2

Design and Modeling of RESTful Web Services

Java Backend Developer I

Objetivos

- Conocer el protocolo HTTP y sus versiones.
- Puntos claves para identificar los recurso de un servicio: Acerca de REST y restricciones
- Proceso de diseño de un servicio RESTful sobre HTTP.
- Analizar un caso de estudio (laboratorio)

Agenda

- Revisar el protocolo HTTP y sus versiones.
- Revisar los Puntos clave para identificar los recurso de un servicio: Overview REST, verificar sus restricciones
- Conocer los 10 pasos para el diseño de un servicio web RESTful sobre HTTP. (laboratorio)
- Analizar un caso de estudio (laboratorio)

Protocolo HTTP

- HTTP es el protocolo en el que se basa la Web. Fue inventado por Tim Berners-Lee entre los años 1989-1991, HTTP ha evolucionado, al inicio era un protocolo destinado al intercambio de archivos en un entorno de un laboratorio semi-seguro, y actualmente sobre internet, sirve para el intercambio de imágenes, vídeos en alta resolución y en 3D.
- Historia: En 1989, Tim Berners-Lee escribió una propuesta para desarrollar un sistema de hipertexto sobre Internet. Inicialmente lo llamó: '*Mesh'* (malla, en inglés), y posteriormente se renombró como *World Wide Web* (red mundial), durante su implementación en 1990.

Protocolo HTTP

- Desarrollado sobre los protocolos existentes TCP e IP, está basado en cuatro bloques:
  - Un formato de texto para representar documentos de hiper-texto: HyperText Markup Language (HTML).
  - Un protocolo sencillo para el intercambio de esos documentos, del inglés: HypertText Transfer Protocol (HTTP) : protocolo de transferencia de hiper-texto.
  - Un cliente que muestre (e incluso pueda editar) esos documentos. El primer navegador Web, llamado: WorldWideWeb.
  - Un servidor para dar acceso a los documentos, una versión temprana: httpd (http daemon)

Protocolo HTTP versiones

- HTTP/0.9 – El protocolo de una sola línea
- HTTP/1.0 – Desarrollando expansibilidad
- HTTP/1.1 – El protocolo estándar
- HTTP/2 – Un protocolo para un mayor rendimiento
- Post-evolución del HTTP/2

Protocolo HTTP versiones

- HTTP/0.9 – El protocolo de una sola línea, no usa cabecera HTTP. Solo se transmitía HTML y no otro tipo de archivo. No usa estado.
- HTTP/1.0 – Desarrollando expansibilidad, se envía el estado y cabeceras con ello se puede transmitir otro tipo de archivos. (cabecera Content-Type)
- HTTP/1.1 – El protocolo estándar, durante 15 años de constante evolución. Transmitía sobre SSL, luego TLS. Permitía las transmisiones seguras. En el año 2000 un nuevo formato para usar HTTP fue diseñado: REST (Transferencia de Estado Representacional)
- HTTP/2 – Un protocolo para un mayor rendimiento. Es un protocolo binario en respuesta al protocolo de cadena de texto en versiones anteriores. Permite peticiones paralelas.
- Post-evolución del HTTP/2, utilizar prefijo de seguridad en la cabecera cookie garantizando que no haya sido alterada. Adicionar cabecera Client-Hints comunicación proactiva con el servidor.

Acerca de REST

- El término "Transferencia de Estado Representacional" (REST) representa un conjunto de características de diseño de arquitecturas software que aportan confiabilidad, eficiencia y escalibilidad a los sistemas distribuidos. Un sistema es llamado RESTful cuando se ajusta a estas características.
- La idea básica de REST es que permite que un recurso, por ejemplo un documento, sea transferido con su estado y su relaciones (hipertexto) mediante formatos y operaciones estandarizadas bien definidas.

Acerca de REST

- Como HTTP, el protocolo estándar de la Web, también transfiere documentos e hipertexto, las APIs HTTP a veces son llamadas APIs RESTful, servicios RESTful, o simplemente servicios REST, aunque no se ajusten del todo a la definición de REST.
- Los que recién inician pueden pensar que una API REST es un servicio HTTP que puede ser llamado mediante librerías y herramientas web estándar.

Restricciones REST

- Podemos identificar las propiedades inducidas por las restricciones de la web al examinar el impacto de cada restricción.
- Se pueden aplicar restricciones adicionales para formar un nuevo estilo arquitectónico que refleje mejor las propiedades deseadas de una arquitectura web moderna.
- En esta sección se describirán con más detalle las restricciones específicas que componen el estilo REST.

Restricciones REST
Perspectivas

- Hay dos perspectivas comunes sobre el proceso de diseño arquitectónico para software.
- La primera es que un diseñador comienza sin nada y construye una arquitectura a partir de componentes familiares hasta que satisfaga las necesidades del sistema previsto.
- El segundo es que un diseñador comienza con las necesidades del sistema como un todo, sin restricciones, y luego identifica y aplica restricciones de forma incremental a los elementos del sistema.

Restricciones REST
Perspectiva: Estilo Nulo

- ***Comenzando con el estilo nulo***
- El estilo nulo (Figura 5-1) es simplemente un conjunto vacío de restricciones. Desde una perspectiva arquitectónica, el estilo nulo describe un sistema en el que no hay límites distinguidos entre los componentes. Es el punto de partida para nuestra descripción de REST.

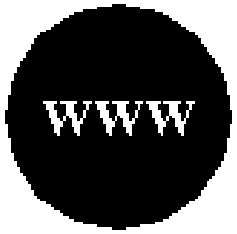


Figure 5-1. Null Style

Fuente: https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm

Restricciones REST
Perspectiva: Cliente Servidor

- ***Cliente servidor***
- Las primeras restricciones agregadas a nuestro estilo híbrido son las del estilo arquitectónico cliente-servidor (Figura 5-2). La separación de es el principio detrás de las restricciones cliente-servidor. Al separar la interfaz de usuario de lo concerniente al almacenamiento de datos, mejoramos la portabilidad.
- Sin embargo, quizás lo más importante para la Web es que la separación permite que los componentes evolucionen de forma independiente, lo que respalda el requisito de escala de Internet de múltiples dominios organizacionales.

Figure 5-2. Client-Server

Fuente: https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm

Restricciones REST
Perspectiva: Sin Estado

- ***Sin Estado***
- Agregamos una restricción a la interacción cliente-servidor: la comunicación *debe ser de naturaleza sin estado*(Figura 5-3), de modo que cada solicitud del cliente al servidor debe contener toda la información necesaria para comprender la solicitud. Por lo tanto, el estado de la sesión se mantiene completamente en el cliente.
- Esta restricción induce las propiedades de visibilidad, confiabilidad y escalabilidad.
- Se mejora la visibilidad porque un sistema de monitoreo no tiene que mirar más allá de un solo dato de solicitud para determinar la naturaleza completa de la solicitud.
- Se mejora la confiabilidad porque facilita la tarea de recuperación de fallas parciales.
- Se mejora la escalabilidad porque al no tener que almacenar el estado entre las solicitudes permitirá que el componente del servidor libere recursos rápidamente.

Restricciones REST
Perspectiva: Sin Estado

- ***Sin Estado***
- Como la mayoría de las opciones arquitectónicas, la restricción sin estado refleja una compensación de diseño.
- Colocar el estado de la aplicación en el lado del cliente reduce el control del servidor sobre el comportamiento consistente de la aplicación, ya que la aplicación se vuelve dependiente de la implementación correcta de la semántica en múltiples versiones del cliente.

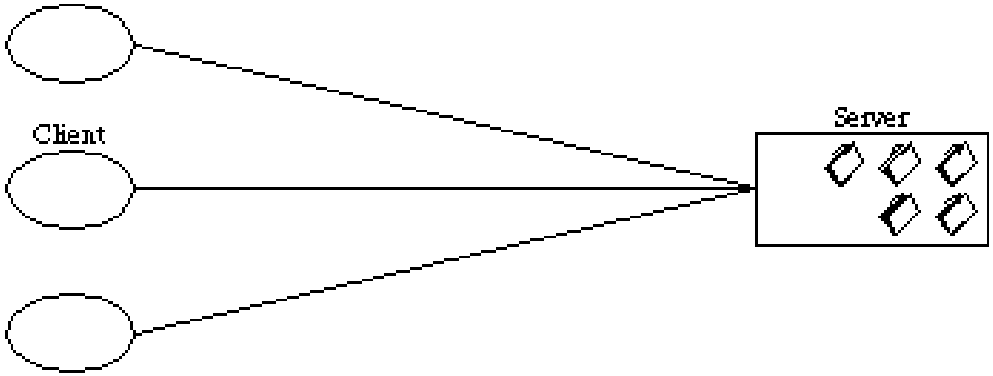


Figure 5-3. Client-Stateless-Server

Fuente: https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm

Restricciones REST
Perspectiva: Cache

- ***Cache***
- Para mejorar la eficiencia de la red, agregamos restricciones de caché para formar el estilo cliente-caché-servidor sin estado (Figura 5-4). Las restricciones de caché requieren que los datos dentro de una respuesta a una solicitud se etiqueten implícita o explícitamente como almacenables en caché o no almacenables en caché. Si una respuesta es almacenable en caché, entonces el caché de un cliente tiene derecho a reutilizar esos datos de respuesta para solicitudes equivalentes posteriores.

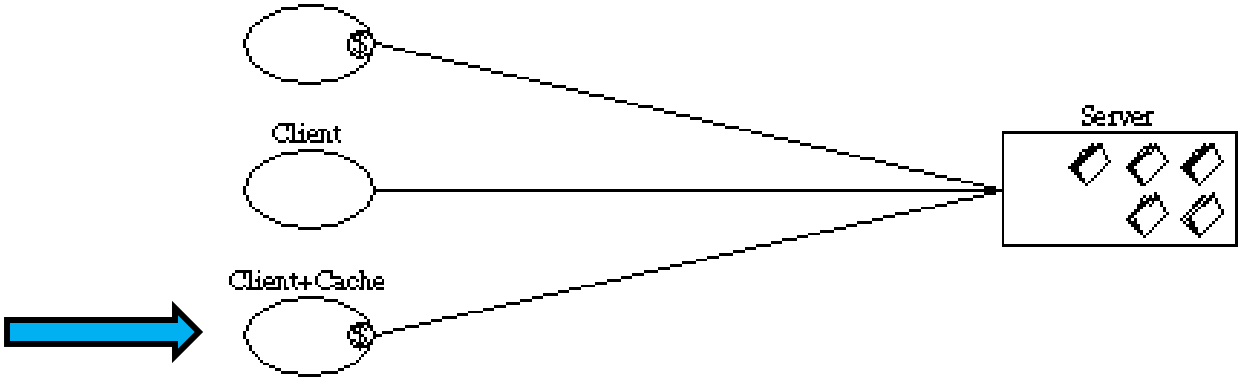


Figure 5-4. Client-Cache-Stateless-Server

Fuente: https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm

Restricciones REST
Perspectiva: Uniform Interface

- ***Uniform Interface***
- La característica central que distingue el estilo arquitectónico REST de otros estilos basados en red es su énfasis en una interfaz uniforme entre componentes (Figura 5-6). Ello implica que las implementaciones están desconectadas de los servicios que brindan, lo que fomenta la capacidad de evolución independiente.
- Sin embargo, la desventaja es que una interfaz uniforme degrada la eficiencia, ya que la información se transfiere de forma estandarizada en lugar de una que sea específica para las necesidades de una aplicación.

Restricciones REST
Perspectiva: Uniform Interface

- ***Uniform Interface*** La interfaz REST está diseñada para ser eficiente para la transferencia de datos hipermedia de gran tamaño, optimizando para el caso común de la Web, pero resultando en una interfaz que no es óptima para otras formas de interacción arquitectónica.

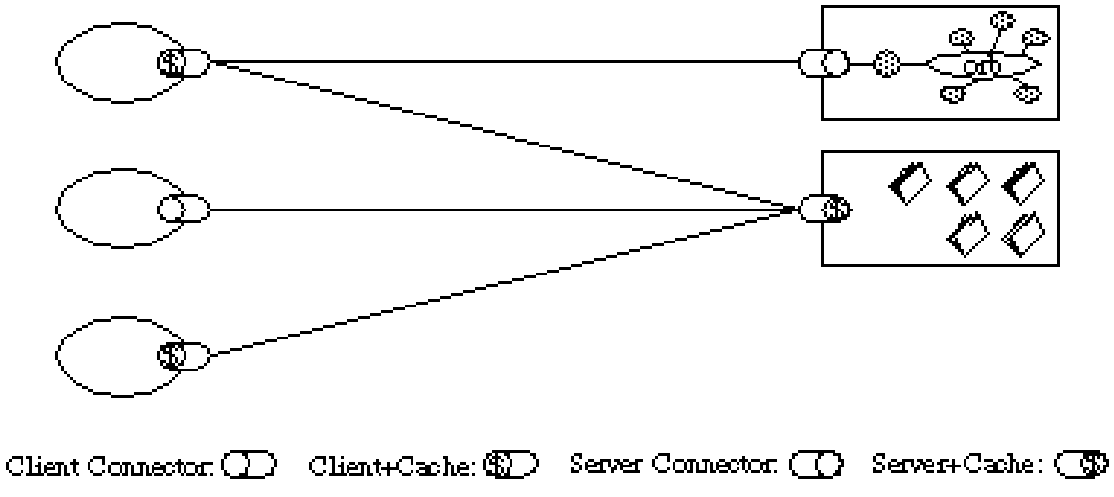


Figure 5-6. Uniform-Client-Cache-Stateless-Server

Fuente: https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm

Restricciones REST
Perspectiva: Sistema de capas

- ***Sistema de capas***
- Para mejorar aún más el comportamiento de los requisitos a escala de Internet, agregamos restricciones de sistema en capas (Figura 5-7). El estilo de sistema en capas permite que una arquitectura esté compuesta de capas jerárquicas al restringir el comportamiento de los componentes de manera que cada componente no pueda "ver" más allá de la capa inmediata con la que están interactuando.
- Las capas se pueden usar para encapsular servicios heredados y para proteger nuevos servicios de clientes heredados, simplificando los componentes al trasladar la funcionalidad de uso poco frecuente a un intermediario compartido. Los intermediarios también se pueden utilizar para mejorar la escalabilidad del sistema al permitir el equilibrio de carga de los servicios en múltiples redes y procesadores.

Restricciones REST
Perspectiva: Sistema de capas

- ***Sistema de capas***

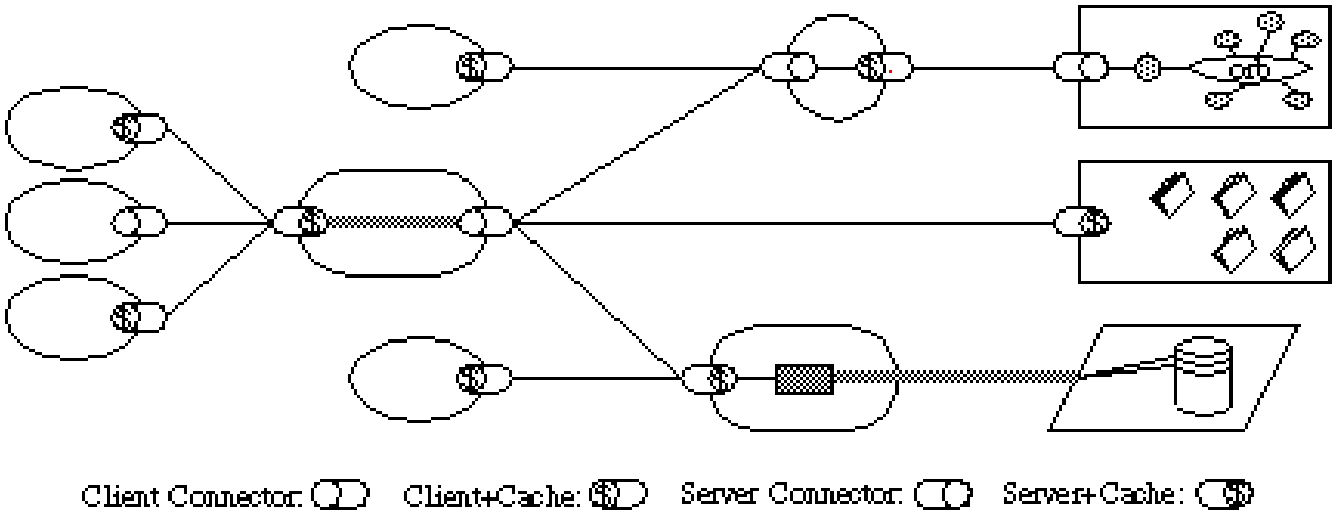


Figure 5-7. Uniform-Layered-Client-Cache-Stateless-Server

Fuente: https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm

Restricciones REST
Perspectiva: Código Bajo Demanda

- ***Código bajo demanda***
- Tenemos el estilo de código bajo demanda (Figura 5-8). REST que permite ampliar la funcionalidad del cliente descargando y ejecutando código en forma de scripts. Esto simplifica a los clientes al reducir la cantidad de características que se deben implementar previamente. Permitir que las características se descarguen después de la implementación mejora la extensibilidad del sistema. Sin embargo, también reduce la visibilidad y, por lo tanto, es solo una restricción opcional dentro de REST.
- Una restricción opcional nos permite diseñar una arquitectura que admita el comportamiento deseado en el caso general, pero el cual puede deshabilitarse en algunos contextos.

Restricciones REST
Perspectiva: Código Bajo Demanda

- ***Código bajo demanda***

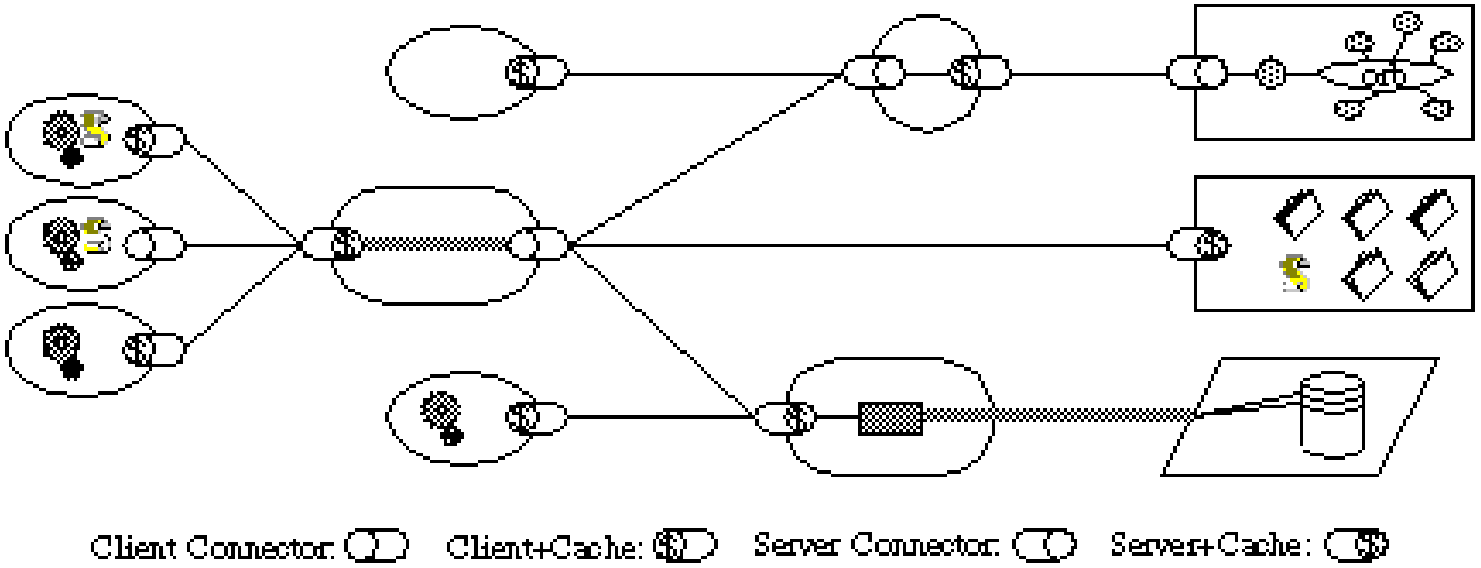


Figure 5-8. REST

Fuente: https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm

Restricciones REST
Resumen

- ***Resumen de derivación de estilo***
- REST consiste en un conjunto de restricciones arquitectónicas. Aunque cada una de estas restricciones puede considerarse de forma aislada, describirlas en términos de su derivación de estilos arquitectónicos comunes hace que sea más fácil entender la razón detrás de su selección. La Figura 5-9 representa la derivación de las restricciones de REST gráficamente en términos de los estilos arquitectónicos basados en red.

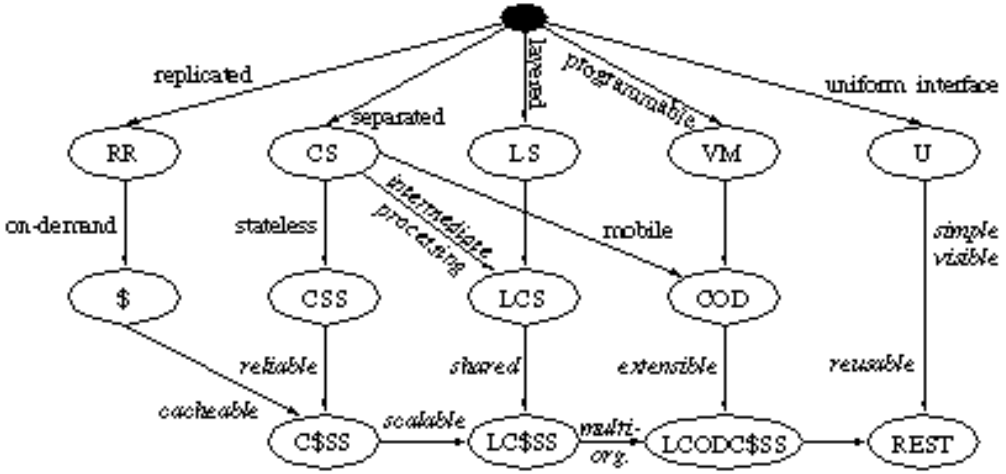


Figure 5-9. REST Derivation by Style Constraints

Fuente: https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm

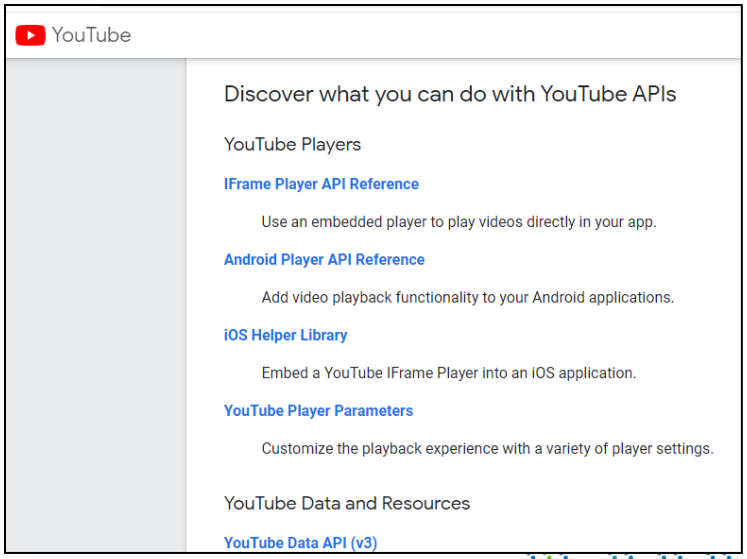
Transferencia de Estado Representacional REST

- ***Importancia***
- REST cambió por completo la ingeniería de software a partir del 2000. Este nuevo enfoque de desarrollo de proyectos y servicios web fue definido por Roy Fielding, el padre de la especificación HTTP y uno los referentes internacionales en todo lo relacionado con la Arquitectura de Redes, en su disertación 'Estilos arquitectónicos y el diseño de arquitecturas de software basadas en red'. En el campo de las APIs, REST (Representational State Transfer- Transferencia de Estado Representacional) es, al día de hoy, lo mas utilizado en el desarrollo de servicios de aplicaciones.
- En la actualidad no existe proyecto o aplicación que no disponga de una API REST para la creación de servicios profesionales a partir de ese software. Twitter, YouTube, los sistemas de identificación con Facebook… hay cientos de empresas que generan negocio gracias a REST y las APIs REST. Sin ellas, todo el crecimiento en horizontal sería prácticamente imposible. Esto es así porque REST es el estándar más lógico, eficiente y habitual en la creación de APIs para servicios de Internet.

Servicios API REST

- ***Algunos servicios:***
- ***Google API: API adMob***

- ***Youtube API: APIs***



Transferencia de Estado Representacional REST

- ***Definición***
- REST es un estilo no es un protocolo entre sistemas que use HTTP para obtener datos o generar operaciones sobre esos datos en todos los formatos posibles, como XML y JSON. Es una alternativa en auge a otros protocolos estándar de intercambio de datos como SOAP (Simple Object Access Protocol), que disponen de una gran capacidad pero también mucha complejidad. A veces es preferible una solución más sencilla de manipulación de datos como REST.
- ***Características***
- Protocolo Cliente/Servidor sin estado.
- Tiene operaciones: POST (crear), GET (leer y consultar), PUT (editar) y DELETE (eliminar), INFO, OPTION y otras operaciones más.
- Los objetos en REST se manipulan a través de la URI.
- Interfaz Uniforme, Sistema de Capas y uso de hipermedios.

Proceso para diseñar un servicio Web API RESTful sobre HTTP

- Este diseño significa que una API sigue el estilo arquitectónico de transferencia de estado representacional (REST) y los siguientes pasos:
- Paso 01: Identificación de sustantivos y verbos
- Paso 02: Extracción de URL y sus métodos de sustantivos y verbos
- Paso 03: Definir condiciones de éxito
- Paso 04: Definir condiciones de error
- Paso 05: Definir la tolerancia a fallas
- Paso 06: Autenticación
- Paso 07: Almacenamiento en caché
- Paso 08: Bloqueo optimista
- Paso 09: Tipos de medios
- Paso 10: Seleccionar el estilo de la Arquitectura.

Proceso para diseñar un servicio Web API RESTful sobre HTTP

- ***Paso 01: identificación de sustantivos y verbos***

  Hemos visto que REST está hecho de recursos, representaciones y acciones. También hemos visto que los recursos son como sustantivos y las acciones son como verbos. Por lo tanto, identificar los sustantivos y verbos disponibles en una especificación del sistema es una buena forma de empezar nuestra especificación en el proceso API.

Proceso para diseñar un servicio Web API RESTful sobre HTTP

- ***Paso 02: Extracción de URL y sus métodos de sustantivos y verbos***

  Algunos verbos se asignan fácilmente a un método HTTP, porque a veces la semántica coincide. Otros verbos necesitarán ser nominados para convertirse en recursos.

```
Request: GET /stations/
Request body: (empty)
Response body:
[{
    "id": 1,
    "name": "John St",
    "location": [40.7, -74]
},
{
    "id": 2,
    "name": "Brooklyn Bridge",
    "location": [40.7, -73]
}]
```

Proceso para diseñar un servicio Web API RESTful sobre HTTP

- ***Paso 03: Definir condiciones de éxito***

  Como ya definimos las URL y los métodos para nuestra API, ahora podemos analizar qué significa el éxito para cada combinación de URL / método. Es importante definir adecuadamente las condiciones de éxito para permitir que los clientes API comuniquen el éxito en sus interfaces a sus usuarios finales.

```
HTTP/1.1 200 OK
Connection: keep-alive
Date: Sat, 15 Jul 2017 13:31:25 GMT
Content-Type: application/json
Content-Length: 29

{
  "text": "Hello world!"
}
```

Proceso para diseñar un servicio Web API RESTful sobre HTTP

- ***Paso 04: Definir condiciones de error***

  ¿En qué condiciones pueden fallar nuestras operaciones API? Los códigos de estado aquí siempre serán 4xx, porque estos son errores que podemos anticipar. Como hemos visto en la descripción de la clase 4xx, los errores anticipados incluyen entradas incorrectas, acceso a recursos inexistentes, estado no sincronizado, usuarios no autenticados, etc.

  Esos son problemas genéricos que pueden ocurrir en cualquier API. Pasemos por cada combinación de URL / método para analizar las posibles fallas que pueden suceder.

- ***HTTP código de status***

  - 1xx Informational responses
  - 2xx Success - 200, 201, 204
  - 3xx Redirection - 301, 304
  - 4xx Client errors - 400, 401, 403, 404
  - 5xx Server errors - 500

Proceso para diseñar un servicio Web API RESTful sobre HTTP

- ***Paso 05: Definir la tolerancia a fallas***

  ¿Qué debe hacer un cliente cuando nunca recibe una respuesta? En ese caso, el cliente no conoce el estado actual del servidor, entonces, ¿está bien volver a intentarlo? La respuesta a estas preguntas se basa en dos conceptos bien definidos: ***seguridad*** e ***idempotencia***.

  Seguridad significa que una solicitud dada no producirá ningún cambio en el estado del servidor. GET

  Idempotencia significa que el estado del recurso no cambia después de Solicitudes posteriores. DELETE, GET

  ¿El método POST / PATCH es seguridad e idempotencia?

Proceso para diseñar un servicio Web API RESTful sobre HTTP

- ***Paso 06: Autenticación***

  La pregunta principal a responder sobre la autenticación en nuestra API es: ¿cómo? Hay muchas formas de autenticar a un usuario en una API HTTP RESTful.

  Autenticación básica, autenticación de token, JSON Web Token (JWT) y Oauth.

Proceso para diseñar un servicio Web API RESTful sobre HTTP

- ***Paso 07: Almacenamiento en caché***

  El almacenamiento en caché significa que no todas las solicitudes deben ser procesadas nuevamente por el servidor.

  Si nada cambió en el recurso, está bien decirle al cliente que use la copia más reciente que almacenó en caché. Dado que lo que es seguro y lo que no se define a nivel de protocolo, el almacenamiento en caché es muy genérico y portátil en HTTP. Debido a que los métodos seguros no producen cambios en el estado del servidor, podemos agregar libremente (pero con cuidado) múltiples capas de almacenamiento en caché a nuestra aplicación.

Proceso para diseñar un servicio Web API RESTful sobre HTTP

- ***Paso 08: Bloqueo optimista***
- Imagine a dos personas compartiendo una sola cuenta de la aplicación de alquiler de bicicletas. Imagine que ambos quieren cambiar el destino de un alquiler. Abren la aplicación en la página de alquiler y seleccionan diferentes ubicaciones nuevas. Ahora imagine que uno de los usuarios hace clic en el botón "actualizar". Antes de que haya pasado suficiente tiempo para que este usuario reciba la respuesta, el segundo usuario también hace clic en "actualizar". ¿Lo que pasa? Bueno, si nuestra aplicación no está lista para lidiar con ese tipo de situación, ambos pensarán que el destino se establecerá en la nueva ubicación que seleccionaron, pero solo la última será la correcta. El problema aquí es que el segundo usuario en presionar "actualizar" está haciendo un cambio basado en una versión del recurso que ya no es válida.

Proceso para diseñar un servicio Web API RESTful sobre HTTP

- ***Paso 09: Tipos de medios (media)***
- Por ejemplo, la aplicación para compartir bicicletas tiene un recurso de estaciones. Hay muchos tipos de medios razonables que podemos usar para representar una lista de estaciones. Hemos utilizado JSON, XML. Existe también Protocol Buffer.

JSon

```
[
  {
   "id": 1,
   "name": "John St",
   "location": [40.7, -74]
  },
  {
   "id": 2,
   "name": "Brooklyn Bridge",
   "location": [40.7, -73]
  }
]
```

But nothing prevents us from using another Media Type, such as XML:

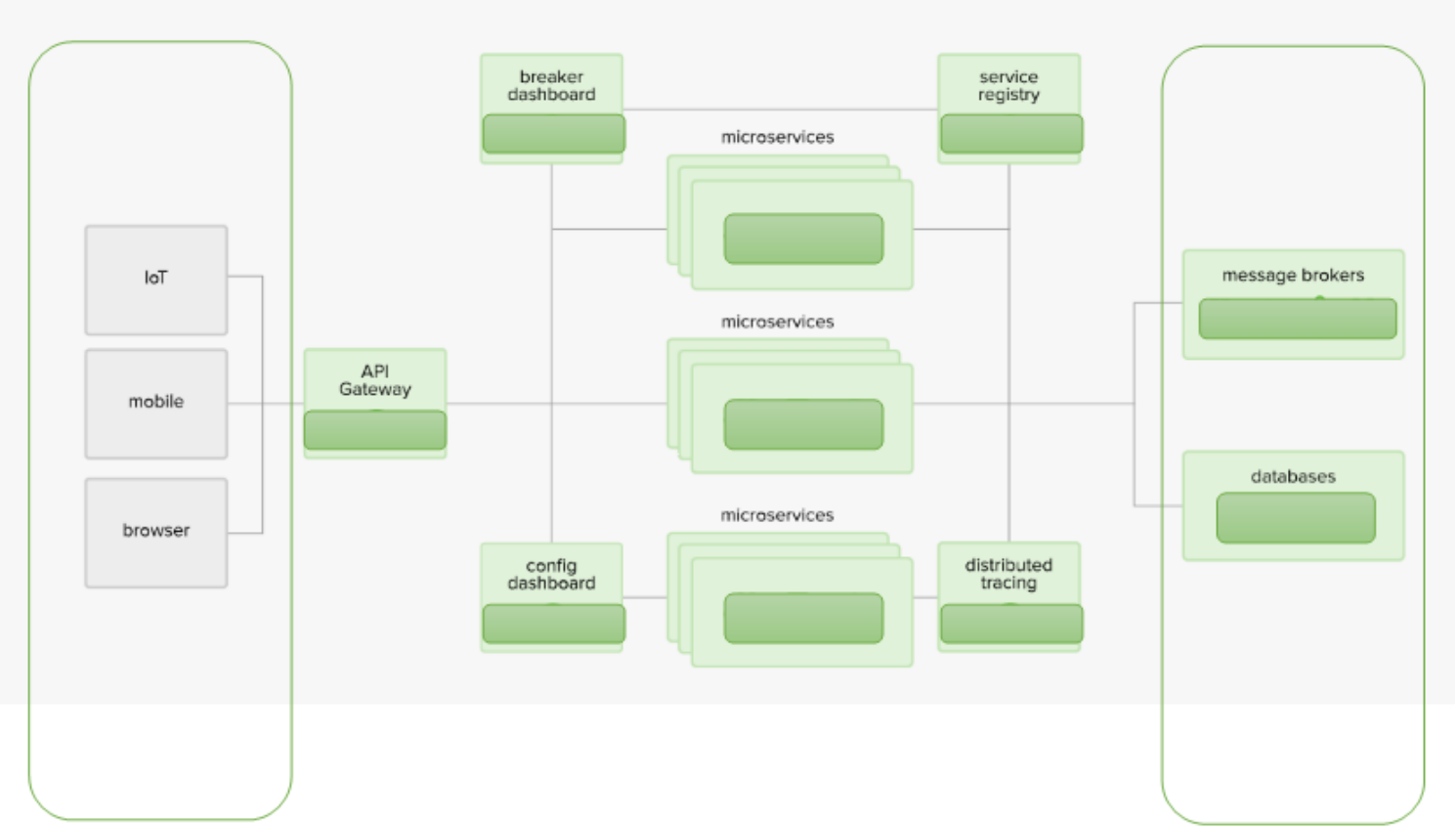
XML

```
<stations>
  <station id="1" name="John St">
    <location>
      <latitude>40.7</latitude>
      <longitude>-74</longitude>
    </location>
  </station>
  <station id="2" name="Brooklyn Bridge">
    <location>
      <latitude>40.7</latitude>
      <longitude>-73</longitude>
    </location>
  </station>
</stations>
```

Proceso para diseñar un servicio Web API RESTful sobre HTTP

- ***<u>Paso 10: Seleccionar el estilo de la Arquitectura</u>***

  Por Ejm: Microservicio.



Transferencia de Estado Representacional REST

- ***Ventajas***
- *Separación entre el cliente y el servidor*. El estilo REST separa totalmente la interfaz de usuario del servidor y el almacenamiento de datos
- *Visibilidad, fiabilidad y escalabilidad.* La separación entre cliente y servidor tiene una ventaja y es que cualquier equipo de desarrollo puede escalar el producto sin excesivos problemas. Se puede migrar a otros servidores o realizar todo tipo de cambios en la base de datos, siempre y cuando los datos de cada una de las peticiones se envíen de forma correcta
- *REST siempre es independiente del tipo de plataformas o lenguajes*. Siempre se adapta al tipo de sintaxis o plataformas con las que se estén trabajando, lo que ofrece una gran libertad a la hora de cambiar o probar nuevos entornos dentro del desarrollo. Con una API REST se pueden tener servidores Java, Python o Node.js. Lo único que es indispensable es que las respuestas a las peticiones se hagan siempre en el lenguaje de intercambio de información usado, normalmente XML o JSON.

Lecturas adicionales

Para obtener información adicional, puede consultar:

- https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm
- Libros: REST API Design Rulebook by Mark Masse
- How a RESTful API represents resources.pdf
- How a RESTful API server reacts to requests.pdf
- How to design a RESTful API architecture from a human.pdf

Resumen

En este capítulo, usted aprendió:

- El protocolo HTTP y sus versiones.
- Puntos claves para identificar los recurso de un servicio: Acerca de REST y restricciones
- Definición, importancia, características y ventajas de usar REST.
- Proceso de diseño de un servicio RESTful sobre HTTP (laboratorio).
- Analizar un caso de estudio (laboratorio)

Tareas 1.1: Diseñar un Servicio Web usando RESTful API

Investigar y diseñar un servicio Web en RESTful de un caso que usted proponga.

- Presentar una ficha técnica del diseño web.