DESARROLLO DE APLICACIONES CON TECNOLOGÍAS WEB (IFCD0210)

ASATA
GOBIERNO DE ESPAÑA
MINISTERIO DE EDUCACIÓN, FORMACIÓN PROFESIONAL Y DEPORTES
SERVICIO PÚBLICO DE EMPLEO
GOBIERNO DEL PRINCIPADO DE ASTURIAS

ASATA

GOBIERNO DE ESPAÑA
MINISTERIO DE EDUCACIÓN, FORMACIÓN PROFESIONAL Y DEPORTES

SERVICIO PÚBLICO DE EMPLEO
GOBIERNO DEL PRINCIPADO DE ASTURIAS

001

MF0492_3: Programación web en el entorno servidor

UF1846: Desarrollo de aplicaciones web distribuidas.

Que es SOA Arquitectura Orientada a Servicio

Modelo conceptual que permite desarrollar aplicaciones web distribuidas utilizando servicios de terceros a través de la red.

Un sistema distribuido consiste en un conjunto de nodos de computo que están comunicados entre si a través de una red de comunicación (local o remota). Actúan como una única unidad.

Sistemas donde diferentes procesos interaccionan entre sí mediante el paso de mensajes, y donde estos procesos están ejecutándose en el mismo nodo o en nodos diferentes

Que es un servicio Web

Un servicio web es un sistema de software diseñado para admitir la interacción interoperable de máquina a máquina a través de una red.

Tiene una interfaz descrita en un formato procesable por máquina (específicamente WSDL).

Otros sistemas interactúan con el servicio web de la manera prescrita por su descripción usando mensajes SOAP, típicamente transmitidos usando HTTP con una serialización XML junto con otros estándares relacionados con la web.

CONPUTACIÓN PARALELA

1. *La computación paralela se refiere a la ejecución simultánea de múltiples cálculos o procesos. Este paradigma se enfoca en dividir una tarea grande en sub-tareas más pequeñas que pueden ejecutarse simultáneamente en diferentes procesadore*
2. *División de Tareas: La tarea principal se divide en sub-tareas independientes.*
3. *Hardware: Requiere múltiples núcleos o procesadores para funcionar eficazmente*

COMPUTACIÓN CONCURRENTE

1. *La computación concurrente se refiere a la capacidad de un sistema para manejar múltiples tareas al mismo tiempo, pero no necesariamente ejecutarlas simultáneamente. La concurrencia se basa en la intercalación de las tareas en pequeños intervalos de tiempo.*
2. *Un solo procesador puede gestionar múltiples tareas "al mismo tiempo" cambiando rápidamente entre ellas*
3. *Usan mecanismos de sincronización para evitar conflictos*

CONPUTACIÓN PARALELA

1. *Ejecución: Tareas se ejecutan simultáneamente.*
2. *Hardware: Necesita múltiples procesadores o núcleos*
3. *Enfoque: Se enfoca en dividir una tarea en sub-tareas independientes*
4. *Aplicaciones típicas: Cálculos intensivos que pueden dividirse en partes independientes*

COMPUTACIÓN CONCURRENTE

1. *Ejecución: Tareas se alternan en la ejecución*
2. *Hardware: Puede ejecutarse en un solo procesador mediante la alternancia rápida entre tareas.*
3. *Enfoque: Se enfoca en gestionar múltiples tareas que pueden no ser completamente independientes*
4. *Aplicaciones tipicas: Gestión de múltiples tareas en sistemas donde la interacción y la latencia son críticas.*

Características de un sistema distribuido

Transparencia

- Transparencia de Acceso: Los usuarios interactúan con el sistema sin preocuparse por la ubicación de los recursos.
- Transparencia de Ubicación: Los recursos pueden estar en cualquier lugar de la red.
- Transparencia de Migración: Los recursos pueden moverse sin afectar el funcionamiento del sistema.
- Transparencia de Replicación: Los usuarios no perciben la existencia de múltiples copias de recursos.

Características de un sistema distribuido

Escalabilidad

- Escalabilidad Horizontal: Capacidad para agregar más nodos al sistema.
- Escalabilidad Vertical: Capacidad para mejorar los nodos existentes (más potencia de CPU, más memoria).

Características de un sistema distribuido

Confiabilidad y Tolerancia a Fallos

- Redundancia: Uso de componentes redundantes para evitar fallos.
- Recuperación Automática: Capacidad del sistema para recuperarse automáticamente de fallos.
- Balanceo de Carga: Distribución eficiente de la carga de trabajo entre los nodos.

Características de un sistema distribuido

Flexibilidad

- Adaptabilidad: Capacidad del sistema para adaptarse a cambios en la carga de trabajo o en el entorno.
- Extensibilidad: Facilidad para agregar nuevas funcionalidades o nodos al sistema.

Características de un sistema distribuido

Heterogeneidad

- Diversidad de Componentes: Capacidad para integrar diferentes tipos de hardware y software.
- Interoperabilidad: Capacidad para que componentes de diferentes proveedores trabajen juntos.

Características de un sistema distribuido

Seguridad

- Autenticación: Verificación de la identidad de los usuarios y nodos.
- Autorización: Control de acceso a recursos según políticas definidas.
- Cifrado: Protección de datos en tránsito y en reposo.
- Integridad: Aseguramiento de que los datos no han sido alterados.

Características de un sistema distribuido

Coherencia y Consistencia

- Consistencia de Datos: Garantía de que todas las copias de los datos son iguales en todos los nodos.
- Protocolos de Consenso: Mecanismos para asegurar la consistencia en decisiones distribuidas.

Características de un sistema distribuido

Desempeño

- Latencia: Tiempo de respuesta del sistema.
- Ancho de Banda: Capacidad de transferencia de datos del sistema.
- Throughput: Cantidad de trabajo que el sistema puede manejar en un tiempo determinado.

Características de un sistema distribuido

Concurrencia

- Procesamiento Concurrente: Capacidad para manejar múltiples tareas al mismo tiempo.
- Sincronización: Mecanismos para coordinar tareas concurrentes y evitar conflictos

Origen SOA

La Arquitectura Orientada a Servicios (SOA) es un paradigma de diseño de software que ha evolucionado a lo largo de varias décadas. Su origen se puede rastrear a diversas influencias tecnológicas y necesidades empresariales que surgieron con el tiempo.

Origen SOA

Años 1970 - 1980: Primeros Intentos y Bases Teóricas

- RPC (Remote Procedure Call): En los años 70 y 80, los primeros conceptos de comunicación remota y distribución de tareas comenzaron a tomar forma con RPC, que permitía a los programas ejecutar procedimientos en servidores remotos.
- Arquitecturas Cliente-Servidor: Durante los años 80, las arquitecturas cliente-servidor comenzaron a ser populares, estableciendo las bases para la interacción entre diferentes sistemas a través de la red.

Origen SOA

Años 1990: Desarrollo de Tecnologías Clave

- CORBA (Common Object Request Broker Architecture): En los años 90, CORBA proporcionó un marco para la comunicación entre objetos distribuidos en una red, influyendo en las ideas que luego conformarían SOA.
- DCOM (Distributed Component Object Model): Introducido por Microsoft, DCOM también jugó un papel en el desarrollo de tecnologías distribuidas.

Origen SOA

Finales de los 1990s - 2000s: Emergencia de SOA

- Evolución de XML y Web Services:
  - XML (eXtensible Markup Language): La adopción de XML en los 90 permitió un formato de datos estructurado y legible por máquinas que facilitó la interoperabilidad entre sistemas heterogéneos.
  - SOAP (Simple Object Access Protocol): Introducido a finales de los 90, SOAP utilizó XML para permitir la comunicación entre aplicaciones a través de HTTP.

Origen SOA

Finales de los 1990s - 2000s: Emergencia de SOA

- Desarrollo de Web Services: El concepto de servicios web, basados en estándares como SOAP, WSDL (Web Services Description Language) y UDDI (Universal Description, Discovery, and Integration), proporcionó los bloques de construcción necesarios para SOA.

- Publicaciones y Estándares: Organizaciones como el W3C y OASIS comenzaron a publicar estándares y especificaciones para servicios web y SOA.

Origen SOA

Años 2000s: Adopción Empresarial y Formalización

- Adopción en Grandes Empresas: Empresas como IBM, Microsoft y Oracle adoptaron y promovieron SOA como una forma de mejorar la integración y reutilización de servicios en sus infraestructuras.

- Formalización del Concepto: Durante esta década, SOA se formalizó como un conjunto de principios y metodologías para diseñar y construir sistemas de software. Se definieron principios como la independencia de plataforma, la interoperabilidad y el uso de interfaces de servicios bien definidas.

Origen SOA

2010s - Presente: Evolución y Nuevas Tendencias

- Microservicios: A partir de los 2010s, el paradigma de microservicios emergió como una evolución de SOA, enfocándose en la creación de servicios aún más pequeños e independientes, mejorando la escalabilidad y la flexibilidad.

- APIs RESTful: Aunque REST no es sinónimo de SOA, muchas implementaciones de SOA adoptaron APIs RESTful debido a su simplicidad y eficiencia en la comunicación entre servicios.

- Computación en la Nube: La popularización de la computación en la nube ha impulsado aún más la adopción de arquitecturas orientadas a servicios, permitiendo una escalabilidad dinámica y una integración más fácil de servicios distribuidos.

Arquitecturas SOA

Arquitectura de Microservicios: Conjunto de servicios pequeños e independientes que se comunican entre sí.

Arquitectura de Servicios Web Basados en SOAP: Utiliza el protocolo SOAP (Simple Object Access Protocol) para la comunicación entre servicios.
Ejemplo: Plataformas empresariales que requieren alta seguridad y transaccionalidad, como servicios financieros y de salud.

Arquitectura RESTful: Servicios que siguen el estilo arquitectónico REST (Representational State Transfer).
Ejemplo: API de Twitter, Google Maps API.

Arquitecturas SOA

Arquitectura Orientada a Eventos: Basada en el envío y recepción de eventos asincrónicos.
Ejemplo: Sistemas de gestión de stock en tiempo real, como el de Walmart.

Arquitectura de Servicios Basada en Mensajes: Utiliza sistemas de mensajería para la comunicación entre servicios.
Ejemplo: MQTT, Amazon SQS (Simple Queue Service), RabbitMQ.

Arquitectura de Servicios Basada en Recursos: Cada servicio expone recursos específicos, usualmente a través de una API REST.
Ejemplo: Servicios de almacenamiento en la nube como Amazon S3.

Arquitecturas SOA

Arquitectura Basada en RPC (Remote Procedure Call) : Permite que un programa ejecute procedimientos en otro espacio de direcciones (en un servidor remoto).
Ejemplo: gRPC de Google, CORBA.

Arquitectura de Servicios Distribuidos en Nube Híbrida: Combina servicios distribuidos en nube pública y privada.
Ejemplo: Implementaciones de Azure Stack en combinación con Azure Cloud.

Arquitectura de Servicios Distribuidos Basada en PaaS (Platform as a Service) : Provee una plataforma que permite a los desarrolladores construir aplicaciones y servicios a través de la nube.
Ejemplo: Google App Engine, Heroku.

Arquitecturas SOA

Arquitectura de Microkernel: Sistema central mínimo que permite la adición de funcionalidades a través de módulos adicionales.
Ejemplo: Sistemas operativos como QNX, servicios empresariales modulares.

Arquitectura de Servicios Distribuidos con Blockchain: Utiliza blockchain para gestionar y verificar transacciones distribuidas.
Ejemplo: Plataformas de contratos inteligentes como Ethereum.

Arquitectura de Servicios Distribuidos de IoT (Internet of Things) : Red de dispositivos inteligentes que se comunican entre sí y con sistemas backend.
Ejemplo: Sistemas de domótica, redes de sensores en ciudades inteligentes.

Arquitecturas SOA

Arquitectura de Computación Distribuida: Redes de computadoras que trabajan juntas para resolver problemas complejos.
Ejemplo: Proyectos de computación distribuida como SETI@home, Folding@home.

Arquitectura de Servicios Distribuidos de Juegos en Línea: Plataformas que gestionan la interacción en tiempo real de múltiples jugadores.
Ejemplo: Arquitectura de servidores de juegos como Fortnite, World of Warcraft.

Arquitectura de Servicios Distribuidos en Redes Sociales: Plataformas que soportan la interacción de millones de usuarios simultáneamente.
Ejemplo: Facebook, LinkedIn.

¿Qué es un microservicio?

- Descomposición Modular:
  - La aplicación se divide en pequeños servicios, cada uno responsable de una única funcionalidad o dominio de negocio.
  - Esta modularidad facilita el desarrollo, mantenimiento y escalabilidad.
- Desacoplamiento:
  - Los microservicios son independientes entre sí, lo que significa que los cambios en uno no afectan directamente a los demás.
  - Permite a los equipos trabajar de manera autónoma en diferentes servicios.

¿Qué es un microservicio?

- Intercomunicación mediante APIs:
  - Los microservicios se comunican entre sí a través de interfaces estándar, como APIs RESTful o protocolos de mensajería (como AMQP).
  - La comunicación suele ser a través de HTTP/HTTPS, gRPC o mensajes asíncronos.
- Implementación Independiente:
  - Cada microservicio puede ser implementado, desplegado y escalado de manera independiente.
  - Permite la utilización de diferentes lenguajes de programación y tecnologías según la necesidad de cada servicio.

¿Qué es un microservicio?

- Escalabilidad:
  - Los microservicios pueden escalarse individualmente, lo que permite una utilización más eficiente de los recursos.
  - Se puede aumentar la capacidad de solo aquellos servicios que lo necesitan.
- Fallo Aislado:
  - Un fallo en un microservicio no necesariamente causa la caída de toda la aplicación.
  - Mejora la resiliencia y la capacidad de recuperación del sistema.

¿Qué es un microservicio?

- Ciclo de Vida Independiente:
  - Los microservicios pueden tener sus propios ciclos de desarrollo, pruebas y despliegue.
  - Esto permite una entrega continua y rápida de nuevas funcionalidades.
- Tecnologías Heterogéneas:
  - Diferentes microservicios pueden ser construidos usando distintas tecnologías y frameworks que sean más adecuados para cada caso.
  - Facilita la utilización de las mejores herramientas y prácticas disponibles.

¿Qué es SOAP?

- SOAP (Simple Object Access Protocol) es un protocolo de comunicación diseñado para permitir la interoperabilidad entre aplicaciones a través de la red.

- Un sistema basado en SOAP tiene varias características distintivas que lo definen y lo hacen adecuado para ciertas aplicaciones empresariales y de servicios web.

¿Qué es SOAP?

- Protocolo Basado en XML:
  - Formato de Mensaje: SOAP utiliza XML para estructurar los mensajes, lo que lo hace independiente del lenguaje de programación y la plataforma.
  - Estándar de Intercambio: XML permite que los datos sean fácilmente intercambiables y entendidos por diferentes sistemas.

¿Qué es SOAP?

- Estructura de Mensaje Bien Definida:

  - Envoltorio (Envelope): La estructura básica de un mensaje SOAP, que contiene un encabezado opcional y un cuerpo obligatorio.
  - Encabezado (Header): Utilizado para especificar información de control y metadatos (por ejemplo, seguridad, transacciones).
  - Cuerpo (Body): Contiene el mensaje de la solicitud o la respuesta, donde se encuentran los datos principales.
  - El<Fault>elemento, contenido en<Body>, se utiliza para notificar errores.

¿Qué es SOAP?

- El sobre SOAP
  - <Envelope>es el elemento raíz en cada mensaje SOAP y contiene dos elementos hijo, un valor opcional<Header>y un elemento obligatorio<Body>.

- La cabecera SOAP
  - <Header>es un subelemento opcional del sobre SOAP y se utiliza para pasar información relacionada con la aplicación que debe procesar los nodos SOAP a lo largo de la vía de acceso del mensaje; consulte La cabecera SOAP.

¿Qué es SOAP?

- El cuerpo SOAP
  - <Body>es un subelemento obligatorio del sobre SOAP, que contiene información destinada al destinatario final del mensaje; consulte El cuerpo SOAP.

- El error SOAP
  - <Fault>es un subelemento del cuerpo SOAP, que se utiliza para notificar errores; consulte El error SOAP.

- Elementos XML en<Header>y<Body>son definidas por las aplicaciones que hacen uso de ellas, aunque la especificación SOAP impone algunas restricciones en su estructura. En el siguiente diagrama se muestra la estructura de un mensaje SOAP.

¿Qué es SOAP?

```
<?xml version='1.0' Encoding='UTF-8' ?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
 <env:Header>
  <m:reservation xmlns:m="http://travelcompany.example.org/reservation"
                                 env:role="http://www.w3.org/2003/05/soap-envelope/role/next">
   <m:reference>uuid:093a2da1-q345-739r-ba5d-pqff98fe8j7d</m:reference>
   <m:dateAndTime>2007-11-29T13:20:00.000-05:00</m:dateAndTime>
  </m:reservation>
  <n:passenger xmlns:n="http://mycompany.example.com/employees"
                                 env:role="http://www.w3.org/2003/05/soap-envelope/role/next">
   <n:name>Fred Bloggs</n:name>
  </n:passenger>
 </env:Header>
 <env:Body>
  <p:itinerary xmlns:p="http://travelcompany.example.org/reservation/travel">
   <p:departure>
     <p:departing>New York</p:departing>
     <p:arriving>Los Angeles</p:arriving>
     <p:departureDate>2007-12-14</p:departureDate>
     <p:departureTime>late afternoon</p:departureTime>
     <p:seatPreference>aisle</p:seatPreference>
   </p:departure>
   <p:return>
     <p:departing>Los Angeles</p:departing>
     <p:arriving>New York</p:arriving>
     <p:departureDate>2007-12-20</p:departureDate>
     <p:departureTime>mid-morning</p:departureTime>
     <p:seatPreference></p:seatPreference>
   </p:return>
  </p:itinerary>
 </env:Body>
</env:Envelope>
```

¿Qué es SOAP?

- Transporte Independiente:
  - Flexibilidad de Transporte: Aunque comúnmente se utiliza sobre HTTP/HTTPS, SOAP puede funcionar sobre otros protocolos de transporte como SMTP, FTP o JMS.
  - Uso de HTTP/HTTPS: La capacidad de funcionar sobre HTTP/HTTPS permite la integración con infraestructuras web existentes y facilita el paso a través de firewalls.

¿Qué es SOAP?

- Extensibilidad:
  - Capacidad de Extensión: SOAP permite la extensión mediante el uso de encabezados personalizados que pueden incluir funcionalidades adicionales como seguridad, transacciones, etc.
  - Modularidad: Los encabezados y cuerpos pueden ser extendidos sin romper la estructura básica del mensaje.

¿Qué es SOAP?

- Seguridad:
  - WS-Security: SOAP soporta estándares de seguridad como WS-Security para proporcionar integridad, confidencialidad y autenticación en los mensajes.
  - Control de Acceso: Se puede integrar con tecnologías de autenticación y autorización para controlar el acceso a los servicios.

- Fiabilidad:
  - Mensajería Fiable: SOAP puede trabajar con protocolos como WS-ReliableMessaging para garantizar la entrega de mensajes en entornos de red no confiables.
  - Transacciones Distribuidas: Soporte para WS-AtomicTransaction para asegurar la consistencia en transacciones distribuidas..

¿Qué es SOAP?

- Desacoplamiento:
  - Desacoplamiento de Clientes y Servicios: Los clientes no necesitan conocer la implementación del servicio, solo la interfaz que expone.
  - Interoperabilidad: Permite que aplicaciones escritas en diferentes lenguajes y ejecutadas en diferentes plataformas se comuniquen de manera efectiva.

- WSDL (Web Services Description Language):
  - Descripción de Servicios: SOAP suele utilizar WSDL para describir la interfaz del servicio web (métodos disponibles, tipos de datos, protocolos de comunicación).
  - Generación Automática de Clientes: WSDL permite la generación automática de clientes que pueden interactuar con el servicio SOAP.

¿Qué es SOAP?

- Mensajería Sincrónica y Asincrónica:
  - Soporte para Ambos Modelos: SOAP permite tanto la comunicación sincrónica (donde el cliente espera una respuesta inmediata) como asincrónica (donde el cliente puede recibir la respuesta en un momento posterior).

- Soporte para Servicios Complejos:
  - Transacciones y Procesos de Negocio: SOAP es adecuado para servicios que requieren transacciones distribuidas y procesos de negocio complejos.
  - Orquestación de Servicios: Compatible con estándares de orquestación como BPEL (Business Process Execution Language) para coordinar múltiples servicios.

¿Qué es RESTful?

- RESTful se refiere a los servicios web que siguen los principios del estilo arquitectónico REST (Representational State Transfer). REST es una forma de diseñar sistemas de software para comunicarse a través de HTTP, y cuando un servicio web implementa estas ideas, se le llama RESTful.

¿Qué es RESTful?

- Uso de HTTP y URL:
  - Protocolos Estándar: RESTful utiliza HTTP/HTTPS para la comunicación entre el cliente y el servidor.
  - Recursos Identificados por URLs: Cada recurso se identifica mediante una URL (Uniform Resource Locator), que actúa como un identificador único.

- Operaciones CRUD a través de Métodos HTTP:

  - GET: Recupera representaciones de un recurso.
  - POST: Crea un nuevo recurso.
  - PUT: Actualiza un recurso existente.
  - DELETE: Elimina un recurso.

¿Qué es RESTful?

- PUT
  - Propósito: El método PUT se utiliza para actualizar un recurso completo. En otras palabras, envías la representación completa del recurso que deseas actualizar.
  - Idempotencia: PUT es idempotente, lo que significa que múltiples solicitudes PUT con la misma carga útil al mismo recurso siempre tendrán el mismo efecto que una única solicitud.
  - Reemplazo Completo: Cuando usas PUT, estás reemplazando todo el recurso en el servidor con la representación que envías en la solicitud. Si algún campo no se incluye en la solicitud, se eliminará del recurso en el servidor.

¿Qué es RESTful?

- PATCH
  - Propósito: El método PATCH se utiliza para realizar actualizaciones parciales en un recurso. En otras palabras, envías solo los cambios que deseas aplicar al recurso.
  - Idempotencia: PATCH no es necesariamente idempotente. La misma solicitud PATCH podría tener efectos diferentes si se realiza varias veces, dependiendo del estado inicial del recurso. Una operación es idempotente si, cuando se aplica varias veces, el resultado es el mismo que cuando se aplica una sola vez.
  - Actualización Parcial: PATCH aplica solo los cambios especificados sin alterar otros campos del recurso que no se mencionan en la solicitud.

¿Qué es RESTful?

- Interacción Sin Estado (Stateless):
  - Sin Estado: Cada solicitud del cliente al servidor debe contener toda la información necesaria para entender y procesar la solicitud. No se mantiene estado del cliente en el servidor entre peticiones.
  - Escalabilidad: Este enfoque simplifica la escalabilidad del sistema, ya que no es necesario mantener sesiones del lado del servidor.

¿Qué es RESTful?

- Representaciones de Recursos:
  - Formatos de Datos: Los recursos pueden representarse en varios formatos, como JSON (JavaScript Object Notation), XML (eXtensible Markup Language), HTML, o texto plano.
  - Negociación de Contenidos: Los clientes pueden solicitar representaciones específicas del recurso a través del encabezado HTTP "Accept".

¿Qué es RESTful?

- HATEOAS (Hypermedia as the Engine of Application State):
  - Hipermedios como Motor del Estado de la Aplicación: Los clientes interactúan con la aplicación enteramente a través de hipermedios proporcionados de manera dinámica por las aplicaciones. Esto significa que las respuestas de los servidores RESTful contienen enlaces a otros recursos o acciones posibles.
  - Descubrimiento de Recursos: Los clientes pueden descubrir dinámicamente las acciones que pueden realizar en los recursos.

¿Qué es RESTful?

- Caché:
  - Capacidad de Caché: Las respuestas pueden ser almacenadas en caché para mejorar el rendimiento. Las respuestas deben indicar su capacidad de ser almacenadas en caché mediante encabezados HTTP.
  - Eficiencia: La utilización de cachés reduce la carga en el servidor y mejora la latencia percibida por el usuario..

¿Qué es RESTful?

- Uniformidad de Interfaz:
  - Interfaz Uniforme: La interfaz uniforme es un principio fundamental de REST que simplifica la arquitectura al restringir las operaciones a un conjunto estándar de operaciones, como las proporcionadas por HTTP.
  - Previsibilidad: La uniformidad de la interfaz mejora la previsibilidad y la comprensión del sistema por parte de los desarrolladores.

- Escalabilidad y Desempeño:
  - Descentralización: La interacción sin estado y la caché facilitan la distribución de la carga y la escalabilidad del sistema.
  - Rendimiento: La simplicidad de las interacciones basadas en HTTP contribuye a un buen rendimiento del sistema.

¿Qué es RESTful?

- Ejemplos y Aplicaciones:
    - APIs Públicas: Muchas APIs de servicios en la nube, redes sociales, y aplicaciones móviles utilizan RESTful, como la API de Twitter, la API de Google Maps, y la API de GitHub.
    - Aplicaciones Móviles y Web: Los servicios RESTful son comúnmente utilizados para la comunicación entre aplicaciones móviles y sus servidores backend.
    - Sistemas de Microservicios: Los microservicios a menudo utilizan RESTful para la comunicación entre servicios debido a su simplicidad y eficiencia

¿Qué es RESTful?

- RESTful es un enfoque para construir servicios web que enfatiza la simplicidad, la escalabilidad y el uso de estándares abiertos como HTTP y URL.

- Los servicios RESTful permiten una fácil comunicación entre sistemas mediante el uso de operaciones HTTP y representaciones de recursos en formatos estándar.

- Este enfoque es ampliamente adoptado debido a su flexibilidad y capacidad para integrarse con una variedad de aplicaciones y tecnologías.

¿Qué es RESTful?

- modelo de madurez de Richardson
  - describe los niveles por los que pasa una especificación de API REST desde que es creada hasta que se perfecciona adquiriendo controles hipermedia.

¿Qué es RESTful?

- Nivel 0. Los servicios cuentan con una sola URI que acepta todo el rango de operaciones admitidas por el servicio, con unos recursos poco definidos. No se considera una API RESTful.

- Nivel 1. Introduce recursos y permite hacer peticiones a URIs individuales para acciones separadas en lugar de exponer un punto de acceso universal. Los recursos siguen estando generalizados pero es posible identificar un ámbito algo más concreto. El primer nivel sigue sin ser RESTful, pero está más orientado a adquirir la capacidad de serlo.

¿Qué es RESTful?

- Nivel 2. El sistema empieza a hacer uso de los verbos HTTP. Esto permite mayor especialización y generalmente conlleva la división de los recursos en dos:
- Uno para obtener únicamente datos (GET)
- Otro para modificarlos (POST),
- Aunque un grado mayor de granularidad también es posible. Una de las desventajas de proveer un sistema distribuido con más de una petición GET y POST por recurso puede ser el aumento de complejidad del sistema, a pesar de que el consumo de datos por clientes de la API se simplifica en gran medida.

¿Qué es RESTful?

- Nivel 3. El último nivel introduce la representación hipermedia, también llamada HATEOAS. Esta representación se realiza mediante elementos incrustados en los mensajes de respuesta de los recursos, que permiten al cliente que envía la petición establecer una relación entre entidades de datos individuales. Por ejemplo, una petición GET a un sistema de reservas de un hotel, podría devolver el número de habitaciones disponibles junto con los enlaces que permiten reservar habitaciones específicas.

SOAP	REST
1. Mas orientada a servicios	1. Arquitectura centrada en datos (recursos)
2. Es un protocolo. Más restringido y más estandarizado.	2. Envían información en HTML, XML o JSON
3. Utiliza XML normalmente	3. Necesitan menos ancho de banda
4. Fuerte tipado	4. deben poder ser probados mediante un navegador para poder ser RESTful
5. Lenguaje más verboso, necesita más ancho de banda.	5. se basa en el principio de lectura como operación más frecuente, por lo que la mayor parte de las peticiones serán de tipo GET
6. más construido con peticiones POST. Debido a esto último y a unos recursos inespecíficos, SOAP puede conllevar el desarrollo de clientes más complejos que REST.	

Seguridad de datos en SOA

- Protección de Datos en Tránsito

- Datos en Tránsito: Son los datos que se transmiten entre sistemas o entre diferentes componentes del sistema. La protección de estos datos es crucial para evitar intercepciones y accesos no autorizados.

- Cifrado en Tránsito: Se utiliza para asegurar que los datos no puedan ser leídos por terceros durante la transmisión. Esto se logra mediante el uso de protocolos de cifrado como TLS (Transport Layer Security).

Seguridad de datos en SOA

- Protección de Datos en Reposo

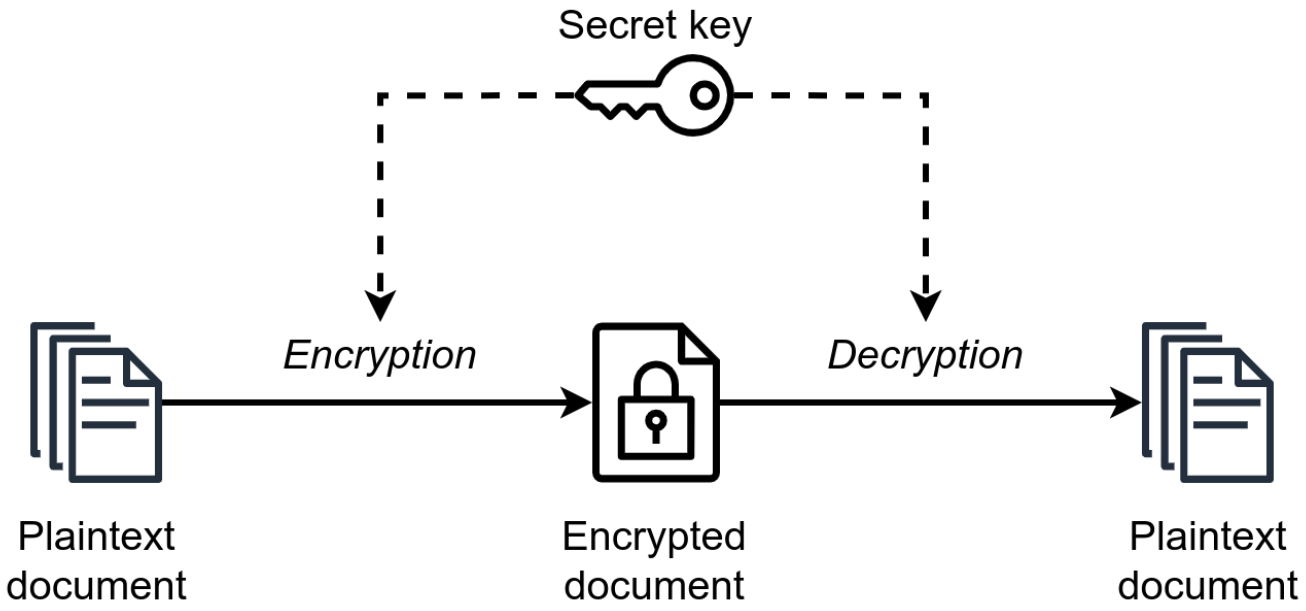
- Datos en Reposo: Son los datos que se almacenan en discos duros, bases de datos o cualquier otro medio de almacenamiento.

- Cifrado en Reposo: Protege los datos almacenados de accesos no autorizados. Herramientas y técnicas comunes incluyen cifrado a nivel de base de datos, cifrado de archivos y discos.

Seguridad de datos en SOA

- Cifrado Simétrico:

- Utiliza la misma clave para cifrar y descifrar los datos. Es rápido y eficiente para grandes volúmenes de datos. Ejemplos: AES (Advanced Encryption Standard), DES (Data Encryption Standard).



Seguridad de datos en SOA

•Cifrados de Bloque: cifran el mensaje dividiendo el flujo en bloques de k bits. Cada bloque se corresponde con otro diferente. Por ejemplo, un bloque con k=3 "010" se podría corresponder con "110". Se utilizan algoritmos como DES, TDES, RC5, RC6, CAST, Serpent, IDEA, Kasumi, Blowfish, Camellia y AES (Rijndael). Estos algoritmos trabajan con bloques de 64 o 128 bits.

Seguridad de datos en SOA

•Cifrados de flujo: se utilizan en aplicaciones donde la velocidad del flujo de datos es variable o en tiempo real, como telefonía o WLAN. Se genera un "flujo de claves" a partir de una semilla aleatoria y se combina bit a bit con la información en claro. Algunos ejemplos son RC4, Trivium, SEAL, WAKE, VEST, SNOW y Rabbit.

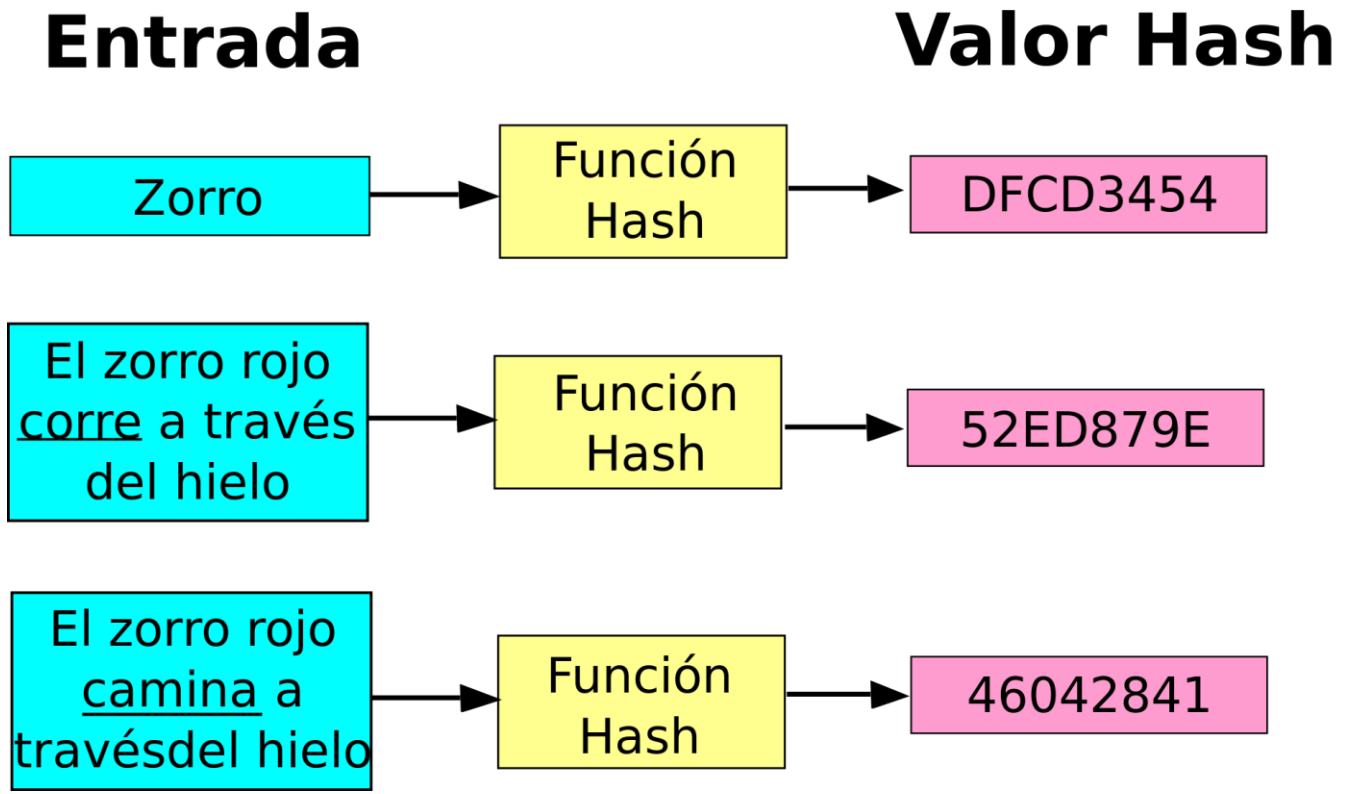
Seguridad de datos en SOA

•Cifrado simétrico de resumen (*hash functions*): se utilizan para garantizar la integridad de los datos. Algunos ejemplos son los algoritmos SHA o Whirlpool.

Una función resumen, hash function ó digest function, también conocida con los híbridos función hash o función de hash, convierte uno o varios elementos de entrada a una función en otro elemento.

La función *hash* tiene como entrada un conjunto de elementos, que suelen ser cadenas, y los convierte en un rango de salida finito, normalmente cadenas de longitud fija. Es decir, la función actúa como una proyección del conjunto U sobre el conjunto M

Seguridad de datos en SOA



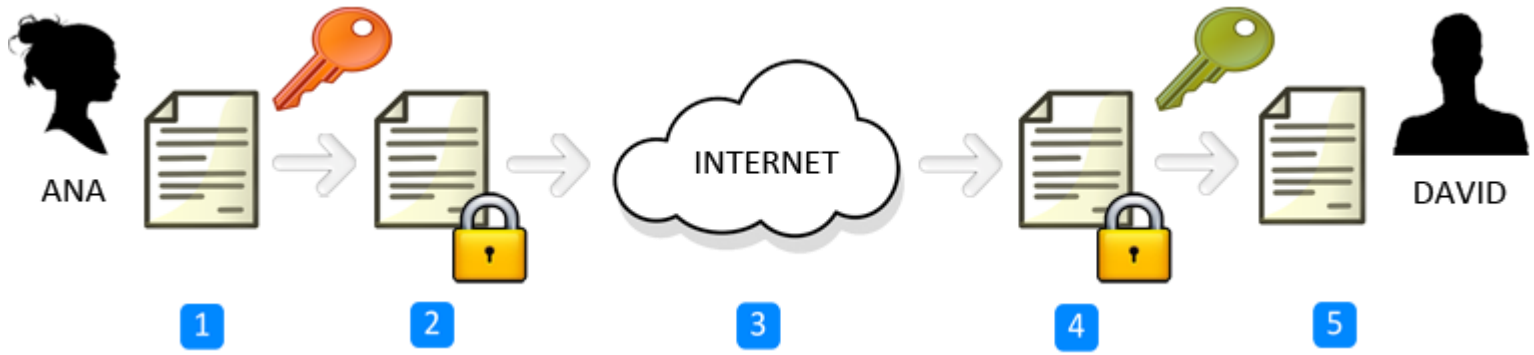
Seguridad de datos en SOA

- Cifrado Asimétrico:

- Utiliza un par de claves (pública y privada). La clave pública cifra los datos y la clave privada los descifra. Es más seguro pero menos eficiente. Ejemplos: RSA, ECC (Elliptic Curve Cryptography).

Seguridad de datos en SOA

Ejemplo de cifrado de mensaje: *Ana envía un mensaje a David*



1. Ana redacta un mensaje.
2. Ana cifra el mensaje con la clave pública de David.
3. Ana envía el mensaje cifrado a David a través de internet, ya sea por correo electrónico, mensajería instantánea o cualquier otro medio.
4. David recibe el mensaje cifrado y lo descifra con su clave privada.
5. David ya puede leer el mensaje original que le mandó Ana

Seguridad de datos en SOA

- Integridad y Autenticidad de Mensajes:

- Integridad de Mensajes: Garantiza que los mensajes no han sido alterados durante el tránsito. Esto se logra mediante el uso de funciones hash.

  - HMAC (Hash-Based Message Authentication Code): Combina una función hash con una clave secreta para verificar tanto la integridad como la autenticidad de un mensaje.

- Autenticidad de Mensajes: Garantiza que el mensaje proviene de una fuente legítima. Esto se puede lograr mediante el uso de firmas digitales.

Seguridad de datos en SOA

- Firmas Digitales y Técnicas de Cifrado:

- Firmas Digitales: Utilizan cifrado asimétrico para firmar un mensaje. La clave privada del remitente firma el mensaje y cualquier persona con la clave pública correspondiente puede verificar la firma.
  - Proceso: El mensaje se hashea, el hash se cifra con la clave privada del remitente, y el hash cifrado se adjunta al mensaje como la firma digital.

- Técnicas de Cifrado Comunes: Se utilizan para proteger el contenido del mensaje durante la transmisión.
  - Ejemplos: TLS para cifrado en tránsito, AES para cifrado de mensajes.

Seguridad de datos en SOA

Ejemplo de firma digital con clave asimétrica: *David envía un mensaje a Ana*

1. David redacta un mensaje.
2. David firma digitalmente el mensaje con su clave privada.
3. David envía el mensaje firmado digitalmente a Ana a través de internet, ya sea por correo, mensajería instantánea o cualquier otro medio.
4. Ana recibe el mensaje firmado digitalmente y comprueba su autenticidad usando la clave pública de David.
5. Ana ya puede leer el mensaje con total seguridad de que ha sido David el remitente.

Seguridad de datos en SOA – Control de accesos

Modelo de Control de Acceso Basado en Roles (RBAC):

- RBAC asigna permisos a roles en lugar de a usuarios individuales. Los usuarios se asignan a roles y adquieren los permisos de esos roles.
- Roles: Grupos predefinidos de permisos que corresponden a funciones específicas dentro de una organización.
- Permisos: Acciones autorizadas que los roles pueden realizar sobre recursos.

Seguridad de datos en SOA – Protocolos seguros

TLS/SSL y su Implementación:

- TLS (Transport Layer Security): Protocolo de seguridad que proporciona comunicaciones seguras a través de una red. Es la versión mejorada de SSL (Secure Sockets Layer).

- Implementación: Involucra la configuración de certificados digitales emitidos por una autoridad de certificación (CA), y la configuración de servidores para aceptar conexiones seguras.

- Proceso de Handshake: Incluye autenticación de servidores y, opcionalmente, de clientes, y establece una sesión cifrada mediante la negociación de claves.

Seguridad de datos en SOA – Protocolos seguros

Protocolos Seguros en Servicios Web:

- HTTPS (HTTP Secure): HTTP sobre TLS. Garantiza que la comunicación entre el navegador web y el servidor web es segura.
- WS-Security: Protocolo de extensión SOAP que permite asegurar mensajes SOAP mediante integridad y confidencialidad.
- OAuth: Protocolo de autorización que permite a los usuarios otorgar acceso limitado a sus recursos sin compartir sus credenciales.
- JWT (JSON Web Token): Utilizado para la autenticación y el intercambio seguro de información entre partes.

Seguridad de datos en SOA – JWT

JWT (JSON Web Token) es un estándar abierto para la creación de tokens de acceso que permiten la transferencia segura de información entre dos partes. Este estándar es ampliamente utilizado en la autenticación y autorización de aplicaciones web y servicios.

Estructura de un JWT: Un JWT consta de tres partes codificadas en Base64 y separadas por puntos (.):

Seguridad de datos en SOA – JWT

Header (Encabezado):

Algoritmo de Firma (alg): Especifica el algoritmo utilizado para firmar el token, como HMAC SHA256 o RSA.
Tipo de Token (typ): Indica que se trata de un JWT

Seguridad de datos en SOA – JWT

Payload (Carga Útil):

- Claims (Declaraciones): Contiene las declaraciones o datos sobre la entidad (generalmente el usuario) y metadatos adicionales. Hay tres tipos de claims:

- Registered Claims: Claims estándar predefinidas por el estándar JWT, como iss (emisor), exp (fecha de expiración), aud (audiencia).

- Public Claims: Claims definidas libremente y que pueden ser utilizadas por cualquiera. Se recomienda evitar colisiones utilizando namespaces.

- Private Claims: Claims personalizadas y específicas de la aplicación.

Seguridad de datos en SOA – JWT

Signature (Firma):

- Proceso de Firma: Se crea combinando el encabezado y la carga útil, luego cifrando este valor con una clave secreta utilizando el algoritmo especificado en el encabezado.

- Propósito: La firma verifica que el mensaje no ha sido alterado y garantiza la autenticidad del emisor.

Seguridad de datos en SOA – JWT

Autenticación:

- Proceso de Inicio de Sesión: El usuario envía sus credenciales al servidor de autenticación. Si las credenciales son válidas, el servidor crea un JWT y lo envía de vuelta al cliente.

- Uso del Token: El cliente almacena el JWT (generalmente en almacenamiento local o cookies) y lo envía con cada solicitud subsecuente en el encabezado HTTP Authorization: Bearer <token>.

- Verificación del Token: En cada solicitud protegida, el servidor verifica el JWT para asegurarse de que es válido y no ha expirado antes de permitir el acceso a los recursos.

Seguridad de datos en SOA – JWT

Autorización:

- Claims: El servidor puede usar las claims en el JWT para determinar los permisos del usuario y limitar el acceso a ciertos recursos o acciones.

VENTAJAS

1. *Compacto y Eficiente: Los tokens están en formato JSON y son compactos, lo que permite transferencias rápidas y eficaces.*
2. *Autocontenido: Los tokens contienen toda la información necesaria sobre el usuario y sus permisos, eliminando la necesidad de acceder a la base de datos en cada solicitud.*
3. *Interoperabilidad: Basado en estándares abiertos, JWT se puede utilizar en cualquier lenguaje de programación*

DESAFIOS

1. *Seguridad de la Clave Secreta: La clave utilizada para firmar el JWT debe mantenerse segura y no ser expuesta.*
2. *Expiración del Token: Es crucial manejar (exp) para minimizar riesgos de uso indebido si un token es comprometido.*
3. *Cliente debe almacenar de forma segura el JWT*
4. *Invalidación de Tokens: JWT no tiene un mecanismo de invalidación incorporado, problemático si se necesita revocar el acceso antes de que el token expire.*