**TYPESCRIPT**

**Introducción a TypeScript:**

TypeScript es un lenguaje de programación desarrollado por Microsoft que se ha convertido en una herramienta esencial en el mundo del desarrollo web y de aplicaciones. A diferencia de JavaScript, TypeScript es un superset que agrega tipado estático al código, lo que significa que puedes declarar el tipo de datos de tus variables, parámetros de función y otros elementos de tu código. Esta característica facilita la detección temprana de errores y proporciona un entorno más robusto para el desarrollo de aplicaciones. En tu exposición, puedes explicar cómo TypeScript se compila a JavaScript estándar, lo que lo hace compatible con todos los navegadores y entornos que admiten JavaScript. Además, es importante destacar que TypeScript es de código abierto y se encuentra en constante evolución.

**Historia y evolución de TypeScript:**

La historia de TypeScript es una parte interesante de su contexto. Puedes comenzar por mencionar que fue desarrollado por Anders Hejlsberg, conocido por su trabajo en el desarrollo de Delphi y C#. TypeScript se presentó por primera vez en 2012 y ha experimentado un crecimiento constante desde entonces. A lo largo de los años, se han lanzado varias versiones, cada una con nuevas características y mejoras. Puedes destacar hitos importantes en su evolución, como la adopción en proyectos de gran envergadura y la estrecha colaboración con la comunidad de desarrollo. Esto demuestra cómo TypeScript ha logrado mantenerse relevante y útil en el cambiante mundo del desarrollo de software.

**¿Por qué usar TypeScript?**

TypeScript ofrece una serie de ventajas que lo hacen una elección atractiva en el desarrollo de software. En primer lugar, su característica principal es la introducción del tipado estático, lo que significa que puedes declarar el tipo de datos de tus variables, parámetros de función y otros elementos en tu código. Esto conlleva varios beneficios, como la detección temprana de errores, lo que ayuda a evitar problemas comunes en el desarrollo de aplicaciones. Además, el tipado estático hace que el código sea más legible y mantenible, ya que proporciona una documentación viva y autoexplicativa. TypeScript también ofrece un entorno de desarrollo más robusto gracias a su integración con herramientas como Visual Studio Code, que proporciona características avanzadas de autocompletado, refactorización y depuración. Otra ventaja es su amplia comunidad y la disponibilidad de definiciones de tipos para muchas bibliotecas y frameworks populares, lo que facilita la integración de terceros en tus proyectos. En resumen, el uso de TypeScript mejora la calidad del código, acelera el desarrollo y aumenta la productividad del equipo, lo que lo convierte en una elección sólida para desarrolladores de aplicaciones modernas.

**Sintaxis básica:**

Declaración de variables (let, const): En TypeScript, puedes declarar variables utilizando las palabras clave "let" y "const". "let" se usa para variables que pueden cambiar su valor a lo largo del tiempo, mientras que "const" se utiliza para constantes cuyo valor no puede modificarse una vez asignado. Esta característica mejora la seguridad y la claridad del código, ya que ayuda a evitar la reasignación accidental de variables.

Tipos de datos básicos (number, string, boolean): TypeScript admite tipos de datos básicos similares a JavaScript, como números (number), cadenas de texto (string) y valores booleanos (boolean). Al declarar una variable con un tipo de dato específico, TypeScript verifica que solo se asignen valores compatibles con ese tipo, lo que ayuda a prevenir errores de tipo comunes. Por ejemplo, si defines una variable como "let edad: number = 30;", TypeScript te dará un error si intentas asignar una cadena de texto a esa variable.

Uso de funciones: En TypeScript, puedes definir funciones de manera similar a JavaScript, pero también puedes especificar los tipos de datos que espera una función para sus parámetros y el tipo de dato que devuelve. Esto se conoce como "anotación de tipos" y es una característica fundamental para garantizar la seguridad de tipos en las funciones. Por ejemplo, puedes escribir una función como "function suma(a: number, b: number): number { return a + b; }", donde se espera que "a" y "b" sean números y la función devuelve un número.

**Tipos de datos en TypeScript:**

**Tipos primitivos:** En TypeScript, los tipos primitivos se refieren a los tipos de datos básicos que ya mencionamos, como number, string y boolean. Estos tipos son simples y representan valores individuales.

**Tipos compuestos** (arrays, objetos, tuplas, uniones, intersecciones): Además de los tipos primitivos, TypeScript ofrece tipos compuestos que permiten estructurar datos de manera más compleja. Los tipos compuestos incluyen arrays, que representan colecciones ordenadas de elementos; objetos, que definen estructuras de datos con propiedades y valores; tuplas, que son arrays con una longitud fija y tipos específicos en cada posición; uniones, que permiten valores de múltiplos tipos; e intersecciones, que combinan múltiples tipos en uno.

**Tipos personalizados** (interfaces y tipos): TypeScript permite crear tipos personalizados mediante interfaces y tipos (también conocidos como "type"). Las interfaces permiten definir una estructura de objeto, especificando las propiedades y sus tipos. Los tipos, por otro lado, son más flexibles y puedenrepresentar cualquier tipo de estructura de datos, no solo objetos. Estos tipos personalizados son especialmente útiles para definir contratos de datos claros y reutilizables en tu código.

**Anotaciones de tipos:**

Uso de anotaciones de tipos: En TypeScript, las anotaciones de tipos son una parte fundamental de la sintaxis. Puedes utilizarlas para declarar el tipo de datos que una variable, función o estructura de datos espera o devuelve. Esto proporciona claridad en el código y permite a TypeScript realizar verificaciones estáticas para prevenir errores de tipo. Por ejemplo, al declarar una variable, puedes anotar su tipo como "let edad: number = 30;", lo que significa que "edad" debe contener un valor numérico. Las anotaciones de tipos se utilizan en todo el código para mejorar la seguridad y la legibilidad.

Inferencia de tipos: TypeScript también es capaz de inferir tipos automáticamente en muchas situaciones. Esto significa que, en lugar de especificar explícitamente el tipo de una variable, TypeScript puede deducirlo en función del valor que se le asigna. Esta característica ahorra tiempo y esfuerzo al escribir código, ya que TypeScript puede identificar el tipo correcto a partir de la asignación. Por ejemplo, si declaras "let nombre = 'Juan';", TypeScript inferirá que "nombre" es de tipo string.

**Ventajas de TypeScript:**

Detección de errores estáticos: Una de las ventajas más notables de TypeScript es su capacidad para detectar errores de tipo en tiempo de compilación. Esto significa que puedes identificar y solucionar problemas antes de que el código se ejecute, lo que aumenta la robustez de las aplicaciones y ahorra tiempo y recursos en fases posteriores del desarrollo.

Mejor autocompletado y sugerencias en el IDE: TypeScript proporciona un soporte avanzado en los entornos de desarrollo integrados (IDE) como Visual Studio Code. Gracias a las anotaciones de tipos, el IDE puede ofrecer autocompletado inteligente, sugerencias de código y una mejor documentación en línea, lo que agiliza la escritura de código y ayuda a los desarrolladores a entender y utilizar bibliotecas y frameworks de manera más efectiva.

Mayor legibilidad y mantenibilidad del código: Con las anotaciones de tipos y la estructura más sólida que aporta TypeScript, el código se vuelve más claro y fácil de entender. Esto facilita la colaboración en proyectos, la identificación de problemas y el mantenimiento a largo plazo, ya que los desarrolladores pueden comprender rápidamente cómo se supone que debe funcionar el código.

Mayor capacidad de refactorización: TypeScript permite realizar refactorizaciones con confianza, ya que las anotaciones de tipos garantizan que los cambios realizados se apliquen de manera coherente en todo el código. Esto facilita la reorganización, optimización y actualización de proyectos a medida que evolucionan con el tiempo.

**Configuración y herramientas:**

**Configuración del archivo tsconfig.json:** El archivo "tsconfig.json" es fundamental en un proyecto TypeScript. En él, se especifican las opciones de configuración que TypeScript debe seguir al compilar el código. Puedes definir aspectos como la versión de ECMAScript a la que se debe compilar (por ejemplo, ES5 o ES6), la ubicación de los archivos de origen TypeScript, la ubicación de los archivos de salida JavaScript, las opciones de módulos, las rutas de inclusión y exclusión, entre otras. La configuración adecuada en este archivo es esencial para asegurarse de que TypeScript se comporte de acuerdo a las necesidades específicas de tu proyecto.

**Compilación de TypeScript a JavaScript:** Una de las tareas esenciales en el desarrollo con TypeScript es la compilación del código TypeScript a JavaScript. Esto se logra mediante el comando "tsc" (TypeScript Compiler) o configurando herramientas de compilación automática. Al ejecutar el comando "tsc" en un proyecto con un archivo "tsconfig.json" válido, TypeScript compilará los archivos TypeScript en JavaScript según las opciones de configuración especificadas. Esto es fundamental para que el código TypeScript sea ejecutable en navegadores web u otros entornos que requieren JavaScript puro. La compilación a JavaScript también puede incluir transpilación de características TypeScript avanzadas a versiones de JavaScript compatibles con navegadores más antiguos, si se configura de esa manera.

**Uso de linters y formateadores de código (Eslint, Prettier):** En el desarrollo de software, mantener un código limpio y consistente es esencial. Para lograrlo, es común utilizar herramientas como Eslint y Prettier. Eslint es un linter que ayuda a detectar y corregir problemas en el código, como errores de sintaxis, estilo y convenciones de codificación. Prettier, por otro lado, es un formateador de código que garantiza que el código se vea limpio y organizado siguiendo un estilo de codificación específico. Ambas herramientas son altamente configurables y se pueden integrar fácilmente en proyectos TypeScript para mantener la calidad del código y facilitar la colaboración en equipos de desarrollo.

**Clases y objetos:**

**Uso de clases en TypeScript:** TypeScript permite utilizar clases como una parte fundamental de la programación orientada a objetos. Puedes definir clases para representar objetos y encapsular propiedades y métodos relacionados. La sintaxis es similar a la de JavaScript, pero TypeScript agrega la capacidad de anotar los tipos de propiedades y argumentos de métodos, lo que brinda seguridad de tipos. Por ejemplo, puedes crear una clase "Persona" con propiedades como "nombre" y "edad", y métodos como "saludar". Luego, puedes crear instancias de esta clase para representar personas reales en tu aplicación.

**Herencia y polimorfismo:** TypeScript admite herencia de clases, lo que permite que una clase herede propiedades y métodos de otra. Esto fomenta la reutilización de código y la organización jerárquica de clases. El polimorfismo, por otro lado, permite que objetos de diferentes clases sean tratados de manera uniforme si comparten una interfaz común. Esto es especialmente útil para la creación de abstracciones y la escritura de código genérico. Por ejemplo, puedes tener una clase base "Vehiculo" con propiedades y métodos comunes, y luego crear clases derivadas como "Coche" y "Motocicleta" que hereden de "Vehiculo". Esto facilita la gestión de vehículos de diferentes tipos de manera consistente.

**Módulos y Namespaces:**

**Organización del código en módulos:** Los módulos en TypeScript son una forma de organizar y estructurar el código en partes separadas y reutilizables. Puedes dividir tu código en módulos para mantenerlo limpio y modular. Cada módulo puede contener clases, funciones, variables y otros elementos, y, de esta manera, puedes evitar la contaminación del espacio global de nombres. Esto es especialmente útil en proyectos grandes y complejos, donde necesitas dividir la lógica en piezas más pequeñas y manejables. Los módulos permiten una mejor organización, facilitan la reutilización del código y mejoran la mantenibilidad.

**Uso de import y export:** En TypeScript, puedes utilizar las palabras clave "import" y "export" para trabajar con módulos. Con "export", puedes exponer elementos (como funciones, clases o variables) desde un módulo para que estén disponibles para otros módulos. Por otro lado, con "import", puedes importar estos elementos en otros módulos y utilizarlos. Esto facilita la composición de aplicaciones a partir de piezas modulares y garantiza la encapsulación y la protección del alcance de las variables. Por ejemplo, si tienes un módulo llamado "utils.ts" con una función de utilidad "calcular", puedes exportarla con "export function calcular(...)", y en otro módulo, puedes importarla con "import { calcular } from './utils'".

**Namespaces y su uso:** Los namespaces son otra forma de organizar el código en TypeScript, pero se utilizan principalmente para agrupar tipos y valores relacionados. Los namespaces evitan conflictos de nombres al proporcionar un contenedor para elementos relacionados. Puedes declarar un namespace usando la palabra clave "namespace". Por ejemplo, si tienes un conjunto de tipos y funciones relacionados para trabajar con fechas, puedes crear un namespace llamado "Fechas" y colocar todos los elementos relacionados dentro de él. Luego, puedes acceder a estos elementos utilizando la notación de puntos, como "Fechas.parseFecha('2023-10-11')".

**Generics:**

**Concepto de generics:** Los generics, o genéricos, son una característica fundamental de TypeScript que permite escribir código que puede trabajar de manera flexible con diferentes tipos de datos. En esencia, los generics permiten parametrizar funciones, clases o interfaces para que puedan trabajar con tipos específicos que se determinan en tiempo de compilación. En lugar de definir un tipo de dato concreto en una función o clase, puedes utilizar un tipo genérico, representado comúnmente por una letra como "T". Esto permite que la función o clase sea reutilizable y admita varios tipos de datos sin necesidad de duplicar código.

**Uso de generics en funciones y clases:** Los generics se utilizan en funciones para definir parámetros y tipos de retorno flexibles que dependen de los tipos proporcionados en la llamada a la función. Por ejemplo, puedes escribir una función genérica que ordene un array de elementos sin importar el tipo de elementos que contiene. En clases, los generics se utilizan para definir tipos de propiedades y métodos que trabajan con tipos dinámicos. Esto es especialmente útil en contenedores de datos y estructuras de colección, como las clases "Array" o "Map".

**Ventajas de los generics:**

**Flexibilidad y reutilización:** Una de las ventajas clave de los generics es la capacidad de escribir código más flexible y reutilizable. Al parametrizar funciones y clases, puedes adaptarlas para trabajar con diferentes tipos de datos sin tener que escribir múltiples versiones del mismo código. Esto ahorra tiempo y reduce la duplicación de código.

**Seguridad de tipos:** Los generics garantizan la seguridad de tipos en tiempo de compilación. Esto significa que los errores de tipo se detectan antes de la ejecución del código. Si intentas usar un tipo incorrecto, TypeScript te informará de ello, lo que mejora la robustez y calidad del software.

**Abstracción y legibilidad:** Los generics permiten abstraer conceptos comunes y crear código más legible y comprensible. Al utilizar nombres de tipo genérico descriptivos, como "T" o "U", puedes comunicar de manera clara la intención de tu código y hacer que sea más fácil de entender y mantener.

**Promesas y async/await:**

**Tratamiento de asincronía en TypeScript:** La asincronía es común en la programación, especialmente en aplicaciones web, donde las operaciones como llamadas a API, solicitudes de red y acceso a bases de datos pueden llevar tiempo. En TypeScript, una forma eficaz de lidiar con la asincronía es a través del uso de promesas y async/await. En lugar de bloquear la ejecución del código mientras se espera una respuesta, las promesas permiten que el código continúe ejecutándose y se ejecute una función de retorno de llamada cuando se completa la operación asincrónica. Esto mantiene la aplicación receptiva y evita bloqueos.

**Uso de promesas y async/await:** Las promesas se crean mediante el constructor "Promise" y se utilizan para envolver operaciones asincrónicas. Puedes encadenar promesas utilizando "then" para ejecutar código después de que la promesa se resuelva o rechace. Sin embargo, el enfoque más moderno y legible en TypeScript es el uso de "async/await". Al marcar una función como "async", puedes utilizar "await" para esperar que una promesa se resuelva antes de continuar con la ejecución del código. Esto hace que el código asincrónico se parezca más al código sincrónico, lo que mejora la legibilidad. Por ejemplo, puedes escribir una función asincrónica que haga una llamada a una API con "await fetch('url')", lo que facilita la gestión de la respuesta de la API.

**Decoradores:**

**Introducción a los decoradores:** Los decoradores son una característica avanzada en TypeScript que permite agregar metadatos a clases, propiedades, métodos y parámetros. Los decoradores proporcionan una forma de extender y modificar el comportamiento de una clase o sus miembros sin alterar el código original. Los decoradores se aplican utilizando el símbolo "@" seguido de un identificador de función que se ejecuta cuando se declara el elemento decorado. Los decoradores se utilizan comúnmente en frameworks y bibliotecas de TypeScript, como Angular, para agregar funcionalidades personalizadas a componentes, servicios y otras partes de una aplicación.

**Uso de decoradores en clases y métodos:** Los decoradores pueden aplicarse a clases, métodos, propiedades y parámetros de una clase. Por ejemplo, un decorador de clase podría agregarse a una clase de componente en Angular para definir cómo se debe comportar el componente. Los decoradores de métodos pueden utilizarse para modificar el comportamiento de los métodos, como la validación de entrada o la administración de eventos. Los decoradores de propiedades pueden utilizarse para agregar propiedades adicionales a una clase. Los decoradores de parámetros pueden utilizarse para modificar argumentos pasados a métodos o constructores. Los decoradores se aplican al declarar el elemento decorado y se ejecutan en tiempo de diseño, lo que significa que pueden afectar el comportamiento del elemento decorado en tiempo de ejecución.

**Bibliotecas y Frameworks populares:**

**Angular:** Angular es un popular framework de desarrollo de aplicaciones web y móviles. Se basa en TypeScript y es conocido por su enfoque en la estructuración de aplicaciones mediante el uso de componentes. Angular utiliza un sistema de inyección de dependencias y proporciona una amplia gama de características, incluyendo enrutamiento, manejo de formularios, animaciones y comunicación con servidores a través de HTTP. Gracias a TypeScript, Angular ofrece una experiencia de desarrollo más segura y más robusta al proporcionar verificación de tipos en tiempo de compilación. Esto ayuda a prevenir errores y facilita el mantenimiento de aplicaciones a medida que crecen en complejidad. La combinación de Angular y TypeScript es muy popular en la creación de aplicaciones empresariales y grandes proyectos web.

**React con TypeScript:** React es una biblioteca de JavaScript muy utilizada para construir interfaces de usuario. Cuando se combina con TypeScript, se obtiene una experiencia de desarrollo más sólida y confiable. TypeScript proporciona anotaciones de tipos para componentes de React, lo que facilita la detección temprana de errores y la escritura de código más seguro. Además, TypeScript permite definir interfaces para las propiedades de los componentes, lo que hace que el código sea más legible y autodocumentado. React y TypeScript se utilizan en conjunto en una amplia variedad de aplicaciones, desde aplicaciones web de una sola página (SPA) hasta aplicaciones móviles nativas e incluso aplicaciones de escritorio con tecnologías como Electron. La combinación de React y TypeScript es especialmente popular en la comunidad de desarrollo web moderno debido a su eficacia y productividad.

**Mejores prácticas:**

**Convenciones de nomenclatura:** En TypeScript, seguir convenciones de nomenclatura claras es fundamental para mantener un código limpio y legible. Algunas convenciones comunes incluyen utilizar nombres descriptivos para variables, funciones y clases, utilizar camelCase para identificadores (por ejemplo, "miVariable" en lugar de "mivariable"), y utilizar PascalCase para nombres de clases y tipos personalizados (por ejemplo, "MiClase" en lugar de "miClase").

**Evitar el uso de tipos any:** Uno de los principales beneficios de TypeScript es la seguridad de tipos. Usar el tipo "any" socava esta seguridad al permitir que una variable pueda ser de cualquier tipo. Es importante evitar el uso excesivo de "any" y, en su lugar, anotar y tipar adecuadamente las variables y funciones para aprovechar al máximo las ventajas de TypeScript.

**Uso de types en lugar de interfaces en ciertos casos:** En TypeScript, puedes usar tanto interfaces como tipos (type) para definir estructuras de datos. Las interfaces suelen utilizarse para definir contratos y especificar formas de objetos, mientras que los tipos son más versátiles y pueden representar cualquier estructura de datos. En ciertos casos, es más apropiado utilizar tipos, especialmente cuando necesitas representar tipos compuestos o realizar transformaciones en tipos. Sin embargo, las interfaces siguen siendo útiles cuando necesitas definir un contrato claro para una clase o un objeto.

**Desafíos y limitaciones:**

**a. Desafíos comunes al trabajar con TypeScript:** Algunos desafíos comunes al trabajar con TypeScript incluyen la curva de aprendizaje inicial, especialmente para aquellos que no están familiarizados con el sistema de tipos, y la necesidad de anotar correctamente los tipos en el código. Además, la integración de TypeScript en proyectos existentes puede ser un desafío, especialmente en proyectos grandes.

**b. Limitaciones y consideraciones al migrar proyectos a TypeScript:** Al migrar proyectos a TypeScript, es importante considerar el tiempo necesario para refactorizar el código existente y adaptarlo a TypeScript. También debes estar preparado para enfrentar desafíos relacionados con la interoperabilidad con bibliotecas y frameworks que no están escritos en TypeScript. Además, es importante planificar la forma en que se administrarán las actualizaciones y mantenimiento a lo largo del tiempo, ya que TypeScript continuará evolucionando.

**TYPESCRIPT-PHP-MYSQL**

**Ejercicio guiado pasos a seguir:**

¡Hola a todos! Hoy vamos a aprender a crear una página de inicio de sesión para tu sitio web utilizando TypeScript, PHP y MySQL. Vamos a dividir este proyecto en varios pasos para que sea más fácil de entender.

**Paso 1: Diseñar la Página de Inicio de Sesión**

Empecemos por diseñar la página de inicio de sesión. Esto significa crear la apariencia de la página usando HTML y darle estilo con CSS para que se vea atractiva y fácil de usar. En esta página, los usuarios ingresarán su nombre de usuario o correo electrónico y su contraseña, y luego harán clic en un botón para iniciar sesión.

**Paso 2: Implementar el Frontend**

En el siguiente paso, vamos a utilizar HTML para crear los campos en los que los usuarios ingresarán su nombre de usuario y contraseña. También podemos usar JavaScript o TypeScript para agregar algunas reglas de validación en el lado del cliente, como asegurarnos de que los campos no estén vacíos y cumplan con ciertos requisitos. Además, vamos a asegurarnos de que la información se envíe al servidor cuando los usuarios hagan clic en "Iniciar sesión".

**Paso 3: Crear un Script en el Backend**

En este paso, vamos a desarrollar un script en el servidor utilizando PHP para manejar la solicitud de inicio de sesión. Este script recibirá los datos del formulario, los validará y verificará si coinciden con la información almacenada en la base de datos. Si las credenciales son correctas, crearemos una sesión o un token para autenticar al usuario.

**Paso 4: Integración con la Base de Datos**

Es importante que nuestro script PHP pueda comunicarse con la base de datos MySQL. Debe buscar y comparar las credenciales proporcionadas con los registros de usuario en la base de datos. Además, vamos a aprender a almacenar contraseñas de manera segura utilizando técnicas como el hash y la sal.

**Paso 5: Autenticación**

A continuación, implementaremos la lógica para autenticar al usuario. Si las credenciales son correctas, configuraremos una sesión o emitiremos un token para identificar al usuario como autenticado. Esto permitirá a los usuarios acceder a partes protegidas de la aplicación.

**Paso 6: Manejo de Errores**

Vamos a aprender a manejar errores de manera que podamos dar retroalimentación a los usuarios en caso de credenciales incorrectas, problemas de base de datos u otros errores. Mostraremos mensajes de error adecuados en la página de inicio de sesión.

**Paso 7: Redirección o Mostrar Contenido**

Si el inicio de sesión es exitoso, podemos redirigir al usuario a un panel de control o mostrar contenido específico para usuarios autenticados. En caso de un inicio de sesión fallido, mantendremos al usuario en la página de inicio de sesión con un mensaje de error.

**Paso 8: Consideraciones de Seguridad**

Hablaremos sobre la importancia de la seguridad, como proteger contra ataques de fuerza bruta, gestionar sesiones de manera segura y utilizar conexiones HTTPS para proteger los datos.

**Paso 9: Pruebas**

Finalmente, realizaremos pruebas exhaustivas de la funcionalidad de inicio de sesión para asegurarnos de que todo funcione como debería. Probaremos tanto los intentos de inicio de sesión correctos como los incorrectos en diferentes navegadores y también consideraremos pruebas de seguridad para detectar vulnerabilidades.

**Paso 10: Registro de Usuarios**

Si aún no lo has hecho, también es posible que necesites crear una página de registro de usuarios para permitir que las personas se registren en tu aplicación.