Entornos de Desarrollo

Tarea 1: Lenguajes de Programación (TypeScript)

En esta actividad hablaremos del lenguaje de programación TypeScript, desde su origen hasta sus versiones más actuales. Hablaremos de sus características más fundamentales y describiremos sus frameworks y bibliotecas, así como algunas de sus ventajas y desventajas.

Origen de Typescript:

TypeScript fue presentado por Microsoft en el año 2012 como respuesta a las limitaciones de JavaScript, por lo que opera a un nivel superior de este. El proyecto fue iniciado por Anders Hejlsberg en el año 2012 al darse cuenta de que dentro de la compañía los equipos de desarrolladores pedían la creación de una tecnología llamada Script#. La idea de esto era desarrollar un transpilador de lenguaje C# a JavaScript, puesto que los proyectos desarrollados en este último lenguaje eran transpilados por los trabajadores desde C# debido a las muchas limitaciones y problemas para llevar a cabo proyectos en JavaScript. En palabras de Anders, “me pregunté, ¿qué está tan roto en JavaScript para que alguien quiera alejarse tanto de su objetivo solamente para obtener una mejor experiencia?”

Paradigmas y características fundamentales de TypeScript

TypeScript fue pensado como una extensión tipada de JavaScript con el objetivo de otorgarle un refinamiento pero más estructural y genérico, y orientado a proyectos grandes. Por esto, es un lenguaje ***transpilado***, es decir, su compilador lo transforma a JavaScript.

A continuación una lista resumida de diferencias entre ambos:

- Tipado Estático y Opcional: JavaScript no verifica de forma nativa los tipos de las variables a la hora de ejecutar el código, por lo que no es necesario especificarlos. Esto se conoce como tipado dinámico que, si bien agiliza el proceso, da lugar a errores en el compilado con más frecuencia. TypeScript soluciona este problema permitiendo definir tipos para variables, funciones y objetos, lo que mejora la detección de errores durante el desarrollo.

- Compilado a JavaScript: Tal como se mencionó previamente, el código TypeScript fue desarrollado como respuesta a los trabajadores de Microsoft transpirando desde C#, así este lenguaje es transpilado a JavaScript para ser ejecutado en cualquier entorno compatible con el mismo.

- Orientado a Objetos: Aunque JavaScript sí ofrece soporte para clases, herencia y métodos, este es en sí mismo muy básico ya que JavaScript no tiene integrada la función para crear interfaces directamente. TypeScript ofrece soporte completo para todo lo anteriormente mencionado, especialmente para crear interfaces que definan qué métodos y propiedades debe tener una clase que la implemente.

```typescript
interface Persona {
  nombre: string;
  edad: number;
  saludar(): void;
}
```

(Ejemplo de interfaz en TypeScript, que permite establecer los métodos y atributos)

- Compatible con ECMAScript: Adopta características modernas de ECMAScript (el estándar que define cómo debe ser el comportamiento y la sintaxis de lenguajes de programación basados en JavaScript) como async/await, módulos y funciones de flecha.

- Escalable: Ideal para proyectos grandes gracias a su tipado estático y herramientas que facilitan el mantenimiento del código.

- Modularización: TS ofrece un soporte directo para módulos, mientras que JS lo hace a través de ECMAScript 6.

- Funcional: Igual que JavaScript, Typescript es un lenguaje funcional, ya que soporta funciones de orden superior y programación inmutable. in embargo, Typescript te permite especificar los tipos de las funciones que puedes pasar como argumento o devolver, lo que ofrece **mayor seguridad y prevención de errores** durante el desarrollo.

- Condicional: Tanto JavaScript como TypeScript soportan funciones, bucles y condicionales. Sin embargo, TypeScript agrega características adicionales como el tipado estático del que hemos hablado anteriormente, en contraposición al tipado dinámico de JavaScript.

- Tuplas: Las tuplas son estructuras de datos de tamaño y estructura fijos que en algunos lenguajes de programación permite almacenar elementos de diferentes tipos en un solo contenedor. JavaScript no soporta tuplas, en lugar de ello utiliza arreglos, aunque al ser estos de tipo dinámico, se presenta el mismo problema del que ya hemos hablado: mayor flexibilidad pero menor seguridad.

Ejemplo de tupla en JavaScript vs TypeScript:

```javascript
let miArreglo = [1, "hola", true];  // Un arreglo con diferentes tipos de datos
console.log(miArreglo[0]);  // 1
console.log(miArreglo[1]);  // "hola"
console.log(miArreglo[2]);  // true
```

Como vemos, aquí se utilizan arreglos para almacenar los datos, pero al no tener estos ningún tipo asignado debido al tipado dinámico, carecen de estructura fija.

```typescript
let miTupla: [number, string, boolean] = [1, "hola", true];

console.log(miTupla[0]);  // 1
console.log(miTupla[1]);  // "hola"
console.log(miTupla[2]);  // true

// Mi tupla no permite más o menos elementos que los definidos:
miTupla = [2, "adiós", false];  // Correcto
// miTupla = [2, "adiós"];  // Error: La tupla debe tener 3 elementos
```

Aquí vemos como, en efecto, se pueden especificar los tipos de las variables.

Limitaciones de TypeScript

- Curva de Aprendizaje: Para desarrolladores sin experiencia en lenguajes tipados, puede ser complejo. De hecho su uso se recomienda para proyectos grandes, por lo que es poco probable que nadie aprenda a utilizarlo hasta que entre en el mundo laboral, y para entonces es presumible que, como mínimo, ya posea un nivel moderado de conocimientos en programación.

- Sobrecarga Inicial: La declaración de tipos puede hacer que el desarrollo inicial sea más lento.

- Configuración: Requiere configurar herramientas como el compilador y linters.

Aplicaciones y usos comunes

El uso más común de TypeScript se encuentra en el desarrollo de aplicaciones web frontend (es decir, lo que ocurre en el lado de usuario, la parte visible con la que interactuamos)

Muchos frameworks (es decir, herramientas, bibliotecas y convenciones básicas para permitir al usuario desarrollar de forma eficiente) tienen un fuerte soporte para TypeScript. De entre estos destacamos Angular, React y Vue.js, que aprovechan el tipado estático y las interfaces de TypeScript para mejorar la productividad de los desarrolladores y reducir los errores en tiempo de ejecución.

Angular: Desde su versión 2, Angular ha sido diseñado para trabajar con TypeScript. Angular permite crear aplicaciones grandes y complejas de forma más mantenible y estructurada. El uso de decoradores en TypeScript en Angular permite una programación más sencilla y ordenada.

Facilita un desarrollo rápido y organizado, con rendimiento mejorado gracias a su sistema de enrutado. Sin embargo, posee una curva de aprendizaje empinada y compleja.

Ejemplo de Angular con TypeScript:

```
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title: string = 'Bienvenidos a mi aplicación Angular!';
}
```

React: Permite crear componentes de UI (Interfaz de Usuario) que encapsulan tanto la lógica como la presentación. Estos componentes pueden ser reutilizados a lo largo de toda la aplicación, lo que facilita el desarrollo, mantenimiento y escalabilidad de la misma. Podemos ver un ejemplo de esto en la siguiente imagen:

```
interface Props {
  nombre: string;
}

const Saludo: React.FC<Props> = ({ nombre }) => {
  return <h1>Hola, {nombre}!</h1>;
};
```

La constante “Saludo” devuelve el mensaje “Hola” seguido del nombre, declarado en la interfaz como String. Tal como ya se ha dicho, esto permite reutilización. React posee buena escalabilidad y rendimiento, además de un respaldo muy activo por parte de la comunidad y ser compatible con herramientas modernas. No obstante, no es la herramienta más intuitiva y su curva de aprendizaje es bastante elevada. En definitiva, se utiliza para aplicaciones tanto grandes como pequeñas y especialmente aplicaciones móviles y sitios web dinámicos.

Vue.js: Es parecido a React, aunque permite más facilidad y sencillez a la hora de crear las interfaces:

```
import { Vue, Component } from 'vue-property-decorator';

@Component
export default class MiComponente extends Vue {
  mensaje: string = '¡Hola desde TypeScript!';

  mostrarMensaje() {
    console.log(this.mensaje);
  }
}
```

Vue.js se utiliza en aplicaciones web tanto grandes como pequeñas (dashboards, paneles de control, aplicaciones móviles…) Es una opción más sencilla de aprender que la anterior, con una comunidad activa y que se actualiza constantemente, aparte de su flexibilidad y escalabilidad, aunque carece de respaldo corporativo y su ecosistema es mucho más reducido debido a su limitada popularidad (más centrada en países asiáticos)

Otros usos comunes de TypeScript son:

- Desarrollo Web: Principalmente en frontend con frameworks como Angular, React y Vue.

- Backend: Utilizado con Node.js para construir APIs robustas.

- Aplicaciones Móviles: A través de frameworks como Ionic y React Native.

- Aplicaciones de Escritorio: Gracias a Electron, es posible desarrollar aplicaciones multiplataforma.

Estado Actual y librerías más comunes

A día de hoy, la versión más reciente de TypeScript es 5.2, lanzada en 2023, si bien en Diciembre de este año 2024 entrará en efecto la versión 5.3. Sin entrar en excesivo detalle, algunas librerías populares son:

- TypeORM: ORM para bases de datos.

- NestJS: Framework para backend.

- RxJS: Programación reactiva.

- Lodash: Utilidades para manipulación de datos.

- Jest: Framework de pruebas.

Resumen final:

Como resumen, se puede decir que TypeScript, siendo un superset de JavaScript, ofrece seguridad sacrificando dinamismo y flexibilidad, lo que de cara a grandes proyectos ofrece muchas más posibilidades pero es mucho más recomendable pero no tan intuitivo para un programador más novato. Es así mismo un lenguaje de paradigma funcional, imperativo y orientado a objetos en constante desarrollo que emplea frameworks y bibliotecas con buen respaldo tanto de la comunidad internacional como de las grandes empresas y corporaciones. Su uso es adecuado tanto para grandes como pequeños proyectos y aplicaciones multiplataforma de todo tipo. Su característica más destacable es su capacidad para definir interfaces y su tipado estático y opcional que ofrece más seguridad contra posibles errores.