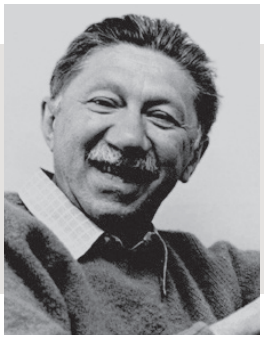
Una guía para el diseño de API y REST

ca technologies

Si la única herramienta que tuviera fuera un martillo...

En su libro “The Psychology of Science” publicado en 1966, el psicólogo estadounidense Abraham Maslow trató la idea de que aquellos que trabajan en el campo de la psicología necesitan abordar el tratamiento desde varias perspectivas para adoptar nuevas ideas, y no solo continuar utilizando las mismas teorías y técnicas creadas por Freud y sus colegas hace tantos años. Tras reconocer que cambiar el punto de vista puede ser difícil, Maslow señaló que “resulta tentador, si la única herramienta disponible fuera un martillo, tratar todo como si fuera un clavo”. Todos lo hemos vivido. Nos hemos adaptado tanto a la manera en que se hicieron las cosas anteriormente que a veces no nos preguntamos los motivos por los cuales las realizamos.

Puede parecer curioso consultar con un psicólogo en un trabajo sobre el diseño de API (interfaz de programación de aplicaciones) y REST (transferencia de estado representacional), pero sirve para ilustrar dos puntos diferentes: (1) todas las decisiones de diseño, independientemente de que estén relacionadas con el software o la arquitectura, se deben tomar en el contexto de los requisitos funcionales, de comportamiento y sociales, y no como tendencias aleatorias; (2) cuando solo se sabe cómo hacer una cosa bien, lo demás tiende a verse idéntico.



> “Si la única herramienta disponible fuera un martillo, todo lo demás se vería como un clavo”.
>
> — Abraham Maslow, The Psychology of Science

En su tesis, “Architectural Styles and the Design of Network-based Software Architectures”,[1] Roy Fielding define la REST (transferencia de estado representacional) de la siguiente manera: “Considere cuán a menudo vemos que los proyectos de software comienzan con la adopción de la última moda en diseño arquitectónico, y únicamente después se descubre si los requisitos del sistema requieren o no este tipo de arquitectura”.

¿No tiene tiempo para leer la tesis completa de Fielding? No hay problema. Creamos esta descripción general de alto nivel pensando en usted. Para comenzar, analicemos la REST más detalladamente.

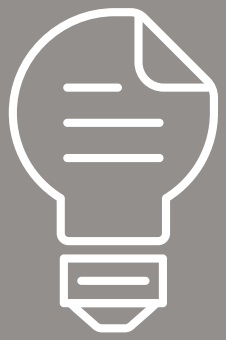
[1] Architectural Styles and the Design of Network-based Software Architectures

Estilo frente a estándar

Un estilo arquitectónico es un concepto abstracto, y no algo concreto. Por ejemplo, considere una catedral gótica. La catedral es diferente del estilo arquitectónico gótico. El estilo gótico define los atributos o las características que apreciaría en una catedral construida con dicho estilo.

De manera comparativa, el NIST (Instituto Nacional de Normas y Tecnología) y los NEC (Códigos eléctricos nacionales) regulan los organismos que formulan las reglas que reconocemos como estándares. Si no se hacen las conexiones eléctricas correctamente en una construcción, el establecimiento podría incendiarse hasta derrumbarse. A menudo, las personas confunden el concepto de *estándares* (lo que sabemos que es correcto o incorrecto) con el concepto de *estilos*, que se refiere a un modo de expresión en particular.

En Internet, la REST es un estilo y el HTTP (protocolo de transferencia de hipertexto) es un estándar. La REST depende de protocolos de comunicaciones con caché de cliente-servidor, sin estado, como el protocolo HTTP, para facilitar el desarrollo de las aplicaciones. Al aplicar los principios del diseño de REST a un protocolo como HTTP, los desarrolladores pueden crear interfaces para su uso en prácticamente cualquier dispositivo o sistema operativo.

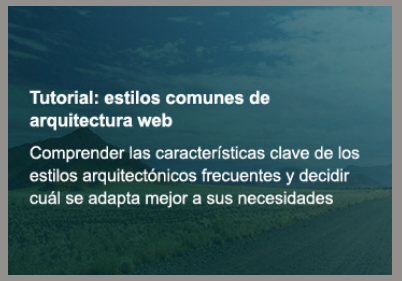


¿Cuál es su estilo?

Los tres estilos de arquitectura web comunes son estos:

- túneles (SOAP [protocolo simple de acceso a objetos]);
- objetos (CRUD [crear, leer, actualizar o eliminar]);
- hipermedios (REST [transferencia de estado representacional]).

Vea este video[2] para comprender las características clave de los estilos arquitectónicos comunes y decidir cuál se adapta mejor a sus necesidades.



[2] Diseño de API, lección 201: estilos arquitectónicos de API web, API Academy.

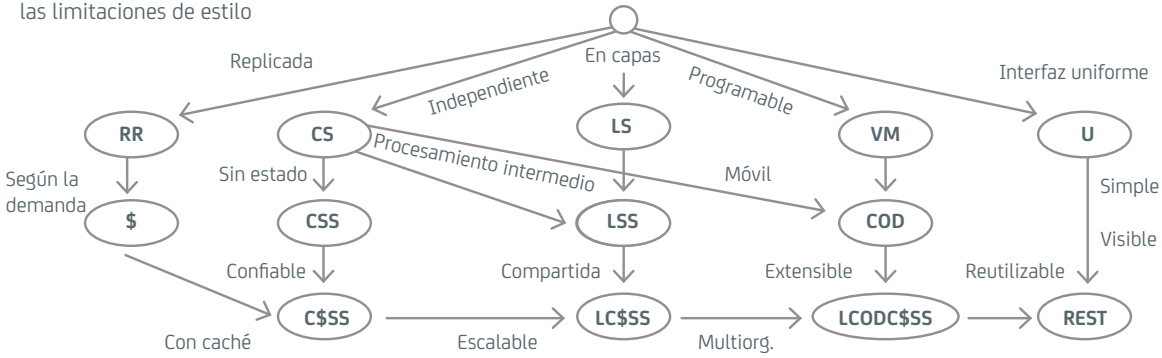
Los estilos se describen mediante limitaciones

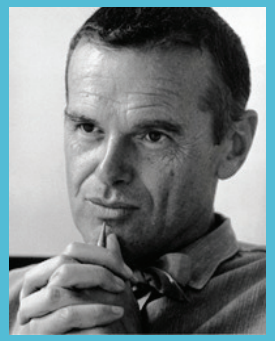
Un estilo arquitectónico se describe mediante las características que hacen que una construcción u otra estructura se pueda distinguir e identificar. Las formas características de una arquitectura gótica, por ejemplo, incluyen arco en punta, bóveda de crucería, contrafuertes, ventanas grandes que a menudo están agrupadas, rosetones, torres, chapiteles, pináculos y fachadas ornamentadas.

De forma similar, la REST se describe mediante un conjunto de limitaciones arquitectónicas que tratan de minimizar la latencia y las comunicaciones de la red y, al mismo tiempo, maximizar la independencia y escalabilidad de las implementaciones de los componentes. Las seis limitaciones de la REST incluyen lo siguiente:

1. **Cliente-servidor**: requiere que un servicio ofrezca una o más operaciones, y que los servicios esperen que los clientes soliciten dichas operaciones.
2. **Sin estado**: requiere que las comunicaciones entre el consumidor del servicio (cliente) y el proveedor del servicio (servidor) no tengan estado.
3. **Caché**: requiere que las respuestas se etiqueten claramente como con caché o son caché.
4. **Interfaz uniforme**: requiere que todos los proveedores del servicio y consumidores dentro de una arquitectura que cumple con la REST compartan una sola interfaz común para todas las operaciones.
5. **Sistema en capas**: requiere la capacidad de agregar o quitar intermediarios en el tiempo de ejecución sin alterar el sistema.
6. **Código según la demanda (opcional)**: permite que la lógica dentro de los clientes (como exploradores web) se actualice independientemente de la lógica del servidor utilizando un código ejecutable enviado por los proveedores del servicio a los consumidores.

Figura: derivación de la REST mediante las limitaciones de estilo





"Esta es una de las pocas claves eficaces del problema de diseño (la capacidad del diseñador de reconocer la mayor cantidad de limitaciones posible), su predisposición y entusiasmo para trabajar dentro de estas limitaciones. Las limitaciones de precio, de tamaño, de solidez, de equilibrio, de superficie, de tiempo, y así sucesivamente".[3]

— Charles Eames, Eames Design.

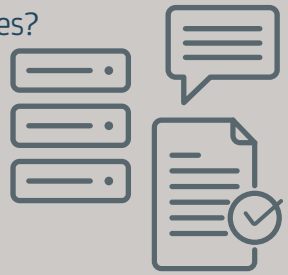
[3] http://www.eamesoffice.com/the-work/design-q-a-text/

Conectores ≠ componentes

Según Fielding, “la [REST] se logra al aplicar limitaciones en la semántica de los conectores, mientras que otros estilos se centran en la semántica de los componentes”. Su diseño apunta a limitar la manera en que las cosas se conectan entre sí, no la manera en que funcionan internamente, y aplica su teoría a toda la red. Al crear aplicaciones a gran escala, a menudo se pasa por alto el concepto de que los conectores no son lo mismo que los componentes. Pero Fielding pone este concepto en un primer plano.

Entonces, ¿cuáles son los componentes? Entre ellos, se incluyen los siguientes:

- Bases de datos
- Sistemas de archivos
- Colas de mensajes
- Herramientas del administrador de transacciones
- Código de fuente



Estos componentes son distintos a los conectores, que incluyen lo siguiente:

- Servidores web
- Agentes del explorador
- Servidores proxy
- Caché compartida

Los componentes sirven para solucionar los problemas de formas exclusivas. MySQL funciona de manera diferente del servidor SQL, al igual que CouchDB o MongoDB. Se puede decir lo mismo para los sistemas de archivos basados en UNIX, Windows o Mac. La manera en que coloca información en la cola, la manera en que decide cuándo iniciar y finalizar una transacción; estas son características totalmente locales de los componentes que el desarrollador puede manipular. Son los componentes del desarrollador, su sistema operativo, sus herramientas e idioma, y, por lo tanto, son privados.

Los conectores, en cambio, son públicos. Son una serie de canales estandarizados con los que trabajan todos los desarrolladores. En función del principio de Fielding, los desarrolladores pueden ser tan creativos o tan rutinarios como deseen respecto de sus componentes privados, siempre y cuando acepten transmitir la información de un lado a otro utilizando los conectores públicos estandarizados.

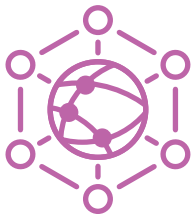
Mantenga los componentes y los conectores por separado, ya que esto le facilitará intercambiarlos más adelante. Por ejemplo, el código que debe introducir para su servidor web está diseñado para dirigirse a muchos dispositivos en la red de Internet pública. Sin embargo, el código que introduzca para sus componentes está diseñado para dirigirse específicamente a las herramientas que tenga disponibles.

Cómo garantizar que los conectores funcionen en conjunto

Al crear aplicaciones basadas en el cliente o servicios para servidores, es este acoplamiento de componentes privados con conectores públicos (conectarlos y unirlos entre sí) el que puede hacer que el desarrollo sea desafiante y muy interesante. Entonces, ¿cómo puede garantizar que un conector funcione? ¿Cómo puede diseñar aplicaciones escalables si se comunican mediante conectores?

Identificación de recursos:
URI, URL y URN como identificadores

Representaciones de recursos:
tipos de medios como formas de representar la información transferida entre las partes

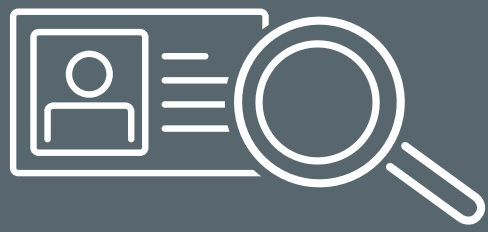


Mensajes autodescriptivos:
combinación de metadatos en encabezados, así como también en el cuerpo de un mensaje, a fin de crear una respuesta autodescriptiva

Hipermedios:
enlaces y formas como una manera de describirle al cliente las acciones disponibles que son respaldadas actualmente por el servicio

La limitación de la interfaz uniforme es fundamental para el diseño de cualquier servicio de REST. Simplifica y desacopla los conectores, lo que permite que cada pieza evoluciones de forma independiente. Debido a la manera en que la Web se utiliza en la actualidad, las cuatro limitaciones mencionadas anteriormente constituyen las herramientas esenciales que les permiten a los desarrolladores implementar la interfaz uniforme de Fielding. En las siguientes páginas, se explican estas herramientas más detalladamente.

URI para la identificación

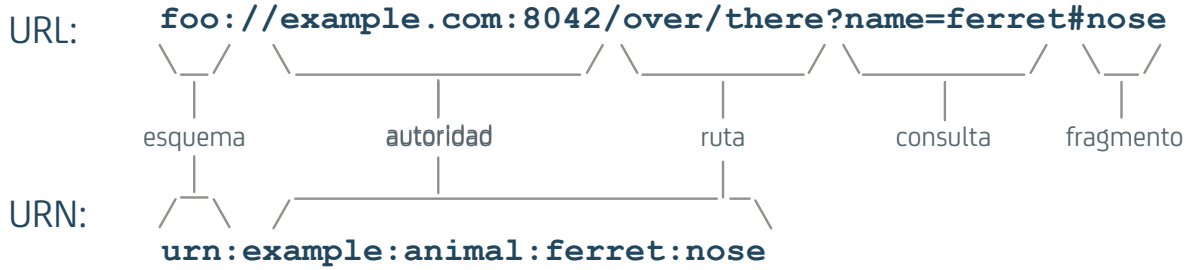


Tal como se describe en RFC2396[4], "un URI (identificador uniforme de recursos) es una cadena compacta de caracteres que permiten identificar un recurso abstracto o físico". El identificador se puede implementar en una de dos maneras: un URL (localizador uniforme de recursos) o un URN (nombre uniforme de recursos). Los URL (p. ej., http://example.org/users/mike) se utilizan para identificar la ubicación en línea de un recurso individual, mientras que los URN (p. ej., urn:usuario:mike) se desarrollan con el fin de funcionar como identificadores persistentes, independientes de la ubicación. El URN funciona como el nombre de una persona y el URL se asemeja al domicilio de una persona. En otras palabras, el URN define la identidad de un elemento ("el nombre de usuario es mike") y el URL proporciona un método para buscarlo ("mike se puede encontrar en ejemplo.org/usuarios/").

Los componentes de un URI incluyen los siguientes:

- **Nombre de esquema**: identifica el protocolo (p. ej., FTP:, HTTP:, HTTPS:, IRC:).
- **Parte jerárquica**: tiene el objetivo de mantener la información jerárquica por naturaleza.
  - **Autoridad**: se refiere a la resolución del DNS (sistema de nombres de dominio) real del servidor (p. ej., nombre de dominio o dirección IP).
  - **Ruta**: se refiere a una secuencia de segmentos separados por una barra ("/").
- **Consulta**: contiene información de identificación adicional que no es jerárquica por naturaleza y, a menudo, está separada por un signo de interrogación ("?").
- **Fragmento**: proporciona la orientación hacia un recurso secundario dentro de uno principal identificado por la autoridad y la ruta, y está separado del resto mediante el signo de numeral ("#").

La estructura de los URI



[4] https://tools.ietf.org/html/rfc2396

Tipos de medios para representación

Según RFC2046[5], los identificadores del tipo MIME (extensiones multipropósito de correo de Internet) (tipos de medios) se deben utilizar para "especificar la naturaleza de los datos en el cuerpo de una entidad de MIME, junto con la información auxiliar que pueda requerirse". Los tipos de MIME se utilizaron primero en las transmisiones de mensajes de correo electrónico, como se demuestra con su nombre completo: Extensiones multipropósito de correo de Internet. En la actualidad, los tipos de MIME permiten que las personas intercambien diferentes tipos de archivos de datos en Internet: audio, video, texto, imágenes y programas de aplicaciones.

Los tipos de MIME (o tipos de medios) identifican la naturaleza de los datos y la información auxiliar. En la Web, los tipos de medios también identifican las reglas de procesamiento de mensajes. La cadena identificadora de tipo de MIME incluye un tipo y subtipo separados por una barra (p. ej., texto/sin formato, imagen/gif, etc.).

Además de las cadenas de tipo de MIME estándar (p. ej., aplicación/json), se pueden crear identificadores utilizando las siguientes convenciones:

- Utilizar vnd. como prefijo del subtipo para los tipos de MIME específicos del proveedor, que forma parte de un producto comercial (p. ej., vnd.bigcompany.report/json).
- Utilizar prs. como prefijo del subtipo para los tipos de MIME personales o de vanidad, que no forma parte de un producto comercial (p. ej., prs.smith.data/json).

Registro del tipo de MIME

Para ver una lista completa de los tipos de MIME oficiales asignados por la IANA (Autoridad de Números Asignados de Internet), haga clic aquí.

[5] https://tools.ietf.org/html/rfc2396

Encabezados + cuerpo de mensajes autodescriptivos

En RFC2616[6], se establece que los mensajes [en HTTP] incluyen una línea inicial, cero campos o más campos de encabezado (también conocidos como “encabezado”), una línea vacía (p. ej., una línea sin texto que preceda CRFL), que indica el final de los campos de encabezado, y posiblemente un cuerpo del mensaje.

Cada cliente realiza una solicitud y la respuesta del servidor es un mensaje; las aplicaciones que cumplen con la REST esperan que cada mensaje sea autodescriptivo. Esto significa que cada mensaje debe contener toda la información necesaria para completar la tarea. Otras formas de describir este tipo de mensaje son “sin estado” o “sin contexto”. Cada mensaje transferido entre el cliente y el servidor puede tener un cuerpo y metadatos.

Las implementaciones de la REST también dependen de la noción de un conjunto de operaciones limitadas que tanto el cliente como el servidor entienden totalmente desde el comienzo. En el protocolo HTTP, las operaciones se describen en la “línea inicial”, y las seis operaciones utilizadas más comúnmente en HTTP son las siguientes:

- **GET:** devuelve la información que se haya identificado mediante el URI de solicitud.
- **HEAD:** es similar a GET, excepto que el servidor no debe devolver un cuerpo de mensaje en la respuesta, sino solo los metadatos.
- **OPTIONS:** devuelve información acerca de las opciones de comunicación disponibles en la cadena de solicitud y respuesta identificada por el URI de solicitud.
- **PUT:** solicita que la entidad adjunta se almacene en el URI de solicitud suministrado.
- **POST:** solicita que el servidor de origen acepte la entidad adjunta en la solicitud como un nuevo subordinado del recurso identificado por el URI de solicitud.
- **DELETE:** solicita que el servidor de origen elimine el recurso identificado por el URI de solicitud.

Las primeras tres operaciones son de solo lectura, mientras que las últimas tres son operaciones de escritura. En el protocolo HTTP, existen reglas bien definidas sobre cómo se espera que los clientes y servidores se comporten cuando utilicen estos operadores. Los nombres y significados de los elementos de metadatos adjuntos (encabezados) también están bien definidos. Las aplicaciones que cumplen con la REST y se ejecutan en el protocolo HTTP comprenden y siguen estas reglas rigurosamente.

[6] https://tools.ietf.org/html/rfc2396

Ejemplo de intercambio “GET” de HTTP

Para recuperar un archivo en http://www.unhost.com/path/file.html, abra un socket en el host www.unhost.com.ar, utilice el puerto predeterminado 80 porque no se especificó ninguno en la dirección URL y envíe lo siguiente a través del socket:

```
GET/path/file.html HTTP/1.0
De: usuario@jmarshall.com
Usuario o agente: HTTPTool/1.0
[dejar línea en blanco]
```

Vuelva a enviar los datos a través del mismo socket, y el servidor responderá con lo siguiente:

```
HTTP/1.0 200 OK
Fecha: vie, 31 de diciembre de 1999 23:59:59 GMT
Tipo o contenido: texto/html
Extensión del contenido: 1354 (Herramientas
de diagnóstico y personalización 300)

<html>
<body>
<h1>Happy New Year!</h1>
(más contenido del archivo)
</body>
</html>
```

Enlaces y formas de hipermedios

Los enlaces y las formas se utilizan dentro de un tipo de medios para respaldar las limitaciones de hipermedios de Fielding. Por ejemplo, existen algunas posibilidades en HTML y el explorador web común comprende las reglas de todas ellas. Los enlaces y las formas en un mensaje de HTML se reconocen fácilmente, pero lo que quizás no sean tan claras son las reglas de procesamiento, o las semánticas, asociadas a ellos.

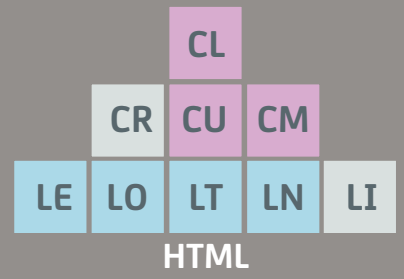
Los enlaces y las formas le proporcionan la capacidad de tomar medidas: son *posibilidades*. HTML tiene un conjunto de posibilidades bien definido. Algunas posibilidades le permiten escribir datos, como una forma que tiene una propiedad de método establecida en "POST". Algunas posibilidades le permiten obtener datos de una ubicación remota para verlos en un documento HTML actual, como un elemento IMG. El elemento A de HTML es una posibilidad para navegar a una nueva ubicación en la Web.

Según Fielding, "el hipermedio se define mediante la presencia de información de control de aplicación incorporada dentro de dicho elemento o como una capa superior, la presentación de la información". Para Fielding, la REST les ofrece a los proveedores de servicios la capacidad de enviar información de control (enlaces y formas) a las aplicaciones del cliente en todo el mundo mediante el envío de posibilidades, los controles de hipermedios.

Factores H

Al comparar los tipos de medios, puede ser útil documentar los factores H existentes en un cuadro visual sencillo. En el ejemplo que sigue a continuación, la fila inferior identifica los factores de enlace básicos (los factores de hipermedios más notables), mientras que las dos filas superiores identifican los factores de datos de control.

Factores de hipermedios



Soporte de enlace
[LE] Enlaces incorporados
[LO] Enlaces salientes
[LT] Consultas enviadas en plantilla
[LN] Actualizaciones no idempotentes
[LI] Actualizaciones idempotentes

Soporte de datos de control
[CR] Datos de control para solicitudes leídas
[CU] Datos de control para solicitudes de actualización
[CM] Datos de control para métodos de la interfaz
[CL] Datos de control para enlaces

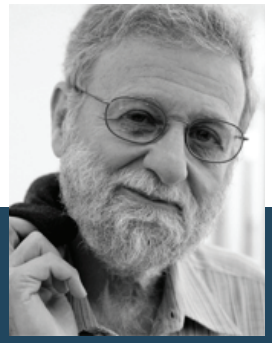
Para obtener más información, sobre los factores H: http://amundsen.com/hypermedia/hfactor/.

Software que extiende la vida útil

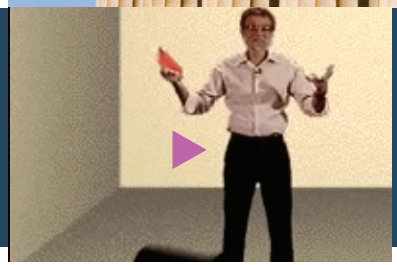
La arquitectura física extiende la vida útil. Los establecimientos construidos hace cientos de años pueden ser tan vívidos, tan útiles, tan vibrantes y reconfortantes actualmente como lo fueron anteriormente cuando las personas entraron en ellos por primera vez, incluso cuando se cambiaron los usos originales de dichos establecimientos, por ejemplo, cuando las iglesias se transformaron en museos o las salas de reuniones se convirtieron en departamentos. ¿Por qué? Porque estos establecimientos eternos dependen de patrones arquitectónicos universales que traspasan el espacio y el tiempo. Una entrada es una entrada; una ventana es una ventana. La manera en que estos elementos se implementan depende de los materiales disponibles. La forma en que se representan es diferente en cada caso, pero se pueden identificar igualmente año tras año.

En el desarrollo de software, el concepto es el mismo. Algunas veces, los desarrolladores y arquitectos de software quieren crear aplicaciones que duren mucho tiempo. La REST, con su conjunto de limitaciones universales (como patrones arquitectónicos) es una manera de cumplir con este fin.

Pero Fielding no dijo que esta sea la única manera de alcanzar el éxito. Cuando escribió su tesis, no incluyó todas las posibilidades, todas las respuestas. De hecho, hubo varias partes que quedaron sin terminar. Sin embargo, lo que Fielding sí hizo fue documentar su estrategia de crear un estilo arquitectónico para software de redes que se basó en la identificación de una serie de limitaciones con el fin de satisfacer este objetivo de minimizar la latencia y maximizar la escalabilidad. De hecho, utilizó las limitaciones de la misma manera que Charles Eames las usó porque ambos lograron cumplir con este objetivo.



Podemos aprovechar la experiencia de Fielding para poder ver las cosas de una manera un poco diferente. Podemos usar las herramientas que proporcionó para experimentar con las limitaciones y expresar nuestro propio estilo y, durante el proceso, crear aplicaciones de software excepcionales que, si así lo deseamos, puedan resistir la prueba del tiempo.



"El valor de un objetivo bien diseñado se adquiere cuando cuenta con un importante conjunto de posibilidades con el cual las personas que lo utilicen puedan hacer cosas que el diseñador jamás haya imaginado".[7]

— Donald Norman, 1994

[7] https://www.youtube.com/watch?v=NK1Zb_5VxuM

Obtenga más información sobre CA API Management

Visite **ca.com/ar/api**

CA Technologies (NASDAQ: CA) crea un software que impulsa la transformación en las empresas y les permite aprovechar las oportunidades de la economía de aplicaciones. El software es el centro de cada empresa, en cada sector. Desde la planificación hasta el desarrollo, la administración y la seguridad, CA trabaja con empresas en todo el mundo para cambiar el estilo de vida, realizar transacciones y comunicarse, mediante entornos móviles, de nubes públicas y privadas, centrales y distribuidos. Obtenga más información en ca.com/ar.

CS200-110010

ca technologies