Taller 1: Análisis de la Incorporación de JavaScript en el Proyecto del Hospital

Allison Acosta F.

GENERALIDADES DEL LENGUAJE JAVASCRIPT.

I. Historia de JavaScript:

JavaScript fue creado en 1995 por Brendan Eich en Netscape, inicialmente bajo el nombre de Mocha y luego renombrado a LiveScript antes de finalmente adoptar el nombre de JavaScript. Fué creado en tan solo 10 días y su propósito era proporcionar una manera de hacer que las páginas web fueran más interactivas y dinámicas sin necesidad de recargar toda la página, respondiendo a la creciente demanda de mayor funcionalidad en la web. Con el tiempo, JavaScript fue estandarizado bajo el nombre de ECMAScript por la organización ECMA en 1997, lo que permitió que se adoptara de manera uniforme en los diferentes navegadores.

A partir de 2005, con la popularización de tecnologías como AJAX, JavaScript permitió la creación de aplicaciones web más interactivas y fluidas, sin necesidad de recargar las páginas. Con el crecimiento de bibliotecas y frameworks como jQuery, React y Angular, JavaScript ha transformado la forma en que se desarrollan las aplicaciones web, convirtiéndose en un lenguaje esencial tanto para el cliente como en el uso de servidores como Node.js, consolidándose como la base para el desarrollo de la web moderna.

II. Uso de JavaScript en Navegadores Web:

JavaScript desempeña un papel fundamental en los navegadores web al permitir que las páginas sean interactivas y dinámicas. A diferencia del HTML y CSS, que definen la estructura y el estilo de una página web, JavaScript se utiliza para añadir funcionalidad, como la manipulación del DOM, la validación de formularios, la gestión de eventos y la actualización de contenido sin necesidad de recargar la página.

JavaScript se ejecuta directamente en el lado del cliente, es decir, en el navegador del usuario, lo que permite que el código se ejecute sin necesidad de interactuar con el servidor para cada acción del usuario. El motor de JavaScript del navegador interpreta y ejecuta el código en tiempo real. Esto permite que los desarrolladores añadan interactividad sin tener que hacer múltiples solicitudes al servidor, mejorando la velocidad y la experiencia del usuario.

III. Entornos Virtuales de JavaScript:

JavaScript se puede ejecutar en diversos entornos, siendo el más común el navegador web, donde se utiliza para crear interactividad en las páginas web a través del DOM y manejar eventos en tiempo real.

Por otro lado, Node.js permite ejecutar JavaScript fuera del navegador, en el servidor, ofreciendo la posibilidad de crear aplicaciones completas utilizando el mismo lenguaje tanto en el cliente como en el servidor.

Aparte de estos entornos, JavaScript también puede ser ejecutado en plataformas como Deno, un entorno moderno y seguro para JavaScript y TypeScript, y en aplicaciones de escritorio mediante frameworks como Electron, que permite crear aplicaciones nativas multiplataforma.

IV. Diferencias entre JavaScript y otros lenguajes:

JavaScript se diferencia de otros lenguajes de programación como Java o Python en varios aspectos. Mientras que Java se usa principalmente para aplicaciones de gran escala y en entornos corporativos, y Python se destaca en la ciencia de datos y automatización, JavaScript está principalmente diseñado para desarrollo web, permitiendo la creación de páginas interactivas en el navegador.
Otras características son:

1. Tipado dinámico: JS es dinámico y flexible a diferencia de lenguajes como Java o C++ que son tipados estáticamente.
2. Lenguaje Interpretado: Se ejecuta línea por línea en lugar de ser compilado previamente.
3. Multiparadigma: Soporta programación orientada a objetos, funcional y procedimental o Imperativo.
    a. Orientado a objetos: Utiliza clases y objetos para modelar el comportamiento del sistema.
    b. Funcional: Usa funciones puras, inmutabilidad y evita el estado global.
    c. Imperativo: Realiza tareas paso a paso mediante instrucciones explícitas.

V. Fortalezas y debilidades de JavaScript:

JS tiene varias fortalezas, una de sus principales ventajas es su capacidad para ejecutarse directamente en el navegador, lo que permite la creación de aplicaciones web altamente interactivas sin necesidad de recargar la página, mejorando la experiencia del usuario. Además, al ser un lenguaje multiparadigma, JavaScript permite a los desarrolladores elegir entre programación orientada a objetos, funcional y procedural, lo que lo hace flexible y adaptable a diferentes necesidades.
La gran cantidad de frameworks y bibliotecas (como React, Node.js, Angular, etc...) también facilita el desarrollo de aplicaciones web escalables y robustas. JS gracias a su naturaleza dinámica, permite una gran flexibilidad a la hora de manipular datos y estructuras.Otro punto a favor es su amplia comunidad y el soporte constante de nuevas herramientas y mejoras, lo que asegura su evolución y relevancia.
Sin embargo, JavaScript también presenta algunas limitaciones. Una de las principales debilidades es su rendimiento (por ser un lenguaje interpretado) en comparación con lenguajes como C++ o Java, lo que puede ser un problema en aplicaciones que requieren procesamiento intensivo, como las de gráficos en tiempo real o cálculos complejos.
Además, debido a su naturaleza dinámica, los errores en JavaScript pueden ser más difíciles de detectar en tiempo de desarrollo, lo que puede generar problemas en la producción si no se manejan adecuadamente. Otra limitación es la falta de un sistema de tipos estáticos por defecto, lo que puede conducir a errores de tipo difíciles de rastrear, aunque herramientas como TypeScript han mitigado este problema al añadir tipado estático al ecosistema de JavaScript.

VI. JavaScript como lenguaje asíncrono:

JavaScript utiliza un modelo asíncrono para manejar tareas de forma eficiente sin bloquear el hilo principal. Estas tareas pueden ser peticiones HTTP, solicitudes de red o lectura de archivos. Esto se logra mediante el uso de callbacks, promises y async/await.

1. Los callbacks son funciones que se pasan como argumentos y se ejecutan una vez que una operación asíncrona se completa, pero pueden llevar a problemas de callback hell (anidación excesiva de callbacks). Para resolver esto, se introdujeron las promises
2. Las promises permiten manejar operaciones asíncronas de manera más legible y controlada, representando un valor que puede estar disponible en el futuro.
3. async/await: JavaScript permite escribir código asíncrono de manera más similar al código síncrono, usando la palabra clave await para esperar el resultado de una promise, lo que mejora la legibilidad y la manejabilidad del flujo asíncrono.

EVOLUCIÓN DE JAVASCRIPT Y EL ESTÁNDAR ECMASCRIPT

I. Lenguaje Interpretado vs. Compilado:

Los lenguajes compilados, como C++ o Java, son convertidos completamente a código máquina por un compilador antes de ser ejecutados, lo que suele resultar en una ejecución más rápida y optimizada.
Por otro lado, los lenguajes interpretados, como JavaScript, son procesados por un intérprete en tiempo real, lo que significa que el código fuente se ejecuta directamente sin necesidad de una fase de compilación previa. Esto proporciona ventajas en términos de flexibilidad y rapidez de desarrollo, ya que el código puede modificarse y ejecutarse de inmediato, pero también puede ser menos eficiente en términos de rendimiento.
Cabe destacar que en el caso de JS, actualmente en motores como V8 de Chrome lo compilan “just.in.time” (JIT) a código máquina mejorando la eficiencia y convirtiéndolo en un lenguaje “híbrido”.

II. Evolución del Estándar ECMAScript:

JavaScript está basado en el estándar ECMAScript, gestionado por ECMA International. Cada versión de ECMAScript introduce nuevas características que mejoran el lenguaje definiendo el comportamiento y las funcionalidades de este.
Cada versión de ECMAScript contribuyó a hacer a JavaScript más potente, eficiente y adecuado para el desarrollo de aplicaciones modernas. Las versiones son:

1. **(ES3) - 1999:** Esta versión consolidó a JavaScript como un estándar, introduciendo características clave como el manejo de cadenas y expresiones regulares, así como mejoras en la compatibilidad de los navegadores.
2. **(ES4) - :** Aunque se propuso en 2008, ES4 nunca fue publicado debido a desacuerdos en la comunidad.

3. **(ES5) - 2009:** Introdujo JSON nativo, strict mode para evitar malas prácticas, así como métodos para mejorar el manejo de arrays, como forEach(), map() y filter(). También agregó la posibilidad de definir propiedades con Object.defineProperty().
4. **(ES6) - 2015:** Esta versión trajo cambios significativos como las clases, módulos (import/export), promises, let y const, además de funciones flecha y la mejora del manejo de objetos y arrays.
5. **(ES7) - 2016:** Introdujo el operador exponenciación (**) y el método Array.prototype.includes() para mejorar la verificación de la existencia de elementos en un array.
6. **(ES8) - 2017:** Agregó características como async/await para un manejo más sencillo de la asincronía, y métodos adicionales para objetos y mejoras en las cadenas de texto
7. **(ES9) - 2018:** Introdujo la expresión de rest y spread para objetos, así como mejoras en la gestión asíncrona, y el soporte para reglas de agrupamiento en expresiones regulares con el flag s.

III. JavaScript vs. ECMAScript:

ECMAScript es el estándar que define las especificaciones del lenguaje, estableciendo las reglas y características básicas que deben seguir los motores de JavaScript para garantizar la compatibilidad y la interoperabilidad entre diferentes entornos. JavaScript es la implementación de esas especificaciones en los navegadores web y en otros entornos, como Node.js.
El estándar ECMAScript influye directamente en las versiones y características que los desarrolladores pueden utilizar, ya que cada nueva versión de ECMAScript introduce mejoras, nuevas funcionalidades y optimizaciones que luego se incorporan en los motores de JavaScript. Así, ECMAScript asegura que el lenguaje evolucione de manera consistente y que los cambios sean adoptados globalmente por todas las implementaciones de JavaScript.

IV. TypeScript y sus Características:

TypeScript es un superconjunto de JavaScript que añade tipado estático y otras características avanzadas, como interfaces y clases, para mejorar la calidad y mantenibilidad del código. Al compilarse a JavaScript, es completamente compatible con él, pero permite detectar errores en tiempo de desarrollo. Es una alternativa popular para proyectos grandes, ya que facilita la gestión de código y mejora la seguridad.

V. Ventajas y Desventajas de TypeScript:

TypeScript ofrece tipado estático, lo que ayuda a detectar errores en tiempo de desarrollo, mejorando la calidad del código y evitando errores comunes que pueden surgir en JavaScript avanzado debido a su tipado dinámico. Esta característica resulta especialmente útil en proyectos grandes o complejos, donde la claridad en las estructuras de datos y funciones facilita la comprensión del código y la colaboración entre equipos. Esto no ocurre en el caso del hospital, el cual es un proyecto pequeño con uno o dos colaboradores (caso de tarea en parejas).

TypeScript cuenta con herramientas de autocompletado y refactorización más potentes debido a su sistema de tipos, lo que mejora la productividad y reduce los errores en la evolución del código. También ofrece características como interfaces y tipos personalizados que no están presentes en JavaScript, lo que aporta una mayor estructura al código.

Una de las principales desventajas de usar TypeScript es que se introduce una curva de aprendizaje adicional, especialmente si los desarrolladores ya están acostumbrados a JavaScript avanzado. Aunque TypeScript ofrece ventajas en cuanto a seguridad y mantenimiento, la configuración y compilaciones adicionales pueden ralentizar el flujo de trabajo, especialmente en proyectos pequeños como el hospital o cuando se requiere desarrollo rápido.

Además, TypeScript no elimina completamente la posibilidad de errores, ya que las verificaciones de tipos son limitadas, y el código aún debe ser validado correctamente. En comparación, JavaScript avanzado permite una mayor flexibilidad y rapidez, sin la necesidad de etapas de compilación o configuraciones adicionales.

PERTINENCIA DE INTEGRAR JAVASCRIPT AVANZADO VS TYPESCRIPT

En un proyecto pequeño con un solo programador, el uso de TypeScript puede no ser la opción más eficiente. Aunque TypeScript ofrece beneficios como el tipado estático y la detección temprana de errores, estos beneficios no siempre se justifican en proyectos donde el código es relativamente simple y el alcance es limitado (caso del hospital). En estos casos, el desarrollo en JavaScript avanzado resulta más rápido y directo, sin necesidad de configurar o compilar el código, lo que permite centrarse en la funcionalidad de la aplicación sin preocuparse por los detalles adicionales del sistema de tipos.

Además, al ser un solo programador el encargado del proyecto, es menos probable que se presenten los desafíos de escalabilidad o mantenimiento que justifican el uso de TypeScript. JavaScript avanzado proporciona suficiente flexibilidad para manejar proyectos pequeños sin la sobrecarga que implica el aprendizaje y la implementación de TypeScript.

En resumen, para proyectos sencillos con un único desarrollador, JavaScript avanzado es una opción más rápida, sencilla y adecuada, ya que permite centrarse en la creación de funcionalidades sin las complejidades adicionales del tipado estático.

Implementación:

1. **Gestión de DOM y eventos:** Utilizar funciones modernas como `querySelector()`, `addEventListener()` y `classList` para manipular el DOM de manera eficiente y simplificar la gestión de eventos, usando delegación de eventos cuando sea posible.
2. **Funciones de orden superior y programación funcional:** Usar métodos como `map()`, `filter()`, y `reduce()` para manipular arrays de forma más declarativa y reducir el código repetitivo.
3. **Modularización del código:** organizar el código en componentes reutilizables, mejorando la mantenibilidad y escalabilidad del proyecto.

Otras implementaciones que se podrían hacer (que no lo haré) son:

- **Asincronía con Promesas y async/await:** Implementar **promesas** y **async/await** para manejar operaciones asíncronas de forma clara, mejorando la legibilidad del código y evitando el "callback hell".
- **Para optimización de rendimiento:** Implementar técnicas como **debounce** y **throttle** en eventos frecuentes para mejorar el rendimiento, y usar caché para reducir solicitudes innecesarias.
- **Herramientas modernas de desarrollo:** Usar **Babel** para compatibilidad entre navegadores, y **ESLint** y **Prettier** para mantener la calidad y consistencia del código.

Beneficios: Esta integración mejora la eficiencia y claridad del desarrollo, optimiza el rendimiento y asegura la escalabilidad del proyecto a medida que crece, sin la complejidad adicional de herramientas como TypeScript.