Curso: Front End - G3

Evaluación Modulo 3 – Taller 1.

Ana Moraga.

1. Generalidades del Lenguaje JavaScript

a) Historia de JavaScript:

JavaScript fue creado en 1995 por Brendan Eich mientras trabajaba en Netscape. El lenguaje se desarrolló inicialmente bajo el nombre de Mocha, luego fue renombrado a LiveScript, y finalmente se estableció como JavaScript. La creación de JavaScript fue impulsada por la necesidad de agregar interactividad a las páginas web, en un momento en que HTML y CSS eran los únicos lenguajes disponibles para el desarrollo web.

JavaScript se volvió fundamental para el desarrollo web moderno debido a su capacidad para manipular el DOM (Document Object Model), lo que permite a los desarrolladores crear aplicaciones web dinámicas y mejorar la experiencia del usuario. Su popularidad creció rápidamente, y en 1997 se estandarizó como ECMAScript por el ECMA International, lo que ayudó a establecer un marco común para su desarrollo y uso.

b) Uso de JavaScript en Navegadores Web:

JavaScript se ejecuta en el navegador del cliente, lo que significa que el código se procesa en la máquina del usuario, en lugar de en el servidor. Esto permite una interacción más rápida y una experiencia más fluida para el usuario sin necesidad de recargar la página.

Cuando un navegador carga una página web, descarga el código HTML, CSS y JavaScript. El motor de JavaScript del navegador interpreta y ejecuta el código, lo que permite la manipulación del contenido de la página, la respuesta a eventos (como clics y desplazamientos) y la comunicación con servidores a través de AJAX (Asynchronous JavaScript and XML).

c) Entornos Virtuales de JavaScript:

JavaScript no solo se limita a los navegadores; también se puede ejecutar en varios entornos. Algunos de los más destacados son:

- **Navegadores Web**: Aquí es donde JavaScript fue inicialmente adoptado y sigue siendo su uso más común.
- **Node.js**: Un entorno de ejecución que permite ejecutar JavaScript en el lado del servidor. Node.js utiliza el motor V8 de Google Chrome y es ideal para crear aplicaciones escalables y de alto rendimiento.
- **Deno**: Un entorno moderno y seguro para ejecutar JavaScript y TypeScript, que también se basa en el motor V8 pero ofrece una serie de mejoras en términos de seguridad y simplicidad.
- **Electron**: Permite crear aplicaciones de escritorio utilizando JavaScript, HTML y CSS, combinando el poder de Node.js y tecnologías web.

d) Diferencias entre JavaScript y otros lenguajes:

JavaScript se diferencia de otros lenguajes de programación en varios aspectos:

- **Propósito**: JavaScript fue diseñado específicamente para el desarrollo web, mientras que otros lenguajes (como Python o Java) son más genéricos y se utilizan en una variedad de aplicaciones, desde ciencia de datos hasta aplicaciones empresariales.
- **Uso**: JavaScript es predominantemente utilizado para crear interfaces de usuario interactivas en la web, mientras que lenguajes como C++ se utilizan para desarrollo de sistemas y aplicaciones de alto rendimiento.
- **Paradigmas soportados**: JavaScript es un lenguaje multiparadigma que soporta programación orientada a objetos, programación funcional y programación imperativa. Otros lenguajes, como Java, son más estrictos en su enfoque (principalmente orientado a objetos).

e) Fortalezas y debilidades de JavaScript:

- **Fortalezas:**
  - **Interactividad**: Permite crear aplicaciones web dinámicas y altamente interactivas.
  - **Amplio soporte**: Todos los navegadores modernos son compatibles con JavaScript, lo que lo convierte en un estándar de facto para el desarrollo web.
  - **Ecosistema rico**: Existen numerosas bibliotecas y frameworks (como React, Angular y Vue.js) que facilitan el desarrollo de aplicaciones complejas.
  - **Desarrollo asíncrono**: Su modelo de ejecución asíncrono permite manejar múltiples tareas simultáneamente sin bloquear la interfaz de usuario.
- **Debilidades:**
  - **Problemas de seguridad**: Al ser un lenguaje interpretado en el lado del cliente, puede ser vulnerable a ataques como XSS (Cross-Site Scripting).
  - **Dificultades en la depuración**: La naturaleza dinámica de JavaScript puede dificultar la identificación de errores en el código.
  - **Variabilidad de implementación**: Aunque hay estándares, las diferencias en la implementación de los motores de JavaScript pueden causar comportamientos inesperados en diferentes navegadores.

f) JavaScript como lenguaje asíncrono:

JavaScript es un lenguaje asíncrono, lo que significa que puede ejecutar operaciones sin bloquear el hilo principal de ejecución. Esto es crucial para mantener la interfaz de usuario receptiva durante operaciones como la carga de datos desde un servidor.

- **Callbacks**: Funciones que se pasan como argumentos a otras funciones y se ejecutan después de que se completa una tarea. Sin embargo, pueden llevar a lo que se conoce como "callback hell", donde la anidación de callbacks se vuelve difícil de manejar.

- **Promises**: Un objeto que representa la eventual finalización (o falla) de una operación asíncrona y su valor resultante. Las promesas permiten encadenar operaciones asíncronas de manera más legible.
- **Async/Await**: Una sintaxis que permite escribir código asíncrono de manera más similar al código síncrono. Las funciones marcadas con async devuelven una promesa, y dentro de ellas, se puede usar await para esperar la resolución de una promesa, facilitando la lectura y el mantenimiento del código.

2. Generalidades del Lenguaje JavaScript

a) Lenguaje Interpretado vs. Compilado:

JavaScript es un lenguaje interpretado, lo que significa que el código se ejecuta línea por línea en tiempo de ejecución. En contraste, los lenguajes compilados se traducen completamente a código máquina antes de su ejecución.

b) Evolución del Estándar ECMAScript:

La evolución del estándar ECMAScript ha sido significativa desde ES3 hasta ES9, introduciendo numerosas mejoras y características que han transformado el lenguaje JavaScript. A continuación, se detalla esta evolución:

- **ECMAScript 3 (ES3) – 1999**
  - Introdujo soporte para caracteres Unicode.
  - Mejoró el manejo de errores con declaraciones try/catch.
  - Agregó expresiones regulares.
  - Implementó el switch case y el bucle do-while.
- **ECMAScript 5 (ES5) – 2009**
  - Introdujo el "strict mode" para mejorar la detección de errores.
  - Agregó soporte para JSON.
  - Implementó getters y setters en JavaScript.
  - Añadió nuevos métodos para manipular objetos, como Object.keys().
- **ECMAScript 2015 (ES6) – 2015**
  - Introdujo las declaraciones let y const para un mejor manejo del ámbito de las variables.
  - Agregó clases para facilitar la programación orientada a objetos.
  - Implementó las funciones flecha (arrow functions) para una sintaxis más concisa.
  - Añadió los módulos import y export para mejorar la organización del código.
  - Introdujo el bucle for...of.
  - Implementó las promesas para un mejor manejo de operaciones asíncronas.
- **ECMAScript 2016 (ES7) – 2016**
  - Agregó el método Array.prototype.includes para facilitar la búsqueda en arrays.
  - Introdujo el operador de exponenciación (**) para cálculos matemáticos más sencillos.
- **ECMAScript 2017 (ES8) – 2017**
  - Implementó async/await para simplificar la programación asíncrona.

    - Agregó los métodos Object.values y Object.entries para facilitar el trabajo con objetos.
    - Introdujo Object.getOwnPropertiesDescriptors para obtener todas las propiedades de un objeto.
- **ECMAScript 2018 (ES9) – 2018**
    - Agregó el operador de propagación (spread operator) para objetos.
    - Implementó los parámetros rest para funciones.
    - Introdujo la iteración asíncrona con for-await-of.
    - Agregó el método Promise.finally() para manejar la finalización de promesas.

c) JavaScript vs. ECMAScript:

JavaScript es la implementación más popular y conocida del estándar ECMAScript. Mientras que ECMAScript es la especificación que define cómo debe funcionar y comportarse el lenguaje, JavaScript es la aplicación práctica de este estándar.

Cada nueva versión de ECMAScript introduce características y funcionalidades que se incorporan a JavaScript.

ECMAScript 6 (ES6 o ES2015) marcó un hito importante, introduciendo numerosas mejoras como:

- Declaraciones let y const.
- Funciones flecha.
- Clases.
- Promesas.
- Módulos import/export.

Los navegadores web implementan motores de JavaScript que interpretan el lenguaje según las especificaciones de ECMAScript. Esto permite una mayor consistencia en el comportamiento de JavaScript entre diferentes plataformas y navegadores.

d) TypeScript y sus Características:

TypeScript es un lenguaje de programación de código abierto desarrollado y mantenido por Microsoft. Es un superconjunto de JavaScript que añade características adicionales al lenguaje, principalmente el tipado estático opcional.

Principales características de TypeScript

- Tipado estático: TypeScript introduce un sistema de tipos estáticos que permite a los desarrolladores definir explícitamente los tipos de datos para variables, parámetros de funciones y propiedades de objetos. Esto ayuda a:
    - Detectar errores en tiempo de compilación.
    - Mejorar la legibilidad del código.
    - Facilitar el mantenimiento de proyectos grandes.
- Compatibilidad con JavaScript: todo código JavaScript válido es también código TypeScript válido. Esto permite una adopción gradual de TypeScript en proyectos existentes de JavaScript.

- Compilación a JavaScript: TypeScript se compila a JavaScript estándar, lo que lo hace compatible con cualquier entorno que soporte JavaScript, incluyendo navegadores web y servidores Node.js.
- Características de lenguaje avanzadas: TypeScript incorpora características modernas de programación, como:
    - Clases y módulos.
    - Interfaces.
    - Genéricos.
    - Decoradores.
    - Espacios de nombres.
- Mejor soporte de herramientas: el tipado estático de TypeScript permite un mejor autocompletado, refactorización y detección de errores en los entornos de desarrollo integrados (IDEs).

Por qué TypeScript es una alternativa a JavaScript

- Detección temprana de errores: El sistema de tipos ayuda a identificar problemas potenciales durante el desarrollo, antes de que el código se ejecute.
- Mejora la mantenibilidad: El código tipado es más fácil de entender y mantener, especialmente en proyectos grandes y complejos.
- Mejor experiencia de desarrollo: Las características avanzadas y el mejor soporte de herramientas hacen que el proceso de desarrollo sea más eficiente y agradable.
- Ecosistema en crecimiento: La popularidad de TypeScript ha llevado a una mejor compatibilidad con bibliotecas y frameworks, facilitando su adopción en diversos proyectos.
- Evolución constante: Al no estar limitado por estándares de la misma manera que JavaScript, TypeScript puede evolucionar más rápidamente e incorporar características de lenguajes modernos.

En resumen, TypeScript ofrece una alternativa robusta a JavaScript, especialmente para proyectos grandes y complejos, al proporcionar un conjunto de herramientas que mejoran la calidad del código, la productividad del desarrollador y la mantenibilidad a largo plazo.

e) Ventajas y Desventajas de TypeScript:

Ventajas de TypeScript

- Tipado estático y seguridad de tipos:

  TypeScript introduce un sistema de tipado estático que ofrece varios beneficios:

    - Detección temprana de errores en tiempo de compilación, lo que puede ahorrar horas de depuración.
    - Mejora la calidad y mantenibilidad del código al hacer que sea más fácil entender la intención y el funcionamiento de este.
    - Proporciona autocompletado y sugerencias más precisas en los entornos de desarrollo integrados (IDEs).

- Mejora en la productividad y eficiencia:
    - Las herramientas de desarrollo mejoradas, como Visual Studio Code, ofrecen refactorización más efectiva y navegación en el código.
    - La capacidad de autocompletado inteligente permite codificar más rápido y con menos errores.

Desventajas de TypeScript

- Curva de aprendizaje: adoptar TypeScript puede requerir tiempo para que los desarrolladores se acostumbren al sistema de tipado estático, especialmente aquellos más familiarizados con JavaScript puro.
- Requisito de compilación: TypeScript necesita ser compilado a JavaScript antes de poder ejecutarse en un navegador o entorno Node.js, lo que añade un paso adicional al flujo de trabajo de desarrollo.
- Posible sobrecarga de anotaciones de tipo: en algunos casos, el requisito de anotar tipos puede resultar verboso y agregar sobrecarga visual al código.

Para ejemplificar las diferencias se tiene una función en Javascript y TypeScript:

Javascript:
```
function calcularAreaCirculo(radio) {
  return Math.PI * radio * radio;
}
console.log(calcularAreaCirculo(5)); // Output: 78.53981633974483
```

TypeScript:
```
function calcularAreaCirculo(radio: number): number {
  return Math.PI * radio * radio;
}
console.log(calcularAreaCirculo(5)); // Output: 78.53981633974483
```

Explicación de las diferencias

- **Tipado**: La principal diferencia es que en TypeScript, especificamos el tipo de dato para el parámetro radio y el tipo de retorno de la función. En JavaScript, los tipos son dinámicos y no se declaran explícitamente.
- **Seguridad de tipos**: En TypeScript, si intentáramos llamar a la función con un argumento que no sea un número (por ejemplo, una cadena), el compilador nos advertiría del error antes de ejecutar el código. En JavaScript, no habría advertencia y podría llevar a errores en tiempo de ejecución.
- **Intellisense y autocompletado**: En un entorno de desarrollo integrado (IDE), TypeScript proporciona mejor soporte para autocompletado y sugerencias basadas en los tipos declarados.
- **Compilación**: El código TypeScript necesita ser compilado a JavaScript antes de poder ejecutarse en un navegador o en Node.js. El código JavaScript puede ejecutarse directamente sin este paso adicional.
- **Legibilidad**: El código TypeScript es más explícito sobre los tipos de datos esperados, lo que puede mejorar la legibilidad y el mantenimiento del código, especialmente en proyectos más grandes.

3. **Análisis de la Pertinencia de Integrar JavaScript Avanzado o TypeScript en el Proyecto del sitio web del hospital.**

a) **Ventajas de utilizar JavaScript avanzado o TypeScript en el proyecto.**
- Mayor organización del código.
- Mejora en la calidad y mantenibilidad del código: el tipado estático de TypeScript permite detectar errores en tiempo de compilación, lo que resulta en un código más robusto y confiable.
- Aumento de la productividad del desarrollador: las herramientas de desarrollo mejoradas, como el autocompletado inteligente y la refactorización más efectiva, agilizan el proceso de codificación.

b) **Desventajas o posibles dificultades que podría traer la implementación de estas tecnologías.**
- Curva de aprendizaje: el desarrollador puede enfrentar una curva de aprendizaje inicial, especialmente con TypeScript, que requiere comprender conceptos como tipado estático, interfaces y genéricos
- Compatibilidad entre navegadores: aunque ha mejorado, aún pueden existir diferencias en cómo los distintos navegadores interpretan y ejecutan el código JavaScript, lo que puede requerir ajustes y pruebas adicionales.
- Dependencia de herramientas: Al adoptar TypeScript, los proyectos se vuelven dependientes de esta herramienta, lo que podría afectar el desarrollo si surgen problemas de mantenimiento o cambios significativos en TypeScript.

c) **Conclusión**: ¿Es recomendable incluir JavaScript avanzado o TypeScript en el proyecto?

Basándome en el análisis de las ventajas y desventajas de JavaScript avanzado y TypeScript, considero que es altamente recomendable incluir TypeScript en el proyecto del hospital. Las razones principales son:

- Seguridad y detección temprana de errores: TypeScript ofrece un sistema de tipado estático que permite detectar errores en tiempo de compilación.
- Mejor mantenibilidad y escalabilidad: el tipado estático y las características avanzadas de TypeScript, como interfaces y clases, mejoran significativamente la legibilidad y estructura del código.
- Productividad mejorada: TypeScript proporciona herramientas de desarrollo mejoradas, como autocompletado más preciso y refactorización más efectiva.
- Compatibilidad con JavaScript moderno: TypeScript es compatible con las últimas características de ECMAScript, permitiendo utilizar funcionalidades modernas de JavaScript con la seguridad adicional del tipado estático.

Consideraciones importantes

A pesar de las ventajas, es importante tener en cuenta:

- La curva de aprendizaje inicial para el desarrollador.

- El tiempo adicional de compilación que puede afectar el flujo de trabajo.
- La necesidad de configuración adicional en el proyecto.

Sin embargo, estos desafíos son superados por los beneficios a largo plazo, especialmente en un proyecto crítico como un sistema hospitalario.

En conclusión, la adopción de TypeScript en el proyecto del hospital proporcionará una base más sólida, segura y mantenible para el desarrollo del sistema, justificando ampliamente su implementación a pesar de los desafíos iniciales.