# Navegación, Rutas y Variables de Sesión en Ionic

Ing. Fausto Viscaino

# Navegación entre páginas

En Ionic, la navegación entre páginas se realiza principalmente utilizando el NavController o el Router de Angular.

# NAVCONTROLLER

NavController es una característica específica de Ionic que proporciona una API para navegar entre páginas en aplicaciones móviles, especialmente útil para mantener un historial de navegación y permitir gestos de deslizamiento para volver atrás

**Características principales de NavController:**
**1.Historial de navegación**: Mantiene una pila de páginas visitadas, facilitando la navegación hacia atrás.
**2.Transiciones animadas**: Proporciona transiciones suaves entre páginas, similares a las aplicaciones nativas.
**3.Gestión de ciclo de vida**: Maneja los eventos de ciclo de vida de las páginas (ionViewWillEnter, ionViewDidEnter, etc.).
**4.Navegación por gestos**: Permite la navegación hacia atrás mediante gestos de deslizamiento en iOS.

**Métodos clave de NavController:**
•**navigateForward(url):** Navega hacia adelante a una nueva página.
•**navigateBack(url):** Navega hacia atrás en el historial.
•**navigateRoot(url):** Establece la página raíz, limpiando el historial.
•**back():** Vuelve a la página anterior.
•**pop():** Elimina la página actual de la pila y vuelve a la anterior

# NavController

Página.ts

```
import { Component } from '@angular/core';
import {                    } from '@ionic/angular';
@Component({
 selector: 'app-home', templateUrl: 'home.page.html',
})
export class HomePage {
    constructor(private navCtrl: NavController)
    {}

    navigateToAbout()
   {
        this.navCtrl.navigateForward('/about');
   }
}
```

Página.html

```
<ion-button (click)="navigateToAbout()">Ir a About</ion-button>
```

# Router de Angular

El Router de Angular es el sistema de enrutamiento estándar de Angular, que Ionic ha integrado completamente. Es más adecuado para aplicaciones web y proporciona características avanzadas de enrutamiento.

**Características principales del Angular Router:**

1. **Rutas anidadas**: Permite definir jerarquías complejas de rutas.
2. **Guardias de ruta**: Proporciona control de acceso a rutas basado en condiciones.
3. **Resolvers**: Permite cargar datos antes de que se active una ruta.
4. **Lazy loading**: Facilita la carga perezosa de módulos para mejorar el rendimiento.
5. **Parámetros de ruta y consulta**: Maneja fácilmente parámetros en las URL.

# Router

Página.ts

```
import { Component } from '@angular/core'; import { Router
} from '@angular/router’;

@Component({
selector: 'app-home’,
templateUrl: 'home.page.html’,
})
export class HomePage {
      constructor(private router: Router)
      {}

      navigateToAbout()
      {
        this.router.navigate(['/about']);
      }
}
```

Página.html

```
<ion-button (click)="navigateToAbout()">Ir a About</ion-button>
```

# Cuando usar cada uno?

**NavController**:
•Ideal para aplicaciones móviles con navegación lineal.
•Mejor para mantener un historial de navegación similar a apps nativas.
•Proporciona transiciones de página más suaves y nativas.

**Angular Router**
•Mejor para aplicaciones web complejas o PWAs.
•Ofrece más control sobre la estructura de la URL.
•Permite lazy loading de módulos para mejor rendimiento.
•Más potente para manejar guardias de ruta y resolvers.

En la práctica, muchas aplicaciones Ionic modernas utilizan una combinación de ambos: Angular Router para la estructura general de la aplicación y NavController para la navegación dentro de secciones específicas que requieren una experiencia más nativa

# Configuración de Rutas

- Las rutas en Ionic se configuran en el archivo app-routing.module.ts. Aquí hay un ejemplo de cómo se podría ver este archivo:

En este ejemplo, tenemos rutas para 'home' y 'about', y una redirección por defecto a 'home'.

```
import { NgModule } from '@angular/core’;
import { PreloadAllModules, RouterModule, Routes } from '@angular/router’;
 const routes: Routes = [
  {
    path: 'home’,
    loadChildren: () => import('./home/home.module').then(     =>     .HomePageModule)
  },
  {
     path: 'about’,
     loadChildren: () => import('./about/about.module').then(     =>     .AboutPageModule)
  },
  {
     path: ‘’,
     redirectTo: 'home’,
     pathMatch: 'full’
  },
];
```

# Variables de Sesión

- En Ionic, puedes manejar variables de sesión de varias maneras. Una forma común es utilizar el almacenamiento local (localStorage) o el servicio de Ionic Preferences.

# LocalStorage

localStorage es una API web estándar que permite almacenar datos de forma local en el navegador del usuario.

**Características de localStorage:**

1. Almacenamiento simple de pares clave-valor.
2. Los datos persisten incluso después de cerrar el navegador.
3. Limitado a strings (los objetos deben ser serializados).
4. Tiene un límite de almacenamiento (generalmente 5-10MB).
5. Síncrono, lo que puede bloquear el hilo principal en operaciones grandes.

```
mensaje: string = '';
```

```
CrearDatos()
{
const dato = { nombre: 'Juan', edad: 30 };
localStorage.setItem('usuarios', JSON.stringify(dato));
}
```

```
obtenerDatos() {
    const almacen = localStorage.getItem('userData');
    if (almacen) {
        const datos = JSON.parse(almacen);
        this.mensaje = 'Nombre: ${datos.nombre}, Edad: ${datos.edad}';
     } else {
         this.mensaje = 'No hay datos almacenados';
     }
}
```

```
eliminarDato() {
localStorage.removeItem(usuarios');
}
```

```
limpiarDatos() { localStorage.clear(); }
```

# Ionic Preferences

Ionic Preferences es una API moderna proporcionada por Capacitor (el motor nativo de Ionic) para almacenar datos de forma local en aplicaciones móviles y web.

**Características de Ionic Preferences:**

1. API asíncrona, lo que mejora el rendimiento.
2. Consistente entre plataformas (web, iOS, Android).
3. Integración nativa en dispositivos móviles.
4. Mejor manejo de errores y excepciones.
5. Soporta encriptación en plataformas nativas.

**Configuración de Ionic Preferences:**

Primero, asegúrate de tener instalado Capacitor en tu proyecto Ionic. Luego, importa Preferences en tu componente o servicio:

```
import { Preferences } from '@capacitor/preferences';
```

```
mensaje: string = '';
```

```
async crearDatos() {
    const datos = { nombre: Juan', edad: 30 };
    await Preferences.set({
        key: 'usuarios',
        value: JSON.stringify(datos)
    });
}
```

```
async loadData() {
   const result = await Preferences.get({ key: 'usuarios' });
    if (result.value) {
        const dato = JSON.parse(result.value);
        this.mensaje = 'Nombre: ${dato.nombre}, Edad: ${dato.edad}';
    } else {
        this.mensaje = 'No hay datos almacenados';
    }
}
```

```
async eliminarDato() {
    await Preferences.remove({ key: usuarios' });
}
```

```
async limpiarDatos() { await Preferences.clear(); }
```

# Compracion LocalStorage y Ionic Preferences

**Sincrónía vs Asincronía:**

- localStorage: Operaciones síncronas, pueden bloquear el hilo principal.
- Preferences: Operaciones asíncronas, mejor rendimiento y no bloquean.

**Compatibilidad:**

- localStorage: Disponible en todos los navegadores modernos.
- Preferences: Funciona en web y aplicaciones nativas (iOS/Android).

**Capacidades:**

- localStorage: Limitado a strings, requiere serialización manual.
- Preferences: Maneja automáticamente la serialización de objetos.

**Seguridad:**

- localStorage: No ofrece encriptación nativa.
- Preferences: Soporta encriptación en plataformas nativas.

**Límites de almacenamiento:**

- localStorage: Generalmente limitado a 5-10MB.
- Preferences: Los límites dependen de la plataforma, generalmente más flexibles.

**Manejo de errores:**

- localStorage: Manejo de errores básico.
- Preferences: Mejor manejo de errores y excepciones.

# Bibliografia

https://ionicframework.com/