REST

Representational State Transfer (REST) is a style of software architecture for distributed hypermedia systems such as the WWW

REST, or REpresentational State Transfer, is an architectural style for providing standards between computer systems on the web, making it easier for systems to communicate with each other. REST-compliant systems, often called RESTful systems, are characterized by how they are stateless and separate the concerns of client and server.

A horizontal abstraction across multiple architectures (vertical abstractions)

- names a repeated architectural pattern
- defined by its design constraints
- chosen for the properties they induce

REST is an architectural style

- for network-based applications

Arquitectura REST Style

- REST: REpresentational State Transfer
- Independiente del protocolo
- Soporte de URIs – Principalmente usado con HTTP
- Estilo de arquitectura
  - No es un estándar (no hay RFC)
  - No hay "hard rules"
- Características principales
  - Sigue el paradigma cliente-servidor
  - Sin estado
  - Sistema basado en capas
  - Interfaz uniforme y desacoplada
  - Hiperenlace como motor
  - Cacheable
- Orientado al recurso
- Un estilo de arquitectura es una serie de restricciones que limitan y moldean la forma de la arquitectura y las relaciones entre los elementos en cualquier arquitectura que sigue el estilo

Perspectives on the process of architectural design:

- Start with nothing
  - Build-up an architecture from familiar components
  - Until it satisfies the needs of the intended system
  - Emphasizes creativity and unbounded vision
- Start with the system needs as a whole
  - Start without constraints
  - Incrementally identify and apply constraints
  - Emphasizes restraint and understanding of the system context

REST Uniform Interface

- All important resources are identified by one (uniform) resource identifier mechanism
- La definición de los recursos consta de
  - URI
  - Formato de intercambio de datos (JSON, XML)
  - Métodos (GET, PUT, POST)
- Los recursos se identifican de forma unívoca por su URI
- Los servicios permiten leer, crear, modificar y eliminar recursos
- Hypermedia as the engine of Application State (HATEOAS)
- Las operaciones se corresponden con los métodos HTTP
  - GET
    - obtener un recurso
  - POST
    - crear un recurso nuevo
  - PUT
    - crear / actualizar un recurso
  - DELETE
    - eliminar un recurso
- Códigos de estado correspondientes a HTTP
  - 200 OK: Petición finalizó con éxito
  - 201 Created: La petición POST finalizó correctamente y el objeto fue creado, se debe acompañar con la URI del nuevo objeto
  - 204 No content: La petición finalizó con éxito pero no devuelve ningún cuerpo de mensaje
  - 400 Bad request: La petición no se pudo realizar por estar mal formateada
  - 401 Unauthorizaed: Faltan datos de autenticación y/o autorización
  - 403 Forbidden: Autenticación OK, pero no se tiene acceso al recurso solicitado
  - 404 Not found: El recurso solicitado no existe
  - 405 Method not allowed: El método HTTP no está soportado en la URI dada

Orientado al recurso

- Con HTTP el recurso se representa en el cuerpo del mensaje
- El tipo de formato se especifica mediante el tipo de contenido
- El cliente puede negociar con Accept
- Content-type
  - text/plain
  - text/html
  - application/xml
  - application/xml
- Contrato de servicio WADL
  - Equivalente a WSDL de SOAP
  - Su uso es opcional, no demasiado extendido, se puede usar WSDL también

    - Útil para generar código de cliente y de servidor
    - Mejor integración con herramientas y frameworks
    - Hay otros estándares mejores como OpenAPI (Swagger)

Sin estado

- El servidor no debe mantener el estado del cliente
    - Simplifica el balanceo de carga
    - Permite el uso de caches
- el servidor almacena únicamente el estado de sus recursos
- La sesiones del lado del servidor no estan permitidas
    - El cliente si puede gestionar el una sesion
    - Cualquier petición realizada desde el cliente tendrá que contener toda la información que se necesite para poderla procesar en el servidor

Rest debería funcionar como máquina de estados

HATEOAS

- En REST una aplicación web se puede ver como una gran máquina virtual de estados
    - Los distintos recursos son los nodos de la máquina
    - Los enlaces representan las transiciones de estado
    - El siguiente estado de la máquina será el resultado de seguir el enlace y el servicio al que se acceda
- Hypermedia As The Engine Of Application State
- Extensión del concepto de hipervínculo
- Los recursos tienen enlaces que apuntan a su vez a otros recursos
- La aplicación sabe tras cada respuesta cuáles son las posibles continuaciones
- El cliente mantiene el estado de la aplicación (máquina de estados implícita)

Seguridad en REST

- La seguridad en servicios REST emplea los mecanismos disponibles en las capas de transporte (SSL/TLS) y HTTP
- Mecanismos de autenticación HTTP
    - HTTP Basic
    - HTTP Digest (not widely supported)
- SSL permite emplear cifrado y firma
- Limitación frente a las soluciones de seguridad basada en mensaje proporcionadas por WS-Security

Ventajas rest

- maximizar el reuso
- escalabilidad
- es sencillo
- Utiliza tecnologías y plataformas bien documentas y maduras (http)
- facilidad de uso, dada una uri todo el mundo sabe como accerder a ella

facilidad de uso, dada una un todo el mundo sabe como acceder a ella

- Nos evitamos añadir otro protocolo
- Permite utilizar gran cantidad de formatos en el mensaje
- Hereda la seguridad de HTTP, tecnología madura
- "Makes sense" usa lo que ya esta establecido para dar un paso más, es la extensión lógica de la web

Servicios RESTful

- Richardson propuso una clasificación de servicios de la web, lo que se conoce como modelo de madurez "maturity model"
- En esta clasificación tenemos varios niveles, basados en el soporte de un servicio para URIs, http y hipermedia
- Es una clasificación general que nos permite describir todos los servicios web

Nivel 0

- nivel más básico
- Servicios que tienen una única URI
- Que solo utilizan un único método HTTP (POST típicamente)
- Ej: Servicios SOAP
- Se usa HTTP pero sin aprovechar las ventajas y las opciones que da (“tunelling” using HTTP)

Nivel 1

- Utilizamos varias URIs pero únicamente un método HTTP (normalmente GET o POST)
- Se exponen varios recursos
- Muchos de los servicios REST actuales se quedan en este nivel

Nivel 2

- Utilizamos varias URIs que además soportan varios métodos HTTP
- Tenemos servicios que soportan las operaciones CRUD
- También se utilizan los códigos de HTTP
- Dejamos de utilizar HTTP simplemente como transporte

Nivel 3

- Se añade la noción de HATEOAS
- Las representaciones incluyen URIs que hacen referencia a otros recursos, la transición de un recurso a otro genera el estado de la aplicación

Buenas prácticas en REST

- nombre de recurso que queramos acceder
  - books
    - http://example.com/api/books o http://example.com/api/books/10
- Asociaciones. La API debe ser intuitiva a la hora de definirlas asociaciones
- http://example.com/api/user/23/books/10
  - Obtener el libro con ID 10 para el usuario con ID 23
- No sobre cargar al cliente ,si el contenido que se va a a servir es muy grande usar paginación (no

devolver todos los resultados)
- Peticiones Get tienen un tamaño máximo
- SI el servicio tiene muchos argumentos o con valores de tamaño variable se debe utilizar POST o PUT
- Usar correctamente los métodos y códigos de error HTTP
- Para buscar, ordenar, filtrar, paginar, etc
    - No se crearan nuevos recursos sino que se añadirán los parámetros al GET
- En una API siempre se debe mantener la compatibilidad hacia atrás, si no se hace, se debe cambiar el nombre
- Usar un mapeo de URIs lo más intuitivo posible y no cambiarlo
- Pensar muy bien cómo va a realizar el consumidor todas las operaciones necesarias
- Aplicar HATEOAS

Estandarización
- Lenguajes de descripción (Description Languajes)
    - Ejemplos
        - WADL
        - WSDL
        - OpenAPI

Webhooks
- cliente se subscribe a un servicio
- envia un mensaje si hay actualización

Rest
- el cliente tiene que esperar y estar atento a los cambios

Ejercicio
tabla verde
interfaz uniforme