Informe sobre Preguntas Teóricas de TypeScript

(La presentación se basa en el siguiente documento)

Allison Acosta - Francisco Jara - Valeria Torrealba

1. **¿Qué es TypeScript y para qué se utiliza?**
   TypeScript es un lenguaje de programación de código abierto desarrollado por Microsoft que extiende las capacidades de JavaScript al incluir un sistema de tipos estáticos. Esto significa que TypeScript permite definir los tipos de datos que las variables, funciones y otros elementos del código pueden contener, lo que facilita la detección de errores en el tiempo de desarrollo.
   Se utiliza principalmente para el desarrollo de aplicaciones web, móviles y de escritorio, especialmente en proyectos de mediana y gran escala, donde la estructura clara y el tipado son esenciales para mantener la calidad del código y su mantenibilidad.

2. **¿Cuáles son las principales diferencias entre TypeScript y JavaScript?**

   Sistema de Tipos: TypeScript es fuertemente tipado y permite definir tipos explícitos, mientras que JavaScript es débilmente tipado y dinámico.
   Transpilación: El código TypeScript debe ser transpilado a JavaScript para ser ejecutado en navegadores o entornos como Node.js.
   Características Avanzadas: TypeScript incluye características adicionales como interfaces, decoradores y tipos genéricos, que no están disponibles en JavaScript nativo.

3. **¿Por qué es útil TypeScript en el desarrollo de aplicaciones ReactJS?**
   TypeScript mejora la experiencia de desarrollo en React al proporcionar tipado estático, lo que facilita la detección de errores en tiempo de desarrollo, especialmente en componentes complejos y grandes. Además, ayuda a documentar mejor las props y estados de los componentes, lo que resulta en un código más claro y fácil de mantener. Esto es particularmente valioso en aplicaciones React escalables o colaborativas.

4. **¿Qué es el sistema de tipos en TypeScript y cómo ayuda a evitar errores en tiempo de desarrollo?**

El sistema de tipos en TypeScript permite definir explícitamente los tipos de variables, funciones, objetos y clases. Esto previene errores comunes como operaciones inválidas entre tipos incompatibles o el acceso a propiedades inexistentes. Al detectar estos problemas antes de ejecutar el código, TypeScript reduce los errores en producción y facilita el desarrollo más confiable.

Pregunta 2:
Se crea un codigo que crea una clase Doctor y luego se “juega” con sus parametros

```
type Doctor = {
  id: number;
  name: string;
  specialty: string;
  available: boolean;
};

function getDoctorInfo(doctors: Doctor[]): string[] {
  return doctors.map((doctor) => doctor.name);
}

const doctors = [
  { id: 1, name: "Dr. Smith", specialty: "Cardiology", available: true },
  { id: 2, name: "Dr. Jones", specialty: "Neurology", available: false },
];

const names = getDoctorInfo(doctors);
console.log(names);
```

Pregunta 3:
Se define una interfaz IDoctor y clases en typeScript

```
interface IDoctor {
  id: number;
  name: string;
  specialty: string;
  available: boolean;
  getDetails(): string;
  updateSpecialty(newSpecialty: string): void;
}

class Doctor implements IDoctor {
  constructor(
    public id: number,
    public name: string,
```

```
    public specialty: string,
    public available: boolean
  ) {}

  getDetails(): string {
    return `${this.name} (${this.specialty}) - ${this.available ? "Available" : "Not Available"}`;
  }

  updateSpecialty(newSpecialty: string): void {
    this.specialty = newSpecialty;
  }
}

// Ejemplo de uso
const doctor = new Doctor(1, "Dr. Tapia", "Pediatra", true);
console.log(doctor.getDetails());
doctor.updateSpecialty("Psicologo");
console.log(doctor.getDetails());
```

4. Implementar Componentes (prop y otros)

```
import React from "react";

type DoctorProps = {
  id: number;
  name: string;
  specialty: string;
  available: boolean;
};

const DoctorCard: React.FC<DoctorProps> = ({ id, name, specialty, available }) => {
  return (
    <div>
      <h2>{name}</h2>
      <p>Specialty: {specialty}</p>
      <p>Status: {available ? "Available" : "Not Available"}</p>
    </div>
  );
};

// Ejemplo de uso
const App = () => {
  const doctor = { id: 1, name: "Dr. Smith", specialty: "Cardiology", available: true };
  return <DoctorCard {...doctor} />;
};

export default App;
```

5. Ventajas de TypeScript

Detección temprana de errores: TypeScript permite detectar errores de tipo en tiempo de desarrollo, antes de ejecutar el código.
Autocompletado: Los editores como VS Code sugieren propiedades y métodos basados en los tipos definidos.
Mantenimiento del código: Al tener tipos definidos, es más fácil escalar y comprender el código.

código de ejemplo:

```
const doctor = { id: 1, name: "Dr. Smith" };
// doctor.specialty; // Error: Property 'specialty' does not exist.
```

Conclusión

TypeScript es una herramienta poderosa para el desarrollo moderno. Su sistema de tipos estáticos y características avanzadas mejoran la calidad del código y la experiencia de desarrollo, especialmente en proyectos escalables y colaborativos.