1. Generalidades del Lenguaje JavaScript

1. Historia de JavaScript

JavaScript fue creado por Brendan Eich en 1995, en solo 10 días, como un lenguaje de scripting para Netscape Navigator. Aunque fue diseñado inicialmente para agregar interactividad básica a los sitios web, ha evolucionado en un lenguaje versátil usado tanto en el frontend como en el backend (gracias a Node.js)

Actualmente es fundamental en el desarrollo web moderno gracias a que permite la interactividad del lado del cliente y la comunicación con la base de datos.

2. Uso de JavaScript en Navegadores Web

JavaScript de lado del cliente extiende el núcleo del lenguaje al proporcionar objetos para controlar un navegador y su Modelo de objetos de documento (DOM por Document Object Model).

Por ejemplo, las extensiones de lado del cliente permiten que una aplicación coloque elementos en un formulario HTML y responda a eventos del usuario, como clics del mouse, formularios para ingreso de datos y navegación de páginas.

3. Entornos Virtuales de JavaScript

JavaScript de lado del servidor amplía el núcleo del lenguaje al proporcionar objetos relevantes para ejecutar JavaScript en un servidor.

Por ejemplo, las extensiones de lado del servidor permiten que una aplicación se comunique con una base de datos, brinde continuidad de información de una invocación a otra de la aplicación o realice manipulación de archivos en un servidor.

Además de ejecutarse en los navegadores, JavaScript puede ejecutarse en servidores utilizando Node.js. Esto permite usar JavaScript para desarrollar aplicaciones backend, servicios REST, o incluso sistemas IoT.

4. Diferencias entre JavaScript y otros lenguajes:

Las principales diferencias por las cuales se destaca JavaScript en comparación con otros lenguaje de programación son:

- **Tipado dinámico**: a diferencia de lenguajes como Java o C++ que son tipados estáticamente, JavaScript es dinámico y flexible.
- **Interpretado:** JavaScript se ejecuta línea por línea en lugar de ser compilado previamente.
- **Multiparadigma:** aporta programación orientada a objetos(No hay distinción entre tipos de objetos), funcional e imperativa.
- Javascript no se puede escribir automáticamente en el disco duro.

5. Fortalezas y debilidades de JavaScript

El lenguaje de programación JavaScript cuenta con múltiples ventajas y algunas limitaciones, las cuales son descritas a continuación:

Ventajas:

- **Popularidad:** al ser uno de los lenguajes más utilizados, existe numerosa información disponible en la web, lo cual, facilita el aprendizaje y trabajo de los desarrolladores.
- **Velocidad**: al ejecutarse inmediatamente en el navegador, javascript se ejecuta más rápido en comparación a otros lenguajes.
- **Simplicidad:** JavaScript cuenta con una sintaxis simple y accesible, lo cual, también facilita su curva de aprendizaje.
- **Versatilidad:** al ser un lenguaje estándar en la industria web, se puede integrar de forma sencilla con otras tecnologías, además, es compatible con otros lenguajes de programación como Java y PHP.

Limitaciones:

- **Depuración:** la detección de errores al utilizar JavaScript puede ser más débil en comparación con otras tecnologías.
- **Compatibilidad con navegadores**: debido a que cada navegador interpreta JS de forma diferente, se debe tener consideraciones y generalmente realizar pruebas en más de un navegador web.
- **Seguridad:** como es un programa que se ejecuta en el lado del cliente, sus códigos podrían ser leídos por otros usuarios, lo cual, lo convierte en un lenguaje susceptible a ataques informáticos.

2. Evolución del Lenguaje JavaScript y el Estándar ECMAScript

1. JavaScript como lenguaje asíncrono

Javascript es un lenguaje asíncrono, ya que permite que se inicie una tarea que puede tener una larga duración, y a la vez, responder a otros eventos, esto es de gran utilidad, ya que permite que no sea necesario esperar que una tarea haya terminado para ejecutar otra.

Para que JavaScript tenga este comportamiento asincrónico, existen algunas soluciones, las cuales serán descritas a continuación:

- **Callbacks:** una función de callback es una función que se pasa dentro de otra función como un argumento y luego se invoca dentro de la función externa para completar algún tipo de rutina o acción. Si bien, es un método sencillo y fácil de implementar, cuenta con inconvenientes cuando se tienen múltiples operaciones asincrónicas en ese caso, se forma una estructura anidada que se vuelve un código complicado y difícil de leer.
- **Promises:** las promesas nacieron como una solución a los inconvenientes que generaban los callbacks, y se trata de una función que toma dos parámetros “resolve” y “reject” donde resolve indica éxito y que la promesa fue resuelta y

reject significa rechazo, lo cual ocurre cuando se produce un error. Uno de sus grandes beneficios es que luego de que finalice el proceso que se estaba ejecutando, se puede realizar cualquier otra tarea.

- **Async/Await:** es una función que se basa en promesas pero la hace más eficiente simplificando su comportamiento. Básicamente el funcionamiento de Async/Await es el siguiente; cuando se llama a una función async, esta devuelve una promesa. Cuando la función async devuelve un valor, la promesa se resolverá con el valor devuelto. Si la función async genera una excepción o algún valor, la promesa se rechazará con el valor generado. a su vez, se debe considerar que una función async puede contener una expresión await, la cual pausa la ejecución de la función asíncrona y espera la resolución de la promesa pasada, posterior a eso reanuda la ejecución de la función async y devuelve el valor resuelto.

2. Lenguaje Interpretado vs. Compilado

Los lenguajes compilados o interpretados son dos categorías de lenguajes de programación basados en cómo son procesados por el compilador o el intérprete. A continuación, se presentarán las características principales de cada tipo, y cómo funciona la interpretación del código JavaScript .

- **Lenguaje interpretado:** un lenguaje interpretado es aquel que es convertido a lenguaje de máquina a medida que es ejecutado. Dentro de sus características destacan un desarrollo más fácil, una mayor portabilidad y más flexibilidad. En el caso de JavaScript su código es interpretado, es decir, es directamente traducido a código de lenguaje de máquina subyacente mediante un motor de JavaScript.
- **Lenguaje compilado:** Los lenguajes compilados son aquellos que son convertidos directamente a código máquina que el procesador puede ejecutar, se caracterizan por brindar un rendimiento más rápido (aunque cada vez es menor la diferencia en relación a los lenguajes interpretados), más control y mejor seguridad. Algunos ejemplos de lenguajes compilados son C, C ++ y C#..

3. Evolución del Estándar ECMAScript

Desde el año 2015 ECMAScript es el estándar encargado de regir como debe ser interpretado y funcionar el lenguaje JavaScript, el cual es interpretado y procesado por una multitud de plataformas como navegadores y entornos de desarrollo. A lo largo de los años, ECMAScript ha evolucionado con el objetivo de mejorar el lenguaje incorporando nuevas funcionalidades y métodos que facilitan la codificación con este lenguaje y amplían sus alcances. A continuación se presentan las principales características y cambios que ha tenido desde su versión ES3 a ES9.

- **ES3 (1999):** esta versión se basó principalmente en la creación de la estandarización de JavaScript y sus funcionalidades esenciales.
- **ES4 (Abandonado) :** esta versión nunca se completó debido a desacuerdos sobre la dirección que debía tomar el lenguaje, lo cual, llevó a que no hubiera

una nueva versión de ECMAScript en varios años y se continuará trabajando en base a la versión ES3.

- **ES5 (2009):** significó un gran avance en comparación a su última versión estable, incluyendo soporte para JSON, propiedades getters/setters, manejo de “strict”, y algunos métodos nuevos relacionados a la manipulación de matrices y cadenas.
- **ES6 (2015):** considera como la evolución más importante de ECMAScript a la actualidad, esta versión incorporó la declaración de las variables let y const, se implementó una plantilla para las cadenas de texto utilizando tildes invertidas, arrow functions, Destructuring, símbolos, iteradores. entre otros. También, se destaca la introducción de conceptos ya mencionados como lo son las promesas.
- **ES7 (2016):** esta versión añadió dos cambios, la introducción del método includes para los arrays y el operador exponencial.
- **ES8 (2017):** en esta versión, se introdujeron cambios muy útiles, entre los que destacan el async y await (mencionado anteriormente), Object.entries y Object.values, padding de string, memoria compartida, acceso atómico y comas al final en listas de parámetros y llamadas a funciones.
- **ES9 (2018):** las actualizaciones más destacadas de esta versión tiene relación con la mejoras al trabajo con expresiones regulares, método finally en promesas y la asignación por destructuring en objetos cambiando lo indicado en la versión ES6, estas nuevas mejoras actualmente permiten la clonación, merge y destrucción de objetos.

4. JavaScript vs. ECMAScript

ECMAScript proporciona las reglas, detalles y directrices que un lenguaje de scripting debe seguir, para que se considere que cumple con ECMAScript. ECMA-262 es el estándar y ECMAScript es la especificación que este estándar representa.

JavaScript es un lenguaje de scripting de propósito general que se ajusta a la especificación ECMAScript.

ECMAScript influye directamente en las implementaciones de JavaScript. Cada versión define funcionalidades y sintaxis más avanzadas que los motores de JavaScript, como V8 (de Google Chrome y Node.js), deben implementar. Un ejemplo de esto es ES6 que introdujo varias características importantes, como el uso de let y const para variables, funciones de flecha, Promises, y class para la programación orientada a objetos.

5. TypeScript y sus Características

TypeScript es un lenguaje de programación fuertemente tipado construido sobre JavaScript, que incluye herramientas adicionales. TypeScript dota a JavaScript de varias características adicionales que hacen posible escribir código con menos errores, más sencillo, coherente y fácil de probar. TypeScript va un paso más allá de ECMAScript 6 o posterior y añade más funcionalidad a ECMAScript, como tipado fuerte, anotaciones o módulos. Adicionalmente, como TypeScript es un

superconjunto de JavaScript, todo lenguaje escrito en JavaScript es válido para TypeScript.

Algunas características de TypeScript son:

- **Tipado estático**: define el tipo de datos que maneja cada variable.
- **Compilación a JavaScript**: el código de TypeScript debe pasar por un proceso de compilación, generando un código JavaScript que pueda ser interpretado por el navegador. Dado a que es compilado, no hay diferencia en rendimiento frente a una aplicación escrita directamente en JavaScript.
- **Compatibilidad con JavaScript**: cualquier código válido de JavaScript también es código válido para TypeScript.
- **Sport para clases, interfaces y modularización**: lo que permite un mejor control y organización del código en un proyecto a escala.

En resumen, TypeScript es una alternativa a JavaScript que manteniendo su compatibilidad, agrega una serie de funcionalidades y que permiten escribir código más robusto que puede ayudar a disminuir la probabilidad de potenciales errores permitiendo una detección temprana de estos.

6. Ventajas y Desventajas de TypeScript

TypeScript provee varias ventajas, tales como las mencionadas en el punto 5, las cuales, permiten manejar proyectos más grandes de forma más consistente y robusta, evitando errores y haciendo el proyecto más fácil de mantener en el tiempo. Sin embargo, esto a su vez introduce un nivel de complejidad adicional y mayor tiempo de desarrollo que no siempre se encuentra justificado. Por lo anterior, es importante evaluar caso a caso si la decisión de su implementación es acorde a la escala del proyecto.

En el caso del proyecto del hospital, si la página web es sencilla y solo es utilizada como una landing page, que permite a los usuarios informarse sobre los servicio y contactar a la empresa, se podría justificar no utilizar TypeScript dada la envergadura del proyecto, sobretodo si es que hay restricciones temporales. Por el contrario, si el proyecto del hospital apunta a ser una aplicación web más compleja, que permita reservar horas, revisar exámenes, manejar datos de usuarios, etc, entonces el uso de TypeScript podría aportar un valor relevante al proyecto, disminuyendo la posibilidad de que se produzcan errores durante el desarrollo, manejo de datos, facilitando el testeo y la reutilización de componentes.

3. Análisis de la pertinencia de Integrar JavaScript Avanzado o TypeScript en el Proyecto

Utilizar JavaScript en el proyecto del hospital puede generar un gran aporte en cuanto a funcionalidades a la aplicación web. Sin embargo, todo depende de la envergadura del proyecto. Si el proyecto está pensado más bien como una landing page simple, podría bastar con utilizar HTML y CSS para el proyecto, sin embargo,

para poder implementar funcionalidades más complejas como, *carruseles* de imágenes, componentes con movimiento, botones con diversos comportamientos, sería necesario implementar el uso de JavaScript dentro de la aplicación web.

JavaScript puro es una herramienta muy potente que permite generar sitios web dinámicos que de otra forma no sería posible. También permite interactuar con APIs, modificar datos, etc, que agregan un valor tremendo a la aplicación, pero a su vez también incrementa su complejidad. Por esta razón existen frameworks altamente utilizados en la industria como lo son Angular y React que permiten la programación de una aplicación web en base a componentes reutilizables, facilitando al equipo de desarrollo mantener el proyecto e implementar nuevas funcionalidades.

Por otro lado, TypeScript es un lenguaje enfocado en proyectos de mayor complejidad, aporta gran valor, permitiendo que las aplicaciones escalen aún más al disminuir la posibilidad de errores en el manejo de los datos. También permite que cuando el proyecto ya se encuentra estable en producción sea más fácil de mantener, ya que al ser un lenguaje tipado es más fácil de leer y también permite que editores de código puedan aprovechar mejor sus funcionalidades advirtiendo al desarrollador si está utilizando mal una función debido a sus definiciones.

En resumen, todo depende de la escala del proyecto, sin embargo, utilizar TypeScript siempre es considerado una buena práctica y de ser posible lo ideal sería adoptar su uso, ya que permitiría aumentar el alcance y funcionalidades del proyecto desarrollado.