Servicios Web, AJAX y Rich in Internet Aplication

Implementación del protocolo HTTP para que el servidor devuelva otro tipo de contenido, el cual estaría destinado a ser consumido por un programa que tome el papel del cliente; no necesariamente un navegador.

Servicios Web: es un servicio ofrecido por un dispositivo a otros dispositivos a traves de la web, están diseñados para comunicación maquina a maquina mediante HTTP.

Ventajas

- ***Interoperabilidad*** entre distintos componentes, independientes del lenguaje de programación en el que estén implementados y le la maquina en la que se ejecuten.

Pila de Protocolos de Servicios Web

- **Protocolo de Transporte**
  Determina el mecanismo de envio de mensajes.Normalmente HTTP.

- **Protocolo de Mensajería**
  Determina como se codifican los mensajes; XML (SOAP) o JSON.

- **Protocolo de Descripción**
  Describe la interfaz de un servicio (operaciones que realiza y parámetros), ejemplo WSDL.

- **Protocolo de Descubrimiento**
  Permite que un servicio publique su URL y descripción en un registro público comun.

Tipos de Servicios Web

- **REST (REpresentational State Transfer)**
  Utilizan las peticiones HTTP para especificar las operaciones a realizar. las operaciones se realizan sobre recursos, cada uno identificado mediante un URL.

- **SOAP (Simple Object Access Protocol)**
  Utilizan mensajes en formato XML para especificar el tipo de operación que se desea realizar y sus parámetros.

- **Formato XML y JSON**

  El servidor responde al cliente con la información que este ha pedido, por lo tanto debe haber un formato en el que se transmite la información que se le entrega al cliente .

  - **Requisitos de un Formato de intercambio de Datos**
    - **Formato XML**

  Sintaxis similar a la de HTML , con ciertas restricciones

  1. ***Toda etiqueta de inicio tiene etiqueta de cierre, estas deben estar correctamente anidadas.***
  2. ***Los atributos de etiquetas están delimitados por comillas dobles.***
  3. ***Existe un único elemento de raíz.***

  El conjunto de etiquetas XML permitidas y su contenido depende de la aplicación concreta.

  - **Utilizar JSON en JavaScript**
    - ***Convierte un texto en formato JSON en un objeto Javascript*: JSON.parse(texto)**
    - ***Obtiene la representación JSON de un objeto Javascript*: JSON.stringify(valor)**

    - **Formato JSON**

  Sintaxis similar a los objetos de Javascript exceptuando que los nombre de los atributos van siempre delimitados por comillas.
  #Array []... Lista de Objetos {}

- **Servicios REST**

Utilizan la semantica de los métodos HTTP para indicar la accion sobre un determinado recurso, identificado por una URL

  - **Principios Basicos**
    1. ***Todo es un recurso.***
    2. ***Los recursos están identificados por URLs.***
    3. ***Se utilizan verbos HTTP estandar .***
    4. ***Un recurso puede representarse de múltiples formas .***
    5. ***La comunicación con los servicios web no tiene estado.***

  - **Recursos y URLs**

  Las URLs no hacen referencia necesariamente a un fichero físico en el servidor, sino a un recurso *abstracto.*

- **Metodos HTTP**

  Suelen utilizarse las acciones CRUD habituales (Create, Read, Update, Delete)

Metodo HTTP	**Semántica**
GET	*Leer un recurso*
POST	*Crear/Anadir un nuevo recurso*
PUT	*Actualizar un recurso existente*
DELETE	*Eliminar un recurso*

La descripción semántica de estos recursos es orientativa y hay algunos métodos imponen de manera implícita ciertas propiedades:

- ➢ GET no altera el estado del recurso al que hace referencia.
- ➢ GET, PUT y DELETE son idempotentes. Realizar una misma operación varias veces seguidas es equivalente a realizar una única vez.

- **Codigo de Respuesta**

Metodo	**Codigo de Respuesta**
GET	**200 OK** **404 Not Found** **500 Internal Server Error**
POST	**201 Created** **500 Internal Server Error**
PUT	**200 OK** **500 Internal Server Error**
DELETE	**200 OK** **204 No Content** **404 Not Found** **500 Internal Server Error**

- **Representaciones de un Recurso**

  Un recurso puede ser representado de distintas formas (JSON, XML)

  -Las peticiones GET el cliente especifica en la cabecera el tipo de representación que desea recibir

  -Las peticiones PUT/ DELETE el cliente especifica en la cabecera de la petición la representación del nuevo objeto

- **Ausencia de Estado**

  En este tipo de aplicaciones se respeta la carencia de estado propia del protocolo HTTP.

#NOTA:No debe hacerse uso de los mecanismos para mantener variables de estado que perviven entre peticiones

- **Implementación de Servicios Web REST con ExpressJS**

  Se pueden utilizar el framework ExpressJS para implementar servicios web de tipo REST, esto también implica que tambien es posible añadir middleware en cada petición

  - ***app.get(url, funcion)***
  - ***app.post(url, funcion)***
  - ***app.put(url, funcion)***
  - ***app.delete(url, funcion)***

  ➢ ***Obtener un Objeto***
    - **Recibe un objeto que transforma en JSON: *response.json(<objeto>)*** es una operación terminal

  ➢ ***Obtener un Objeto a Partir de un índice***
    - **URLs Parametricas: *request.params***
    - **query strings: *request.query***
    - **Cuerpo de Petición(PUT Y POST): *body-parser***

  ➢ ***Añadir un Objeto***
    - **El objeto JSON debe ir en el cuerpo de la misma: *app.use(bodyParser.json());***

  ➢ ***Eliminar un Objeto***
    - ***La url debe tener el índice del elemento a eliminar***

  ➢ ***Actualizar un Objeto***
    - ***La url debe tener el indice a actualizar***
    - ***Valor nuevo de los atributos en formato JSON***

- **Comprobar el Funcionamiento de un Servicio Web**

  Existen varias herramientas que permiten al usuario realizar cualquier tipo de peticiones HTTP a un servidor.

- **Envío de Ficheros Binarios**

En las peticiones PUT y POST que se quieren adjuntar un fichero binario el formato JSON resulta inadecuado. En este caso es mejor adjuntar a la petición un cuerpo con **multipatf/form-data** utilizando **multer**

- **Peticiones AJAX**

Existe otro mecanismo para realizar peticiones HTTP mediante JavaScript en el navegador **AJAX(Asynchronous Javascript and XML)** . En nuestro caso realizaremos las peticiones AJAX mediante la librería jQuery, que abstrae todas estas diferencias en una función llamada:

$.ajax(opciones)

opciones es un objeto que especifica:

1. **method:** Metodo HTTP.
2. **url:** URL sobre la que realiza la petición.
3. **data:** datos a adjuntar en la petición*(query string o body)*
4. **contentType:** tipo de cuerpo de la petición **application/x-www-form-urlencoded**
5. Funciones callback cuando la peticion ha tenido éxito o ha fallado.
    a. **success:** el servidor responde la petición de éxito (2xx)
        i. **(data, statusText, jqXHR)=>{}**
            1. **data:** Datos contenidos en el cuerpo de la respuesta HTTP(*JSON->JavaScript).*
            2. **statusText:** Cadena que describe el estado de la petición "*success".*
            3. **jqXHR:** Objeto con más información sobre la respuesta.
    b. **error:**
        i. **(jqXHR,statusText,errorThrown)=>{}**
            1. **jqXHR:** XMLHTTPRequest
            2. **statusText:** ("timeout","parseerror", "abort")
            3. **errorThrown:** ("Not Found","Internal Server Error")

#NOTA: Cuando se pasa un objeto Javascript a la opcion data, este se transforma en una query string.

#NOTA2: Si el valor viene dado en el cuerpo de la petición en forma de JSON solo son peticiones de POST y debemos utilizar JSON.stringify() del lado del cliente he indicar en el contentType: **application/json**

- **Autenticación Basica**

Es usual y es necesario restringir el uso de un servicio web a un determinado conjunto de usuarios, la solución a ello es mediante un sistema de autenticación.

El protocolo HTTP incorpora mecanismos de control de acceso autenticado

Esquema de Autenticacion HTTP

- **En el servidor** debemos reconocer y procesar la cabecera Authorization de las Peticiones HTTP.

a. **Instalar el paquete de autenticación:**
   i. **npm install --save passport**
   ii. **npm install --save passport-http**
b. **Importamos los módulos**
   i. **let passport =require("passport");**
   ii. **let passportHTTP =require("passport-http");**
c. **Cadena de middlewares de autenticacion app.use(passport.initialize());**
d. **Definir estrategia de autenticación**
   i. **passport.use(new passportHTTP.BasicStrategy({realm:"Página Protegida"}, funcion);)**

La función de autenticación que debemos implementar comprobará que los datos son correctos

-error: callback(error)
-Datos incorrectos: callback(null,false)
-Datos correctos: callback(null, user)
#NOTA los datos de autenticacion se guardaran en **request.user**

e. **Middleware de comprobación de usuario passport.authenticate('basic', {session:false})** ***{session:false} indica que la autenticacion no involucra ni cookies ni sesiones***

- **En el cliente** debemos incluir la cabecera Authorization en las peticiones AJAX.
  a. Anadir a la cabecera= btoa(user+":"+pass);
  b. beforSend: (req)=>{ req.setRequestHeader("Authorization"+, "Basic"+ cadena);}

- **Introducción a SOAP**

Los servicios SOAP no esta basados en el acceso a recursos mediante el uso de metodos HTTP. **Esencialmente hacen uso de peticiones POST y GET utilizando XML** como mecanismo de comunicación; para **cada servicio** existe un fichero **(WSDL**) que especifica el catalogo de operaciones ofrecidas por el servicio.

<u>Acceso a Servicios SOAP desde JAVA</u>

Java dispone de una herramienta que genera clases de acceso a servicios web a partir de un fichero WSDL.

NetBeans: File /New File/ Web Service /Web Service Client

genera una clase llamada **Periodictable** que proporciona acceso a los servicios

- **El protocolo HTTPS**

El protocolo HTTP es un protocolo inseguro, ya que toda la información que viaja a traves de la web viaja en forma de texto, por lo que es propenso a ***ataques intermediarios*** .

El atacante intercepta los mensajes entre el cliente y el servidor, pudiendo acceder a su contenido y manipularlos. Ni el cliente ni el servidor son concientes de la existencia del intermediario.

El **protocolo HTTPS** pretende remediar esos problemas añadiendo **encriptación en los mensajes enviados** . Utiliza una tecnica **criptografía asimétrica**, en los que cada parte utiliza dos claves para envio de mensajes

- **Clave Privada:** sólo conocida por el propietario
- **Clave Pública:** conocida por todos

"Cualquier mensaje encriptado con la clave pública solo puede ser encriptado por la clave privada y viceversa"

1. **Confidencialidad de Envío**

Este esquema puede garantizar la confidencialidad de envio entre las dos partes ya que la ***clave pública*** tiene ***rol de candado*** y la ***clave privada*** tiene rol de ***llave.***

Proceso de Envio de Informacion entre Usuarios

Si una persona A quiere enviar información a una persona B requiere de pública de B para encriptar el mensaje. Luego, para leer un mensaje encriptado recibido se requiere de la clave privada .

El protocolo HTTPS funciona de forma similar. El **servidor** posee sus dos claves: ***pública y privada*** . Sin embargo la diferencia es que

1. El **cliente** genera su propio par de **claves por cada petición que realiza.**
2. El ***cliente*** **envía su clave privada encriptada con la clave pública del servidor** y este es el único que puede desencriptar el mensaje. Por lo tanto, ***el servidor*** **obtiene la clave privada del cliente para esta petición**.
3. El ***cliente*** **envia al servidor su peticion HTTPs** encriptada **utilizando su clave pública** y el ***servidor*** **podrá desencriptar la peticion .**

Problema: Este proceso no es inmune a ataques de intermediarios. El intermediario puede generar sus propias claves e interceptar al cliente y al servidor para obtener sus claves sin consentimiento de estos ya que no hay ninguna manera de comprobar la autenticidad de la clave

Solucion: Autoridad externa (Certification Authority) que le asegure que dicha clave pública es del servidor , esta autoridad puede firmar una clave pública si tiene garantías de la procedencia de la misma .

2. Identificacion y Autenticacion

La criptografía de las claves permite garantizar autenticidad del remitente de los mensajes. En este caso la ***clave privada*** tiene rol de **bolígrafo** y la ***clave pública*** simula **gafas** que comprueban la **autenticidad de las firmas** del bolígrafo (clave privada)

Proceso de Envio de Informacion entre Usuarios

Un usuario A quiere recibir un mensaje de B con garantia de que es B. A obtiene la clave pública de B, luego b firma (encripta) el mensaje con su clave privada y se lo envía a A.

A al recibir el mensaje lo desencripta utilizando la clave pública y la clave privada correspondiente por lo que el remitente a de ser B, y en el caso de B, solo puede encriptar los mensajes descifrables con su clave pública

Proceso con una Autoridad de Certificación

El servidor quiere que la CA certifique la autenticidad de su clave privada , para ello el servidor genera un certificado que contiene datos del servidor y la clave pública del servidor, se lo envía a la CA. La CA comprueba que los datos del certificado concuerdan con la identidad de quien lo solicita y procese a firmar el certificado .

Por otro lado suponemos que el navegador conoce a la entidad certificadora y que este viene instalado con la clave pública de la CA

Proceso con Cliente/Servidor

El cliente solicita la clave pública del servidor, el servidor le envía la clave pública metida dentro de un certificado con la firma CA

El cliente comprueba que la firma de la CA es autentica y como tiene la certeza pues ya tiene la clave pública del servidor .