Buenos Aires aprende
Agencia de Habilidades para el Futuro

Talento Tech

Front-End JS

Clase 15 - “API y procesamiento de datos”

Buenos Aires Ciudad

¡Les damos la bienvenida!

 Vamos a comenzar a grabar la clase

Buenos Aires aprende
Agencia de Habilidades para el Futuro

Buenos Aires Ciudad

Talento Tech

Clase 14.

JS 6 - Asincronía

1. Asincronía
2. Consumo de API REST a través de fetch
3. Procesamiento de los datos
4. Incluir los datos consumidos y procesados por medio de fetch en nuestro proyecto

Clase 15.

API y Procesamiento de Datos

1. Asincronía Desarrollo de un proyecto integrador que combine HTML, CSS, y JavaScript
2. Consumo de API REST
3. Incorporación y procesamiento de los datos en nuestro HTML
4. Incorporación de buenas prácticas de accesibilidad y SEO
5. Presentación del proyecto final

Clase 16.

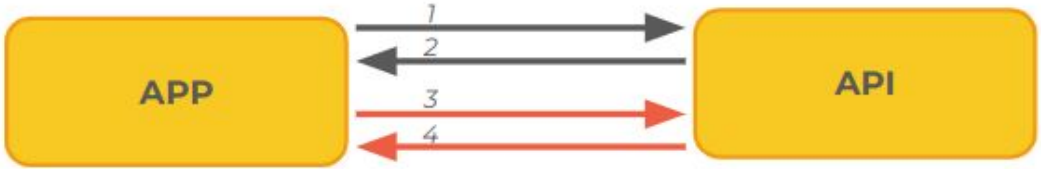
Revisión final y despedida

1. Presentación y revisión de los proyectos finales
2. Feedback personalizado a cada proyecto
3. Comparación con proyectos profesionales
4. Despedida y recomendaciones para el futuro

¿Qué es una API REST?

Comunicación entre aplicaciones

- **Definición de API:** una API es una interfaz que permite la comunicación entre diferentes aplicaciones.
- **API REST:** es un tipo de API que usa HTTP para acceder y manipular datos en un servidor.
- **Aplicación en E-commerce:** en un sitio de E-commerce, una API REST permite acceder a catálogos de productos, realizar operaciones de carrito de compras y más.



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.

Json vs Xml

La Evolución de APIs

En los inicios del desarrollo web, **XML** era el formato estándar para compartir datos en APIs. Sin embargo, **JSON** se volvió preferido por su simplicidad, menor tamaño y compatibilidad directa con JavaScript, facilitando la velocidad y eficiencia en aplicaciones modernas. Hoy, JSON es el formato más usado en APIs REST.

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,
            }
                        ]
      }
}
```

Consumo de API en JavaScript

Función fetch()

- Uso de fetch(): Realiza solicitudes HTTP de manera asíncrona.


```
fetch('https://fakestoreapi.com/products')
  .then(response => response.json())  // Convertir respuesta a JSON
  .then(data => { console.log(data); })  // Manipular los datos
  .catch(error => { console.error('Error:', error); });  // Manejo de errores
```

Procesamiento de Datos en HTML

Renderizado dinámico de productos

- Renderizado en HTML: usar JavaScript para insertar datos dinámicamente en HTML.
- Ejemplo para mostrar productos:

```
fetch('https://fakestoreapi.com/products' )
 .then((response) => response.json())
 .then((data) => {
   const productosContainer = document.getElementById("productos-container" );
   data.forEach((producto) => {
     productosContainer.innerHTML += `
       <div class="card">
           <img src=" ${producto.image}" alt="${producto.title}">
           <h3> ${producto.title}</h3>
           <p>Precio: $ ${producto.price}</p>
           <button onclick="addToCart( ${producto.id})">Añadir al carrito</button>
       </div>
     `;
   });
 })
 .catch((error) => console.error("Error al obtener productos:" , error));
```

Buenas Prácticas de Accesibilidad y SEO

Mejorando la experiencia y visibilidad del sitio

- **Accesibilidad**:
  - Añadir etiquetas alt a las imágenes.
  - Navegación accesible con teclado.
  - Uso de etiquetas semánticas (<header>, <main>, <footer>) para organizar el contenido.
- **SEO**:
  - Uso de metaetiquetas (<meta name="description" content="...">).
  - Encabezados (<h1>, <h2>) para resaltar la jerarquía del contenido.

Manejo de errores

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.

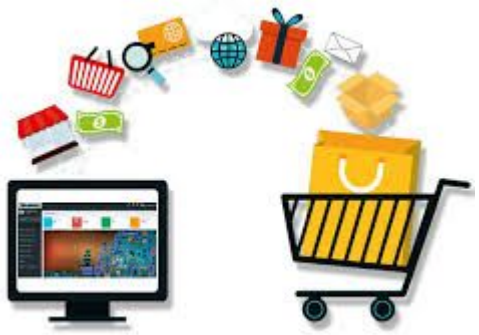
Manejo de errores - 400

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

Manejo de errores - 500

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

Guia para el proyecto final



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.

Guia para el proyecto final

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.

¡Vamos a la práctica! 

Ejercicios Prácticos

Optativos | No entregables

Revisión Final y Documentación en GitHub

Realiza una revisión completa de tu proyecto para asegurarte de que cumple con todos los requisitos. Documenta los pasos importantes, decisiones de diseño, y cualquier información relevante en el archivo README.md

Pasos:

- Verifica que cada parte del proyecto funcione correctamente (carrito, consumo de API, formulario de contacto).
- Revisa el diseño responsivo y la accesibilidad, asegurándote de que el sitio sea navegable en diferentes dispositivos.
- **Actualiza el README.md:**
  - Incluye una descripción detallada del proyecto, sus objetivos y funcionalidades.
  - Añade instrucciones claras sobre cómo clonar el repositorio y ejecutar el proyecto.
  - Explica cualquier tecnología o recurso externo utilizado (como la API o Bootstrap).
  - Adjunta una captura de pantalla de la interfaz y el enlace del despliegue en GitHub Pages.
- **Tip**: Un README.md bien documentado puede ser muy útil para otros usuarios y desarrolladores que exploren tu proyecto.

Ejercicios Prácticos

Optativos | No entregables

Preparación de la Presentación del Proyecto

Prepara una breve presentación de tu proyecto para la clase siguiente. La presentación debe incluir una introducción al proyecto, una demostración en vivo, y una explicación de los principales desafíos y aprendizajes.

Pasos:

- **Introducción (1-2 minutos):** Explica el objetivo de tu proyecto de E-commerce y describe brevemente sus funcionalidades principales.
- **Demostración en Vivo (2-3 minutos):** Muestra el sitio en funcionamiento, destacando la visualización de productos, el carrito de compras, y el formulario de contacto.
- **Desafíos y Aprendizajes (1-2 minutos):** Menciona los principales retos que enfrentaste durante el desarrollo (por ejemplo, el uso de fetch() o el almacenamiento en localStorage), y lo que aprendiste de la experiencia.
- **Tip:** Practica la presentación para ajustarte al tiempo y asegúrate de que el proyecto esté accesible y funcional durante la demostración.

Buenos Aires aprende
Agencia de Habilidades para el Futuro

Talento Tech

Proyecto Final Integrador

Clase 15 - “API y procesamiento de datos”

Buenos Aires Ciudad

Entrega de Proyecto

Obligatorio | Entregable

Formato de entrega: Compartir un link al drive (público) que contenga los archivos y carpetas que conforman tu proyecto o compartí el link de tu repositorio de Github. Los links deberán ser entregados en el apartado de “Pre-Entrega de Proyecto” en el Campus Virtual.

Requisitos para la entrega:

1. Estructura Básica de HTML.

- **Estructura semántica**: El HTML debe estar dividido en las etiquetas semánticas principales: **header**, **nav**, **main**, **section**, **footer**.
- **README.md**: Incluir un archivo que explique brevemente el propósito de la página.

2. Formulario de Contacto.

- **Formulario funcional**: Crear un formulario de contacto con campos para **nombre**, **correo electrónico** y **mensaje**, utilizando **Formspree** para manejar el envío de datos.

Entrega de Proyecto

Requisitos para la entrega

3. Estilos básicos aplicados con CSS

- **Archivo styles.css**: El proyecto debe contar con un archivo CSS externo que incluya:
  - Estilos básicos aplicados a las secciones de **header**, **footer** y **lista de navegación**.
  - **Fuentes de Google Fonts** correctamente implementadas.
  - Propiedades de **background** aplicadas en alguna sección de la página (color, imagen, degradado, etc.).

4. Diseño responsivo con Flexbox y Grid

- **Sección "Productos"**: Organizada en **cards** de forma responsiva utilizando **Flexbox**.
- **Sección "Reseñas"**: Organizada utilizando **Grid**, con una distribución lógica y estética.
- **Sección "Contacto"**: Debe ser responsiva mediante el uso de **Media Queries** para adaptarse a diferentes tamaños de pantalla.

Entrega de Proyecto

Requisitos para la entrega:

5. Contenido Multimedia y Navegación

- **Multimedia**: deberá incluir archivos multimedia (imágenes, video o iframe) correctamente integrado en la página.
- **Lista de navegación**: Implementar una **lista desordenada** con enlaces que simulen una navegación interna (Inicio, Productos, Contacto, etc.).

6. Subida del Proyecto.

- El proyecto debe estar subido a un **hosting gratuito** (Netlify o GitHub Pages), con una **URL funcional** para visualizar el sitio.

Entrega de Proyecto

Requisitos para la entrega:

7. JavaScript

- **Script.js**: deberá incluir un archivo Debes crear un archivo `script.js` para manejar toda la interactividad de la página.

  Asegúrate de enlazarlo correctamente en tu archivo HTML.

- **DOM**: Implementa funciones para validar formularios (ej., campos requeridos y formato de correo).

  Usa JavaScript para manipular elementos del DOM, por ejemplo, actualizar el carrito y mostrar mensajes al usuario

- **Fetch Api**

  Consume datos desde una API REST usando fetch.

  Muestra los productos obtenidos de la API en la página en forma de tarjetas (cards).

- **Visualización de Productos**:

  Cada producto debe tener su imagen, título y precio, mostrando una lista atractiva para el usuario.

Entrega de Proyecto

Requisitos para la entrega:

8. Carrito de compras dinámico

- **Agregar Productos al Carrito**: Implementa un carrito de compras donde los usuarios puedan añadir productos desde las tarjetas.
- **Uso de localStorage o sessionStorage**: Guarda el estado del carrito en `localStorage` o `sessionStorage` para que no se pierda al actualizar o cerrar la página.
- **Contador Dinámico**: Muestra el número total de productos en el carrito y asegúrate de actualizarlo en tiempo real.

9. Edición y visualización del carrito

- **Visualización de Productos en el Carrito**: Muestra una lista de productos añadidos al carrito, incluyendo cantidad, precio y total.
- **Edición de Cantidades y Eliminación de Productos**: Implementa funciones para que el usuario pueda editar la cantidad de cada producto o eliminarlo del carrito.
- **Total Dinámico**:Actualiza el total de la compra cada vez que se modifiquen los productos en el carrito.

Entrega de Proyecto

Requisitos para la entrega:

10. SEO & Accesibilidad

Buenas Prácticas de Accesibilidad:

- Usa `alt` en las imágenes para mejorar la accesibilidad.
- Asegúrate de que se pueda navegar fácilmente con el teclado.

SEO Básico:

- Usa metaetiquetas en el `head` del HTML para optimizar el SEO.
- Organiza los encabezados lógicamente usando `<h1>`, `<h2>`, etc.

11. Git & Github

Historial de Cambios en GitHub:

- Realiza commits regulares que documenten el progreso del proyecto.
- Crea un repositorio en GitHub y sube el proyecto.

README.md:

- Documenta el propósito del proyecto, instrucciones de configuración, y cómo utilizarlo.
- Incluye una breve guía de usuario y cualquier dependencia necesaria.

Entrega de Proyecto

Requisitos para la entrega:

12. Funcionalidad esperada del proyecto

Interactividad Completa:

- La página debe permitir al usuario ver productos, añadirlos al carrito, editar el carrito, y simular la compra.

Formulario de Contacto:

- Implementa un formulario funcional que envíe datos a través de Formspree.

Diseño Responsivo:

- Asegúrate de que el diseño sea adaptable a diferentes tamaños de pantalla.

Persistencia del Carrito:

- El carrito debe mantenerse activo incluso si el usuario cierra o actualiza la página, usando `localStorage` o `sessionStorage`.