Parte 1: Opción Múltiple (10 Preguntas)
Marca con X la respuesta correcta.

1. **¿Qué es Angular en el contexto de Ionic?**

    a. (X) Un framework de desarrollo frontend basado en TypeScript
    b. ( ) Un framework de diseño de interfaces para aplicaciones móviles
    c. ( ) Un compilador de código nativo para aplicaciones móviles
2. **¿Cómo se crea un nuevo proyecto en Ionic con Angular?**

    a. ( ) ng new myApp --ionic
    b. (X) ionic start myApp blank --type=angular
    c. ( ) ionic generate project myApp
3. **¿Cuál de los siguientes comandos genera un nuevo componente en un proyecto Angular?**

    a. ( ) ionic component generate myComponent
    b. (X) ng generate component components/myComponent
    c. ( ) ionic g module myComponent
4. **¿Cuál de las siguientes opciones describe mejor un módulo en Angular?**

    a. ( ) Un archivo CSS que agrupa estilos
    b. ( ) Una función de JavaScript que maneja eventos
    c. ( ) Un tipo especial de componente
    d. (X) Un conjunto de componentes, directivas y servicios organizados
5. **¿Cuál es la forma correcta de hacer una inyección de dependencia en un componente de Angular?**

    a. ( ) constructor(service: ApiService) {}
    b. (X) constructor(private apiService: ApiService) {}
    c. ( ) service = new ApiService();
6. **¿Qué método del ciclo de vida de un componente en Angular se ejecuta justo después de la creación del componente?**

    a. ( ) ngOnDestroy()
    b. ( ) ngAfterViewInit()
    c. (X) ngOnInit()
7. **¿Cómo se maneja una petición HTTP con HttpClient en Angular?**

    a. ( ) this.http.get('https://api.com/data').then(response => console.log(response));
    b. (X) this.http.get('https://api.com/data').subscribe(response => console.log(response));
    c. ( ) fetch('https://api.com/data').then(response => console.log(response));
8. **¿Cuál es la diferencia entre LocalStorage y SessionStorage?**

    a. (X) LocalStorage almacena datos de manera persistente, mientras que SessionStorage se borra al cerrar la pestaña o navegador
    b. ( ) LocalStorage almacena datos por sesión y SessionStorage almacena datos de forma permanente
    c. ( ) No hay diferencia, ambos funcionan igual
9. **¿Cómo se define una ruta en Angular?**

    a. ( ) const routes: Router = [{ url: '/home', component: HomePage }];
    b. (X) const routes: Routes = [{ path: 'home', component: HomePage }];
    c. ( ) const routes = new Router([{ route: '/home', component: HomePage }]);
10. **¿Qué operador de RxJS se usa para transformar los datos de un observable?**

11. ( ) catchError()
12. ( ) subscribe()
13. (X) map()

1. Diferencia entre Observables y Promises en Angular

- **Promises**: Son objetos que representan el valor de una operación asincrónica. Una promesa puede estar en uno de tres estados: pendiente, resuelta o rechazada. Se resuelve o se rechaza una sola vez y no se puede volver a resolver. Son ideales para operaciones que se completan una sola vez, como una llamada a una API.

- **Observables**: Son más flexibles y permiten emitir múltiples valores a lo largo del tiempo. Se usan para manejar flujos de datos, como la respuesta a múltiples solicitudes HTTP, eventos o flujos de datos que cambian con el tiempo. Los observables también pueden ser cancelados, lo que los hace más poderosos para manejar secuencias de eventos.

2. Ciclo de vida de un componente en Angular

El ciclo de vida de un componente en Angular incluye varias fases que van desde su creación hasta su destrucción. Los principales métodos del ciclo de vida son:

- **ngOnInit()**: Se ejecuta después de que Angular haya inicializado todas las propiedades del componente. Ideal para la inicialización de datos.

- **ngOnChanges()**: Se llama cuando una propiedad de entrada del componente cambia.

- **ngDoCheck()**: Se invoca para permitir al desarrollador realizar comprobaciones personalizadas durante cada ciclo de detección de cambios.

- **ngAfterContentInit()**: Se ejecuta después de que el contenido del componente se haya proyectado en el DOM.

- **ngAfterContentChecked()**: Se ejecuta después de cada comprobación de cambios en el contenido proyectado.

- **ngAfterViewInit()**: Se ejecuta después de que las vistas de un componente y sus vistas secundarias se hayan inicializado.

- **ngAfterViewChecked()**: Se invoca después de cada comprobación de cambios en la vista del componente.

- **ngOnDestroy()**: Se ejecuta cuando el componente es destruido, útil para limpiar recursos o cancelar suscripciones.

3. Comunicación entre componentes en Angular

En Angular, los componentes pueden comunicarse entre sí de varias formas:

- **A través de Input y Output**: Los datos fluyen de un componente padre a un componente hijo mediante @Input y desde un componente hijo a un padre mediante @Output

- **Servicios compartidos**: Cuando los componentes no están directamente relacionados (no padre e hijo), se puede usar un servicio común que actúe como intermediario para compartir datos o manejar la comunicación entre componentes.

4. Diferencia entre ngOnInit() y ngOnDestroy()

- **ngOnInit()**: Es parte del ciclo de vida del componente y se ejecuta cuando el componente se ha inicializado. Es común usarlo para realizar configuraciones iniciales, como la carga de datos o la suscripción a observables.

- **ngOnDestroy()**: Se ejecuta cuando el componente va a ser destruido. Se usa para liberar recursos, como cancelar suscripciones o eliminar manejadores de eventos.

5. ¿Qué son los Guards en Angular y cómo funcionan?

Los **Guards** en Angular son funciones que se ejecutan antes de activar o desactivar una ruta. Permiten proteger rutas de acceso no autorizado o impedir la navegación bajo ciertas condiciones. Los tipos de guards más comunes son:

- **CanActivate**: Previene la activación de una ruta.
- **CanDeactivate**: Previene que un usuario salga de una ruta.
- **CanLoad**: Previene que una ruta se cargue de forma perezosa.
- **Resolve**: Carga los datos antes de navegar a una ruta.

6. Funcionamiento del Lazy Loading en Angular

El **Lazy Loading** en Angular permite cargar módulos solo cuando son necesarios, en lugar de cargarlos todos al inicio. Esto mejora el rendimiento de la aplicación al dividirla en módulos más pequeños y cargar solo los que se necesitan en ese momento, reduciendo el tiempo de carga inicial.

7. ¿Qué es el HttpInterceptor en Angular y para qué se usa?

Un **HttpInterceptor** es un servicio en Angular que permite interceptar todas las solicitudes HTTP que se hacen desde la aplicación. Se puede usar para:

- Modificar las solicitudes antes de enviarlas (por ejemplo, agregar encabezados de autorización).
- Modificar las respuestas antes de que lleguen a los componentes.
- Manejar errores globalmente.

Es útil para casos como autenticación o gestión de errores de manera centralizada.

8. ¿Qué son los Guards en Angular y cómo funcionan?

Los **Guards** son funciones que determinan si se puede o no navegar a una ruta o salir de ella, asegurando que se cumplan ciertas condiciones antes de permitir la navegación.