«Talento Tech»

Front-End JS

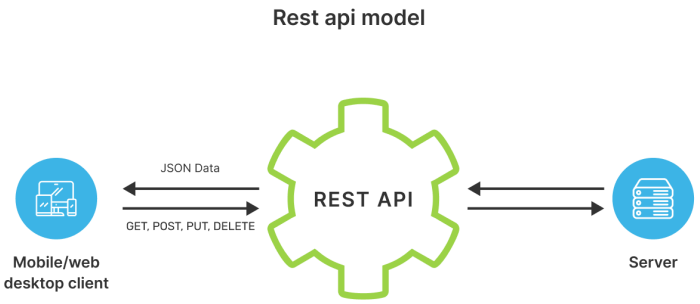
Clase 15

Buenos Aires Ciudad

Clase 15: API y Procesamiento de Datos en E-commerce

Índice

1. Introducción al Consumo de API REST



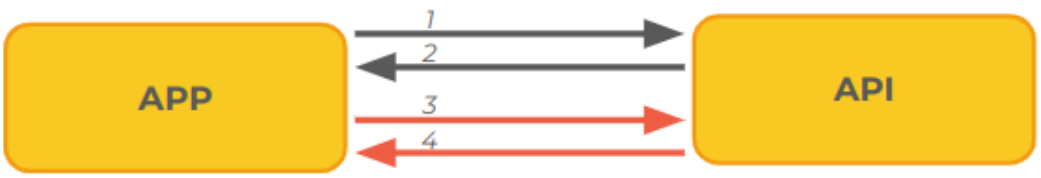
1.1. **¿Qué es una API REST?**

- Una API REST permite que dos sistemas se comuniquen de forma estructurada y eficiente. Con APIs, las aplicaciones pueden obtener y enviar datos de un servidor, facilitando la creación de experiencias dinámicas y en tiempo real.

1.2. **Ejemplos de Uso en Empresas de Tecnología**

- **Redes Sociales**: Facebook, Twitter e Instagram ofrecen APIs que permiten acceder a perfiles, publicaciones y seguidores.
- **E-commerce**: Plataformas como Amazon y eBay usan APIs para facilitar la integración de productos, precios y disponibilidad en tiendas online externas.
- **Mapas y Geolocalización**: Google Maps y OpenStreetMap tienen APIs que permiten a las aplicaciones obtener información geográfica, rutas y mapas en tiempo real.
- **Pago y Autenticación**: PayPal, Stripe, y MercadoPago ofrecen APIs para gestionar pagos seguros y verificar identidades.
- **Noticias y Entretenimiento**: APIs como las de Spotify y YouTube permiten que las aplicaciones integren contenido musical o audiovisual.

Las APIs REST son fundamentales para la conectividad y el intercambio de datos entre sistemas de diversas industrias, ofreciendo flexibilidad y potencia para crear experiencias digitales modernas.



1.3. **¿Por qué se usa JSON en lugar de XML?**

- **Ligereza**: JSON (JavaScript Object Notation) es más ligero y fácil de leer que XML. Esto permite transferir datos de manera más rápida, reduciendo el tiempo de carga en la aplicación.
- **Compatibilidad**: JSON es más compatible con JavaScript, ya que se puede convertir directamente en objetos JavaScript. Con XML, era necesario un procesamiento adicional para extraer los datos.
- **Legibilidad y Sencillez**: JSON es menos verboso, haciendo que el código sea más claro y fácil de interpretar.

Ejemplo de JSON vs. XML:

XML

```
<empinfo>
   <employees>
      <employee>
         <name>James Kirk</name>
         <age>40></age>
      </employee>
      <employee>
         <name>Jean-Luc Picard</name>
         <age>45</age>
      </employee>
      <employee>
         <name>Wesley Crusher</name>
         <age>27</age>
      </employee>
   </employees>
</empinfo>
```

JSON

```
{  "empinfo" :
      {
            "employees" :  [
            {
                  "name" : "James Kirk",
                  "age" : 40,
            },
            {
                  "name" : "Jean-Luc Picard",
                  "age" : 45,
            },
            {
                  "name" : "Wesley Crusher",
                  "age" : 27,
            }
                              ]
      }
}
```

2. Consumo de APIs con `fetch()`

2.1. **Uso de `fetch()` para Hacer Solicitudes HTTP**

- `fetch()` permite realizar solicitudes HTTP a una API de forma asíncrona. Esto es útil para obtener datos sin recargar la página, mejorando la experiencia del usuario.

Ejemplo básico de fetch:

```
fetch('https://fakestoreapi.com/products')
  .then(response => response.json())
  .then(data => console.log(data))
  .catch(error => console.error('Error al obtener datos:', error));
```

2.2. **Procesamiento de la Respuesta en JSON**

- Usamos `response.json()` para convertir los datos en un objeto JSON, lo cual facilita el procesamiento en JavaScript.

Ejemplo:

```
fetch('https://fakestoreapi.com/products')
  .then(response => response.json())
  .then(data => {
      // Aquí se procesan los datos obtenidos y se integran en el DOM
  });
```

2.3. **Manejo de Errores con `.catch()` y Códigos 400/500**

- Cuando una solicitud a la API falla, es esencial manejar el error para que el usuario reciba un mensaje claro.
- **Errores Comunes**:
    - **Errores 400**: Problemas con la solicitud, como parámetros incorrectos o falta de permisos. Ejemplo: 404 (No Encontrado), 401 (No Autorizado).
    - **Errores 500**: Problemas en el servidor. Estos errores son más difíciles de predecir, ya que ocurren en el lado del servidor.

4XX Client Error	
400	Bad Request
401	Unauthorized
402	Payment Required
403	Forbidden
404	Not Found
405	Method Not Allowed
406	Not Acceptable
407	Proxy Authentication Required
408	Request Timeout

5XX Server Error	
500	Internal Server Error
501	Not Implemented
502	Bad Gateway
503	Service Unavailable
504	Gateway Timeout
505	HTTP Version Not Supported
506	Variant Also Negotiates
507	Insufficient Storage
508	Loop Detected
510	Not Extended
511	Network Authentication Required
599	Network Connect Timeout Error

Manejo de Errores:

```
fetch('https://fakestoreapi.com/products')
  .then(response => {
      if (!response.ok) {
          throw new Error(`HTTP error! Status:
${response.status}`);
      }
      return response.json();
  })
```

```
  .then(data => {
      // Procesamiento de datos
  })
  .catch(error => {
      console.error('Error en la comunicación con la API:', error);
      // Aquí podrías mostrar un mensaje de error al usuario
  });
```

3. Renderizado Dinámico de Productos

3.1. **Creación de Contenedor en HTML para Productos**

En el HTML, necesitamos un contenedor para mostrar los productos de forma dinámica:

```
<div id="productos-container"></div>
```

3.2. **Procesamiento y Estructura de Datos en JavaScript**

- Con los datos obtenidos de la API, usamos JavaScript para generar elementos HTML que representen cada producto (imagen, título, precio, botón de compra).

3.3. **Integración de Productos en el DOM Usando JavaScript**

Insertamos cada producto en el contenedor usando JavaScript y manipulación del DOM:

```
fetch('https://fakestoreapi.com/products')
  .then(response => response.json())
  .then(data => {
      const contenedor = document.getElementById("productos-container");
```

```
        data.forEach(producto => {
            const productoCard = `
                <div class="card">
                    <img src="${producto.image}"
alt="${producto.title}">
                    <h3>${producto.title}</h3>
                    <p>Precio: $${producto.price}</p>
                    <button
onclick="agregarAlCarrito(${producto.id})">Añadir al
carrito</button>
                </div>
            `;
            contenedor.innerHTML += productoCard;
        });
    });
```

---

4. Buenas Prácticas de Accesibilidad y SEO

4.1. **Accesibilidad en el Proyecto**

- **Etiquetas Alt en Imágenes**: Cada imagen debe tener un atributo `alt` que describa el contenido de la imagen, para mejorar la experiencia de usuarios con discapacidades visuales.
- **Navegación por Teclado**: Asegurarse de que los elementos interactivos, como botones y enlaces, puedan ser accesados con el teclado.
- **Estructura Semántica**: Usar etiquetas como `<header>`, `<main>`, `<section>`, `<footer>` para que el contenido sea fácilmente interpretable.

4.2. **Optimización SEO en el Proyecto**

- **Metaetiquetas**: Las etiquetas `<meta>` en el `<head>` mejoran la visibilidad en los motores de búsqueda.
- **Encabezados Lógicos**: Usar encabezados (`<h1>`, `<h2>`, `<h3>`) en una estructura jerárquica para que los motores de búsqueda puedan entender la relevancia del contenido.
- **Contenido Optimizado**: Asegurarse de que las descripciones y los nombres de productos sean claros y relevantes para los usuarios.

---

5. Guía para el Proyecto Final de E-commerce

5.1. **Estructura y Requisitos del Proyecto**

- **HTML**: Uso de etiquetas semánticas para organizar la página.
- **CSS**: Implementación de un diseño responsivo y atractivo usando Bootstrap y Flexbox.
- **JavaScript**: Integración de una API REST para obtener datos y renderizar productos en el DOM, además de la funcionalidad de un carrito de compras usando `localStorage`.
- **Accesibilidad y SEO**: Implementar prácticas que mejoren la experiencia del usuario y optimicen la página para los motores de búsqueda.

5.2. **Puntos Clave para Revisión y Entrega**

- **Subida del Proyecto**: Debe estar disponible en GitHub Pages o Netlify para facilitar su acceso.
- **Control de versiones**: Mantener un historial de commits detallado para documentar cada avance.

- **Presentación**: El archivo README.md debe incluir una descripción del proyecto, las tecnologías usadas, instrucciones de instalación y cualquier detalle relevante.

Buenos Aires aprende

Agencia de Habilidades para el Futuro

Buenos Aires Ciudad