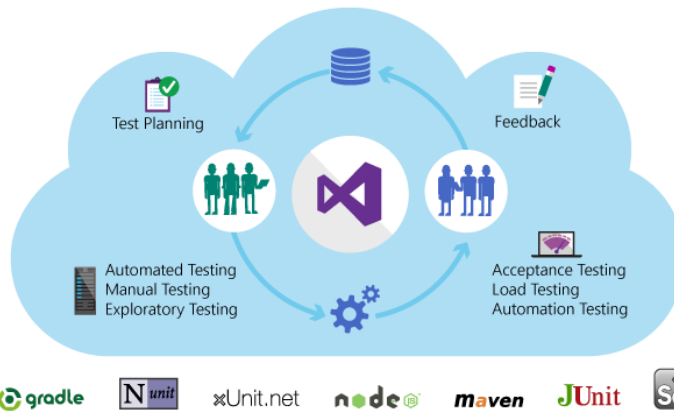
© JMA 2020. All rights reserved

Desventajas

- Expectativas poco realistas de la herramienta (incluyendo funcionalidad y facilidad de uso).
- Exceso de confianza en la herramienta (sustitución por el diseño de pruebas o uso de pruebas automatizadas cuando sería mejor llevar a cabo pruebas manuales).
- Subestimar:
 - La cantidad de tiempo, coste y esfuerzo necesario para la introducción inicial de una herramienta (incluyendo formación y experiencia externa).
 - El tiempo y el esfuerzo necesarios para conseguir ventajas significativas y constantes de la herramienta (incluyendo la necesidad de cambios en el proceso de pruebas y la mejora continua de la forma en la que se utiliza la herramienta).
 - El esfuerzo necesario para mantener los activos de prueba generados por la herramienta.
- Desprecio del control de versión de los activos de prueba en la herramienta.
- Desprecio de problemas de relaciones e interoperabilidad entre herramientas críticas tales como las herramientas de gestión de requisitos, herramientas de control de versiones, herramientas de gestión de incidencias, herramientas de seguimiento de defectos y herramientas procedentes de varios fabricantes.
- Coste de las herramientas comerciales o ausencia de garantías en las herramientas open source.
- Riesgo de que el fabricante de la herramienta cierre, retire la herramienta o venda la herramienta a otro proveedor.
- Mala respuesta del fabricante para soporte, actualizaciones y corrección de defectos.
- Imprevistos, tales como la incapacidad de soportar una nueva plataforma.

© JMA 2020. All rights reserved

Microsoft DevOps



© JMA 2020. All rights reserved

Ecosistema de pruebas en Java



© JMA 2020. All rights reserved

TestLink

- <http://testlink.org/>
- TestLink es una herramienta web de gestión de pruebas que ayuda a gestionar las pruebas funcionales de un proyecto permitiendo realizar las tareas relacionadas con el proceso de aseguramiento de calidad de software tales como: gestión de requerimientos, diseño de casos de prueba, planes de pruebas, ejecución de casos de prueba, seguimiento de informes de resultados del proceso de pruebas.
- Es una herramienta gratuita que permite crear y gestionar casos de pruebas y organizarlos en planes de prueba. Estos planes permiten a los miembros del equipo ejecutar casos de test y registrar los resultados dinámicamente, generar informes, mantener la trazabilidad con los requerimientos, así como priorizar y asignar tareas.
- Permite gestionar tantos proyectos de pruebas como sean necesarios, accesibles con diferentes roles de usuario con distintos permisos para los integrantes de un equipo de desarrollo.
- Los proyectos pueden definir diferentes plataformas de pruebas y mantener un inventario de las mismas.

© JMA 2020. All rights reserved

Mantis

- <https://www.mantisbt.org/>
- Mantis Bug Tracker es un sistema de gestión de incidencias, control de cambios y control de errores, es una aplicación web OpenSource multiplataforma que permite gestionar las incidencias de la empresa, sistemas o proyectos.
- Es un sistema fácil de usar y adaptable a muchos escenarios, tanto para incidencias técnicas, peticiones de soporte o bugs de un sistema.
- Es una aplicación realizada con php y mysql que destaca por su facilidad y flexibilidad de instalar y configurar.
- Mantis es una aplicación que permite a distintos usuarios crear tickets de cualquier tipo.
- A la hora de notificar una incidencia, el usuario tiene muchas opciones y campos a rellenar con el fin de hacer más fácil el trabajo del encargado de resolver el ticket.
- Mantis permite notificar a los usuarios novedades por correo electrónico.
- Se puede especificar un número indeterminado de estados para cada tarea (nueva, se necesitan más datos, aceptada, confirmada, asignada, resuelta, cerrada) y configurar la transición de estados.
- Permite la carga de plugins específicos de la plataforma que añaden funcionalidades extra.

© JMA 2020. All rights reserved

Selenium

- <http://www.seleniumhq.org/>
- El Selenium es un conjunto de herramientas para automatizar los navegadores web, robot que simula la interacción del usuario con el navegador, originalmente pensado como entorno de pruebas de software para aplicaciones basadas en la web.
- Como principales herramientas Selenium cuenta con:
 - Selenium IDE: una herramienta para grabar y reproducir secuencias de acciones con el navegador que permite crear pruebas sin usar un lenguaje de scripting para pruebas.
 - Selenium Core: API para escribir pruebas automatizadas y de regresión en un amplio número de lenguajes de programación populares incluyendo Java, C#, Ruby, Groovy, Perl, Php y Python.
 - WebDriver: interfaces que permite ejecutar las pruebas de forma nativa usando la mayoría de los navegadores web modernos en diferentes sistemas operativos como Windows, Linux y OSX.
 - Selenium Grid: Permite ejecutar muchas pruebas de un mismo grupo en paralelo o pruebas en múltiples entornos. Tiene la ventaja que un conjunto de pruebas muy grande puede dividirse en varias maquinas remotas para una ejecución más rápida o si se necesitan repetir las mismas pruebas en múltiples entornos.

© JMA 2020. All rights reserved

Postman

- <https://www.postman.com/>
- Postman surgió originariamente como una extensión para el navegador Google Chrome que permitía realizar peticiones API REST con métodos diferentes al GET. A día de hoy dispone de aplicaciones nativas para MAC, Windows y algunas producciones Linux.
- Está compuesto por diferentes herramientas y utilidades gratuitas (en la versión free) que permiten realizar tareas diferentes dentro del mundo API REST: creación de peticiones a APIs internas o de terceros, elaboración de tests para validar el comportamiento de APIs, posibilidad de crear entornos de trabajo diferentes (con variables globales y locales), y todo ello con la posibilidad de ser compartido con otros compañeros del equipo de manera gratuita (exportación de toda esta información mediante URL en formato JSON).
- Además, dispone de un modo cloud colaborativo (de pago) para que equipos de trabajo puedan desarrollar entre todos colecciones para APIs sincronizadas en la nube para una integración más inmediata y sincronizada.
- Quizás sea una de las herramientas más utilizadas para hacer testing exploratorio de API REST.

© JMA 2020. All rights reserved

SoapUI

<https://www.soapui.org>

- SoapUI es una herramienta para probar servicios web que pueden ser servicios web SOAP, servicios web RESTful u otros servicios basados en HTTP.
- SoapUI es una herramienta de código abierto y completamente gratuita con una versión comercial, SoapUI Pro, que tiene una funcionalidad adicional para empresas con servicios web de misión crítica.
- SoapUI se considera el estándar de facto para las Pruebas de servicio API. Esto significa que hay mucho conocimiento en la red sobre la herramienta y blogs para obtener más información sobre el uso de SoapUI en la vida real.
- SoapUI permite realizar pruebas funcionales, pruebas de rendimiento, pruebas de interoperabilidad, pruebas de regresión y mucho más. Su objetivo es que la prueba sea bastante fácil de comenzar, por ejemplo, para crear una Prueba de carga, simplemente hay que hacer clic derecho en una prueba funcional y ejecutarla como una prueba de carga.
- SoapUI puede simular servicios web (mocking). Se puede grabar pruebas y usarlas más tarde.

© JMA 2020. All rights reserved

JMeter

<http://jmeter.apache.org>

- JMeter es una herramienta de pruebas de carga para llevar acabo simulaciones sobre cualquier recurso de Software.
- Inicialmente diseñada para pruebas de estrés en aplicaciones web, hoy en día, su arquitectura ha evolucionado no sólo para llevar acabo pruebas en componentes habilitados en Internet (HTTP), sino también en Bases de Datos , programas en Perl, peticiones FTP y prácticamente cualquier otro medio.
- Además, posee la capacidad de realizar desde una solicitud sencilla hasta secuencias de peticiones que permiten diagnosticar el comportamiento de una aplicación en condiciones de producción.
- En este sentido, simula todas las funcionalidades de un navegador, o de cualquier otro cliente, siendo capaz de manipular resultados en determinada requisición y reutilizarlos para ser empleados en una nueva secuencia.

© JMA 2020. All rights reserved

Sonar

- <http://www.sonarqube.org/>
- SonarQube (conocido anteriormente como Sonar) es una plataforma para la revisión y evaluación del código fuente.
- Es open source y realiza el análisis estático de código fuente integrando las mejores herramientas de medición de la calidad de código como Checkstyle, PMD o FindBugs, para obtener métricas que pueden ayudar a mejorar la calidad del código de un programa.
- Informa sobre código duplicado, estándares de codificación, pruebas unitarias, cobertura de código, complejidad ciclomática, posible errores, comentarios y diseño del software.
- Aunque pensado para Java, acepta mas de 20 lenguajes mediante extensiones.
- Se integra con Maven, Ant y herramientas de integración continua como Atlassian Bamboo, Jenkins y Hudson).

© JMA 2020. All rights reserved

Análisis estático Web

- HTML
 - <https://validator.w3.org/>
- CSS
 - <http://jigsaw.w3.org/css-validator/>
- WAI
 - <https://www.w3.org/WAI/ER/tools/>
- JavaScript
 - <http://jshint.com/>
 - <http://www.jshint.com/>
- Chrome Dev Tools

© JMA 2020. All rights reserved

ESTILOS DE COMUNICACIÓN

© JMA 2020. All rights reserved

Estilos de comunicación

- El cliente y los servicios, o los servicios entre si, pueden comunicarse a través de muchos tipos diferentes de comunicación, cada uno destinado a un escenario y unos objetivos distintos. Inicialmente, estos tipos de comunicaciones se pueden clasificar por diferentes criterios.
- El primer criterio define si el proceso es sincrónico o asincrónico:
 - Protocolo síncrono: HTTP es el protocolo síncrono mas utilizado. El cliente envía una solicitud y espera una respuesta del servicio, solo puede continuar su tarea cuando recibe la respuesta del servidor. Es independiente de la ejecución de código de cliente, que puede ser síncrono (el subproceso está bloqueado) o asíncrono (el subproceso no está bloqueado y la respuesta dispara una devolución de llamada).
 - Protocolo asíncrono: Otros protocolos como AMQP (un protocolo compatible con muchos sistemas operativos y entornos de nube) usan mensajes asíncronos. Normalmente el código de cliente o el remitente del mensaje no espera ninguna respuesta. Simplemente se envía el mensaje a una cola de un agente de mensajes, que son escuchadas por los consumidores.
- El segundo criterio define si la comunicación tiene un único receptor o varios:
 - Receptor único 1:1 (comando, punto a punto, P2P): Cada solicitud debe ser procesada por un receptor o servicio exactamente (como en el patrón Command).
 - Varios receptores 0:N (eventos, publicación/subscription, Pub/Sub): Cada solicitud puede ser procesada por entre cero y varios receptores. Este tipo de comunicación debe ser asíncrona (basada en un bus de eventos o un agente de mensajes).

© JMA 2020. All rights reserved

Criterio según su base

- **Basados en Recursos:** Los servicios exponen información, documentos que incluyen tanto identificadores de datos como de acciones (enlaces y formularios), las operaciones CRUD están predefinidas. REST es un estilo arquitectónico que separa las preocupaciones del consumidor y del proveedor de la API al depender de comandos que están integrados en el protocolo de red subyacente. REST (Representational State Transfer) es extremadamente flexible en el formato de sus cargas útiles de datos, lo que permite una variedad de formatos de datos populares como JSON y XML, entre otros.
- **Basados en Procedimientos:** Las llamadas a procedimiento remoto, o RPC, generalmente requieren que los desarrolladores ejecuten bloques específicos de código en otro sistema: operaciones. RPC es independiente del protocolo, lo que significa que tiene el potencial de ser compatible con muchos protocolos, pero también pierde los beneficios de usar capacidades de protocolo nativo (por ejemplo, almacenamiento en caché). La utilización de diferentes estándares da como resultado un acoplamiento más estrecho entre los consumidores y los proveedores de API y las tecnologías implicadas, lo que a su vez sobrecarga a los desarrolladores involucrados en todos los aspectos de un ecosistema de APIs impulsado por RPC. Los patrones de arquitectura de RPC se pueden observar en tecnologías API populares como SOAP, GraphQL y gRPC.
- **Basados en Eventos/Streaming:** a veces denominadas arquitecturas de eventos, en tiempo real, de transmisión, asíncronas o push, las APIs impulsadas por eventos no esperan a que un consumidor de la API las llame antes de entregar una respuesta. En cambio, una respuesta se desencadena por la ocurrencia de un evento. Estos servicios exponen eventos, con una información mínima, a los que los clientes pueden suscribirse para recibir actualizaciones cuando cambian los valores del servicio. Hay un puñado de variaciones para este estilo que incluyen (entre otras) reactivo, publicador/suscriptor, notificación de eventos y CQRS.

© JMA 2020. All rights reserved

Evolución histórica de los servicios

- **Precusores:**
 - RPC: Llamadas a Procedimientos Remotos
 - Binarios: CORBA, Java RMI, .NET Remoting
 - XML-RPC: Precursor del SOAP
- **Actuales:**
 - Servicios Web XML o Servicios SOAP
 - Servicios Web REST o API REST
 - WebHooks
 - Servicios GraphQL
 - Servicios gRPC

AÑO	Descripción
1976	Aparición de RPC (Remote Procedure Call) en Sistema Unix
1990	Aparición de DCE (Distributed Computing Environment) que es un sistema de software para computación distribuida, basado en RPC.
1991	Aparición de Microsoft RPC basado en DCE para sistemas Windows.
1992	Aparición de DCOM (Microsoft) y CORBA (ORB) para la creación de componentes software distribuidos.
1997	Aparición de Java RMI en JDK 1.1
1998	Aparición de XML-RPC
1999	Aparición de SOAP 1.0, WSDL, UDDI
2000	Definición del REST
2012	Propuesta de GraphQL por Facebook
2015	Desarrollo de gRPC por Google

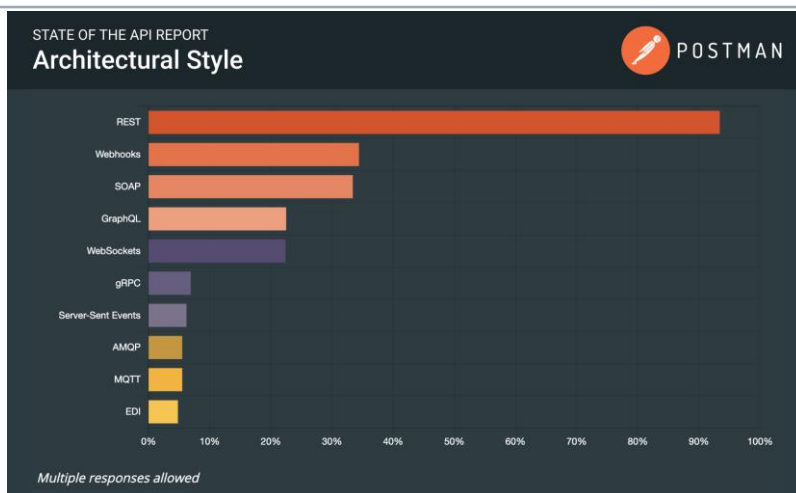
© JMA 2020. All rights reserved

Protocolos y Estándares

- Síncronos:
 - SOAP: Basado en operaciones, de tipos texto, en formato XML y comunicaciones síncronas.
 - REST: Basado en recursos, independiente de formato (texto/binario) y comunicaciones síncronas.
 - GraphQL: Basado en consultas, en formato JSON y comunicaciones síncronas.
 - gRPC: Basado en contratos, en el formato binario Protocol Buffers y comunicaciones síncronas/asíncronas.
- Asíncronos:
 - WebSockets: protocolo estándar abierto, elemental, texto y binario.
 - STOMP (Simple/Streaming Text Oriented Messaging Protocol): protocolo estándar abierto, basado en texto, soluciones simples y ligero.
 - AMQP (Advanced Message Queuing Protocol): protocolo estándar abierto, binario, encolamiento, P2P y Pub/Sub, exactitud y seguridad
 - MQTT (Message Queue Telemetry Transport): protocolo estándar abierto, binario, P2P, liviano, soluciones simples y seguridad
 - JMS (Java Message Service): API, binario Java, P2P y Pub/Sub

© JMA 2020. All rights reserved

Estilos arquitectónicos mas utilizados



© JMA 2020. All rights reserved

<https://www.postman.com/state-of-api/api-technologies/#api-technologies>

Estilos de comunicación

SERVICIOS REST

© JMA 2020. All rights reserved

REST (REpresentational State Transfer)

- En 2000, Roy Fielding propuso la transferencia de estado representacional (REST) como enfoque de arquitectura para el diseño de servicios web. REST **es un estilo de arquitectura** para la creación de sistemas distribuidos basados en hipermedia. REST es independiente de cualquier protocolo subyacente y no está necesariamente unido a HTTP. Sin embargo, en las implementaciones más comunes de REST se usa HTTP como protocolo de aplicación, y esta guía se centra en el diseño de API de REST para HTTP.
- Originalmente se basaba en lo que ya estaba disponible en HTTP:
 - URL como identificadores de recursos
 - HTTP ya define 8 métodos (algunas veces referidos como "verbos") que indica la acción que desea que se efectúe sobre el recurso identificado: HEAD, GET, POST, PUT, DELETE, TRACE, OPTIONS, CONNECT + PATCH (HTTP1.1)
 - HTTP permite transmitir en el encabezado la información de comportamiento: Content-type, Accept, Authorization, Cache-control, ...
 - HTTP utiliza códigos de estado en la respuesta para indicar como se ha completado una solicitud HTTP específica: respuestas informativas (1xx), respuestas satisfactorias (2xx), redirecciones (3xx), Errores en la petición (4xx) y errores de los servidores (5xx).

© JMA 2020. All rights reserved

Petición HTTP

- Cuando realizamos una petición HTTP, el mensaje consta de:
 - Primera línea de texto indicando la versión del protocolo utilizado, el verbo y el URI
 - El verbo indica la acción a realizar sobre el recurso web localizado en la URI
 - Posteriormente vendrían las cabeceras (opcionales)
 - Después el cuerpo del mensaje, que contiene un documento, que puede estar en cualquier formato (XML, HTML, JSON → Content-type)

```
POST /server/payment HTTP/1.1
Host: www.myserver.com
Content-Type: application/x-www-form-urlencoded
Accept: application/json
Accept-Encoding: gzip, deflate, sdch
Accept-Language: en-US,en;q=0.8
Cache-Control: max-age=0
Connection: keep-alive

orderId=34fry423&payment-method=visa&card-number=2345123423487648&sn=345
```

© JMA 2020. All rights reserved

Respuesta HTTP

- Los mensajes HTTP de respuesta siguen el mismo formato que los de envío.
- Sólo difieren en la primera línea
 - Donde se indica un código de respuesta junto a una explicación textual de dicha respuesta.
 - El código de respuesta indica si la petición tuvo éxito o no.

```
HTTP/1.1 201 Created
Content-Type: application/json; charset=utf-8
Location: https://www.myserver.com/services/payment/3432
Cache-Control: max-age=21600
Connection: close
Date: Mon, 23 Jul 2012 14:20:19 GMT
ETag: "2cc8-3e3073913b100"
Expires: Mon, 23 Jul 2012 20:20:19 GMT

{"id": "https://www.myserver.com/services/payment/3432", "status": "pending"}
```

© JMA 2020. All rights reserved

Recursos

- Un recurso es cualquier elemento que dispone de un URI correcto y único, cualquier tipo de objeto, dato o servicio que sea direccionable a través de internet.
- Un recurso es un objeto que es lo suficientemente importante como para ser referenciado por sí mismo. Un recurso tiene datos, relaciones con otros recursos y métodos que operan contra él para permitir el acceso y la manipulación de la información asociada. Un grupo de recursos se llama colección. El contenido de las colecciones y los recursos depende de los requisitos de la organización y de los consumidores.
- En REST todos los recursos comparten una interfaz única y constante, la URI. (https://...)
- Todos los recursos tienen las mismas operaciones (CRUD)
 - CREATE, READ, UPDATE, DELETE
- Normalmente estos recursos son accesibles en una red o sistema.
- Las URI son el único medio por el que los clientes y servidores pueden realizar el intercambio de representaciones.
- Para que un URI sea correcto, debe cumplir los requisitos de formato, REST no indica de forma específica un formato obligatorio.
 - <esquema>://<host>:<puerto>/<ruta><querystring><fragmento>
- Los URI asociados a los recursos pueden cambiar si modificamos el recurso (nombre, ubicación, características, etc)

© JMA 2020. All rights reserved

Tipos MIME

- Otro aspecto muy importante es la posibilidad de negociar distintos formatos (representaciones) a usar en la transferencia del estado entre servidor y cliente (y viceversa). La representación de los recursos es el formato de lo que se envía un lado a otro entre clientes y servidores. Como REST utiliza HTTP, podemos transferir múltiples tipos de información.
- Los datos se transmiten a través de TCP/IP, el navegador sabe cómo interpretar las secuencias binarias (CONTENT-TYPE) por el protocolo HTTP.
- La representación de un recurso depende del tipo de llamada que se ha generado (Texto, HTML, PDF, etc).
- En HTTP cada uno de estos formatos dispone de su propio tipos MIME, en el formato <tipo>/<subtipo>.
 - application/json application/xml text/html text/plain image/jpeg
- Para negociar el formato:
 - El cliente, en la cabecera ACCEPT, envía una lista priorizada de tipos MIME que entiende.
 - Tanto cliente como servidor indican en la cabecera CONTENT-TYPE el formato MIME en que está codificado el body.
- Si el servidor no entiende ninguno de los tipos MIME propuestos (ACCEPT) devuelve un mensaje con código 406 (incapaz de aceptar petición).

© JMA 2020. All rights reserved

Métodos HTTP

HTTP	REST	Descripción
GET	RETRIEVE	Sin identificador: Recuperar el estado completo de un recurso (HEAD + BODY) Con identificador: Recuperar el estado individual de un recurso (HEAD + BODY)
HEAD		Recuperar la cabecera del estado de un recurso (HEAD)
POST	CREATE or REPLACE	Crea o modifica un recurso (sin identificador)
PUT	CREATE or REPLACE	Crea o modifica un recurso (con identificador)
DELETE	DELETE	Sin identificador: Elimina todo el recurso Con identificador: Elimina un elemento concreto del recurso
CONNECT		Comprueba el acceso al host
TRACE		Solicita al servidor que introduzca en la respuesta todos los datos que reciba en el mensaje de petición
OPTIONS		Devuelve los métodos HTTP que el servidor soporta para un URL específico
PATCH	REPLACE	HTTP 1.1 Reemplaza parcialmente un elemento del recurso

© JMA 2020. All rights reserved

Códigos HTTP (status)

status	statusText	Descripción
100	Continue	Una parte de la petición (normalmente la primera) se ha recibido sin problemas y se puede enviar el resto de la petición
101	Switching protocols	El servidor va a cambiar el protocolo con el que se envía la información de la respuesta. En la cabecera Upgrade indica el nuevo protocolo
200	OK	La petición se ha recibido correctamente y se está enviando la respuesta. Este código es con mucha diferencia el que mas devuelven los servidores
201	Created	Se ha creado un nuevo recurso (por ejemplo una página web o un archivo) como parte de la respuesta
202	Accepted	La petición se ha recibido correctamente y se va a responder, pero no de forma inmediata
203	Non-Authoritative Information	La respuesta que se envía la ha generado un servidor externo. A efectos prácticos, es muy parecido al código 200
204	No Content	La petición se ha recibido de forma correcta pero no es necesaria una respuesta
205	Reset Content	El servidor solicita al navegador que inicialice el documento desde el que se realizó la petición, como por ejemplo un formulario
206	Partial Content	La respuesta contiene sólo la parte concreta del documento que se ha solicitado en la petición

© JMA 2020. All rights reserved

Códigos de redirección

status	statusText	Descripción
300	Multiple Choices	El contenido original ha cambiado de sitio y se devuelve una lista con varias direcciones alternativas en las que se puede encontrar el contenido
301	Moved Permanently	El contenido original ha cambiado de sitio y el servidor devuelve la nueva URL del contenido. La próxima vez que solicite el contenido, el navegador utiliza la nueva URL
302	Found	El contenido original ha cambiado de sitio de forma temporal. El servidor devuelve la nueva URL, pero el navegador debe seguir utilizando la URL original en las próximas peticiones
303	See Other	El contenido solicitado se puede obtener en la URL alternativa devuelta por el servidor. Este código no implica que el contenido original ha cambiado de sitio
304	Not Modified	Normalmente, el navegador guarda en su caché los contenidos accedidos frecuentemente. Cuando el navegador solicita esos contenidos, incluye la condición de que no hayan cambiado desde la última vez que los recibió. Si el contenido no ha cambiado, el servidor devuelve este código para indicar que la respuesta sería la misma que la última vez
305	Use Proxy	El recurso solicitado sólo se puede obtener a través de un proxy, cuyos datos se incluyen en la respuesta
307	Temporary Redirect	Se trata de un código muy similar al 302, ya que indica que el recurso solicitado se encuentra de forma temporal en otra URL

© JMA 2020. All rights reserved

Códigos de error en la petición

status	statusText	Descripción
400	Bad Request	El servidor no entiende la petición porque no ha sido creada de forma correcta
401	Unauthorized	El recurso solicitado requiere autorización previa
402	Payment Required	Código reservado para su uso futuro
403	Forbidden	No se puede acceder al recurso solicitado por falta de permisos o porque el usuario y contraseña indicados no son correctos
404	Not Found	El recurso solicitado no se encuentra en la URL indicada. Se trata de uno de los códigos más utilizados y responsable de los típicos errores de <i>Página no encontrada</i>
405	Method Not Allowed	El servidor no permite el uso del método utilizado por la petición, por ejemplo por utilizar el método GET cuando el servidor sólo permite el método POST
406	Not Acceptable	El tipo de contenido solicitado por el navegador no se encuentra entre la lista de tipos de contenidos que admite, por lo que no se envía en la respuesta
407	Proxy Authentication Required	Similar al código 401, indica que el navegador debe obtener autorización del proxy antes de que se le pueda enviar el contenido solicitado
408	Request Timeout	El navegador ha tardado demasiado tiempo en realizar la petición, por lo que el servidor la descarta

© JMA 2020. All rights reserved

Códigos de error en la petición

status	statusText	Descripción
409	Conflict	El navegador no puede procesar la petición, ya que implica realizar una operación no permitida (como por ejemplo crear, modificar o borrar un archivo)
410	Gone	Similar al código 404. Indica que el recurso solicitado ha cambiado para siempre su localización, pero no se proporciona su nueva URL
411	Length Required	El servidor no procesa la petición porque no se ha indicado de forma explícita el tamaño del contenido de la petición
412	Precondition Failed	No se cumple una de las condiciones bajo las que se realizó la petición
413	Request Entity Too Large	La petición incluye más datos de los que el servidor es capaz de procesar. Normalmente este error se produce cuando se adjunta en la petición un archivo con un tamaño demasiado grande
414	Request-URI Too Long	La URL de la petición es demasiado grande, como cuando se incluyen más de 512 bytes en una petición realizada con el método GET
415	Unsupported Media Type	Al menos una parte de la petición incluye un formato que el servidor no es capaz de procesar
416	Requested Range Not Suitable	El trozo de documento solicitado no está disponible, como por ejemplo cuando se solicitan bytes que están por encima del tamaño total del contenido
417	Expectation Failed	El servidor no puede procesar la petición porque al menos uno de los valores incluidos en la cabecera Expect no se pueden cumplir

© JMA 2020. All rights reserved

Códigos de error del servidor

status	statusText	Descripción
500	Internal Server Error	Se ha producido algún error en el servidor que impide procesar la petición
501	Not Implemented	Procesar la respuesta requiere ciertas características no soportadas por el servidor
502	Bad Gateway	El servidor está actuando de proxy entre el navegador y un servidor externo del que ha obtenido una respuesta no válida
503	Service Unavailable	El servidor está sobrecargado de peticiones y no puede procesar la petición realizada
504	Gateway Timeout	El servidor está actuando de proxy entre el navegador y un servidor externo que ha tardado demasiado tiempo en responder
505	HTTP Version Not Supported	El servidor no es capaz de procesar la versión HTTP utilizada en la petición. La respuesta indica las versiones de HTTP que soporta el servidor

© JMA 2020. All rights reserved

Estilo de arquitectura

- Las APIs de REST se diseñan en torno a recursos, que son cualquier tipo de objeto, dato o servicio al que puede acceder el cliente.
- Un recurso tiene un identificador, que es un URI que identifica de forma única ese recurso.
- Los clientes interactúan con un servicio mediante el intercambio de representaciones de recursos.
- Las APIs de REST usan una interfaz uniforme, que ayuda a desacoplar las implementaciones de clientes y servicios. En las APIs REST basadas en HTTP, la interfaz uniforme incluye el uso de verbos HTTP estándar para realizar operaciones en los recursos. Las operaciones más comunes son GET, POST, PUT, PATCH y DELETE. El código de estado de la respuesta indica el éxito o error de la petición.
- Las APIs de REST usan un modelo de solicitud sin estado. Las solicitudes HTTP deben ser independientes y pueden producirse en cualquier orden, por lo que no es factible conservar la información de estado transitoria entre solicitudes. El único lugar donde se almacena la información es en los propios recursos y cada solicitud debe ser una operación atómica.
- Las APIs de REST se controlan mediante vínculos de hipermedia.

© JMA 2020. All rights reserved

Estilo de arquitectura

- **Métodos GET**
 - Normalmente, un método GET correcto devuelve el código de estado HTTP 200 (Correcto). Si no se encuentra el recurso, el método debe devolver HTTP 404 (No encontrado).
- **Métodos POST**
 - Si un método POST crea un nuevo recurso, devuelve el código de estado HTTP 201 (Creado). El URI del nuevo recurso se incluye en el encabezado Location de la respuesta. El cuerpo de respuesta contiene una representación del recurso.
 - Si el método realiza algún procesamiento pero no crea un nuevo recurso, puede devolver el código de estado HTTP 200 e incluir el resultado de la operación en el cuerpo de respuesta. O bien, si no hay ningún resultado para devolver, el método puede devolver el código de estado HTTP 204 (Sin contenido) sin cuerpo de respuesta.
 - Si el cliente coloca datos no válidos en la solicitud, el servidor debe devolver el código de estado HTTP 400 (Solicitud incorrecta). El cuerpo de respuesta puede contener información adicional sobre el error o un vínculo a un URI que proporciona más detalles.
- **Métodos DELETE**
 - El servidor web debe responder con un 204 (Sin contenido), que indica que la operación de eliminación es correcta, pero que el cuerpo de respuesta no contiene información adicional. Si el recurso no existe, el servidor web puede devolver un 404 (No encontrado).

© JMA 2020. All rights reserved

Estilo de arquitectura

- Métodos PUT

- Si un método PUT crea un nuevo recurso, devuelve el código de estado HTTP 201 (Creado), al igual que con un método POST. Si el método actualiza un recurso existente, devuelve un 200 (Correcto) o un 204 (Sin contenido). Si el cliente coloca datos no válidos en la solicitud, el servidor debe devolver un 400 (Solicitud incorrecta), si no es posible actualizar el recurso existente devolverá un 409 (Conflicto) o el recurso no existe, puede devolver un 404 (No encontrado).
- Hay que considerar la posibilidad de implementar operaciones HTTP PUT masivas que pueden procesar por lotes las actualizaciones de varios recursos de una colección. La solicitud PUT debe especificar el URI de la colección y el cuerpo de solicitud debe especificar los detalles de los recursos que se van a modificar. Este enfoque puede ayudar a reducir el intercambio de mensajes y mejorar el rendimiento.

- Métodos PATCH

- Con una solicitud PATCH, el cliente envía un conjunto de actualizaciones a un recurso existente, en forma de un documento de revisión. El servidor procesa el documento de revisión para realizar la actualización.
- Si el método actualiza un recurso existente, devuelve un 200 (Correcto) o un 204 (Sin contenido). Si el cliente coloca datos no válidos en la solicitud o el documento de revisión tiene un formato incorrecto, el servidor debe devolver un 400 (Solicitud incorrecta), si no se admite el formato de documento de revisión devolverá un 415 (Tipo de medio no compatible), si no es posible actualizar el recurso existente devolverá un 409 (Conflicto) o el recurso no existe, puede devolver un 404 (No encontrado).

© JMA 2020. All rights reserved

Encabezado HTTP Cache-Control

- El encabezado HTTP Cache-Control especifica directivas (instrucciones) para almacenar temporalmente (caching) tanto en peticiones como en respuestas. Una directiva dada en una petición no significa que la misma directiva estar en la respuesta.
- Los valores estándar que pueden ser usados por el servidor en una respuesta HTTP son:
 - public: La respuesta puede estar almacenada en cualquier memoria cache.
 - private: La respuesta puede estar almacenada sólo por el cache de un navegador.
 - no-cache: La respuesta puede estar almacenada en cualquier memoria cache pero DEBE pasar siempre por una validación con el servidor de origen antes de utilizarse.
 - no-store: La respuesta puede no ser almacenada en cualquier cache.
 - max-age=<seconds>: La cantidad máxima de tiempo un recurso es considerado reciente.
 - s-maxage=<seconds>: Anula el encabezado max-age o el Expires, pero solo para caches compartidos (e.g., proxies).
 - must-revalidate: Indica que una vez un recurso se vuelve obsoleto, el cache no debe usar su copia obsoleta sin validar correctamente en el servidor de origen.
 - proxy-revalidate: Similar a must-revalidate, pero solo para caches compartidos (es decir, proxies). Ignorado por caches privados.
 - no-transform: No deberían hacerse transformaciones o conversiones al recurso.

© JMA 2020. All rights reserved

Encabezados HTTP ETag, If-Match y If-None-Match

- El encabezado de respuesta de HTTP ETag es un identificador (resumen hash) para una versión específica de un recurso y los encabezados If-Match e If-None-Match de la solicitud HTTP hace que la solicitud sea condicional.
- Para los métodos GET y HEAD con If-None-Match: si el ETag no coincide con los datos, el servidor devolverá el recurso solicitado con un estado 200, si coincide el servidor debe devolver el código de estado HTTP 304 (No modificado) y DEBE generar cualquiera de los siguientes campos de encabezado que se habrían enviado en una respuesta 200 (OK) a la misma solicitud: Cache-Control, Content-Location, Date, ETag, Expires y Vary.
- Para los métodos PUT y DELETE con If-Match: si el ETag coincide con los datos, se realiza la actualización o borrado y se devuelve un estado HTTP 204 (sin contenido) incluyendo el Cache-Control y el ETag de la versión actualizada del recurso en el PUT. Si no coinciden, se ha producido un error de concurrencia, la versión del servidor ha sido modificada desde que la recibió el cliente, debe devolver una respuesta HTTP con un cuerpo de mensaje vacío y un código de estado 412 (Precondición fallida).
- Si los datos solicitados ya no existen, el servidor debe devolver una respuesta HTTP con el código de estado 404 (no encontrado).

© JMA 2020. All rights reserved

Estilo de arquitectura

- Request: Método /uri?parámetros
 - GET: Recupera el recurso (200)
 - Todos: Sin identificador
 - Uno: Con identificador
 - POST: Crea o reemplaza un nuevo recurso (201)
 - PUT: Crea o reemplaza el recurso identificado (200, 204)
 - DELETE: Elimina el recurso (204)
 - Todos: Sin identificador
 - Uno: Con identificador
- Cabeceras:
 - Accept: Indica al servidor el formato o posibles formatos esperados, utilizando MIME.
 - Content-type: Indica en que formato está codificado el cuerpo, utilizando MIME
- HTTP Status Code: Código de estado con el que el servidor informa del resultado de la petición.

© JMA 2020. All rights reserved

Peticiones

- Request: GET /users
 - accept:application/json
- Response: 200
 - content-type:application/json
 - BODY
- Request: GET /users/11
 - accept:application/json
- Response: 200
 - content-type:application/json
 - BODY
- Request: POST /users
 - accept:application/json
 - content-type:application/json
 - BODY
- Response: 201
 - content-type:application/json
- BODY
- Request: PUT /users/11
 - accept:application/json
 - content-type:application/json
 - BODY
- Response: 200
 - content-type:application/json
 - BODY
- Request: DELETE /users/11
- Response: 204 no content

© JMA 2020. All rights reserved

Diseño de un Servicio Web REST

- Para el desarrollo de los Servicios Web's REST es necesario definir una serie de cosas:
 - Analizar el/los recurso/s a implementar
 - Diseñar la REPRESENTACION del recurso.
 - Deberemos definir el formato de trabajo del recurso: XML, JSON, HTML, imagen, RSS, etc
 - Definir el URI de acceso.
 - Deberemos indicar el/los URI de acceso para el recurso
 - Establecer los métodos soportados por el servicio
 - GET, POST, PUT, DELETE
 - Fijar qué códigos de estado pueden ser devueltos
 - Los códigos de estado HTTP típicos que podrían ser devueltos
- Todo lo anterior dependerá del servicio a implementar.

© JMA 2020. All rights reserved

Definir operaciones

- Sumario y descripción de la operación.
- Dirección: URL
 - Sin identificador
 - Con identificador
 - Con parámetros de consulta
- Método: GET | POST | PUT | DELETE | PATCH
- Solicitud:
 - Cabeceras:
 - ACCEPT: formatos aceptables si espera recibir datos
 - CONTENT-TYPE: formato de envío de los datos en la solicitud
 - Otras cabeceras: Authorization, Cache-control, X-XSRF-TOKEN, ...
 - Cuerpo: en caso de envío, estructura de datos formateados según el CONTENT-TYPE.
- Respuesta:
 - Cabeceras:
 - Códigos de estado HTTP: posibles y sus causas.
 - CONTENT-TYPE: formato de envío de los datos en la respuesta
 - Otras cabeceras
 - Cuerpo: en caso de respuesta, estructura de datos según código de estado y formateados según el CONTENT-TYPE.

© JMA 2020. All rights reserved

Richardson Maturity Model

<http://www.crummy.com/writing/speaking/2008-QCon/act3.html>

- Nivel 0: Definir un URI y todas las operaciones son solicitudes POST a este URI.
- Nivel 1 (Pobre): Crear distintos URI para recursos individuales pero utilizan solo un método.
 - Se debe identificar un recurso
/entities/?invoices=2 → entities/invoices/2
 - Se construyen con nombres nunca con verbos
/getUser/{id} → /users/{id}/
/users/{id}/edit/login → users/{id}/access-token
 - Deberían tener una estructura jerárquica
/invoices/user/{id} → /user/{id}/invoices
- Nivel 2 (Medio): Usar métodos HTTP para definir operaciones en los recursos.
- Nivel 3 (Óptimo): Usar hipermedia (HATEOAS, se describe a continuación).

© JMA 2020. All rights reserved

Hypermedia

- Uno de los principales propósitos que se esconden detrás de REST es que debe ser posible navegar por todo el conjunto de recursos sin necesidad de conocer el esquema de URI. Cada solicitud HTTP GET debe devolver la información necesaria para encontrar los recursos relacionados directamente con el objeto solicitado mediante los hipervínculos que se incluyen en la respuesta, y también se le debe proporcionar información que describa las operaciones disponibles en cada uno de estos recursos.
- Este principio se conoce como HATEOAS, del inglés Hypertext as the Engine of Application State (Hipertexto como motor del estado de la aplicación). El sistema es realmente una máquina de estado finito, y la respuesta a cada solicitud contiene la información necesaria para pasar de un estado a otro; ninguna otra información debería ser necesaria.
- Se basa en la idea de enlazar recursos: propiedades que son enlaces a otros recursos.
- Para que sea útil, el cliente debe saber que en la respuesta hay contenido hypermedia.
- En content-type es clave para esto
 - Un tipo genérico no aporta nada:
Content-Type: text/xml
 - Se pueden crear tipos propios
Content-Type: application/servicio+xml

© JMA 2020. All rights reserved

JSON Hypertext Application Language

- RFC4627 <http://tools.ietf.org/html/draft-kelly-json-hal-00>
- HATEOAS: Content-Type: application/hal+json

```
{
  "_links": {
    "self": {"href": "/orders/523" },
    "warehouse": {"href": "/warehouse/56" },
    "invoice": {"href": "/invoices/873"}
  },
  "currency": "USD"
  , "status": "shipped"
  , "total": 10.20
}
```

© JMA 2020. All rights reserved

Características de una API bien diseñada

- **Fácil de leer y trabajar:** con una API bien diseñada será fácil trabajar, y sus recursos y operaciones asociadas pueden ser memorizados rápidamente por los desarrolladores que trabajan con ella constantemente.
- **Difícil de usar mal:** la implementación e integración con una API con un buen diseño será un proceso sencillo, y escribir código incorrecto será un resultado menos probable porque tiene comentarios informativos y no aplica pautas estrictas al consumidor final de la API.
- **Completa y concisa:** Finalmente, una API completa hará posible que los desarrolladores creen aplicaciones completas con los datos que expone. Por lo general, la completitud ocurre con el tiempo, y la mayoría de los diseñadores y desarrolladores de API construyen gradualmente sobre las APIs existentes. Es un ideal por el que todo ingeniero o empresa con una API debe esforzarse.

© JMA 2020. All rights reserved

Guía de diseño

- Organización de la API en torno a los recursos
- Definición de operaciones en términos de métodos HTTP
- Conformidad con la semántica HTTP
- Filtrado y paginación de los datos
- Compatibilidad con respuestas parciales en recursos binarios de gran tamaño
- Uso de HATEOAS para permitir la navegación a los recursos relacionados
- Control de versiones en la API RESTful
- Documentación Open API

© JMA 2020. All rights reserved

Definición de recursos

- La organización de la API en torno a los recursos se centran en las entidades de dominio que debe exponer la API. Por ejemplo, en un sistema de comercio electrónico, las entidades principales podrían ser clientes y pedidos. La creación de un pedido se puede lograr mediante el envío de una solicitud HTTP POST que contiene la información del pedido. La respuesta HTTP indica si el pedido se realizó correctamente o no.
- Un recurso no tiene que estar basado en un solo elemento de datos físico o tablas de una base de datos relacional. La finalidad de REST es modelar entidades y las operaciones que un consumidor externo puede realizar sobre esas entidades, no debe exponerse a la implementación interna.
- Es necesario adoptar una convención de nomenclatura coherente para los URI. Los URI de recursos deben basarse en nombres (de recurso), nunca en verbos (las operaciones en el recurso) y, en general, resulta útil usar nombres plurales que hagan referencia a colecciones. Debe seguir una estructura jerárquica que refleje las relaciones entre los diferentes tipos de recursos.
- Hay que considerar el uso del enfoque HATEOAS para permitir el descubrimiento y la navegación a los recursos relacionados o el enfoque del patrón agregado.

© JMA 2020. All rights reserved

Definición de recursos

- Exponer una colección de recursos con un único URI puede dar lugar a que las aplicaciones capturen grandes cantidades de datos cuando solo se requiere un subconjunto de la información. A través de la definición de parámetros de cadena de consulta se pueden realizar particiones horizontales con filtrado, ordenación y paginación, o particiones verticales con la proyección de las propiedades a recuperar:
 - `https://host/users?page=1&rows=20&projection=userId,name,lastAccess`
- La definición de operaciones con el recurso se realiza en términos de métodos HTTP, estableciendo cuales serán soportadas. Las operaciones no soportadas por métodos HTTP deben sustantivarse al crearles una URI específica y utilizar el método HTTP semánticamente mas próximo.
 - DELETE `https://host/users/bloqueo` (desbloquear)
 - POST `https://host/pedido/171/factura` (facturar)
- Hay que establecer el tipo o tipos de formatos mas adecuados para las representaciones de recursos. Los formatos se especifican mediante el uso de tipos de medios, también denominados tipos MIME. En el caso de datos no binarios, la mayoría de las APIs web admiten JSON (`application/json`) y, posiblemente, XML (`application/xml`).

© JMA 2020. All rights reserved

Políticas de versionado

- Es muy poco probable que una API permanezca estática. Conforme los requisitos empresariales cambian, se pueden agregar nuevas colecciones de recursos, las relaciones entre los recursos pueden cambiar y la estructura de los datos de los recursos debe adecuarse.
- Los cambios rupturistas no son compatibles con la versión anterior, el consumidor tendrá que adaptar su código para pasar su aplicación existente a la nueva versión y evitar que se rompa.
- Hay dos razones principales por las que las APIs HTTP se comportan de manera diferente al resto de las APIs:
 - El código del cliente dicta lo que lo romperá: Un proveedor de API no tiene control sobre las herramientas que un consumidor puede usar para interpretar una respuesta de la API y la tolerancia al cambio que tienen esas herramientas varían ampliamente, si es rupturista o no.
 - El proveedor de API elige si los cambios son opcionales o transparentes: Los proveedores de API pueden actualizar su API y los cambios en las respuestas afectarán inmediatamente a los clientes. Los clientes no pueden decidir si adoptar o no la nueva versión, lo que puede generar fallos en cascada en los cambios rupturistas.

© JMA 2020. All rights reserved

Políticas de versionado

- El control de versiones permite que una API indique la versión expuesta y que una aplicación cliente pueda enviar solicitudes que se dirijan a una versión específica con una característica o un recurso.
 - Sin control de versiones: Este es el enfoque más sencillo y puede ser aceptable para algunas APIs internas. Los grandes cambios podrían representarse como nuevos recursos o nuevos vínculos.
 - Control de versiones en URI: Cada vez que modifica la API web o cambia el esquema de recursos, agrega un número de versión al URI para cada recurso. Los URI ya existentes deben seguir funcionando como antes y devolver los recursos conforme a su esquema original.
`http://host/v2/users`
 - Control de versiones en cadena de consulta: En lugar de proporcionar varios URI, se puede especificar la versión del recurso mediante un parámetro dentro de la cadena de consulta anexada a la solicitud HTTP:
`http://host/users?versión=2.0`

© JMA 2020. All rights reserved

Políticas de versionado

- Control de versiones en encabezado: En lugar de anejar el número de versión como un parámetro de cadena de consulta, se podría implementar un encabezado personalizado que indica la versión del recurso. Este enfoque requiere que la aplicación cliente agregue el encabezado adecuado a las solicitudes, aunque el código que controla la solicitud de cliente puede usar un valor predeterminado (versión actual) si se omite el encabezado de versión.
GET https://host/users HTTP/1.1
Custom-Header: api-version=1
- Control de versiones por MIME (tipo de medio): Cuando una aplicación cliente envía una solicitud HTTP GET a un servidor web, debe prever el formato del contenido que puede controlar mediante el uso de un encabezado Accept.
GET https:// host/users/3 HTTP/1.1
Accept: application/vnd.mi-api.v1+json
- Si la versión no está soportada, el servicio podría generar un mensaje de respuesta HTTP 406 (no aceptable) o devolver un mensaje con un tipo de medio predeterminado.
- Los esquemas de control de versiones de URI y de cadena de consulta son compatibles con la caché HTTP puesto que la misma combinación de URI y cadena de consulta hace referencia siempre a los mismos datos.

© JMA 2020. All rights reserved

Políticas de versionado

- Dentro de la política de versionado es conveniente planificar la obsolescencia y la política de desaprobación.
- La obsolescencia programada establece el periodo máximo, como una franja temporal o un número de versiones, en que se va a dar soporte a cada versión, evitando los sobrecostos derivados de mantener versiones obsoletas indefinidamente.
- Dentro de la política de desaprobación, para ayudar a garantizar que los consumidores tengan tiempo suficiente y una ruta clara de actualización, se debe establecer el número de versiones en que se mantendrá una característica marcada como obsoleta antes de su desaparición definitiva.
- La obsolescencia programada y la política de desaprobación beneficia a los consumidores de la API porque proporcionan estabilidad y sabrán qué esperar a medida que las APIs evolucionen.
- Para mejorar la calidad y avanzar las novedades, se podrán realizar lanzamientos de versiones Beta y Release Candidatos (RC) o revisiones para cada versión mayor y menor. Estas versiones provisionales desaparecerán con el lanzamiento de la versión definitiva.

© JMA 2020. All rights reserved

Guía de implementación

- **Procesamiento de solicitudes**
 - Las acciones GET, PUT, DELETE, HEAD y PATCH deben ser idempotentes.
 - Las acciones POST que crean nuevos recursos no deben tener efectos secundarios no relacionados.
 - Evitar implementar operaciones POST, PUT y DELETE que generen mucha conversación.
 - Seguir la especificación HTTP al enviar una respuesta.
 - Admitir la negociación de contenido.
 - Proporcionar vínculos que permitan la navegación y la detección de recursos de estilo HATEOAS.
-

© JMA 2020. All rights reserved

Guía de implementación

- **Administración de respuestas y solicitudes de gran tamaño**
 - Optimizar las solicitudes y respuestas que impliquen objetos grandes.
 - Admitir la paginación de las solicitudes que pueden devolver grandes cantidades de objetos.
 - Implementar respuestas parciales para los clientes que no admitan operaciones asíncronas.
 - Evitar enviar mensajes de estado 100-Continuar innecesarios en las aplicaciones cliente.
 - **Mantenimiento de la capacidad de respuesta, la escalabilidad y la disponibilidad**
 - Ofrecer compatibilidad asíncrona para las solicitudes de ejecución prolongada.
 - Comprobar que ninguna de las solicitudes tenga estado.
 - Realizar un seguimiento de los clientes e implementar limitaciones para reducir las posibilidades de ataques de denegación de servicio.
 - Administrar con cuidado las conexiones HTTP persistentes.
-

© JMA 2020. All rights reserved

Guía de implementación

- **Control de excepciones**
 - Capturar todas las excepciones y devolver una respuesta significativa a los clientes.
 - Distinguir entre los errores del lado cliente y del lado servidor.
 - Evitar las vulnerabilidades por exceso de información.
 - Controlar las excepciones de una forma coherente y registrar la información sobre los errores.
- **Optimización del acceso a los datos en el lado cliente**
 - Admitir el almacenamiento en caché del lado cliente.
 - Proporcionar ETags para optimizar el procesamiento de las consultas.
 - Usar ETags para admitir la simultaneidad optimista.

© JMA 2020. All rights reserved

Guía de implementación

- **Publicación y administración de una API web**
 - Todas las solicitudes deben autenticarse y autorizarse, y debe aplicarse el nivel de control de acceso adecuado.
 - Una API web comercial puede estar sujeta a diversas garantías de calidad relativas a los tiempos de respuesta. Es importante asegurarse de que ese entorno de host es escalable si la carga puede variar considerablemente con el tiempo.
 - Puede ser necesario realizar mediciones de las solicitudes para fines de monetización.
 - Es posible que sea necesario regular el flujo de tráfico a la API web e implementar la limitación para clientes concretos que hayan agotado sus cuotas.
 - Los requisitos normativos podrían requerir un registro y una auditoría de todas las solicitudes y respuestas.
 - Para garantizar la disponibilidad, puede ser necesario supervisar el estado del servidor que hospeda la API web y reiniciarlo si hiciera falta.

© JMA 2020. All rights reserved

Guía de implementación

- Pruebas de la API
 - Ejercitar todas las rutas y parámetros para comprobar que invocan las operaciones correctas.
 - Verificar que cada operación devuelve los códigos de estado HTTP correctos para diferentes combinaciones de entradas.
 - Comprobar que todas las rutas estén protegidas correctamente y que estén sujetas a las comprobaciones de autenticación y autorización apropiadas.
 - Verificar el control de excepciones que realiza cada operación y que se devuelve una respuesta HTTP adecuada y significativa de vuelta a la aplicación cliente.
 - Comprobar que los mensajes de solicitud y respuesta están formados correctamente.
 - Comprobar que todos los vínculos dentro de los mensajes de respuesta no están rotos.
-

© JMA 2020. All rights reserved

<https://www.postman.com/>

POSTMAN

© JMA 2020. All rights reserved

INTRODUCCIÓN

© JMA 2020. All rights reserved

Pruebas de la API

- Ejercitar todas las rutas y parámetros para comprobar que invocan las operaciones correctas.
 - Verificar que cada operación devuelve los códigos de estado HTTP correctos para diferentes combinaciones de entradas.
 - Comprobar que todas las rutas estén protegidas correctamente y que estén sujetas a las comprobaciones de autenticación y autorización apropiadas.
 - Verificar el control de excepciones que realiza cada operación y que se devuelve una respuesta HTTP adecuada y significativa de vuelta a la aplicación cliente.
 - Comprobar que los mensajes de solicitud y respuesta están formados correctamente.
 - Comprobar que todos los vínculos dentro de los mensajes de respuesta no están rotos.
-

© JMA 2020. All rights reserved

Introducción

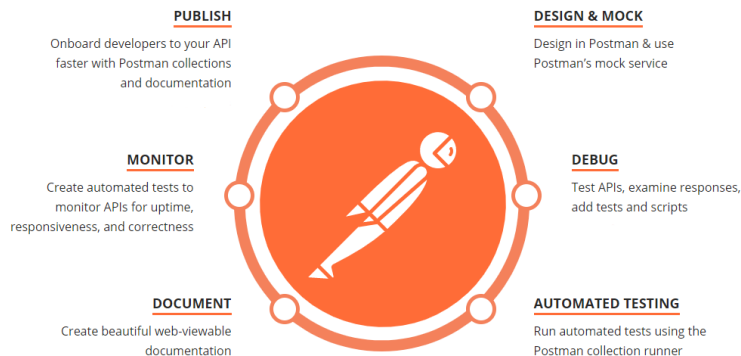
- Postman surgió originariamente como una extensión para el navegador Google Chrome que permitía realizar peticiones API REST con métodos diferentes al GET. A día de hoy dispone de aplicaciones nativas para MAC, Windows y algunas producciones Linux.
- Está compuesto por diferentes herramientas y utilidades gratuitas (en la versión free) que permiten realizar tareas diferentes dentro del mundo API REST: creación de peticiones a APIs internas o de terceros, elaboración de tests para validar el comportamiento de APIs, posibilidad de crear entornos de trabajo diferentes (con variables globales y locales), y todo ello con la posibilidad de ser compartido con otros compañeros del equipo de manera gratuita (exportación de toda esta información mediante URL en formato JSON).
- Además, dispone de un modo cloud colaborativo (de pago) para que equipos de trabajo puedan desarrollar entre todos colecciones para APIs sincronizadas en la nube para una integración más inmediata y sincronizada.
- Quizás sea una de las herramientas más utilizadas para hacer testing exploratorio de API REST.

© JMA 2020. All rights reserved

Soporte del ciclo de vida

Postman Tools Support Every Stage of the API Lifecycle

Through design, testing and full production, Postman is there for faster, easier API development—without the chaos.



© JMA 2020. All rights reserved

Instalación

- Postman está disponible como una aplicación nativa para los sistemas operativos macOS, Windows (32 bits y 64 bits) y Linux (32 bits y 64 bits). La extensión para Chrome de Postman ha quedado en desuso. Existe una versión online (con limitaciones) pero requiere instalar Postman Desktop Agent para superar las restricciones CORS y acceder a recursos locales.
- Para descargar Postman accedemos a su página oficial:
 - <https://www.postman.com/downloads/>
- 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 gratuita.
- 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.

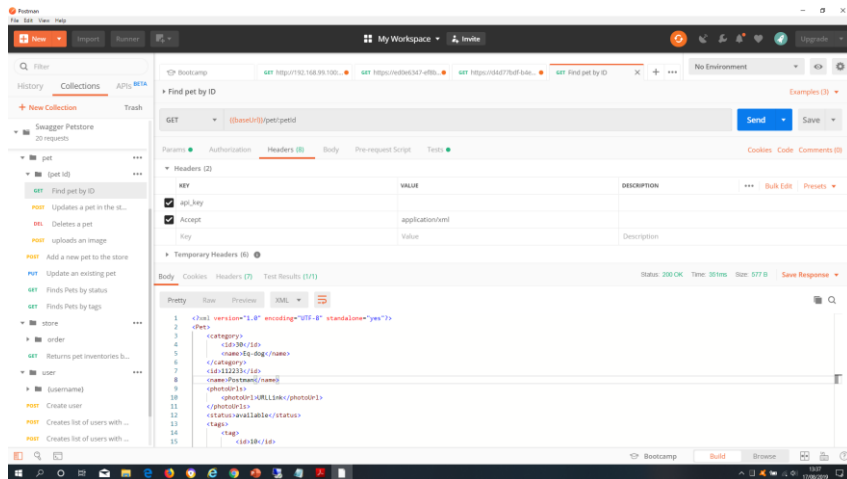
© JMA 2020. All rights reserved

Postman Interceptor

- Postman Interceptor es una extensión de Chrome que actúa como complemento del navegador para la aplicación de escritorio Postman. Interceptor permite capturar solicitudes de red y cookies directamente desde un navegador Chrome.
- Una vez que Interceptor se ejecuta en Chrome, se puede iniciar una sesión de depuración, que es una sesión de captura de tráfico con límite de tiempo.
- Se puede iniciar, pausar y detener una sesión de depuración del interceptor y luego iniciar otra.
- Cada sesión de depuración se registra en la pestaña Historial y muestra el tiempo total de la sesión y todo el tráfico capturado. Desde la sesión registrada, se pueden enviar solicitudes y respuestas a una colección y guardar cookies en el contenedor de cookies de Postman.

© JMA 2020. All rights reserved

Interfaz de Postman



© JMA 2020. All rights reserved

Espacios de trabajo

- Un espacio de trabajo es un repositorio de todos los artefactos de Postman que pueden ser persistentes: colecciones, entornos, simulacros, monitores y otros.
- Las personas pueden organizar su trabajo en espacios de trabajo personales y los equipos pueden colaborar en espacios de trabajo en equipo.
- Independientemente del tipo de espacio de trabajo, se puede compartir elementos en múltiples espacios de trabajo al mismo tiempo.
- Si no se tiene una cuenta de Postman, solo se puede tener un espacio de trabajo personal. Si se dispone de una cuenta, se pueden crear un número ilimitado de espacios de trabajo.
- Los espacios de trabajo de equipo (disponibles solo con usuarios de Postman) permiten el desarrollo continuo o colaboraciones a corto plazo. Cada equipo (sin coste un máximo de 25 usuarios) tiene un espacio de trabajo de equipo predeterminado y puede crear un número ilimitado de espacios de trabajo. Se puede compartir una colección con otros espacios de trabajo. Una Colección Postman compartida con los miembros del equipo es visible para todos los miembros de su equipo Postman con un acceso predeterminado de solo lectura. Si se tiene los permisos adecuados para la colección, se puede editar y actualizar en cualquier espacio de trabajo donde exista, permitiendo el seguimiento de cambios.

© JMA 2020. All rights reserved

Espacios de trabajo

- **Organización personal:**
 - Si un espacio de trabajo personal está abarrotado, se puede crear un nuevo espacio de trabajo para reflejar su flujo de trabajo personal de una manera más granular.
- **Organización del equipo:**
 - Del mismo modo, cree nuevos espacios de trabajo para reflejar el flujo de trabajo de su equipo.
 - Estos pueden ser espacios de trabajo permanentes alineados por función, producto, proyecto o socio.
 - Estos también pueden ser espacios de trabajo temporales para proyectos y actividades temporales.
- **Fuente de la verdad:**
 - Los equipos necesitan una única fuente de verdad para sus API, y los espacios de trabajo pueden dar confianza a un equipo o sub-equipo de que están trabajando con las últimas versiones de las colecciones Postman y saber que sus cambios se están sincronizando en tiempo real.
- **Permisos del equipo:**
 - Los espacios de trabajo del equipo son de solo lectura de forma predeterminada.
 - Se pueden controlar los permisos de edición y visualización de las colecciones y todos sus elementos asociados al proporcionar capacidades de edición a todo el equipo o solo a individuos específicos del equipo.

© JMA 2020. All rights reserved

Espacios de trabajo

- **Descubrimiento:**
 - Es un escenario común que alguien estará trabajando en algo de lo que el equipo más amplio no sabe nada.
 - Si hay colecciones compartidas en un espacio de trabajo común, esto permite a todos los miembros del equipo comprender el alcance de un proyecto y ver esos elementos con mayor claridad.
- **Feed de actividad actualizado:**
 - Ver una fuente de actividad de operaciones CUD (Crear, Actualizar, Eliminar) dentro de una colección.
 - Mantiene al tanto de quién actualizó una colección, cuáles fueron las actualizaciones y cuándo se completaron.
- **Depuración en tiempo real con Historial:**
 - Cuando los miembros del equipo se unen a un espacio de trabajo, incluso el historial de solicitudes es una entidad compartida.
 - Todos los que forman parte de ese espacio de trabajo pueden ver las solicitudes enviadas más recientemente y observar el comportamiento en sus propias instancias en tiempo real.

© JMA 2020. All rights reserved

Colecciones

- Las colecciones Postman permiten agrupar solicitudes individuales y organizar estas solicitudes en carpetas.
 - Organización: Se puede guardar las solicitudes en colecciones y carpetas, para no tener que buscar en el historial repetidamente.
 - Documentación: Se puede agregar un nombre y descripciones a solicitudes, carpetas y colecciones. En Postman, se puede usar el navegador de colección para ver la documentación o generar y publicar páginas con la documentación del API.
 - Suites de prueba: Se pueden adjuntar scripts de prueba a las solicitudes y crear conjuntos de pruebas de integración.
 - Flujos de trabajo condicionales: Se pueden usar scripts para pasar datos entre solicitudes de API y crear flujos de trabajo que reflejen su caso de uso de API real.
 - Compartir: Se pueden compartir colecciones directamente a través de espacios de trabajo o mediante la exportación a archivos.
- Las colecciones son los puntos de partida para las simulaciones (Service Mocking), la documentación, la monitorización y otros.

© JMA 2020. All rights reserved

Variables y Entornos

- Postman permite sustituir determinados valores constantes por el valor almacenado en una variable (o propiedad). Las variables le permiten:
 - Reutilizar los valores para mantener el código DRY (Don't Repeat Yourself).
 - Configurar diferentes juegos de valores para diferentes entornos.
 - Extraer datos de respuestas y solicitudes en cadena en una colección.
- Las variables son nombres simbólicos que representan la información que almacena en ellas. La información que representan las variables puede cambiar, pero las operaciones con la variable siguen siendo las mismas. Las variables en Postman funcionan de la misma manera.
- Un entorno es un conjunto de pares clave-valor. La clave representa el nombre de la variable. Mientras se trabaja con las API, a menudo se necesita diferentes configuraciones para la máquina local, el servidor de desarrollo o la API de producción. Los entornos permiten personalizar las solicitudes utilizando variables para cambiar entre diferentes configuraciones sin cambiar las solicitudes.
- Las variables globales proporcionan un conjunto de variables que siempre están disponibles en todos los ámbitos. Se pueden tener múltiples entornos, pero solo uno puede estar activo a la vez con un conjunto de variables globales, que siempre están disponibles.
- Más allá del alcance, las variables globales y de entorno también se pueden definir de tipo Secret para evitar la divulgación no intencional de datos confidenciales, incluidos los API secrets, las contraseñas, los tokens y las claves.

© JMA 2020. All rights reserved

Variables

- Las variables se pueden definir a diferentes niveles que determinan el ámbito y alcance (si una variable comparte su nombre con una variable superior, la variable mas local tendrá prioridad).
 1. Global
 2. Colección
 3. Entorno
 4. Datos
 5. Local
- Las variables globales y de entorno se gestionan mediante el "Manage Environments", las de colección mediante la edición de la colección, las de datos cargando el fichero (CSV o JSON) en el Collection Runner y las locales en los script.
- Para acceder a variables en el generador de solicitudes, la cadena {{variableName}} se reemplazará con su valor actual cuando Postman resuelva la variable.
- En los script, las variables locales son accesibles de la forma habitual. Para acceder a niveles superiores es necesario indicar el nivel (globals, environment, collectionVariables, variables, iterationData, cookies) con los métodos: `pm.nivel.get()` o `pm.nivel.set()`.
`pm.variables.get("variable_key");`
- Salvo las variables locales, los valores siempre se almacenan como cadenas. Si se está almacenando objetos o matrices, es necesario serializarlos a cadena JSON con `JSON.stringify()` o des serializarlos con `JSON.parse()`.



© JMA 2020. All rights reserved

Variables

© JMA 2020. All rights reserved

Variables dinámicas

- Postman tiene algunas variables dinámicas que puede usar en las solicitudes. Las variables dinámicas solo son accesible por el generador de solicitudes en la URL, encabezados y cuerpo.

```
{
  "name": "{{${RandomCompanyName}}}",
  "email": "{{${RandomEmail}}}",
  "description": "{{${RandomLoremParagraph}}}"
}
```

- Para usar variables dinámicas en scripts de prueba o solicitud previa, se debe reemplazar con `pm.variables.replaceIn`:

```
console.log(pm.variables.replaceIn('{{${RandomCatchPhrase}}'))
```

Comunes: \$guid, \$timestamp, \$randomUUID

Texto, números y colores: \$randomAlphaNumeric, \$randomBoolean, \$randomInt, \$randomColor, \$randomHexColor, \$randomAbbreviation

Fechas: \$randomDateFuture, \$randomDatePast, \$randomDateRecent, \$randomWeekday, \$randomMonth

Archivos y directorios: \$randomFileName, \$randomFileType, \$randomFileExt, \$randomCommonFileName, \$randomCommonFileType, \$randomCommonFileExt, \$randomFilePath, \$randomDirectoryPath, \$randomMimeType

Bases de datos: \$randomDatabaseColumn, \$randomDatabaseType, \$randomDatabaseCollation, \$randomDatabaseEngine

© JMA 2020. All rights reserved

Variables dinámicas

Internet y direcciones IP: \$randomIP, \$randomIPv6, \$randomMACAddress, \$randomPassword, \$randomLocale, \$randomUserAgent, \$randomProtocol, \$randomSemver

Dominios, correos electrónicos y nombres de usuario: \$randomDomainName, \$randomDomainSuffix, \$randomDomainWord, \$randomEmail, \$randomExampleEmail, \$randomUserName, \$randomUrl

Nombres: \$randomFirstName, \$randomLastName, \$randomFullName, \$randomNamePrefix, \$randomNameSuffix

Profesión: \$randomJobArea, \$randomJobDescriptor, \$randomJobTitle, \$randomJobType

Teléfono, dirección y ubicación: \$randomPhoneNumber, \$randomPhoneNumberExt, \$randomCity, \$randomStreetName, \$randomStreetAddress, \$randomCountry, \$randomCountryCode, \$randomLatitude, \$randomLongitude

Negocio: \$randomCompanyName, \$randomCompanySuffix, \$randomBs, \$randomBsAdjective, \$randomBsBuzz, \$randomBsNoun

Financieras: \$randomBankAccount, \$randomBankAccountName, \$randomCreditCardMask, \$randomBankAccountBic, \$randomBankAccountIban, \$randomTransactionType, \$randomCurrencyCode, \$randomCurrencyName, \$randomCurrencySymbol, \$randomBitcoin

Ventas: \$randomPrice, \$randomProduct, \$randomProductAdjective, \$randomProductMaterial, \$randomProductName, \$randomDepartment

Frases con gancho: \$randomCatchPhrase, \$randomCatchPhraseAdjective, \$randomCatchPhraseDescriptor, \$randomCatchPhraseNoun

Gramática: \$randomNoun, \$randomVerb, \$randomIngverb, \$randomAdjective, \$randomWord, \$randomWords, \$randomPhrase

Lorem Ipsum: \$randomLoremWord, \$randomLoremWords, \$randomLoremSentence, \$randomLoremSentences, \$randomLoremParagraph, \$randomLoremParagraphs, \$randomLoremText, \$randomLoremSlug, \$randomLoremLines

Imágenes: \$randomImage, \$randomAvatarImage, \$randomImageUrl, \$randomAbstractImage, \$randomAnimalsImage, \$randomBusinessImage, \$randomCatsImage, \$randomCityImage, \$randomFoodImage, \$randomNightlifeImage, \$randomFashionImage, \$randomPeopleImage, \$randomNatureImage, \$randomSportsImage, \$randomTechnicsImage, \$randomTransportImage, \$randomImageDataUri

© JMA 2020. All rights reserved

Solicitudes

- Una solicitud al API permite ponerse en contacto con un servidor con puntos finales del API y realizar alguna acción. Las acciones son métodos HTTP.
- Los métodos más comunes son GET, POST, PUT y DELETE. Los nombres de los métodos se explican por sí mismos. Por ejemplo, GET le permite recuperar datos de un servidor. POST le permite agregar datos a un archivo o recurso existente en un servidor. PUT le permite reemplazar un archivo o recurso existente en un servidor. Y DELETE le permite eliminar datos de un servidor.
- Postman simplifica el envío de solicitudes de API. En lugar de probar las API a través de una línea de comando o terminal, ofrece una interfaz gráfica intuitiva que es rápida de aprender y fácil de dominar.

© JMA 2020. All rights reserved

Peticiones

- **Método:** indica la acción HTTP que se desea efectuar sobre el recurso identificado
- **URL:** identificador del recurso, opcionalmente puede contar con Path Variables (/id/) y Query Params (?clave1=valor1& clave2=valor2).
- **Params:** pares clave-valor, agrupados en Path Variables y Query Params, Postman combina todo en la cadena de consulta anterior (URL).
- **Headers:** pares clave-valor, HTTP cuenta con un amplio conjunto de cabeceras y valores predefinidos, se pueden crear cabeceras personalizadas. Postman proporciona sugerencias de encabezados HTTP comunes y sus valores. Manage Presets permite cachear cabeceras predefinidas.
- **Cookies:** Puede administrar cookies en aplicaciones nativas utilizando el administrador de cookies para editar las cookies asociadas con cada dominio.
 - nombre=valor; path=/; domain=localhost; HttpOnly; Expires=Tue, 19 Jan 2038 03:14:07 GMT;

© JMA 2020. All rights reserved

Peticiones

- Authorization: Permite establecer la autenticación, las opciones son: No Auth, Inherit auth from parent, Basic auth, Digest Auth, NTLM Authentication, Bearer Token, OAuth 1.0, OAuth 2.0, Hawk Authentication, AWS Signature.
- Body (Request): Cuerpo de la petición, acepta las opciones: none (sin cuerpo), form-data (simula el relleno de un formulario codificado en multipart/form-data por lo que acepta el envío de ficheros), x-www-form-urlencoded (pares clave-valor similares a los query params), raw (formato libre pero debe establecerse el Content-Type: text, JSON, XML, HTML, Javascript), binary (hay que seleccionar un fichero), GraphQL (cuerpo en formato de consulta GraphQL).
- Pre-request scripts: Fragmentos de código asociados con una solicitud de recopilación que se ejecutan antes de enviar la solicitud.
- Test: Fragmentos de código que se ejecutan después de realizar la petición y recibir la respuesta, contienen las aserciones asociadas a la prueba.

© JMA 2020. All rights reserved

Respuestas

- El visor de respuestas de Postman ayuda a garantizar la exactitud de las respuestas API.
- Una respuesta API consiste en el cuerpo, las cookies, los encabezados y el código de estado.
- Postman organiza el cuerpo, las cookies y los encabezados en diferentes pestañas. La pestaña Body brinda varias vistas para ayudar a comprender la respuesta rápidamente: Pretty, Raw, Preview y Visualize.
- El código de estado, la información de red, el tiempo de duración de la llamada API y el tamaño de la respuesta están visibles junto a las pestañas.
- La pestaña test contiene los resultados de los script de test que se ejecutaron contra la solicitud.
- Cuando se trabaja con colecciones, la respuesta se puede guardar como fichero para un análisis posterior o como un ejemplo a efectos de documentación.

© JMA 2020. All rights reserved

Visualizadores

- Los visualizadores permiten presentar los datos de respuesta de manera que ayuden a darle sentido.
- Para establecer la visualización de los datos de la respuesta, se agrega una plantilla ([Handlebars](#)) en el código a la secuencia de comandos Pre-solicitud o Pruebas para la solicitud. El método aplicará la plantilla a los datos y lo presentará en la pestaña Visualize cuando se ejecute la solicitud.`pm.visualizer.set()`

```
var template = `




```

© JMA 2020. All rights reserved

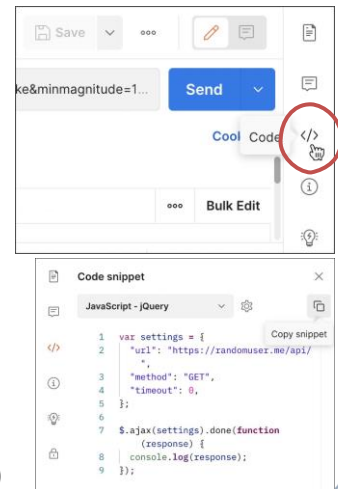
Ejemplos de especificación

- Los ejemplos muestran los puntos finales de su API en acción y brindan más detalles sobre cómo funcionan las solicitudes y las respuestas. Se puede agregar un ejemplo a una solicitud guardando una respuesta, o se puede crear manualmente con una respuesta personalizada para ilustrar un caso de uso específico. Una vez se hayan creado ejemplos, se pueden usar para repetir las peticiones con diferentes casos de uso, configurar un servidor simulado o agregar más detalles a la documentación del API.
- Un ejemplo es un emparejamiento formado por una solicitud y una respuesta relacionada. Cada ejemplo incluye una parte de solicitud (método, URL, parámetros, encabezados y cuerpo) y una parte de respuesta (código de estado, cuerpo y encabezados). Los ejemplos se crean agregándolos a solicitudes en colecciones, y una solicitud puede tener varios ejemplos.
- Tener múltiples ejemplos para una solicitud es útil para ilustrar las diferentes formas en que un punto final puede responder a una solicitud. Se pueden tener ejemplos que respondan con diferentes códigos de estado (como 200 o 404) o que devuelvan datos diferentes (o ningún dato).

© JMA 2020. All rights reserved

Generación de código de cliente

- Postman puede convertir una solicitud de API en un fragmento del código fuente a usar en la aplicación cliente del API.
- Se puede elegir entre múltiples lenguajes de programación y, en algunos casos, entre diferentes marcos soportados por el lenguaje, tanto de front end como de back end.
- Postman admite los siguientes:
 - C (LibCurl), C# (HttpClient, RestSharp), cURL (cURL), Dart (Dart), Go (http package), HTTP (Raw HTTP request), Java (OkHttp, Unirest), JavaScript (Fetch, jQuery, XHR), NodeJS (Axios, Native, Request, Unirest), Objective-C (NSURLSession), OCaml (Cohttp), PHP (cURL, Guzzle, Http_Request2, pecl_http), PowerShell (RestMethod), Python (http.client (Python 3), Requests), R (httr, RCurl), Ruby (NET::Http), Shell (Httpie, wget), Swift (URLSession)




© JMA 2020. All rights reserved

Solucionar problemas de solicitudes

- Si la solicitud al API no se comporta como se esperaba, puede haber muchas razones posibles. Para averiguar cuál es el problema, se puede usar Postman Console para solucionar su solicitud.
- Cada solicitud enviada por Postman se registra en la consola, por lo que se pueden ver los detalles de lo que sucedió cuando envió una solicitud.
- El objeto console permite escribir en la consola mensajes de depuración desde los script.
- La consola Postman registra la siguiente información:
 - La solicitud principal que se envió, incluidos todos los encabezados de solicitud subyacentes, los valores de las variables y las redirecciones
 - La configuración del proxy y los certificados utilizados para la solicitud.
 - Información de red, como direcciones IP, cifrados y protocolos utilizados
 - Instrucciones console y solicitudes asíncronas de los scripts de prueba o solicitud previa
 - La respuesta sin procesar enviada por el servidor antes de que Postman la procese
- Mantener la consola abierta mientras trabaja aumentará la visibilidad de las llamadas de su red y registrará los mensajes durante la depuración.

© JMA 2020. All rights reserved

Captura de solicitudes

- Capturar el tráfico HTTP es una herramienta importante para el desarrollo y las pruebas de API. Cuando se habilita la captura de solicitudes en Postman, se puede inspeccionar las solicitudes que pasan entre las aplicaciones cliente y el API para guardarlas en una colección. Luego se puede usar la información de las solicitudes guardada para comprender cómo se usa y comporta el API, ayudar con la depuración y generar las baterías de pruebas.
- El proxy integrado de Postman y el complemento de navegador Postman Interceptor proporcionan dos formas de capturar el tráfico HTTP y HTTPS. También puede usar el proxy o Interceptor para capturar y sincronizar cookies con el contenedor de cookies de Postman. La configuración de seguridad puede limitar o impedir la captura del tráfico.
- Una sesión de depuración representa un período de tiempo específico durante el cual desea capturar tráfico (por ejemplo, mientras una aplicación cliente envía una serie de solicitudes que desea observar o depurar). Para capturar el tráfico, debe iniciarse el proxy o el Interceptor antes de iniciar una sesión de depuración (Capturar solicitudes barra del pie). 
- Después de comenzar una sesión de depuración, puede pausar y reanudar la captura, o borrar el tráfico capturado, sin detener el proxy o el Interceptor. Utilice las capacidades de búsqueda y filtrado de Postman para limitar las solicitudes según los criterios que elija.

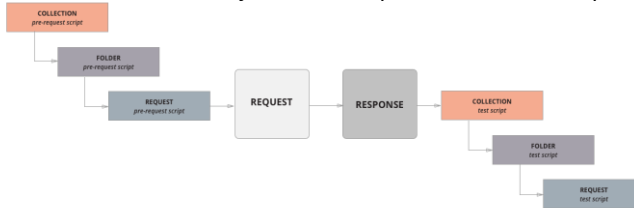
© JMA 2020. All rights reserved

PRUEBAS AUTOMATIZADAS

© JMA 2020. All rights reserved

Secuencia de comandos

- Postman contiene un poderoso motor de ejecución basado en Node.js que permite agregar un comportamiento dinámico a las solicitudes y colecciones. Esto permite escribir conjuntos de pruebas, crear solicitudes que pueden contener parámetros dinámicos, pasar datos entre solicitudes y mucho más.
- Puede agregar código JavaScript para ejecutar durante dos puntos del flujo:
 - Antes de que una solicitud se envíe al servidor, como un script de pre-solicitud (pestaña Pre-request Script).
 - Después de recibir una respuesta, como un script de prueba (pestaña Tests).
- En caso de que estén disponibles, para cada solicitud de una colección, los scripts siempre se ejecutarán de acuerdo con la siguiente jerarquía: script de nivel de colección, script de nivel de carpeta, script de nivel de solicitud. Este orden de ejecución se aplica tanto a los scripts de solicitud previa como de prueba.



© JMA 2020. All rights reserved

Objeto pm

<https://learning.postman.com/docs/writing-scripts/script-references/postman-sandbox-api-reference/>

- El objeto pm contiene toda la información relacionada con el script que se está ejecutando y permite a uno acceder a una copia de la solicitud que se envía o de la respuesta recibida.
- También permite obtener y establecer el entorno y las variables globales.
- Dispone de métodos para enviar solicitudes, crear casos de pruebas y aserciones.

© JMA 2020. All rights reserved

Propiedades de pm

- `pm.info`: contiene información perteneciente al script que se está ejecutando como el nombre de la solicitud, el ID de la solicitud y el recuento de iteraciones se almacenan dentro de este objeto.
- `pm.variables`: da acceso a todas las variables se ajustan a una jerarquía específica y permite modificar sus valores en función de su ámbito.
- `pm.environment`: da acceso a las variables de entorno y permite modificar sus valores.
- `pm.globals`: da acceso a las variables de globales y permite modificar sus valores.
- `pm.request`: es una representación de la solicitud (URL y cabeceras) para la que se ejecuta el script. Para un script de pre-solicitud es la solicitud que está a punto de enviarse y en un script de prueba es la representación de la solicitud que se envió.
- SOLO están disponibles en los scripts de prueba:
 - `pm.iterationData`: da acceso a los valores actuales de variables obtenidos del archivo de datos proporcionado durante una ejecución de recopilación.
 - `pm.cookies`: da acceso a los valores almacenados en las cookies.

© JMA 2020. All rights reserved

pm.response

- Solo disponible dentro de los scripts de prueba, contiene toda la información relacionada con la respuesta que se recibió.
- Los detalles accesibles de la respuesta son:
 - `pm.response.code` → Number: Versión numérica del HTTP Status Code.
 - `pm.response.reason()` → String: Versión textual del HTTP Status Code.
 - `pm.response.headers` → HeaderList: Colección con las cabeceras recibidas (Objetos: {key:"", value:""}).
 - `pm.response.responseTime` → Number: Tiempo de duración de la llamada API, en milisegundos.
 - `pm.response.text()` → String: Versión de textual del cuerpo de la respuesta para su tratamiento como cadena.
 - `pm.response.json()` → Object: Objeto JavaScript obtenido al hacer la des serialización con `JSON.parse()` del cuerpo de la respuesta (el content-type debe ser JSON) para su tratamiento como objeto.

© JMA 2020. All rights reserved

Scripts previos a la solicitud

- Los scripts de pre-solicitud son fragmentos de código JavaScript asociados con una solicitud de recopilación que se ejecutan antes de enviar la solicitud.
- Esto es perfecto para los casos de uso que requieran una preparación previa a la solicitud, como incluir una marca de tiempo en los encabezados de solicitud, dar formato a los datos de salida o incluir una traza en el log.
`console.info('Inicio de la prueba');`
`pm.variables.set('ahora', new Date());`
- Se puede agregar scripts de pre-solicitud a una colección, una carpeta o una sola solicitud. Los script de colección y carpeta permiten reutilizar el código ejecutado comúnmente antes de cada solicitud.

© JMA 2020. All rights reserved

Scripts de prueba

- Los scripts de prueba son fragmentos de código JavaScript asociados con una solicitud de recopilación que se ejecutan después de recibir la respuesta de la solicitud. Aunque están destinados a implementar los casos de prueba pueden incluir trazas en el log así como la captura y cacheo en variables de los valores recibidos.
- Para añadir las especificaciones de prueba se utiliza el método `pm.test`. El método recibe una cadena con el título o descripción de la especificación y la función que contiene el caso de prueba y asegura que el resto del script no se bloquee, incluso si hay errores dentro de la función.
`pm.test("Status code is 200", function () {
 pm.response.to.have.status(200);
});`
- La función con el caso de prueba puede recibir como parámetro la función con la que indicar que una prueba asíncrona a concluido:
`pm.test('async test', function (done) {
 setTimeout(() => { pm.expect(pm.response.code).to.equal(200); done(); }, 1500);
});`
- Con `pm.test.index()` se obtiene el número total de pruebas de una ubicación específica.

© JMA 2020. All rights reserved

Scripts de prueba

- Una especificación contiene una o más expectativas (algo que se espera) que ponen a prueba el estado de la respuesta. Una expectativa es una afirmación que debe ser verdadera pero puede ser falsa.
- Una especificación con todas las expectativas verdaderas es una especificación que pasa la prueba, pero con una o más falsas es una especificación que falla.
- Las expectativas se construyen con el método `pm.expect` que obtiene un valor real de una expresión y se encadenan con afirmaciones del lenguaje natural para compararlo con el valor esperado (constante).
`pm.expect(valor actual).to.equal(valor esperado);`
- Postman utiliza la biblioteca [ChaiJS](#) con el estilo `apto` para cadenas del BDD que proporciona un lenguaje expresivo y un estilo legible.

© JMA 2020. All rights reserved

Expectativas

- Los siguientes elementos se proporcionan como captadores encadenables para mejorar la legibilidad de sus afirmaciones:
 - `.to .be .been .is .that .which .and .has .have .with .at .of .same .but .does .still`
 - `.not`: Niega todas las afirmaciones que siguen en la cadena.
 - `.equal(val)`: verifica si ambos objetos son idénticos `===`.
 - `.eq(obj)`: verifica si ambos objetos son idénticos `===` (búsqueda profunda).
 - `.ok`: verifica si el valor se evalúa como verdadero.
 - `.true`: verifica si el valor es verdadero.
 - `.false`: verifica si el valor es falso.
 - `.null`: verifica si el valor es nulo.
 - `.undefined`: verifica si el valor es indefinido.
 - `.NaN`: verifica si el valor es NaN.

© JMA 2020. All rights reserved

Expectativas

- `.exist`: verifica si el valor es distinto de null y undefined.
- `.empty`: verifica si el valor de la propiedad `length` es 0.
- `.finite`: verifica si el valor es distinto de NaN y NaN.
- `.arguments`: verifica si el valor es la propiedad `arguments` de una función.
- `.above(n)`: verifica si el valor actual es mayor que el esperado.
- `.least(n)`: verifica si el valor actual es mayor o igual que el esperado.
- `.below(n)`: verifica si el valor actual es menor que el esperado.
- `.most(n)`: verifica si el valor actual es menor o igual que el esperado.
- `.within(start, finish)`: verifica si el valor actual esta en el rango esperado.
- `.oneOf(list)`: verifica si el valor actual es uno de la lista esperada.
- `.a(type)`: verifica si el valor actual es del tipo esperado.

© JMA 2020. All rights reserved

Expectativas

- `.instanceof(constructor)`: verifica si el objeto actual es una instancia del tipo esperado.
- `.include(val)`: verifica si el objeto actual incluye el esperado.
- `.string(str)`: verifica si la cadena actual contiene el esperado.
- `.property(name[, val])`: verifica si el objeto actual es tiene la propiedad esperado o el valor de la propiedad es el esperado.
- `.keys(key1[, key2[, ...]])`: verifica si el objeto actual contiene las propiedades o metodos esperados.
- `.lengthOf(n)`: verifica si la longitud del valor actual es mayor es la esperada.
- `.match(regex)`: verifica si el valor pertenece al patrón establecido.
- `.throw([errorLike], [errMsgMatcher])`: verifica si una función lanza una excepción.

© JMA 2020. All rights reserved

Expectativas personalizadas

`.satisfy (matcher)`): el método recibe la función `matcher` que con el valor actual debe devolver verdadero si satisface la aserción o falso si debe fallar.

```
expect(value).to.satisfy(function(num) {  
  return num > 0;  
});
```

`.fail([message])`: Marca directamente la expectativa como fallida y por lo tanto también la especificación. Se utiliza en las verificaciones por código que desean utilizar el mecanismo de fallos de las expectativas:

```
if ...  
  expect.fail();
```

© JMA 2020. All rights reserved

API de respuesta

- `pm.response.to.have.status(code:Number)`
- `pm.response.to.have.status(reason:String)`
- `pm.response.to.have.header(key:String)`
- `pm.response.to.have.header(key:String, optionalValue:String)`
- `pm.response.to.have.body()`
- `pm.response.to.have.body(optionalValue:String)`
- `pm.response.to.have.body(optionalValue:RegExp)`
- `pm.response.to.have.jsonBody()`
- `pm.response.to.have.jsonBody(optionalExpectEqual:Object)`
- `pm.response.to.have.jsonBody(optionalExpectPath:String)`
- `pm.response.to.have.jsonBody(optionalExpectPath:String, optionalValue:*)`
- `pm.response.to.have.jsonSchema(schema:Object)`
- `pm.response.to.have.jsonSchema(schema:Object, ajvOptions:Object)`

© JMA 2020. All rights reserved

API de respuesta

- `pm.response.to.be.info`: Comprueba el código de estado 1XX
- `pm.response.to.be.success`: Comprueba el código de estado 2XX
- `pm.response.to.be.redirection`: Comprueba el código de estado 3XX
- `pm.response.to.be.clientError`: Comprueba el código de estado 4XX
- `pm.response.to.be.serverError`: Comprueba el código de estado 5XX
- `pm.response.to.be.error`: Comprueba el código de estado 4XX o 5XX
- `pm.response.to.be.ok`: El código de estado debe ser 200
- `pm.response.to.be.accepted`: El código de estado debe ser 202
- `pm.response.to.be.badRequest`: El código de estado debe ser 400
- `pm.response.to.be.unauthorized`: El código de estado debe ser 401
- `pm.response.to.be.forbidden`: El código de estado debe ser 403
- `pm.response.to.be.notFound`: El código de estado debe ser 404
- `pm.response.to.be.rateLimited`: El código de estado debe ser 429

© JMA 2020. All rights reserved

pm.sendRequest

- El método `pm.sendRequest` permite enviar solicitudes HTTP/HTTPS de forma asíncrona. Con scripts asíncronos, se puede ejecutar la lógica en segundo plano si una tarea es computacional pesada o se están enviando múltiples solicitudes. En lugar de esperar a que se complete una llamada y bloquear las siguientes solicitudes, se puede designar una función de devolución de llamada y recibir una notificación cuando haya finalizado una operación subyacente.
- El método acepta una solicitud compatible con el SDK de recopilación y una devolución de llamada. La devolución de llamada recibe dos argumentos, un error (si corresponde) y una respuesta compatible con SDK.
- Se puede utilizar en el script de pre-solicitud o en el script de prueba.

```
pm.sendRequest('https://postman-echo.com/get', function (err, res) {  
  if (err) { console.log(err); }  
  pm.test('response should be okay to process', function () {  
    pm.expect(err).to.equal(null);  
    pm.expect(res).to.have.property('code', 200);  
    pm.expect(res).to.have.property('status', 'OK');  
  });  
});
```

© JMA 2020. All rights reserved

Ejemplos

```
pm.test("response is ok", function () {  
  pm.response.to.have.status(200);  
});  
  
pm.test("environment to be production", function () {  
  pm.expect(pm.environment.get("env")).to.equal("production");  
});  
  
pm.test("response should be okay to process", function () {  
  pm.response.to.not.be.error;  
  pm.response.to.have.jsonBody("");  
  pm.response.to.not.have.jsonBody("error");  
});  
  
pm.test("response must be valid and have a JSON body", function () {  
  pm.response.to.be.success; // info, success, redirection, clientError, serverError, are other variants  
  pm.response.to.be.withBody;  
  pm.response.to.be.json;  
});
```

© JMA 2020. All rights reserved

Fragmentos

- Si bien hay muy pocas cosas que recordar al escribir pruebas, Postman intenta facilitar el proceso al enumerar fragmentos de uso común al lado del editor.
- Se puede seleccionar el fragmento que se desea agregar y el código apropiado se completa en el editor de prueba.
- Esta es una excelente manera de crear rápidamente casos de prueba.
- Se dispone de SNIPPETS para recuperar y modificar variables de diferentes ámbitos, casos de prueba que evalúan de diferentes formas el cuerpo, las cabeceras, el estado, la duración, ...

© JMA 2020. All rights reserved

Ver resultados

- Postman ejecuta pruebas cada vez que ejecuta una solicitud.
- Los resultados se muestran en una pestaña Test del visor de respuestas.
- El encabezado de la pestaña muestra cuántas pruebas pasaron, y los resultados de las pruebas se enumeran aquí.
- Si la prueba se evalúa como verdadera, la prueba pasó.
- Se puede filtrar los resultados por pruebas pasadas o fallidas.

© JMA 2020. All rights reserved

JavaScript Object Notation
<http://tools.ietf.org/html/rfc4627>

JSON

© JMA 2020. All rights reserved

Introducción

- JSON (JavaScript Object Notation) es un formato sencillo para el intercambio de información.
- El formato JSON permite representar estructuras de datos (arrays) y objetos (arrays asociativos) en forma de texto.
- La notación de objetos mediante JSON es una de las características principales de JavaScript y es un mecanismo definido en los fundamentos básicos del lenguaje.
- En los últimos años, JSON se ha convertido en una alternativa al formato XML, ya que es más fácil de leer y escribir, además de ser mucho más conciso.
- No obstante, XML es superior técnicamente porque es un lenguaje de marcado, mientras que JSON es simplemente un formato para intercambiar datos.
- La especificación completa del JSON es la RFC 4627, su tipo MIME oficial es application/json y la extensión recomendada es .json.

© JMA 2020. All rights reserved

Estructuras

- JSON está constituido por dos estructuras:
 - Una colección de pares de nombre/valor. En los lenguajes esto es conocido como un objeto, registro, estructura, diccionario, tabla hash, lista de claves o un arreglo asociativo.
 - Una lista ordenada de valores. En la mayoría de los lenguajes, esto se implementa como tablas, arreglos, vectores, listas o secuencias.
- Estas son estructuras universales; virtualmente todos los lenguajes de programación las soportan de una forma u otra. Es razonable que un formato de intercambio de datos que es independiente del lenguaje de programación se base en estas estructuras.

© JMA 2020. All rights reserved

Sintaxis

- Un array es un conjunto de valores separados por comas (,) que se encierran entre corchetes [...]
- Un objeto es un conjunto de pares nombre:valor separados por comas (,) que se acotan entre llaves { ... }
- Los nombres son cadenas, entre comillas dobles (").
- El separador entre el nombre y el valor son los dos puntos (:)
- El valor debe ser un objeto, un array, un número, una cadena o uno de los tres nombres literales siguientes (en minúsculas):
 - true, false o null
- Se codifica en Unicode, la codificación predeterminada es UTF-8.

© JMA 2020. All rights reserved

Valores numéricos

- La representación de números es similar a la utilizada en la mayoría de los lenguajes de programación.
- Un número contiene una parte entera que puede ser prefijada con un signo menos opcional, que puede ser seguida por una parte fraccionaria y / o una parte exponencial.
- La parte fraccionaria comienza con un punto (como separador decimal) seguido de uno o más dígitos.
- La parte exponencial comienza con la letra E en mayúsculas o minúsculas, lo que puede ser seguido por un signo más o menos, y son seguidas por uno o más dígitos.
- Los formatos octales y hexadecimales no están permitidos. Los ceros iniciales no están permitidos.
- No se permiten valores numéricos que no se puedan representar como secuencias de dígitos (como infinito y NaN).

© JMA 2020. All rights reserved

Valores cadena

- La representación de las cadenas es similar a las convenciones utilizadas en la familia C de lenguajes de programación.
- Una cadena comienza y termina con comillas (").
- Se pueden utilizar todos los caracteres Unicode dentro de las comillas con excepción de los caracteres que se deben escapar: los caracteres de control (U + 0000 a U + 001F) y los caracteres con significado.
- Cuando un carácter se encuentra fuera del plano multilingüe básico (U + 0000 a U + FFFF), puede ser representado por su correspondiente valor hexadecimal. Las letras hexadecimales A-F puede ir en mayúsculas o en minúsculas.
- Secuencias de escape:
 - `\\, \/, \", \n, \r, \b, \f, \t`
 - `\u[0-9A-Fa-f]{4}`

© JMA 2020. All rights reserved

Objeto con anidamientos

```
{
  "Image": {
    "Width": 800,
    "Height": 600,
    "Title": "View from 15th Floor",
    "Thumbnail": {
      "Url": "/image/481989943",
      "Height": 125,
      "Width": "100"
    },
    "IDs": [116, 943, 234, 38793]
  }
}
```

© JMA 2020. All rights reserved

Array de objetos

```
[
  {
    "precision": "zip",
    "Latitude": 37.7668,
    "Longitude": -122.3959,
    "City": "SAN FRANCISCO",
    "State": "CA",
    "Zip": "94107"
  },
  {
    "precision": "zip",
    "Latitude": 37.371991,
    "Longitude": -122.026020,
    "City": "SUNNYVALE",
    "State": "CA",
    "Zip": "94085"
  }
]
```

© JMA 2020. All rights reserved

JSON en JavaScript

- El Standard Built-in ECMAScript Objects define que todo interprete de JavaScript debe contar con un objeto JSON como miembro del objeto Global.
- El objeto debe contener, al menos, los siguientes miembros:
 - **JSON.parse** (Función): Convierte una cadena de la notación de objetos de JavaScript (JSON) en un objeto de JavaScript.
 - **JSON.stringify** (Función): Convierte un valor de JavaScript en una cadena de la notación de objetos JavaScript (JSON).

© JMA 2020. All rights reserved

JSONPath

- JSONPath es una forma estandarizada para consultar elementos de un objeto JSON. JSONPath utiliza expresiones de ruta similares a XPath para desplazarse por elementos, elementos anidados y matrices en un documento JSON.
- JSONPath solo cubre las partes esenciales de XPath 1.0.
- Las expresiones JSONPath siempre se refieren a una estructura JSON de la misma manera que las expresiones XPath se usan en combinación con un documento XML. Dado que una estructura JSON suele ser anónima y no necesariamente tiene un "objeto miembro raíz", JSONPath asume el nombre abstracto \$ asignado al objeto de nivel externo.
- Las expresiones JSONPath pueden usar la notación de puntos
 - \$.store.book[0].title
- o la notación corchetes
 - \$('store')['book'][0]['title']

© JMA 2020. All rights reserved

JSONPath

- JSONPath permite el símbolo comodín * para nombres de miembros e índices de matriz.
 - \$.store.*
- Toma prestado el operador descendiente '..' de E4X y la sintaxis de recorte de matriz [start:end:step] de ECMAScript 4.
 - \$..author
 - \$..book[0:]
- Las expresiones del lenguaje de script subyacente (<expr>) se pueden usar como una alternativa a los nombres o índices explícitos, usando el símbolo '@' para el objeto actual, como en
 - \$.store.book[(@.length-1)].title
- Las expresiones de filtro son compatibles a través de la sintaxis ?(<boolean expr>) como en
 - \$.store.book[?(@.price < 10)].title

© JMA 2020. All rights reserved

JSONPath

XPath	JSONPath	Descripción
/	\$	el objeto / elemento raíz
.	@	el objeto / elemento actual
/	. or []	operador hijo
..	n/a	operador padre
//	..	descenso recursivo
*	*	comodín. Todos los objetos / elementos independientemente de sus nombres.
@	n/a	acceso al atributo. Las estructuras JSON no tienen atributos.
[]	[]	operador de subíndice. XPath lo usa para iterar sobre colecciones de elementos y para predicados . En Javascript y JSON es el operador de matriz nativo.
	[,]	el operador de unión en XPath da como resultado una combinación de conjuntos de nodos. JSONPath permite nombres alternativos o índices de matriz como un conjunto.
n/a	[start:end:step]	operador de recorte de matriz.
[]	?()	aplica una expresión de filtro
n/a	()	expresión de script, utilizando el motor de script subyacente.

© JMA 2020. All rights reserved

JSONPath

XPath	JSONPath	Resultado
/store/book/author	\$.store.book[*].author	los autores de todos los libros en la tienda
//author	\$..author	todos los autores
/store/*	\$.store.*	todas las cosas en la tienda
/store//price	\$.store..price	el precio de todo en la tienda
//book[3]	\$..book[2]	el tercer libro
//book[last()]	\$..book[(@.length-1)] \$..book[-1:]	el último libro en orden.
//book[position()<3]	\$..book[0,1] \$..book[:2]	los dos primeros libros
//book[isbn]	\$..book[?(@.isbn)]	filtrar todos los libros con número isbn
//book[price<10]	\$..book[?(@.price<10)]	filtrar todos los libros más baratos que 10
//*	\$..*	todos los elementos en el documento XML o todos los miembros de la estructura JSON.

© JMA 2020. All rights reserved

JSON Patch

- [JSON Patch](#) es un formato (identificado por el MIME "application/json-patch+json") para especificar las actualizaciones que se aplicarán a un recurso, expresado como una secuencia de operaciones a aplicar en un documento JSON de destino. Es adecuado para usar con el método HTTP PATCH o en cualquier situación que requiera realizar actualizaciones parciales a un documento JSON.
- Un documento JSON Patch es un documento JSON que representa una matriz de objetos. Cada objeto representa una operación única a aplicar al documento JSON de destino con las propiedades:
 - **op**: indica el tipo de operación sobre las propiedades de un objeto o los elementos de una matriz.
 - **path**: indica el elemento que se va a actualizar en notación [JSON Pointer](#).
 - **value**: proporciona el nuevo valor.

© JMA 2020. All rights reserved

JSON Patch

```
PATCH /my/data HTTP/1.1
Host: example.org
Content-Length: 326
Content-Type: application/json-patch+json
If-Match: "abc123"
```

```
[
  { "op": "test", "path": "/a/b/c", "value": "foo" },
  { "op": "remove", "path": "/a/b/c" },
  { "op": "add", "path": "/a/b/c", "value": [ "foo", "bar" ] },
  { "op": "replace", "path": "/a/b/c", "value": 42 },
  { "op": "move", "from": "/a/b/c", "path": "/a/b/d" },
  { "op": "copy", "from": "/a/b/d", "path": "/a/b/e" }
]
```

© JMA 2020. All rights reserved

JSON Patch

- Las operaciones admitidas son:

- add** Agrega una propiedad a un objeto o un elemento de matriz. Si la propiedad existe establece su valor.
- remove** Quita una propiedad de un objeto o un elemento de la matriz.
- replace** Lo mismo que remove seguido de add en la misma ubicación.
- move** Lo mismo que remove desde el origen seguido de add al destino con el valor del origen.
- copy** Lo mismo que add al destino con el valor del origen.
- test** Genera un error si el valor referenciado en el path no coincide con el value proporcionado para impedir una actualización.

© JMA 2020. All rights reserved

JSON Pointer

- JSON Pointer ([IETF RFC 6901](#)) define un formato de cadena para identificar un valor específico dentro de un documento JSON. Lo utilizan todas las operaciones en JSON Patch para especificar la parte del documento en la que operar.
- Un puntero JSON es una cadena de tokens separados por caracteres /, estos tokens especifican propiedades en objetos o índices en matrices. Por ejemplo, dado el JSON

```
{
  "biscuits": [
    { "name": "Digestive" },
    { "name": "Choco Leibniz" }
  ]
}
```

 - /biscuits apuntaría a la matriz de galletas y /biscuits/1/name apuntaría a "Choco Leibniz".
 - Para apuntar a la raíz del documento, se usa una cadena vacía para el puntero. El puntero / no apunta a la raíz, apunta a una clave "" de la raíz (que es totalmente válida en JSON).
 - Si se necesita hacer referencia a una clave con ~ o / en el nombre, se debe escapar los caracteres con ~0 y ~1 respectivamente. Para obtener "baz" de { "foo/bar~": "baz" } se usaría el puntero /foo~1bar~0.
 - Finalmente, si es necesario referirse al final de una matriz, se puede usar el - en lugar del índice. Por ejemplo, para referirse al final de la matriz de galletas anterior, se usaría /biscuits/-. Esto es útil cuando se necesita insertar un valor al final de una matriz.

© JMA 2020. All rights reserved

JSON Schema

<https://json-schema.org/>

- JSON Schema es una especificación para definir, anotar y validar las estructuras de datos JSON. La especificación tiene varias definiciones formales y versiones.
- Un esquema de JSON contiene a qué versión de la especificación se ajusta, el identificador o la ubicación del esquema, un título, una descripción y el tipo de objeto del documento raíz.
- Define qué propiedades junto con sus tipos ha de contener el documento JSON al que se aplica, cuáles de esas propiedades son obligatorias y las validaciones sobre los datos como restricciones en los valores de los datos o elementos de un array.
- Un esquema permite el anidamiento de estructuras en las que también se definen que propiedades contienen y cuáles son requeridas.
- Un esquema JSON permite referenciar un esquema JSON externo.

© JMA 2020. All rights reserved

Esquemas de datos

- Los tipos base son string, number, integer, boolean, array y object.
- Con la propiedad format se pueden especificar otros tipos especiales partiendo de los tipos base: long, float, double, byte, binary, date, dateTime, password.
- Los tipos array se definen como una colección de ítems y en dicha propiedad se define el tipo y la estructura de los elementos que lo componen. Los objetos son un conjunto de propiedades, cada una definida dentro de properties.
- Cada tipo y propiedad se identifica por un nombre que no debe estar repetido en su ámbito.
- Cada propiedad puede definir description, default, minimum, maximum, maxLength, minLength, pattern, required, readOnly, ...
- Para una propiedad se pueden definir varios tipos (tipos mixtos o unión).
- Los tipos pueden hacer referencia a otros tipos.

© JMA 2020. All rights reserved

Tipos de datos

type	format	Comentarios
boolean		Booleanos: true y false
integer	int32	Enteros con signo de 32 bits
integer	int64	Enteros con signo de 64 bits (también conocidos como largos)
number	float	Reales cortos
number	double	Reales largos
string		Cadenas de caracteres
string	password	Una pista a las IU para ocultar la entrada.
string	date	Según lo definido por full-date RFC3339 (2018-11-13)
string	date-time	Según lo definido por date-time- RFC3339 (2018-11-13T20:20:39+00:00)
string	byte	Binario codificados en base64
string	binary	Binario en cualquier secuencia de octetos
array		Colección de items
object		Colección de properties

© JMA 2020. All rights reserved

Propiedades de los objetos de esquema

- type: integer, number, boolean, string, array, object
- format: long, float, double, byte, binary, date, dateTime, password
- title: Nombre a mostrar en el UI
- description: Descripción de su uso
- maximum: Valor máximo
- exclusiveMaximum: Valor menor que
- minimum: Valor mínimo
- exclusiveMinimum: Valor mayor que
- multipleOf: Valor múltiplo de
- maxLength: Longitud máxima
- minLength: Longitud mínima
- pattern: Expresión regular del patrón
- deprecated: Si está obsoleto y debería dejar de usarse
- nullable: Si acepta nulos
- default: Valor por defecto
- enum: Lista de valores con nombre
- example: Ejemplo de uso
- externalDocs: referencia a documentación externa adicional
- items: Definición de los elementos del array
- maxItems: Número máximo de elementos
- minItems: Número mínimo de elementos
- uniqueItems: Elementos únicos
- properties: Definición de las propiedades del objeto,
- maxProperties: Número máximo de propiedades
- minProperties: Número mínimo de propiedades
- readOnly: propiedad de solo lectura
- writeOnly: propiedad de solo escritura
- additionalProperties: permite referenciar propiedades adicionales
- required: Lista de propiedades obligatorias

© JMA 2020. All rights reserved

schema

```
{
  "definitions": {},
  "$schema": "http://json-schema.org/draft-07/schema#",
  "$id": "https://example.com/personas.schema.json",
  "title": "Root",
  "type": "array",
  "default": [],
  "items": {
    "$id": "#root/items",
    "title": "Items",
    "type": "object",
    "required": [
      "id",
      "nombre"
    ],
    "properties": {
      "id": {
        "$id": "#root/items/id",
        "title": "Id",
        "type": "integer",
        "examples": [
          2
        ],
        "default": 0
      },
      "nombre": {
        "$id": "#root/items/nombre",
        "title": "Nombre",
        "type": "string",
        "default": ""
      },
      "apellidos": {
        "$id": "#root/items/apellidos",
        "title": "Apellidos",
        "type": "string",
        "default": "",
        "examples": [
          "Grillo"
        ],
        "pattern": "A.*$"
      },
      "edad": {
        "$id": "#root/items/edad",
        "title": "Edad",
        "type": "integer",
        "examples": [
          67
        ],
        "default": 0
      }
    }
  }
}
```

© JMA 2020. All rights reserved

Validar un JSON

- Validar el esquema JSON con el validador de esquema con [ajv](#):

```
var schema = {
  "type": "object",
  "properties": {
    "id": {
      "type": "integer"
    },
    "nombre": {
      "type": "string"
    },
    "apellidos": {
      "type": "string"
    }
  },
  required: ["id", "nombre"]
};

pm.test('Schema is valid', function() {
  pm.response.to.have.jsonSchema(schema);
});
```

© JMA 2020. All rights reserved

Validar un JSON

- Realizar la validación del esquema JSON con Tiny Validator V4 (tv4):

```
const schema = {  
  "items": {  
    "type": "boolean"  
  }  
};  
const data1 = [true, false];  
const data2 = [true, 123];  
  
pm.test('Schema is valid', function() {  
  pm.expect(tv4.validate(data1, schema)).to.be.true;  
  pm.expect(tv4.validate(data2, schema)).to.be.true;  
});
```

© JMA 2020. All rights reserved

COLECCIONES Y FLUJOS

© JMA 2020. All rights reserved

Ejecución de colección

- Las colecciones son grupos de solicitudes que se pueden ejecutar juntas como una serie de solicitudes, en un entorno correspondiente.
- Ejecutar una colección es útil cuando se desea automatizar las pruebas de API. Cuando se ejecuta una colección, envían todas las solicitudes de la colección una tras otra.
- Cuando se usan scripts, se pueden crear conjuntos de pruebas de integración, pasar datos entre solicitudes de API y crear flujos de trabajo que reflejen un caso de uso real de API.
- Para ejecutar una colección se utiliza la utilidad Collection Runner donde habrá que indicar:
 - La colección o carpeta que desea ejecutar.
 - El entorno a utilizar cuando se ejecuta una colección.
 - El número de veces que se ejecutará la colección.
 - El intervalo (en milisegundos) entre cada solicitud en una ejecución de recopilación.
 - El nivel de registro de respuestas cuando se ejecuta la colección.
 - Proporciona un archivo de datos para usar en la ejecución de la recopilación.
 - Si los valores de las variables son persistentes (pruebas irrepetibles)
 - Tratamiento de las cookies.

© JMA 2020. All rights reserved

Flujos de trabajo

- Cuando se inicia una ejecución de recopilación, todas las solicitudes se ejecutan en el orden en que se ven en la aplicación principal, por defecto se ejecutan primero por orden las carpetas y luego cualquier solicitud en la raíz de la colección.
- Se puede alterar el orden de la colección arrastrando y soltando en el panel de navegación de la aplicación principal.
- Para una ejecución concreta se puede alterar el orden en el Collection Runner arrastrando y soltando en el panel RUN ORDER.
- Sin embargo, se puede anular este comportamiento utilizando la función integrada `postman.setNextRequest()`. Esta función permite especificar como argumento el nombre o ID de la solicitud que se desea ejecutar a continuación y establecer una lógica condicional u omitir solicitudes innecesarias.
`postman.setNextRequest("request_name");`

© JMA 2020. All rights reserved

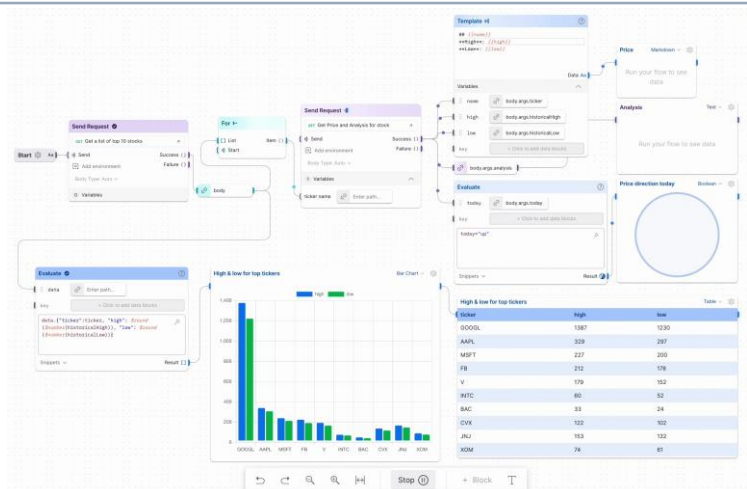
Flujos de trabajo

- Solo funciona con Collection Runner y Newman donde la intención es ejecutar una colección, en lugar de enviar una sola solicitud.
- Siempre se ejecuta al final de la solicitud actual, por lo que se puede colocar en cualquier punto de los scripts de pre-solicitud o test, sin interferir con el resto del código. Si hay más de una asignación, el último valor establecido tiene prioridad.
- Si se ejecuta una colección, se puede saltar a cualquier solicitud de la colección (incluso solicitudes dentro de carpetas). Sin embargo, si se ejecuta una carpeta, el alcance se limita a dicha carpeta y sus subcarpetas.
- Si una solicitud no indica la siguiente se pasa por defecto a la ejecución lineal y continúa por orden con la siguiente solicitud.
- Para detener la ejecución del flujo de trabajo:
`postman.setNextRequest(null);`

© JMA 2020. All rights reserved

Postman Flows

- Postman Flows es una herramienta visual para crear aplicaciones impulsadas por API, una herramienta para crear software para el mundo API-First.
- Se puede usar flujos para encadenar solicitudes, manejar datos y crear flujos de trabajo del mundo real en su espacio de trabajo de Postman.



© JMA 2020. All rights reserved

Ejecución de colecciones

- En Postman, una colección puede ser un grupo de solicitudes guardadas, un flujo de trabajo o un conjunto de pruebas. Se puede ejecutar una colección para enviar solicitudes a los puntos finales de su nueva API, para recuperar datos de una fuente de datos o para probar la funcionalidad de una API. Postman permite ejecutar colecciones de varias maneras:
 - Se puede ejecutar colecciones manualmente, con Collection Runner. Esto es útil cuando se envía una pequeña cantidad de solicitudes y no necesita repetirlas muchas veces.
 - Se puede ejecutar recopilaciones con frecuencia de forma automatizada programando ejecuciones de recopilación o configurando monitores. Las ejecuciones de recopilación programadas son útiles para automatizar las pruebas, y los monitores son útiles para verificar el rendimiento, la disponibilidad y la confiabilidad.
 - Se puede automatizar las ejecuciones de recopilación en el CI/CD con Postman CLI o Newman.
 - Se puede usar webhooks para desencadenar ejecuciones de recopilación en determinados momentos o cuando ocurre un evento específico en su aplicación.

© JMA 2020. All rights reserved

Integración de línea de comando

- Newman es una utilidad de línea de comandos para ejecutar colecciones de Postman y permite ejecutar y probar una Colección Postman directamente desde la línea de comandos. Está destinado a permitir integrar las pruebas de Postman con los servidores de integración continua y sistemas de compilación (Jenkins, Travis CI, AppVeyor, ...).
- Newman se mantiene sincronizado con las características de Postman para permitir ejecutar colecciones de la misma forma en que se ejecutan dentro del Collection Runner de la aplicación Postman.
- Newman se basa en Node.js por lo que es necesario tenerlo instalado. Para instalar Newman:
`npm install -g newman`
- Para ejecutar una colección es necesario exportarla a su fichero JSON. Una vez exportada, se ejecuta con el comando:
`newman run mycollection.json`
- Newman ofrece un amplio conjunto de opciones para personalizar la ejecución:
`newman run -h`

© JMA 2020. All rights reserved

Monitorización

- La supervisión de Postman permite ejecutar una colección periódicamente para verificar su rendimiento y respuesta, un proceso de prueba continuo. Se puede configurar un monitor para que se ejecute con una frecuencia de 5 minutos para verificar si todas las solicitudes de la colección están activas y en buen estado.
- Cuando se configura un monitor, los servidores de Postman realizarán las solicitudes de la colección de acuerdo con la frecuencia especificada. También se puede seleccionar el entorno correspondiente para usar y almacenar variables.
- Si hay pruebas escritas para las solicitudes, el monitor ejecutará estas pruebas para validar la respuesta y notificar cuando falle una prueba. También se puede configurar cómo recibir las alertas de una gran cantidad de integraciones disponibles.
- Cada usuario de Postman recibe 1.000 llamadas de monitoreo gratis al mes. Las cuentas de pago tienen límites más altos: los equipos Postman Pro tienen 10.000 llamadas de monitoreo mensuales incluidas y los equipos Enterprise tienen 100.000 solicitudes mensuales incluidas.

© JMA 2020. All rights reserved

Data Driven Testing (DDT)

- Se basa en la creación de tests para ejecutarse en simultáneo con sus conjuntos de datos relacionados en un framework. El framework provee una lógica de test reusable para reducir el mantenimiento y mejorar la cobertura de test. La entrada y salida (del criterio de test) pueden ser resguardados en uno o más lugares del almacenamiento central o bases de datos, el formato real y la organización de los datos serán específicos para cada caso.
- Todo lo que tiene potencial de cambiar (también llamado "variabilidad," e incluye elementos como el entorno, puntos de salida, datos de test, ubicaciones, etc) está separado de la lógica del test (scripts) y movido a un 'recurso externo'. Esto puede ser configuración o conjunto de datos de test. La lógica ejecutada en el script está dictada por los valores.
- Los datos incluyen variables usadas tanto para la entrada como la verificación de la salida.
- Se pueden usar archivos de datos para pasar conjuntos de valores a Postman para usar en una ejecución de recopilación. Al seleccionar un archivo de datos en Collection Runner, se puede probar sus solicitudes con varios valores como parte de una sola ejecución.
- El uso de archivos de datos es una potente forma de probar el comportamiento de las API en múltiples escenarios. Un archivo de datos contiene los valores de sustitución de las variables para cada iteración de una ejecución de colección.
- Actualmente Postman admite archivos con formato JSON y CSV.

© JMA 2020. All rights reserved

Archivos de datos

- En la forma típica de CSV, la primera fila representa todos los nombres de variables, y las filas siguientes representan valores para estas variables para cada iteración.
accept,content-type,status
application/json,application/json,200
application/xml,application/xml;charset=UTF-8,200
text/plain,application/json,406
- En formato JSON, el fichero contendrá un array de objetos donde el nombre de la propiedad será el nombre de la variable y el valor de la propiedad será el valor de la variable:
[{ "accept": "application/json", "content-type": "application/json", "status": 200 }, ...]
- Solo se puede usar un archivo de datos para una ejecución Collection Runner que es común para todas las solicitudes de la colección por lo que debe contar con todos los valores necesarios.

```
pm.test('Content-Type is ${pm.iterationData.get("content-type")}', function () {  
    pm.response.to.have.header("Content-Type", pm.iterationData.get("content-type"));  
});  
pm.test("Status code is " + pm.iterationData.get("status"), function () {  
    pm.response.to.have.status(+pm.iterationData.get("status"));  
});
```

© JMA 2020. All rights reserved

DOCUMENTACIÓN

© JMA 2020. All rights reserved

Enfoque API First

- El enfoque basado en API First significa que, para cualquier proyecto de desarrollo dado, las APIs se tratan como "ciudadanos de primera clase": que todo sobre un proyecto gira en torno a la idea de que el producto final es un conjunto de APIs consumido por las aplicaciones del cliente.
- El enfoque de API First implica que los desarrollos de APIs sean consistentes y reutilizables, lo que se puede lograr mediante el uso de un lenguaje formal de descripción de APIs para establecer un contrato sobre cómo se supone que se comportará la API. Establecer un contrato implica pasar más tiempo pensando en el diseño de una API.
- A menudo también implica una planificación y colaboración adicionales con las partes interesadas, proporcionando retroalimentación de los consumidores sobre el diseño de una API antes de escribir cualquier código evitando costosos errores.

© JMA 2020. All rights reserved

Beneficios de API First

- Los equipos de desarrollo pueden trabajar en paralelo.
 - Los equipos pueden simular APIs y probar sus dependencias en función de la definición de la API establecida.
- Reduce el coste de desarrollar aplicaciones
 - Las APIs y el código se pueden reutilizar en muchos proyectos diferentes.
- Aumenta la velocidad de desarrollo.
 - Gran parte del proceso de creación de API se puede automatizar mediante herramientas que permiten importar archivos de definición de API y generar el esqueleto del backend y el cliente frontend, así como un mocking server para las pruebas.

© JMA 2020. All rights reserved

Beneficios de API First

- Asegura buenas experiencias de desarrollador
 - Las APIs bien diseñadas, bien documentadas y consistentes brindan experiencias positivas para los desarrolladores porque es más fácil reutilizar el código y los desarrollos integrados, reduciendo la curva de aprendizaje.
- Reduce el riesgo de fallos
 - Reduce el riesgo de fallos al facilitar las pruebas para garantizar que las APIs sean confiables, consistentes y fáciles de usar para los desarrolladores.

© JMA 2020. All rights reserved

Documentar servicios Rest

- Dado que las API están diseñadas para ser consumidas, es importante asegurarse de que el cliente o consumidor pueda implementar rápidamente una API y comprender qué está sucediendo con ella. Desafortunadamente, muchas API hacen que la implementación sea extremadamente difícil, frustrando su propósito.
- La documentación es uno de los factores más importantes para determinar el éxito de una API, ya que la documentación sólida y fácil de entender hace que la implementación de la API sea muy sencilla, mientras que la documentación confusa, desincronizada, incompleta o intrincada hace que sea una aventura desagradable, una que generalmente conduce a desarrolladores frustrados a utilizar las soluciones de la competencia.
- Una buena documentación debe actuar como referencia y como formación, permitiendo a los desarrolladores obtener rápidamente la información que buscan de un vistazo, mientras también leen la documentación para obtener una comprensión de cómo integrar el recurso / método que están viendo.
- Con la expansión de especificaciones abiertas como OpenApi, RAML, ... y las comunidades que las rodean, la documentación se ha vuelto mucho más fácil, aun así requiere invertir tiempo y recursos, todo ello con una cuidadosa planificación.

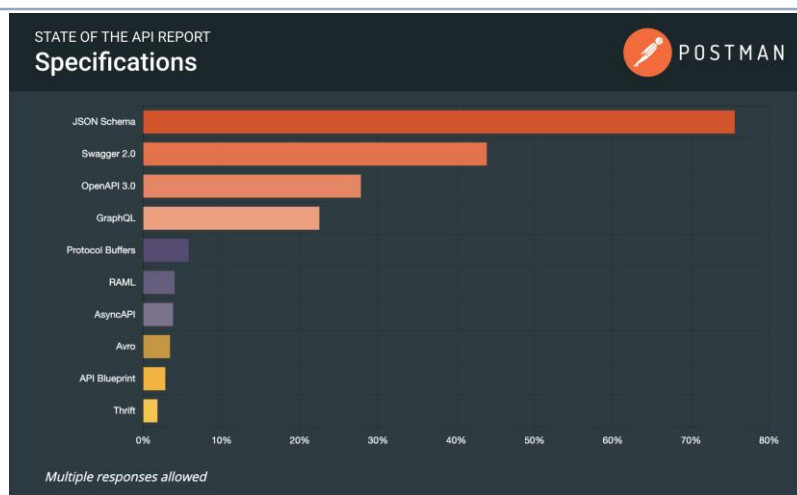
© JMA 2020. All rights reserved

Documentar servicios Rest

- Web Application Description Language (WADL) (<https://www.w3.org/Submission/wadl/>)
 - Especificación de W3C, que la descripción XML legible por máquina de aplicaciones web basadas en HTTP (normalmente servicios web REST). Modela los recursos proporcionados por un servicio y las relaciones entre ellos. Está diseñado para simplificar la reutilización de servicios web basados en la arquitectura HTTP existente de la web. Es independiente de la plataforma y del lenguaje, tiene como objetivo promover la reutilización de aplicaciones más allá del uso básico en un navegador web.
- Spring REST Docs (<https://spring.io/projects/spring-restdocs>)
 - Documentación a través de los test (casos de uso), evita enterrar el código entre anotaciones.
- RAML (<https://raml.org/>)
 - RESTful API Modeling Language es una forma práctica de describir un API RESTful de una manera que sea muy legible tanto para humanos como para máquinas.
- Open API (anteriormente Swagger)
 - Especificación para describir, producir, consumir y visualizar servicios web RESTful. Es el más ampliamente difundido y cuenta con un ecosistema propio.
- JSON Schema (<https://json-schema.org/>)
 - JSON Schema es una especificación para definir, anotar y validar las estructuras de datos JSON.

© JMA 2020. All rights reserved

Especificaciones mas utilizadas



© JMA 2020. All rights reserved

Swagger

<https://swagger.io/>

- Swagger (OpenAPI Specification) es una especificación abierta y su correspondiente implementación para probar y documentar servicios REST. Uno de los objetivos de Swagger es que podamos actualizar la documentación en el mismo instante en que realizamos los cambios en el servidor.
- Un documento Swagger es el equivalente de API REST de un documento WSDL para un servicio web basado en SOAP.
- El documento Swagger especifica la lista de recursos disponibles en la API REST y las operaciones a las que se puede llamar en estos recursos.
- El documento Swagger especifica también la lista de parámetros de una operación, que incluye el nombre y tipo de los parámetros, si los parámetros son necesarios u opcionales, e información sobre los valores aceptables para estos parámetros.

© JMA 2020. All rights reserved

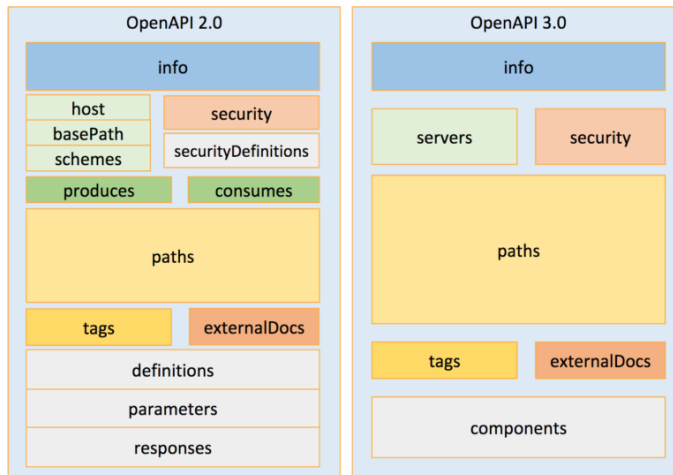
OpenAPI

<https://www.openapis.org/>

- OpenAPI es un estándar para definir contratos de API. Los cuales describen la interfaz de una serie de servicios que vamos a poder consumir por medio de una signature. Conocido previamente como Swagger, ha sido adoptado por la Linux Foundation y obtuvo el apoyo de compañías como Google, Microsoft, IBM, Paypal, etc. para convertirse en un estándar para las APIs REST.
- Las definiciones de OpenAPI se pueden escribir en JSON o YAML. La versión actual de la especificación es la 3.0.3 y orientada a YAML y la versión previa la 2.0, que es idéntica a la especificación 2.0 de Swagger antes de ser renombrada a "Open API Specification".
- Actualmente nos encontramos en periodo de transición de la versión 2 a la 3, sin soporte en muchas herramientas.

© JMA 2020. All rights reserved

Cambio de versión



© JMA 2020. All rights reserved

Sintaxis

- Un documento de OpenAPI que se ajusta a la especificación de OpenAPI es en sí mismo un objeto JSON con propiedades, que puede representarse en formato JSON o YAML.
- YAML es un lenguaje de serialización de datos similar a XML pero que utiliza el sangrado para indicar el anidamiento, estableciendo la estructura jerárquica, y evitar la necesidad de tener que cerrar los elementos.
- Para preservar la capacidad de ida y vuelta entre los formatos YAML y JSON, se RECOMIENDA la versión 1.2 de YAML junto con algunas restricciones adicionales:
 - Las etiquetas DEBEN limitarse a las permitidas por el conjunto de reglas del esquema JSON .
 - Las claves utilizadas en los mapas YAML DEBEN estar limitadas a una cadena escalar, según lo definido por el conjunto de reglas del esquema YAML Failsafe.
- Todos los nombres de propiedades o campos de la especificación distinguen entre mayúsculas y minúsculas. Esto incluye todas las propiedades que se utilizan como claves asociativas, excepto donde se indique explícitamente que las claves no distinguen entre mayúsculas y minúsculas .
- El esquema expone dos tipos de propiedades:
 - propiedades fijas: tienen el nombre establecido en el estándar
 - propiedades con patrón: sus nombres son de creación libre pero deben cumplir una expresión regular (patrón) definida en el estándar y deben ser únicos dentro del objeto contenedor.

© JMA 2020. All rights reserved

Sintaxis

- El sangrado utiliza espacios en blanco, no se permite el uso de caracteres de tabulación.
- Los miembros de las listas van entre corchetes ([]) y separados por coma espacio (,), o uno por línea con un guion (-) inicial.
- Los vectores asociativos se representan usando los dos puntos seguidos por un espacio, "clave: valor", bien uno por línea o entre llaves ({ }) y separados por coma seguida de espacio (,).
- Un valor de un vector asociativo viene precedido por un signo de interrogación (?), lo que permite que se construyan claves complejas sin ambigüedad.
- Los valores sencillos (o escalares) por lo general aparecen sin entrecomillar, pero pueden incluirse entre comillas dobles ("), o apostrofes (').

© JMA 2020. All rights reserved

Sintaxis

- Los comentarios vienen encabezados por la almohadilla (#) y continúan hasta el final de la línea.
- Es sensible a mayúsculas y minúsculas, todas las propiedades (palabras reservadas) de la especificación deben ir en minúsculas y terminar en dos puntos (:).
- Las propiedades requieren líneas independiente, su valor puede ir a continuación en la misma línea (precedido por un espacio) o en múltiples líneas (con sangrado)
- Las descripciones textuales pueden ser multilínea y admiten el dialecto CommonMark de Markdown para una representación de texto enriquecido. El HTML es compatible en la medida en que lo proporciona CommonMark (Bloques HTML en la Especificación 0.27 de CommonMark).
- \$ref permite sustituir, reutilizar y enlazar una definición local con una externa.

© JMA 2020. All rights reserved

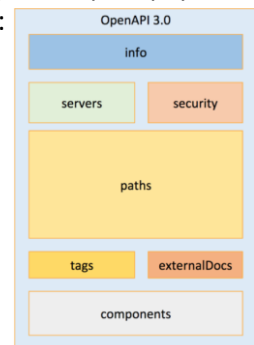
CommonMark de Markdown

- Requiere doble y triple salto de línea para saltos de párrafos y cierre de bloques. Dos espacios al final de la línea lo convierte en salto de línea.
- Regla horizontal (separador): ---
- Énfasis: **cursiva** ****negrita**** ******cursiva y negrita******
- Enlaces: <http://www.example.com> [texto](http://www. example.com)
- Imágenes: ![Image](http://www.example.com/logo.png "icon")
- Citas: > Texto de la cita con sangría
- Bloques de códigos: `Encerrados entre tildes graves`
- Listas: Dos espacios en blanco por nivel de sangrado.
 - + Listas desordenadas 1. Listas ordenadas
- Encabezado: dos líneas debajo del texto, añadir cualquier número de caracteres = para el nivel de título 1, <h1> ... <h6> (el # es interpretado como comentario).

© JMA 2020. All rights reserved

Estructura básica

- Un documento de OpenAPI puede estar compuesto por un solo documento o dividirse en múltiples partes conectadas a discreción del usuario. En el último caso, los campos \$ref deben utilizarse en la especificación para hacer referencia a esas partes.
- Se recomienda que el documento raíz de OpenAPI se llame: openapi.json u openapi.yaml.
- La especificación de la API se puede dividir en 3 secciones principales:
 - Meta información
 - Elementos de ruta (puntos finales):
 - Parámetros de las solicitud
 - Cuerpo de las solicitud
 - Respuestas
 - Componentes reutilizables:
 - Esquemas (modelos de datos)
 - Parámetros
 - Respuestas
 - Otros componentes



© JMA 2020. All rights reserved

Estructura básica

```
openapi: 3.0.0
info:
  title: Sample API
  description: Optional multiline or single-line description in ...
  version: 0.1.9
servers:
- url: http://api.example.com/v1
  description: Optional server description, e.g. Main (production) server
- url: http://staging-api.example.com
  description: Optional server description, e.g. Internal staging server for testing
paths:
  /users:
    get:
      summary: Returns a list of users.
      description: Optional extended description in CommonMark or HTML.
      responses:
        '200':
          # status code
          description: A JSON array of user names
          content:
            application/json:
              schema:
                type: array
                items:
                  type: string
```

© JMA 2020. All rights reserved

Estructura básica (cont)

```
components:
  schemas:
    User:
      properties:
        id:
          type: integer
        name:
          type: string
      # Both properties are required
      required:
        - id
        - name
    securitySchemes:
      BasicAuth:
        type: http
        scheme: basic
  security:
    - BasicAuth: []
```

© JMA 2020. All rights reserved

Prologo

- Cada definición de API debe incluir la versión de la especificación OpenAPI en la que se basa el documento en la propiedad openapi.
- La propiedad info contiene información de la API:
 - title es el nombre de API.
 - description es información extendida sobre la API.
 - version es una cadena arbitraria que especifica la versión de la API (no confundir con la revisión del archivo o la versión del openapi).
 - también admite otras palabras clave para información de contacto (nombre, url, email), licencia (nombre, url), términos de servicio (url) y otros detalles.
- La propiedad servers especifica el servidor API y la URL base. Se pueden definir uno o varios servidores (elementos precedidos por -).
- Con la propiedad externalDocs se puede referenciar la documentación externa adicional.

© JMA 2020. All rights reserved

Rutas

- La sección paths define los puntos finales individuales (rutas) en la API y los métodos (operaciones) HTTP admitidos por estos puntos finales.
- Las ruta es relativa a la ruta del objeto Server.
- Los parámetros de la ruta se pueden usar para aislar un componente específico de los datos con los que el cliente está trabajando. Los parámetros de ruta son parte de la ruta y se expresan entre llaves (/users/{userId}), participan en la jerarquía de la URL y, por lo tanto, se apilan secuencialmente. Los parámetros de ruta deben describirse obligatoriamente en parameters (común para todas las operaciones) o a nivel de operación individual.
- No puede haber dos rutas iguales o ambiguas, que solo se diferencian por el parámetro de ruta.
- La definición de la ruta puede tener con un resumen (summary) y una descripción (description).
- Una ruta debe contar con un conjunto de operaciones, al menos una.
- Opcionalmente, servers permite dar una matriz alternativa de server que den servicio a todas las operaciones en esta ruta.

© JMA 2020. All rights reserved

Rutas

```
'/users/{id}/roles':
  get:
    summary: Returns a list of users's roles.
    operationId: getDirecciones
    parameters:
      - in: path
        name: id
        description: User ID
        required: true
        schema:
          type: number
      - in: query
        name: size
        schema:
          type: string
          enum: [long, medium, short]
        required: true
      - in: query
        name: page
        schema:
          type: integer
          minimum: 0
          default: 0
    responses:
      '200':
        description: List of roles
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/Roles'
      '400':
        description: Bad request. User ID must be an integer and larger than 0.
      '401':
        description: Authorization information is missing or invalid.
      '404':
        description: A user with the specified ID was not found.
      '5XX':
        description: Unexpected error.
    default:
      description: Default error sample response
```

© JMA 2020. All rights reserved

Operaciones

- Describe una única operación de API en una ruta y se identifica con el nombre del método HTTP: get, put, post, delete, options, head, patch, trace.
- Una definición de operación puede incluir un breve resumen de lo que hace (summary), una explicación detallada del comportamiento (description), una referencia a documentación externa adicional (externalDocs), un identificador único para su uso en herramientas y bibliotecas (operationId) y si está obsoleta y debería dejar de usarse (deprecated).
- Las operaciones pueden tener parámetros pasados a través de la ruta URL (/users/{userId}), cadena de consulta (/users?role=admin), encabezados (X-CustomHeader: Value) o cookies (Cookie: debug=0).
- Si la petición (POST, PUT, PATCH) envía un cuerpo en la solicitud (body), la propiedad requestBody permite describir el contenido del cuerpo y el tipo de medio.
- Para cada las respuestas de la operación, se pueden definir los posibles códigos de estado y el schema del cuerpo de respuesta. Los esquemas pueden definirse en línea o referenciarse mediante \$ref. También se pueden proporcionar ejemplos para los diferentes tipos de respuestas.

© JMA 2020. All rights reserved

Parámetros

- Un parámetro único se define mediante una combinación de nombre (name) y ubicación (in: "query", "header", "path" o "cookie") en la propiedad parameters.
- Opcionalmente puede ir acompañado por una breve descripción del parámetro (description), si es obligatorio (required), si permite valores vacíos (allowemptyvalue) y si está obsoleto y debería dejar de usarse (deprecated).
- Las reglas para la serialización del parámetro se especifican dos formas:
 - Para los escenarios más simples, con schema y style se puede describir la estructura y la sintaxis del parámetro.
 - Para escenarios más complejos, la propiedad content puede definir el tipo de medio y el esquema del parámetro.
- Un parámetro debe contener la propiedad schema o content, pero no ambas.
- Se puede proporcionar un example o examples pero debe seguir la estrategia de serialización prescrita para el parámetro.

© JMA 2020. All rights reserved

Parámetros

```
paths:
  /users:
    get:
      description: Returns a list of users
      parameters:
        - name: rows
          in: query
          description: Limits the number of items on a page
          schema:
            type: integer
        - name: page
          in: query
          description: Specifies the page number of the users to be displayed
          schema:
            type: integer
```

© JMA 2020. All rights reserved

Cuerpo de la solicitud

- En versiones anteriores, el cuerpo de la solicitud era un parámetro mas in: body.
- Actualmente se utiliza la propiedad requestBody con una breve descripción (description) y si es obligatorio para la solicitud (required), ambas opcionales.
- La descripción del contenido (content) es obligatoria y se estructura según los tipos de medios que actúan como identificadores. Para las solicitudes que coinciden con varias claves, solo se aplica la clave más específica (text/plain → text/* → */*).
- Por cada tipo de medio se puede definir el esquema del contenido de la solicitud (schema), uno (example) o varios (examples) ejemplos y la codificación (encoding).
- El requestBody sólo se admite en métodos HTTP donde la especificación HTTP 1.1 RFC7231 haya definido explícitamente semántica para cuerpos de solicitud.

© JMA 2020. All rights reserved

Cuerpo de la solicitud

```
paths:
  /users:
    post:
      description: Lets a client post a new user
      requestBody:
        required: true
        content:
          application/json:
            schema:
              type: object
              required:
                - username
            properties:
              username:
                type: string
              password:
                type: string
                format: password
            name:
              type: string
```

© JMA 2020. All rights reserved

Respuestas

- Es obligaría la propiedad `responses` con la lista de posibles respuestas que se devuelven al ejecutar esta operación.
- No se espera necesariamente que la documentación cubra todos los códigos de respuesta HTTP posibles porque es posible que ni se conozcan de antemano. Sin embargo, se espera que cubra la respuesta de la operación cuando tiene éxito y cualquier error previsto.
- Las posibles respuestas se identifican con el código de respuesta HTTP. Con default se puede definir la respuesta por defecto para todos los códigos HTTP que no están cubiertos por la especificación individual.
- La respuesta cuenta con una breve descripción de la respuesta (`description`) y, opcionalmente, el contenido estructurado según los tipos de medios (`content`), los encabezados (`headers`) y los enlaces de operaciones que se pueden seguir desde la respuesta (`links`).

© JMA 2020. All rights reserved

Respuestas

```
paths:
  /users:
    post:
      description: Lets a client post a new user
      requestBody: # ...
      responses:
        '201':
          description: Successfully created a new user
        '400':
          description: Invalid request
          content:
            application/json:
              schema:
                type: object
                properties:
                  code:
                    type: integer
                  message:
                    type: string
```

© JMA 2020. All rights reserved

Etiquetas

- Las etiquetas son metadatos adicionales que permiten organizar la documentación de la especificación de la API y controlar su presentación. Las etiquetas se pueden utilizar para la agrupación lógica de operaciones por recursos o cualquier otro calificador. El orden de las etiquetas se puede utilizar para reflejar un orden en las herramientas de análisis.
- Cada nombre de etiqueta en la lista debe ser único (name) y puede ir acompañado por una explicación detallada (description) y una referencia a documentación externa adicional (externalDocs).
- Las etiquetas se pueden declarar en la propiedad tags del documento:
tags:
 - name: security-resource
 - description: Gestión de la seguridad

© JMA 2020. All rights reserved

Etiquetas

- Las etiquetas se aplican en la propiedad tags de las operaciones:
paths:
 - /users:
 - get:
 - tags:
 - security-resource
 - /roles:
 - get:
 - tags:
 - security-resource
 - read-only-resource
- No es necesario declarar todas las etiquetas, se pueden usar directamente pero no se podrá dar información adicional y se mostraran ordenadas al azar o según la lógica de las herramientas.

© JMA 2020. All rights reserved

Componentes

- La propiedad global `components` permite definir las estructuras de datos comunes utilizadas en la especificación de la API: Contiene un conjunto de objetos reutilizables para diferentes aspectos de la especificación.
- Todos los objetos definidos dentro del objeto de componentes no tendrán ningún efecto en la API a menos que se haga referencia explícitamente a ellos desde propiedades fuera del objeto de componentes.
- La sección `components` dispone de propiedades para `schemas`, `responses`, `parameters`, `examples`, `requestBodies`, `headers`, `securitySchemes`, `links` y `callbacks`.
- Se puede hacer referencia a ellos con `$ref` cuando sea necesario. `$ref` acepta referencias internas con `#` o externas con el nombre de un fichero. La referencia debe incluir la trayectoria para encontrar el elemento referenciado:
 `$ref: '#/components/schemas/Rol'`
 `$ref: 'responses.yaml#/404Error'`
- El uso de referencias permite la reutilización de elementos ya definidos, facilitando la mantenibilidad y disminuyendo sensiblemente la longitud de la especificación, por lo que se deben utilizar extensivamente. Las referencias no interfieren con la presentación en el UI.

© JMA 2020. All rights reserved

Esquemas de datos

- Los `schemas` definen los modelos de datos consumidos y devueltos por la API.
- Los tipos de datos OpenAPI se basan en un subconjunto extendido del JSON Schema Specification Wright Draft 00 (también conocido como Draft 5).
- Los tipos base son `string`, `number`, `integer`, `boolean`, `array` y `object`.
- Con la propiedad `format` se pueden especificar otros tipos especiales partiendo de los tipos base: `long`, `float`, `double`, `byte`, `binary`, `date`, `dateTime`, `password`.
- Los tipos `array` se definen como una colección de ítems y en dicha propiedad se define el tipo y la estructura de los elementos que lo componen. Los objetos son un conjunto de propiedades, cada una definida dentro de `properties`.
- Cada tipo y propiedad se identifica por un nombre que no debe estar repetido en su ámbito.
- Cada propiedad puede definir `description`, `default`, `minimum`, `maximum`, `maxLength`, `minLength`, `pattern`, `required`, `readOnly`, ...
- Para una propiedad se pueden definir varios tipos (tipos mixtos o unión).
- Los tipos pueden hacer referencia a otros tipos.

© JMA 2020. All rights reserved

Tipos de datos

type	format	Comentarios
boolean		Booleanos: true y false
integer	int32	Enteros con signo de 32 bits
integer	int64	Enteros con signo de 64 bits (también conocidos como largos)
number	float	Reales cortos
number	double	Reales largos
string		Cadenas de caracteres
string	password	Una pista a las IU para ocultar la entrada.
string	date	Según lo definido por full-date RFC3339 (2018-11-13)
string	date-time	Según lo definido por date-time- RFC3339 (2018-11-13T20:20:39+00:00)
string	byte	Binario codificados en base64
string	binary	Binario en cualquier secuencia de octetos
array		Colección de items
object		Colección de properties

© JMA 2020. All rights reserved

Propiedades de los objetos de esquema

- type: integer, number, boolean, string, array, object
- format: long, float, double, byte, binary, date, dateTime, password
- title: Nombre a mostrar en el UI
- description: Descripción de su uso
- maximum: Valor máximo
- exclusiveMaximum: Valor menor que
- minimum: Valor mínimo
- exclusiveMinimum: Valor mayor que
- multipleOf: Valor múltiplo de
- maxLength: Longitud máxima
- minLength: Longitud mínima
- pattern: Expresión regular del patrón
- deprecated: Si está obsoleto y debería dejar de usarse
- nullable: Si acepta nulos
- default: Valor por defecto
- enum: Lista de valores con nombre
- example: Ejemplo de uso
- externalDocs: referencia a documentación externa adicional
- items: Definición de los elementos del array
- maxItems: Número máximo de elementos
- minItems: Número mínimo de elementos
- uniqueItems: Elementos únicos
- properties: Definición de las propiedades del objeto,
- maxProperties: Número máximo de propiedades
- minProperties: Número mínimo de propiedades
- readOnly: propiedad de solo lectura
- writeOnly: propiedad de solo escritura
- additionalProperties: permite referenciar propiedades adicionales
- required: Lista de propiedades obligatorias

© JMA 2020. All rights reserved

Modelos de entrada y salida

```
components:
  schemas:
    Roles:
      type: array
      items:
        $ref: '#/components/schemas/Rol'
    Rol:
      type: object
      description: Roles de usuario
      properties:
        rolId:
          type: integer
          format: int32
          minimum: 0
          maximum: 255
        name:
          type: string
          maxLength: 20
        description:
          type: string

last_updated:
  type: string
  format: dateTime
  readOnly: true
level:
  type: string
  description: Nivel de permisos
  enum:
    - high
    - normal
    - low
  default: normal
required:
  - rolId
  - name
```

© JMA 2020. All rights reserved

Autenticación

- La propiedad `securitySchemes` de `components` y la propiedad `security` del documento se utilizan para describir y establecer los métodos de autenticación utilizados en la API.
- `securitySchemes` define los esquemas de seguridad que pueden utilizar las operaciones. Los esquemas admitidos son la autenticación HTTP, una clave API (ya sea como encabezado, parámetro de cookie o parámetro de consulta), los flujos comunes de OAuth2 (implícito, contraseña, credenciales de cliente y código de autorización) tal y como se define en RFC6749 y OpenID Connect Discovery. Cada esquema cuenta con un identificador, un tipo (`type: "apiKey", "http", "oauth2", "openIdConnect"`) y opcionalmente puede ir acompañado por una breve descripción (`description`).
- Según el tipo seleccionado será obligatorio:
 - `apiKey`: ubicación (`in: "query", "header" o "cookie"`) y su nombre (`name`) de parámetro, encabezado o cookie.
 - `http`: esquema de autorización HTTP que se utilizará en el encabezado `Authorization` (`scheme`): `Basic`, `Bearer`, `Digest`, `OAuth`, ... y, si es `Bearer`, prefijo del token de portador (`bearerFormat`).
 - `openIdConnect`: URL de OpenID Connect para descubrir los valores de configuración de OAuth2 (`openIdConnectUrl`).
 - `oauth2`: objeto que contiene información de configuración para los tipos de flujo admitidos (`flows`).
- La propiedad `security` enumera los esquemas de seguridad que se pueden utilizar en la API.

© JMA 2020. All rights reserved

Autenticación

```
components:
  securitySchemes:
    BasicAuth:
      type: http
      scheme: basic
    JWTAuth:
      type: http
      scheme: bearer
      bearerFormat: JWT
    ApiKeyAuth:
      type: apiKey
      name: x-api-key
      in: header
    ApiKeyQuery:
      type: apiKey
      name: api-key
      in: query
  security:
    - ApiKeyAuth: []
    - ApiKeyQuery: []
```

© JMA 2020. All rights reserved

Ejemplos

- Los ejemplos son fundamentales para la correcta comprensión de la documentación. La especificación permite proporcionar uno (example) o varios (examples) ejemplos asociados a las estructuras de datos.
- Por cada uno se puede dar un resumen del ejemplo (summary), una descripción larga (description), el juego de valores de las propiedades de la estructura (value) o una URL que apunta al ejemplo literal para ejemplos que no se pueden incluir fácilmente en documentos JSON o YAML (externalValue). value y externalValue son mutuamente excluyentes. Cuando son varios ejemplos deber estar identificados por un nombre único.

```
examples:
  first-page:
    summary: Primera página
    value: 0
  second-page:
    summary: Segunda página
    value: 1
```

- Los ejemplos pueden ser utilizados automáticamente por las herramientas de UI y de generación de pruebas.

© JMA 2020. All rights reserved

Ecosistema Swagger

- **Swagger Open Source Tools**
 - Swagger Editor: Diseñar APIs en un potente editor de OpenAPI que visualiza la definición y proporciona comentarios de errores en tiempo real.
 - Swagger Codegen: Crear y habilitar el consumo de su API generando la fontanería del servidor y el cliente.
 - Swagger UI: Generar automáticamente la documentación desde la definición de OpenAPI para la interacción visual y un consumo más fácil.
- **Swagger Pro Tools**
 - SwaggerHub: La plataforma de diseño y documentación para equipos e individuos que trabajan con la especificación OpenAPI.
 - Swagger Inspector: La plataforma de pruebas y generación de documentación de las APIs
- <https://openapi.tools/>

© JMA 2020. All rights reserved

Documentación Postman

- La función de documentación de API de Postman permite ver la documentación privada del API o compartir documentación pública del API en una página web.
- Postman genera y aloja automáticamente en tiempo real la documentación basada en navegador del API para las colecciones. Cada colección tiene una vista de documentación pública y privada que Postman genera a partir de datos sincronizados en los servidores (es necesario tener un usuario registrado).
- La documentación para el API incluye:
 - Solicitudes de muestra, encabezados y otros metadatos
 - Descripciones asociadas con solicitudes, carpetas y colecciones.
 - Fragmentos de código generados en algunos de los lenguajes de programación más populares.
- Postman utiliza la ordenación de las solicitudes y carpetas para organizar la documentación en secciones y reflejar la estructura de la colección.
- Se pueden personalizar las descripciones utilizando el estilo Markdown con gráficos incrustados para complementar su documentación.

© JMA 2020. All rights reserved

Desarrollo de API

- Postman admite el desarrollo de API First con API Builder. Utilice API Builder para diseñar su API en Postman. Su definición de API puede actuar como la única fuente de verdad para su proyecto de API.
- Se pueden conectar varios elementos de su proceso de desarrollo y prueba de API a su definición de API, como colecciones, documentación, pruebas y monitores.
- Postman le permite usar el control de versiones basado en Git para desarrollar y administrar cambios en su API. Puede conectar su API a un repositorio remoto de GitHub, Bitbucket, GitLab o Azure DevOps.
- Para ver y trabajar con API, se selecciona API en la barra lateral. Puede abrir y editar cualquier API existente desde aquí, o crear o importar nuevas API.
- Postman admite definiciones OpenAPI (versiones 1.0, 2.0, 3.0 y 3.1), RAML (0.8 y 1.0), protobuf (búfer de protocolo) (2.0 y 3.0), GraphQL o WSDL (1.0 y 2.0). Las definiciones de OpenAPI pueden ser JSON o YAML. Las definiciones RAML deben ser YAML. Las definiciones de Protobuf son .protoarchivos. Las definiciones de GraphQL pueden ser JSON o GraphQL SDL. Las definiciones WSDL deben ser XML.

© JMA 2020. All rights reserved

Servidores simulados

- Los retrasos en el front-end o back-end dificultan que los equipos dependientes completen su trabajo de manera eficiente. Los servidores simulados de Postman pueden aliviar esos retrasos en el proceso de desarrollo.
- Antes de enviar una solicitud real, los desarrolladores front-end pueden crear un mock server para simular cada punto final y su respuesta correspondiente en una Colección Postman. Los desarrolladores pueden ver las posibles respuestas, sin tener que acceder al back-end.
- Postman le permite crear dos tipos de servidores simulados: privados (hay que pasar una clave API en el encabezado x-api-key de la solicitud) y públicos.
 - `https://{{mockId}}.mock.pstmn.io/{{mockPath}}`
- Los servidores simulados residen en la infraestructura de Postman por lo que es necesario tener un usuario registrado (con límites en el número de solicitudes).
- Para crear un mock server se puede tomar como partida los ejemplos de una colección existente o crear manualmente las solicitudes y respuestas (genera una colección asociada al mock server).
- Una vez creados los mock server no se pueden modificar, pero se pueden crear tantos como diferentes escenarios sean necesarios y asociarlos a la misma colección.

© JMA 2020. All rights reserved