Programación Web II – Desarrollador de Software



Práctico Nº 0 – Conceptos de Prog. Web

1.- Que diferencias existen entre URI, URL y URN.

* URI (Identificador Uniforme de Recurso)
  + Puede ser una URL o URN
  + Es un identificador genérico que puede referirse a cualquier recurso (dentro y fuera de la web).
  + <https://www.youtube.com/recurso>
* URL (Localizador Uniforme de Recursos)
  + Es un tipo específico de URI que **indica la ubicación** de un recurso y cómo acceder a él
  + Contiene esquema (protocolo), dominio, ruta, y opcionalmente puerto y parámetros
* URN (Nombre Uniforme de Recursos)
* Es un tipo de URI que **solo identifica** un recurso de manera única, sin especificar su ubicación o cómo acceder a él.
* Puede ser un identificador persistente dentro de un sistema.
* Ejemplo: urn:isbn:0451450523 (URN de un libro basado en su ISBN).

2.- ¿Cuáles son las partes que componen una URL? Explique que es la querystring.

* https → protocolo comunicación ((HTTP, HTTPS, FTP, etc.)
* ejemplo.com → dominio (nombre dek servidor)
* Subdominio (opcional) (www)
* Puerto (Opcional) → :8080
  + Indica el puerto en el servidor (por defecto: 80 para HTTP y 443 para HTTPS).
* /index.html → ruta del recurso en el servidor (path)
* Query String (Cadena de consulta) → ?id=89&telefono=2664859913
  + Se usa para enviar parámetros a la página o aplicación.

3.- Cuales son los diferentes métodos http?

1. ¿Para qué se utilizan?:

* Los métodos HTTP definen las acciones que se pueden realizar sobre un recurso en un servidor web. Los más comunes son:
  + **GET → Recupera información sin modificar el recurso.**
  + **POST → Envía datos al servidor para crear un nuevo recurso.**
  + **PUT → Modifica o crea un recurso en una ubicación específica.**
  + **PATCH → Modifica parcialmente un recurso existente.**
  + **DELETE → Elimina un recurso.**
  + **HEAD → Similar a GET, pero solo devuelve los encabezados.**
  + **OPTIONS → Muestra los métodos disponibles en el servidor.**

1. ¿Cuándo un método es seguro (safe)?
   * Un método **HTTP** es seguro **cuando no modifica el estado del servidor**, es decir, cuando solo recupera información sin causar efectos secundarios.
     + Métodos **seguros**: **GET, HEAD, OPTIONS**
     + Métodos **no seguros: POST, PUT, PATCH, DELETE** (porque pueden modificar datos en el servidor).
2. ¿Cuándo un método es idempotente?
   * Un método es **idempotente** si, sin importar cuántas veces se ejecute, el resultado en el servidor es el mismo.
     + **Métodos idempotentes:** GET, PUT, DELETE, HEAD, OPTIONS
     + **Métodos no idempotentes:** POST, PATCH (porque pueden generar efectos diferentes en cada solicitud).
     + Ejemplo:
       - DELETE /usuario/5 → **Idempotente**, porque eliminar al usuario 5 siempre da el mismo resultado (si no existe, sigue sin existir).
       - POST /usuario → **No idempotente**, porque cada solicitud puede crear un usuario nuevo.

4.- Que es un CDN? ¿Que ventajas y desventajas existen entre usar un CDN y disponer los recursos en el mismo servidor de la aplicación?

Un **CDN (Content Delivery Network)** es una red de servidores distribuidos en diferentes ubicaciones geográficas que almacenan en caché y entregan contenido estático (como imágenes, archivos CSS/JS, videos, etc.) para mejorar el rendimiento y la disponibilidad de una aplicación web.

Ejemplos de CDN:

* Cloudflare
* Amazon CloudFront
* Akamai
* Google Cloud CDN

**Ventajas de usar un CDN vs. Servidor propio**

| **Ventaja** | **CDN** | **Servidor propio** |
| --- | --- | --- |
| Velocidad de carga | Alta (servidores cercanos al usuario). | Más lenta si el usuario está lejos del servidor. |
| Disponibilidad | Alta (varios servidores en diferentes ubicaciones). | Menos disponible si el servidor cae. |
| Carga del servidor | Reduce la carga en el servidor principal. | Todo el tráfico es manejado por un solo servidor. |
| Seguridad | Protección contra ataques DDoS y filtrado de tráfico malicioso. | Mayor riesgo de sobrecarga en ataques. |
| Escalabilidad | Se adapta fácilmente a más tráfico. | Puede necesitar más infraestructura para escalar. |

**Desventajas de usar un CDN vs. Servidor propio**

| **Desventaja** | **CDN** | **Servidor propio** |
| --- | --- | --- |
| **Costo** | Puede ser caro dependiendo del tráfico. | Solo se paga por el hosting del servidor. |
| **Control** | Menos control sobre los datos y el caché. | Control total sobre los archivos y configuración. |
| **Latencia en caché** | Si la caché no está actualizada, puede servir contenido viejo. | Siempre entrega la última versión del contenido. |
| **Dependencia externa** | Si el CDN falla, los recursos pueden no cargarse. | No depende de terceros. |

**En resumen:**

### ****¿Cuándo usar un CDN?****

* Cuando se tiene una aplicación con usuarios globales.
* Si se quiere reducir la carga en el servidor y mejorar la velocidad.
* Para mejorar la seguridad y mitigar ataques DDoS.

### ****¿Cuándo NO usar un CDN?****

* Si la aplicación es pequeña y solo tiene tráfico local.
* Si se necesita control absoluto sobre los archivos.
* Cuando los costos de un CDN no justifican su uso.

5.- Explique a que se debe el término cliente liviano y cliente pesado en la programación web?

### ****Cliente Liviano vs. Cliente Pesado en Programación Web****

En programación web, los términos **cliente liviano** y **cliente pesado** hacen referencia a cómo se distribuye la carga de procesamiento entre el cliente (navegador del usuario) y el servidor.

### ****1. Cliente Liviano (Thin Client)****

El cliente liviano depende en gran medida del servidor para procesar la lógica de la aplicación.

#### **Características:**

✅ Requiere menos recursos en el dispositivo del usuario.  
✅ La mayor parte del procesamiento ocurre en el servidor.  
✅ Se usa en aplicaciones donde los dispositivos cliente tienen poca capacidad de cómputo.  
✅ Menos carga en el lado del cliente, lo que lo hace más rápido en dispositivos de baja gama.

#### **Ejemplo:**

* Aplicaciones web tradicionales basadas en HTML/CSS con poca lógica en el cliente.
* Aplicaciones que dependen de renderizado en el servidor, como PHP con Laravel o JSP.

### ****2. Cliente Pesado (Thick Client o Fat Client)****

El cliente pesado realiza más procesamiento en el navegador del usuario, reduciendo la carga en el servidor.

#### **Características:**

✅ Mayor parte del procesamiento ocurre en el cliente.  
✅ Reduce la carga en el servidor, ya que el cliente ejecuta lógica.  
✅ Permite aplicaciones más interactivas y rápidas.  
✅ Requiere más recursos del dispositivo cliente.

#### **Ejemplo:**

* Aplicaciones web modernas con **JavaScript**, como **React, Angular o Vue.js**.
* Aplicaciones de una sola página (**SPA - Single Page Application**).
* Juegos en el navegador que procesan gráficos sin enviar cada acción al servidor.

### ****Comparación Cliente Liviano vs. Cliente Pesado****

| **Característica** | **Cliente Liviano** | **Cliente Pesado** |
| --- | --- | --- |
| **Procesamiento** | En el servidor | En el cliente (navegador) |
| **Requerimientos en el cliente** | Bajos | Altos (CPU, memoria, gráficos) |
| **Carga del servidor** | Alta | Baja |
| **Interactividad** | Menor (requiere recargar páginas) | Mayor (carga dinámica sin recargar) |
| **Ejemplo de tecnología** | HTML + PHP + Servidor | React, Angular, Vue.js |

### ****¿Cuándo usar cada uno?****

✅ **Cliente liviano:** cuando se necesita compatibilidad con dispositivos de baja potencia y una carga mínima en el cliente.  
✅ **Cliente pesado:** cuando se requiere una experiencia de usuario fluida y rápida sin depender demasiado del servidor.

6.- Como es el funcionamiento (pasos que realiza) de un browser cuando realiza un requerimiento o solicitud?

| **Paso** | **Descripción** |
| --- | --- |
| 1. URL ingresada | El usuario ingresa una dirección en el navegador. |
| 2. DNS Lookup | Se busca la IP del dominio. |
| 3. Conexión | Se establece conexión TCP y, si es HTTPS, se usa TLS. |
| 4. Solicitud HTTP | El navegador envía una petición al servidor. |
| 5. Respuesta del servidor | El servidor devuelve el contenido solicitado. |
| 6. Procesamiento del navegador | Se analizan HTML, CSS, JS y se descargan archivos adicionales. |
| 7. Renderizado | El navegador muestra la página en pantalla. |
| 8. Interacción del usuario | El usuario puede navegar e interactuar. |

7.- Cuales son las tareas que realiza el browser para renderizar el contenido de una respuesta en el navegador?

* + Cuando el navegador recibe una respuesta HTTP del servidor, realiza varias tareas para renderizar el contenido en pantalla. Este proceso involucra el análisis del HTML, CSS y JavaScript, y la construcción de la interfaz gráfica que el usuario ve y con la que interactúa.

8.- Explique la diferencia entre Client Side Rendering (CSR) y Server Side Rendering (SSR)

El **renderizado** es el proceso de convertir el código HTML, CSS y JavaScript en una página visible en el navegador. Existen dos enfoques principales para esto:

## **Server Side Rendering (SSR) – Renderizado en el Servidor**

En **SSR**, el servidor procesa y genera la página completa antes de enviarla al navegador.

🔹 **Cómo funciona:**

* + El usuario solicita una página (https://ejemplo.com).
  + El servidor procesa la solicitud, genera el **HTML completo** y lo envía al navegador.
  + El navegador recibe la página ya lista y la muestra en pantalla.

🔹 **Ejemplo:**  
Un sitio en **PHP** o **Node.js con Express y Handlebars** que genera el HTML en el servidor.

🔹 **Ventajas de SSR:**  
✅ **Mejor para SEO:** Los motores de búsqueda ven el contenido listo.  
✅ **Carga rápida en la primera visita:** El usuario ve contenido inmediatamente.  
✅ **Funciona sin JavaScript habilitado.**

🔹 **Desventajas de SSR:**  
❌ **Más carga en el servidor:** Debe procesar cada solicitud.  
❌ **Interacciones más lentas:** Cada clic puede requerir una nueva carga de página.

## **Client Side Rendering (CSR) – Renderizado en el Cliente**

En **CSR**, el servidor envía solo un archivo HTML base y el navegador descarga JavaScript para generar el contenido dinámicamente.

🔹 **Cómo funciona:**

1. El usuario solicita una página (https://ejemplo.com).
2. El servidor envía un **HTML vacío** con referencias a archivos JavaScript.
3. El navegador descarga y ejecuta el **JavaScript**, que genera el contenido.

**🔹 Ventajas de CSR:  
✅ Experiencia más fluida: No es necesario recargar la página.  
✅ Menos carga en el servidor: Solo envía datos en formato JSON.  
✅ Más interacción en tiempo real: Ideal para aplicaciones como redes sociales o chats.**

🔹 **Desventajas de CSR:**  
❌ **Mala experiencia en la primera carga:** La página aparece vacía hasta que se ejecuta el JavaScript.  
❌ **Problemas con SEO:** Google puede no ver el contenido antes de que se cargue.  
❌ **Mayor uso de recursos en el navegador.**

9.- Realice un resumen sobre las características más relevantes de una aplicación SPA.

* Una **SPA (Single Page Application)** es un tipo de aplicación web que se carga en una sola página HTML y se actualiza dinámicamente a medida que el usuario interactúa con la aplicación, sin necesidad de recargar la página

|  |  |
| --- | --- |
| Característica | Descripción |
| **Carga Inicial Única** | Solo se realiza una carga completa de la página, el resto se carga dinámicamente. |
| **Navegación sin Recargas** | La aplicación cambia de vista sin recargar toda la página, usando AJAX o Fetch API. |
| **Mejor Experiencia de Usuario (UX)** | Transiciones rápidas y suaves entre vistas, sin interrupciones. |
| **Uso Extensivo de JavaScript** | El renderizado y la lógica se manejan en el cliente utilizando tecnologías como React, Angular o Vue.js. |
| **Enrutamiento en el Cliente** | Se usa JavaScript para gestionar las URLs y cambiar entre vistas sin recargar la página. |
| **Manejo de Estado del Cliente** | El estado de la aplicación se maneja en el cliente (usando localStorage, Redux, Vuex). |
| **SEO y Accesibilidad** | El SEO puede ser complicado, pero se pueden implementar estrategias como SSR o pre-renderizado. |
| **Actualización de Datos Dinámica** | Los datos se actualizan de forma dinámica con solicitudes asíncronas (RESTful APIs, WebSockets). |
| **Ventajas** | |  | | --- | |  |  |  | | --- | | Carga rápida, interactividad fluida, menor carga en el servidor. | |
| **Desventajas** | SEO desafiante, mayor consumo de recursos en el navegador, carga inicial más pesada. |

**Ejemplos:**

**Facebook, instagram, etc (La acción de scrollear se va cargando el contenido)**

10.- Realice un relevamiento sobre hostings y/o PaaS para desplegar una aplicación Node.js. Mencione las plataformas que permitan desplegar de forma gratuita.

### ****Vercel****

* **Descripción:** Vercel es una plataforma optimizada para desplegar aplicaciones front-end y back-end, ideal para proyectos Node.js. Es muy popular por su integración con **Next.js**, pero funciona perfectamente para cualquier aplicación Node.js.
* **Características:**
  + Despliegue automático desde repositorios en GitHub, GitLab o Bitbucket.
  + Permite la creación de funciones serverless (sin servidor).
  + Despliegue continuo, optimización automática y escalabilidad.
* **Gratuito:**
  + Plan gratuito con despliegue ilimitado.
  + Ideal para aplicaciones de tamaño pequeño a mediano.

