11086 - Programación en Ambiente Web - UNLU
Segundo Parcial 2020

Germán Fernández
Legajo: 127390

1. ¿Qué diferencia existe entre una petición HTTP generada por un agente de usuario de forma asincrónica respecto a una sincrónica? ¿Cómo puede distinguir una aplicación web entre ambas?

La diferencia existe en que en una petición HTTP generada de forma sincrónica, el navegador dejará de hacer cualquier operación y se dedicará a realizar la petición síncrona. Esto puede correr el riesgo de congelar al navegador y así no poder usar la página hasta que te retorne una respuesta el servidor. En cambio, una petición asincrónica mientras está siendo procesada, deja al navegador libre para que pueda hacer otras operaciones en segundo plano. Cuando se envía una petición, se crea una instancia de la función "XMLHttpRequest", en la cual se debe indicar como parámetro el tipo de petición (Por ejemplo, GET, POST, etc.) y la dirección a la cual está dirigida. Existe además un último parámetro que indica si la petición es asincrónica o sincrónica, de esta forma la aplicación web puede distinguir de que petición se trata (en caso de ser asincrónica, dicho parámetro debe ser definido como true).

2. ¿Qué diferencias existen entre el diseño responsivo y el universal? ¿En qué conceptos hay que hacer hincapié al momento de definir las media querys en cada caso?

Un diseño responsivo es aquel que se adapta y es accesible por cualquier dispositivo, buscando que la experiencia del usuario sea enriquecida, ya que permitirá que el contenido de nuestro sitio sea vea bien y pueda ser bastante legible desde el dispositivo que el usuario utilice. En cambio, un diseño universal se utiliza el mismo diseño para todos los dispositivos a través de los cuales se accedan al sitio diseñado.

Al definir las media querys se puede indicar al navegador que se comporte de una u otra forma dependiendo de la condición especificada. En primer lugar, deberíamos tener en cuenta el ancho de ventana del navegador y no la resolución de pantalla del dispositivo. También podemos definir otros parámetros como por ejemplo la orientación del dispositivo. Para estos puntos de ruptura deberíamos conocer cuáles son los anchos determinados de ciertos dispositivos y de esta forma realizar una aproximación del punto de ruptura de forma intuitiva, ya que el mercado de dispositivos está en constante crecimiento incorporando nuevas características.
Otro concepto importante es el uso de medidas relativas como porcentajes o unidades "em" para mantener a los elementos de forma tan flexible como sea posible.

3. ¿Por qué decimos que no son directamente comparables REST y SOAP en el contexto de los Web Services?

En primer lugar, se podría decir que REST y SOAP no son realmente comparables ya que REST es un estilo arquitectónico, mientras que SOAP es un formato de intercambio de mensajes.
SOAP es un protocolo estandarizado que envía mensajes utilizando otros protocolos como HTTP. Por el contrario, REST establece un conjunto de pautas que se deben seguir si se desea proporcionar un servicio web RESTful, por ejemplo, stateless y el uso de códigos de estados HTTP. Además, es mucho más flexible ya que permite más formatos de mensajes como XML, JSON, HTML y otros. Por el contrario, SOAP solo utiliza XML. Se podría decir que REST es orientado al recurso, mientras que SOAP es orientado al servicio.

4. Explique brevemente tres principios de desarrollo seguro y de un ejemplo para cada uno.

Algunos de los principios de desarrollo seguro son:

- El principio de privilegios mínimo: este principio establece que un usuario debe tener el conjunto mínimo de privilegios necesarios para realizar una tarea específica. Se puede aplicar a todos los aspectos de una aplicación web, incluidos los derechos de usuario y el acceso a los recursos.

Por ejemplo, un usuario que se ha registrado en una aplicación de blog como "autor" no debe tener privilegios administrativos que le permitan agregar o eliminar usuarios. Solo se les debe permitir publicar artículos en la aplicación.

- El principio de seguridad en profundidad: Los múltiples controles de seguridad que abordan los riesgos de diferentes maneras son la mejor opción para proteger una aplicación. Por lo tanto, en lugar de tener un control de seguridad para el acceso de los usuarios, tendría varias capas de validación, herramientas adicionales de auditoría de seguridad y herramientas de registro. Por ejemplo, en lugar de permitir que un usuario inicie sesión con solo un nombre de usuario y contraseña, usaría una verificación de IP, un sistema Captcha, el registro de sus intentos de inicio de sesión, etc.

- Fail securely: Establece que solo se debe mostrar información limitada cuando el sistema encuentra errores. Por ejemplo, en un sistema de inicio de sesión con cuenta y contraseña, cuando falla el inicio de sesión, no se debería revelar información confidencial al usuario sobre cuál de los campos fue incorrecto, sino que se debería indicar que uno de los dos se ingresó incorrectamente y se podría dar la opción de restablecer la contraseña.

5. ¿Cómo se relaciona el header HTTP Content-Security-Policy con la seguridad de un sistema web y por qué es fundamental su uso hoy en día? ¿Se puede implementar esto mismo de otra forma que no sea vía header HTTP (a nivel del server web)?

El header HTTP Content-Security-Policy está fuertemente relacionado con la seguridad de un sistema web ya que nos permite prevenir y mitigar ataques como, por ejemplo, Cross Site Scripting (XSS) o inyección de datos, ataques que pueden ser utilizados para robar información, para desfigurar los sitios, etc. XSS es de los ataques más realizados en los sistemas web por eso es fundamental el uso de CSP.
Estos ataques se aprovechan de la confianza del navegador sobre el contenido que recibe, el cual ejecuta los scripts maliciosos porque confía en la fuente de los cuales provienen. CSP permite reducir o eliminar estos ataques especificando los dominios que el navegador deberá considerar como confiables de los scripts ejecutables.
CSP consiste en agregar a una página web la cabecera HTTP Content-Security-Policy, y darle valores para controlar los recursos que el agente de usuario puede cargar para esa página. Alternativamente esto mismo se puede implementar mediante el elemento <meta> utilizándolo para configurar la política.

<meta http-equiv="Content-Security-Policy" content="script-src {src}">

6. ¿Por qué es útil un buen análisis de riesgos a la hora de priorizar las mejoras de seguridad que podamos aplicar a nuestro sistema web?

Un buen análisis de riesgos es útil ya que nos permite pensar y saber cuáles son las principales vulnerabilidades de nuestro sistema web, y que amenazas podrían explotar dichas vulnerabilidades. De esta manera tendremos bien en claro los riesgos y que tanto preocuparnos por las amenazas posibles estableciendo así las medidas preventivas que garanticen mayores niveles de seguridad, reduciendo el impacto que puedan provocar en nuestro sitio web.
Una vez realizado el análisis sobre nuestro sistema podremos decidir cómo afrontar los riesgos detectados (aceptarlo, transferirlo, mitigarlo o evitarlo).

7. Describa cómo generar una buena estrategia de SEO a partir del uso de herramientas semánticas.

El SEO es el proceso de mejorar la visibilidad de un sitio web en los resultados de los buscadores. Una buena estrategia de SEO permite ayudar a los motores de búsqueda a entender sobre qué trata cada página y si es o no útil para los usuarios.
Una buena estrategia de SEO a partir del uso de herramientas semánticas puede incluir los siguientes puntos:

- Investigar las palabras claves y frases de búsqueda deseables.
- Identificar frases de búsqueda para apuntar, relevante para nuestro sitio web.
- Optimizar el código HTML de nuestro sitio, para obtener la densidad de palabras clave adecuada, la optimización de la etiqueta del título, la estructura de enlaces internos, encabezados, subtítulos, etc.

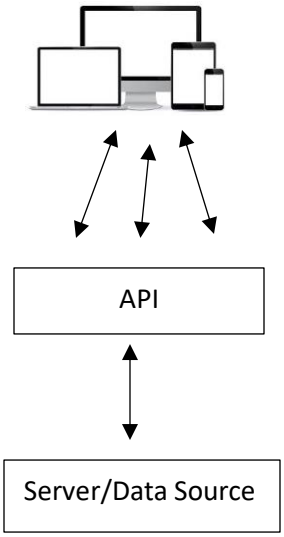
- Ayuda en la redacción de copias para atraer tanto a los motores de búsqueda como a los visitantes reales del sitio web
- Estudiar a los sitios web competidores y motores de búsqueda
- Añadir contenido de calidad
- Monitorear constantemente clasificaciones para términos de búsqueda específicos.

8. ¿Cuáles son las ventajas y desventajas del modelo serverless en el cloud respecto al modelo tradicional basado en infraestructura (servers físicos / VMs).

Un modelo serverless es un modelo de ejecución en el que el proveedor en el cloud es responsable de ejecutar un fragmento de código mediante la asignación dinámica de los recursos. Y cobrando solo por la cantidad de recursos utilizados para ejecutar el código. Un ejemplo de esto es la plataforma AWS Lambda de Amazon.
La principal ventaja de este modelo es que no es necesario la realización de tareas de administración por ejemplo administrar los servidores, solo tenemos responsabilidad del código. Gracias a esto podemos prescindir del hardware necesario para implementarlo por nuestra cuenta y como ventaja adicional, el coste y precio de estos servicios se basa en el consumo efectuado (solo pagaríamos por lo que usemos).
Otro aspecto es que, si se necesita por ejemplo más espacio, más usuarios, más capacidad de cómputo, todo es escalable más fácilmente para estas necesidades.
Como desventaja se podría decir que en un modelo serverless en el cloud los entornos de programación (Ej. Lenguajes) están limitados por el proveedor. También probablemente la decisión de cambiar de proveedor implique un costo de actualización del sistema, para cumplir con las especificaciones del proveedor nuevo.

9. Imagine que tiene que implementar un sistema de firma digital: dado un pdf de entrada debe devolverlo firmado digitalmente. Para ello, y dado que debe integrarse a sistemas web existentes, debe diseñar una arquitectura que facilite dicha integración. Comente sobre los componentes de la misma y qué cuestiones contempla, dificultades, etc.

Para implementar un sistema de firma digital podríamos realizarlo mediante una API, de esta forma facilitaría la integración en sistemas web ya existentes que requieran del uso de la misma. Una arquitectura de una API podría verse de la siguiente manera:



La API debería contar con dos componentes para poder realizar la firma digital, en primer lugar, un primer componente debería poder obtener la colección de documentos a firmar. Un segundo componente donde podrá entregar al sistema existente los documentos ya firmados por el usuario.

10. **Suponga que está desarrollando una API que puede ser consumida utilizando diferentes formatos de intercambio de datos ¿De qué forma puede determinar el backend el formato a utilizar para atender un cliente determinado? ¿Cómo debería comportarse el mismo en caso de no conocer el formato solicitado?**

Para poder determinar el backend el formato a utilizar para atender las a un cliente determinado, debería verificar en la petición HTTP del cliente el formato de intercambio de datos indicado en la cabecera “Accept”, de esta forma podemos saber en qué formato el cliente quiere recibir un recurso. El backend devolverá devolverá el recurso en el primer formato disponible y, de no poder mostrar el recurso en ninguno de los formatos indicados por el cliente, devolverá el código de estado HTTP 406 Not Acceptable, lo cual significa que el servidor no es capaz de devolver los datos en ninguno de los formatos aceptados por el cliente, indicados por éste en la cabecera “Accept” de la petición.