Universidad Icesi

Navegación en Flutter

Pila de rutas, datos entre pantallas y pestañas

Seminario de I+D+I — Universidad Icesi

Universidad ICESI

Mapa de la sesión

La pila

Cada pantalla se apila sobre la anterior

Los datos

Enviar argumentos y recibir un resultado

Reestructurar

Login, logout y fin de compra

Pestañas

Navegar sin apilar: setState

Universidad ICESI

La pila de rutas

Navegar es apilar pantallas

Pila frente a cola

Pila · LIFO

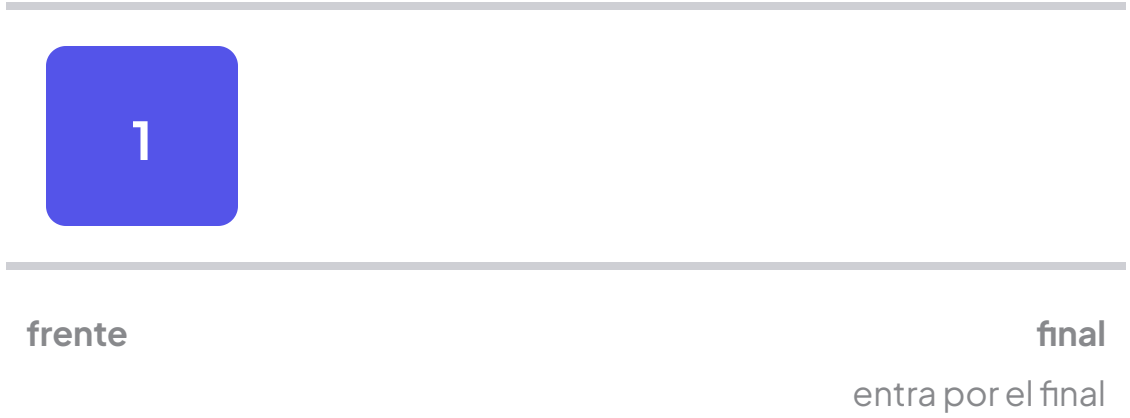
Cola · FIFO

Dos estructuras vacías.

Pila frente a cola

Pila · LIFO

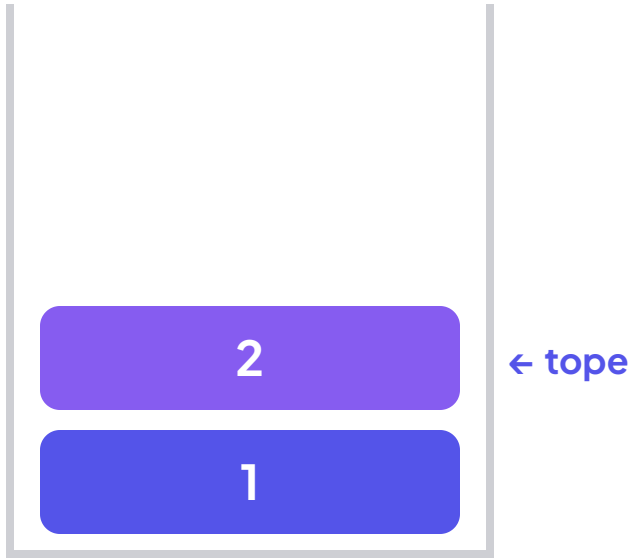
Cola · FIFO



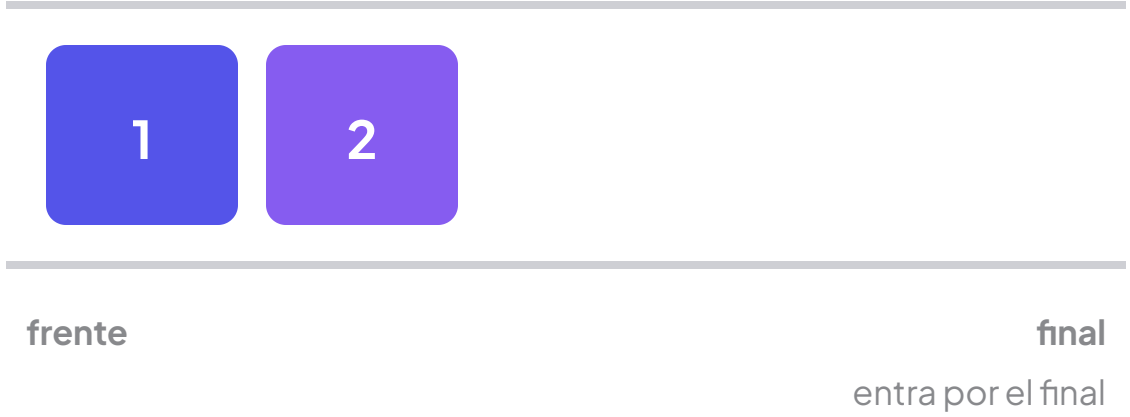
Agregamos el 1.

Pila frente a cola

Pila · LIFO



Cola · FIFO

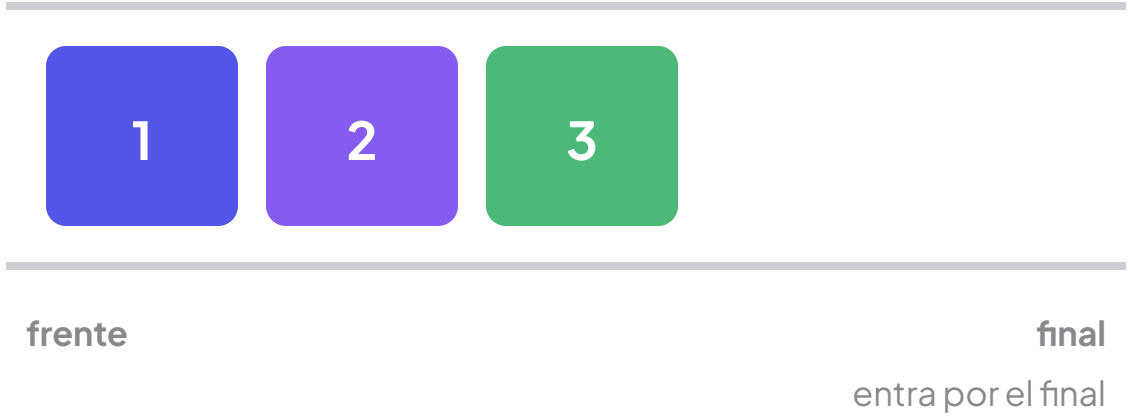


Agregamos el 2.

Pila frente a cola

Pila · LIFO

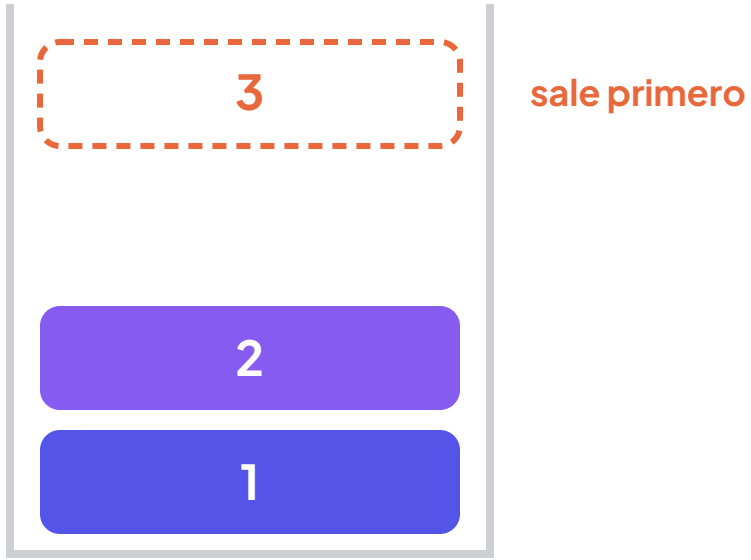
Cola · FIFO



Agregamos el 3, en el mismo orden.

Pila frente a cola

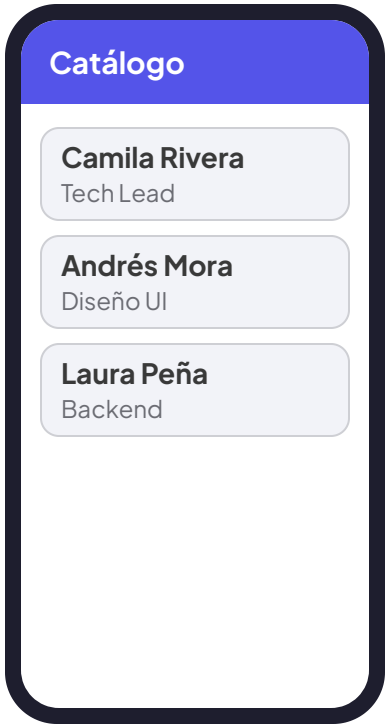
Pila · LIFO



Cola · FIFO

Al sacar, la pila entrega el 3 y la cola el 1.

Navegar es apilar pantallas



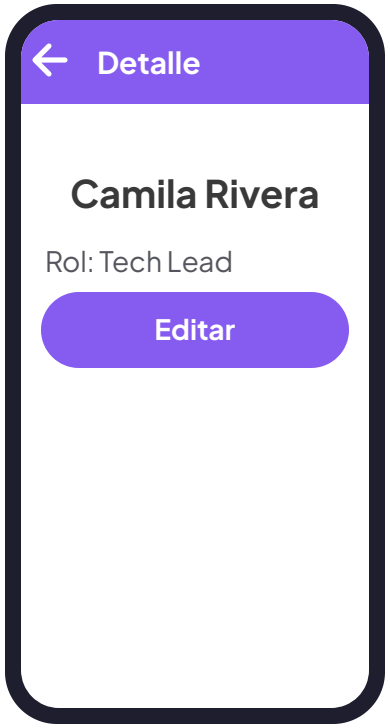
PILA DE RUTAS

main.dart

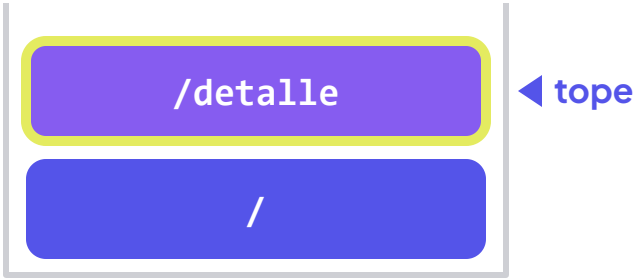
```
MaterialApp(
  initialRoute: '/',
  routes: { ... },
)
```

Arranca con una sola ruta: la raíz.

Navegar es apilar pantallas



PILA DE RUTAS

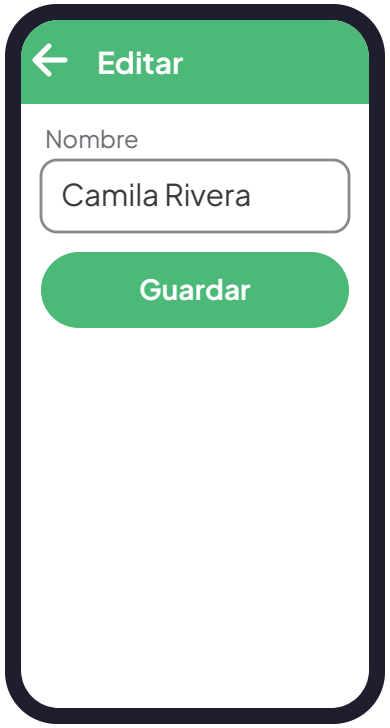


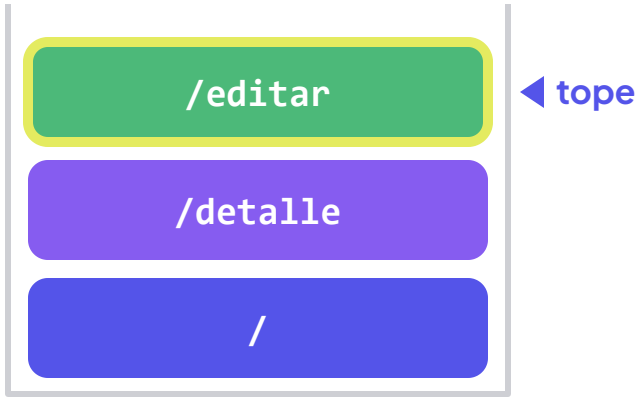
catalogo_screen.dart

```
onPressed: () {
  Navigator.pushNamed(context, '/detalle');
}
```

pushNamed apila. La de abajo sigue viva.

Navegar es apilar pantallas



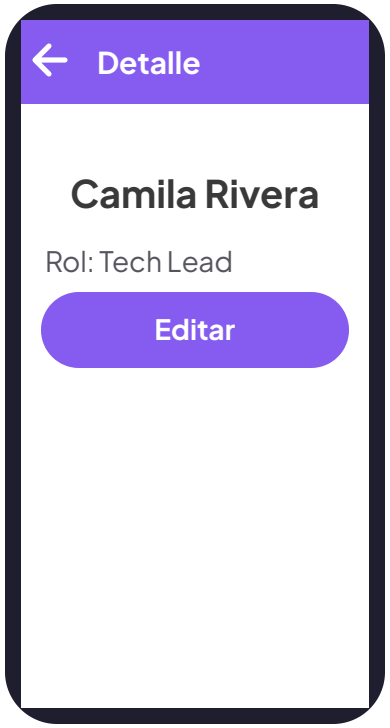


detalle_screen.dart

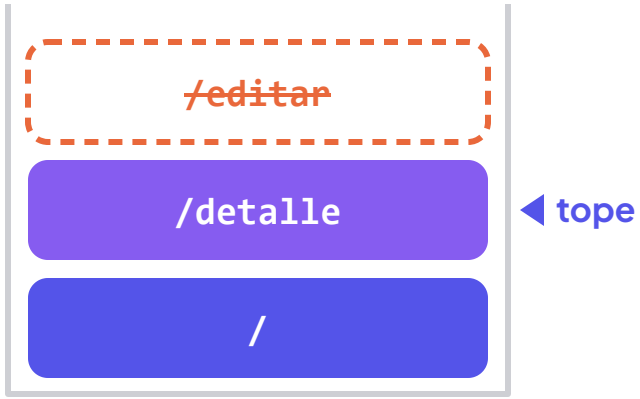
```
onPressed: () {
  Navigator.pushNamed(context, '/editar');
}
```

Otra más al tope. Solo se ve la del tope.

Navegar es apilar pantallas



PILA DE RUTAS

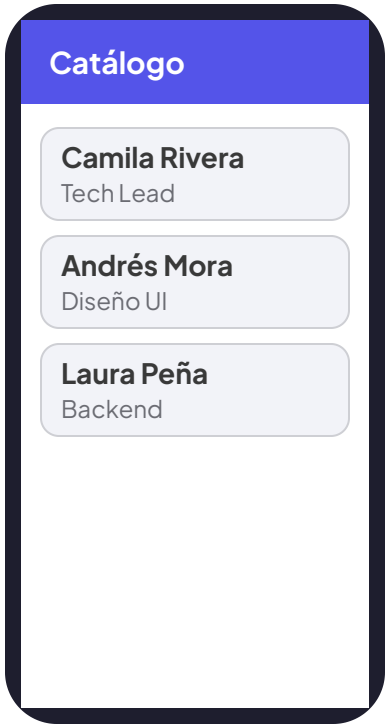


editar_screen.dart

```dart
// Atrás o la flecha de la AppBar
Navigator.pop(context);
```

pop desapila el tope y reaparece la anterior.

Navegar es apilar pantallas



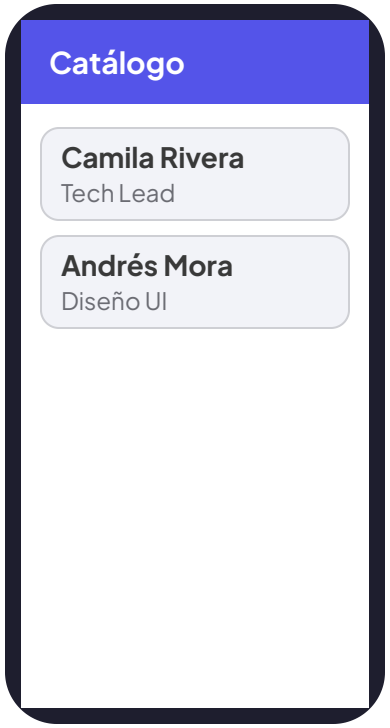
PILA DE RUTAS

detalle_screen.dart

```dart
// Atrás o la flecha de la AppBar
Navigator.pop(context);
```

Otro pop: volvemos al catálogo.

¿Cuándo aparece la flecha?

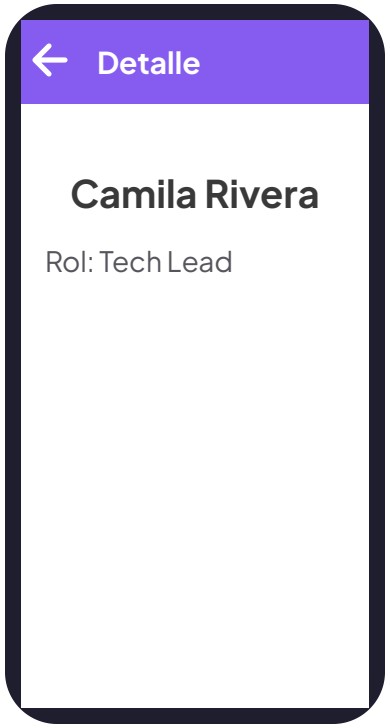


PILA DE RUTAS

`canPop() es false`

Una sola ruta: no hay a dónde volver. Atrás cierra la app.

¿Cuándo aparece la flecha?



PILA DE RUTAS

`canPop() es true`

Hay una ruta debajo: el AppBar dibuja la flecha y la enlaza a pop.

Rutas nombradas en MaterialApp

lib/main.dart

```dart
MaterialApp(
  title: 'Catálogo de Contactos',
  initialRoute: '/',
  routes: {
    '/': (context) => const HomeScreen(),
    '/detalle': (context) => const DetalleScreen(),
    '/seleccion': (context) => const SeleccionScreen(),
    '/login': (context) => const LoginScreen(),
  },
)
```

`initialRoute`

La ruta con la que arranca la app.

Rutas nombradas en MaterialApp

lib/main.dart

```dart
MaterialApp(
  title: 'Catálogo de Contactos',
  initialRoute: '/',
  routes: {
    '/': (context) => const HomeScreen(),
    '/detalle': (context) => const DetalleScreen(),
    '/seleccion': (context) => const SeleccionScreen(),
    '/login': (context) => const LoginScreen(),
  },
)
```

routes

Un mapa: clave es el nombre de la ruta, valor es la función que devuelve la pantalla.

Rutas nombradas en MaterialApp

lib/main.dart

```
MaterialApp(
  title: 'Catálogo de Contactos',
  initialRoute: '/',
  routes: {
    '/': (context) => const HomeScreen(),
    '/detalle': (context) => const DetalleScreen(),
    '/seleccion': (context) => const SeleccionScreen(),
    '/login': (context) => const LoginScreen(),
  },
)
```

Convención: `'/'`

Los nombres empiezan con barra. La raíz es **`/`**.

Cuidado con home y la ruta /

lib/main.dart

```dart
MaterialApp(
  home: const HomeScreen(),
  routes: {
    '/': (context) => const HomeScreen(),
  },
)
```

Colisión

Si **routes** ya define **'/'**, no declares **home**. Dos raíces: el framework lanza una excepción.

Organización del proyecto

Estructura del proyecto

```
lib/
├── main.dart
├── routes/
│   └── app_routes.dart
├── screens/
│   ├── catalogo_screen.dart
│   ├── detalle_screen.dart
│   └── editar_screen.dart
└── models/
    └── contacto.dart
```

`lib/`

Pocos archivos, cada uno con una responsabilidad.

Organización del proyecto

Estructura del proyecto

```
lib/
├── main.dart
├── routes/
│   └── app_routes.dart
├── screens/
│   ├── catalogo_screen.dart
│   ├── detalle_screen.dart
│   └── editar_screen.dart
└── models/
    └── contacto.dart
```

`main.dart`

Solo arranca la app y configura MaterialApp. No contiene pantallas.

Organización del proyecto

Estructura del proyecto

```
lib/
├── main.dart
├── routes/
│   └── app_routes.dart
├── screens/
│   ├── catalogo_screen.dart
│   ├── detalle_screen.dart
│   └── editar_screen.dart
└── models/
    └── contacto.dart
```

routes/

Los nombres de rutas como constantes y la tabla routes. Evita errores de tipeo.

Organización del proyecto

Estructura del proyecto

```
lib/
├── main.dart
├── routes/
│   └── app_routes.dart
├── screens/
│   ├── catalogo_screen.dart
│   ├── detalle_screen.dart
│   └── editar_screen.dart
└── models/
    └── contacto.dart
```

`screens/`

Una pantalla completa por archivo, con nombre terminado en Screen.

Organización del proyecto

Estructura del proyecto

```
lib/
├── main.dart
├── routes/
│   └── app_routes.dart
├── screens/
│   ├── catalogo_screen.dart
│   ├── detalle_screen.dart
│   └── editar_screen.dart
└── models/
    └── contacto.dart
```

`models/`

Las clases de datos, como Contacto, que viajan en arguments.

Datos entre pantallas

Ir con argumentos, volver con un resultado

Enviar datos hacia adelante

Emisor: catalogo_screen.dart

```dart
Navigator.pushNamed(
  context,
  '/detalle',
  arguments: Contacto(
    id: 'c1',
    nombre: 'Camila Rivera',
    rol: 'Tech Lead',
  ),
);
```

`arguments`

Cualquier objeto: un String, un Map o un modelo.

Enviar datos hacia adelante

Emisor

```dart
Navigator.pushNamed(
  context,
  '/detalle',
  arguments: Contacto(
    id: 'c1',
    nombre: 'Camila Rivera',
    rol: 'Tech Lead',
  ),
);
```

Receptor: detalle_screen.dart

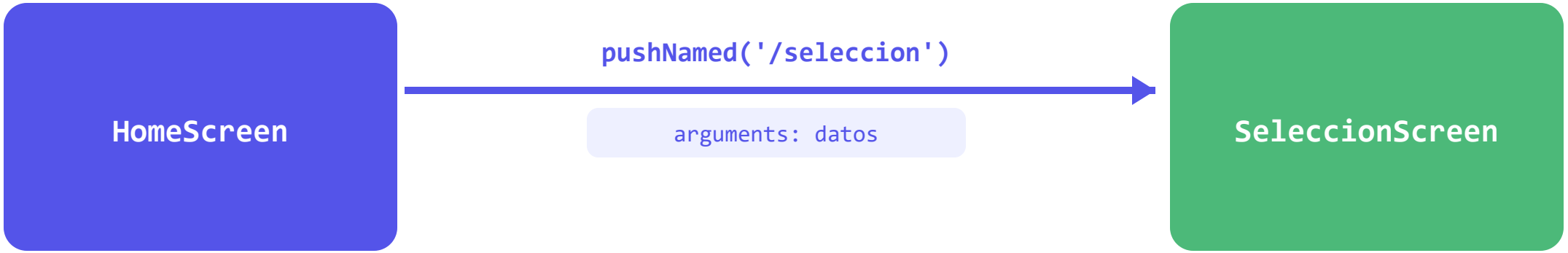
```dart
@override
Widget build(BuildContext context) {
  final contacto = ModalRoute.of(context)!
      .settings.arguments as Contacto;

  return Scaffold(
    appBar: AppBar(title: Text(contacto.nombre)),
  );
}
```

El receptor lee los argumentos con ModalRoute.of(context).

La tubería de ida y vuelta

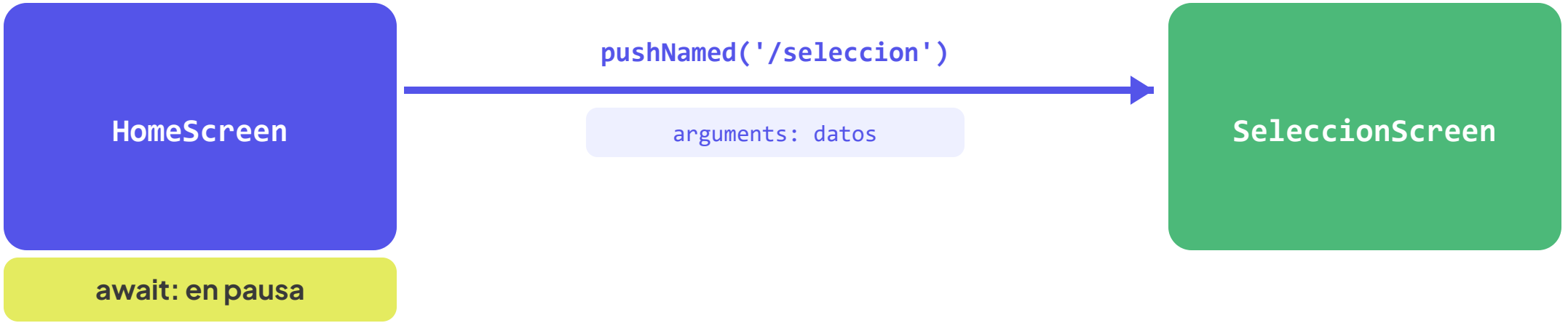
Dos pantallas: una abierta y otra todavía no.

La tubería de ida y vuelta



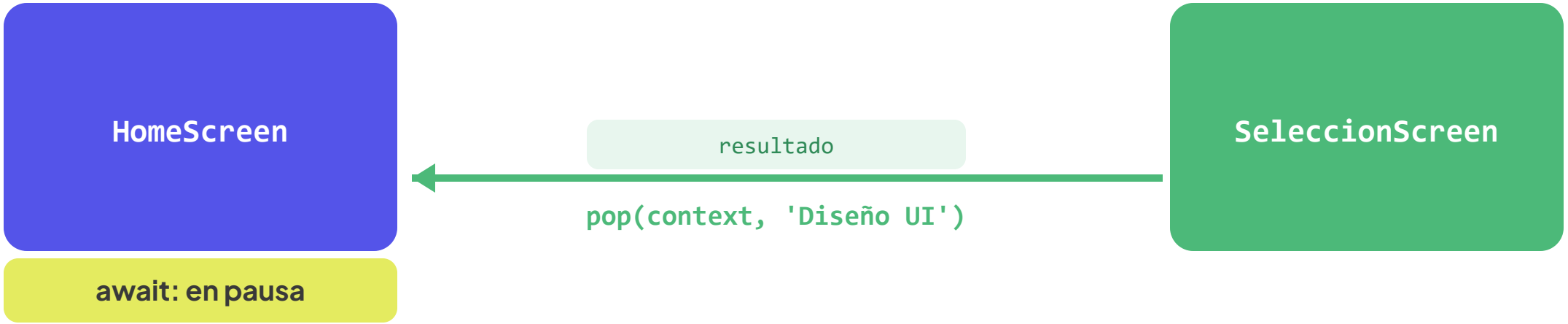
Ida: pushNamed lleva los argumentos.

La tubería de ida y vuelta



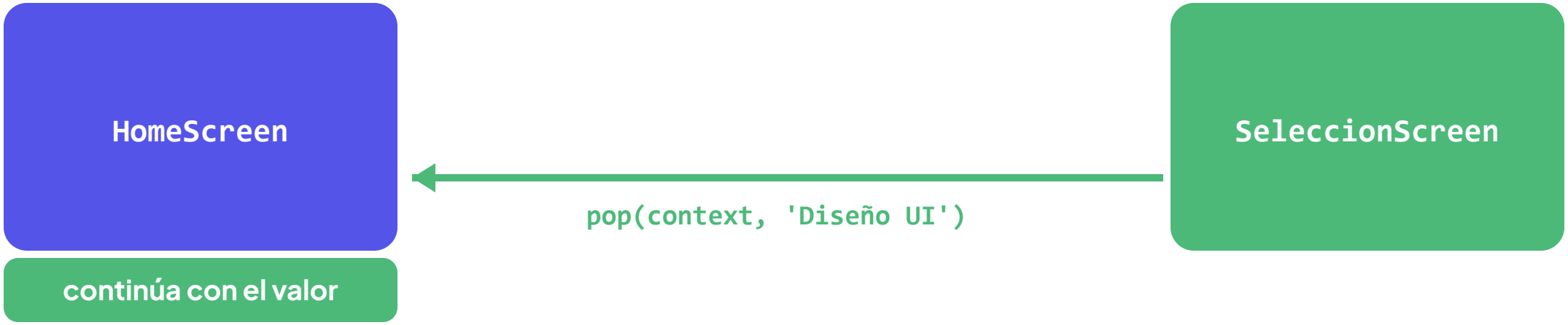
Home espera: el Future de pushNamed aún no se completa.

La tubería de ida y vuelta



Vuelta: pop devuelve el resultado.

La tubería de ida y vuelta



El await termina y Home usa el valor.

Recibir el resultado

Emisor con await

```dart
Future<void> _abrirSelector(BuildContext context) async {
  final String? categoria = await Navigator.pushNamed<String>(
    context,
    '/seleccion',
  );

  if (!context.mounted) return;

  if (categoria != null) {
    ScaffoldMessenger.of(context).showSnackBar(
      SnackBar(content: Text('Seleccionaste: $categoria')),
    );
  }
}
```

await

pushNamed devuelve un Future. Se completa cuando la otra pantalla hace pop.

Recibir el resultado

Emisor con await

```dart
Future<void> _abrirSelector(BuildContext context) async {
  final String? categoria = await Navigator.pushNamed<String>(
    context,
    '/seleccion',
  );

  if (!context.mounted) return;

  if (categoria != null) {
    ScaffoldMessenger.of(context).showSnackBar(
      SnackBar(content: Text('Seleccionaste: $categoria')),
    );
  }
}
```

`context.mounted`

Después de un await, comprueba que la pantalla siga viva antes de usar el context.

Recibir el resultado

Emisor con await

```dart
Future<void> _abrirSelector(BuildContext context) async {
  final String? categoria = await Navigator.pushNamed<String>(
    context,
    '/seleccion',
  );

  if (!context.mounted) return;

  if (categoria != null) {
    ScaffoldMessenger.of(context).showSnackBar(
      SnackBar(content: Text('Seleccionaste: $categoria')),
    );
  }
}
```

`null`

Si sale con Atrás, pop no trae valor: el resultado es null. Por eso el tipo es String?.

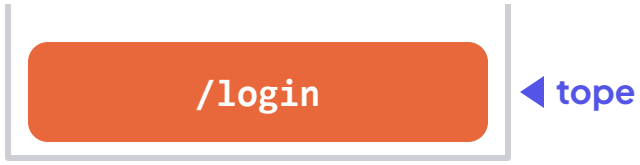
Universidad ICESI

Reestructurar la pila

Login, logout y fin de compra

Login con pushNamed: el problema

PILA DE RUTAS

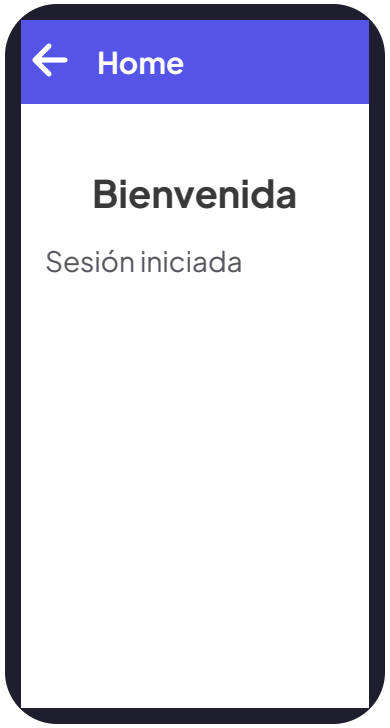


login_screen.dart

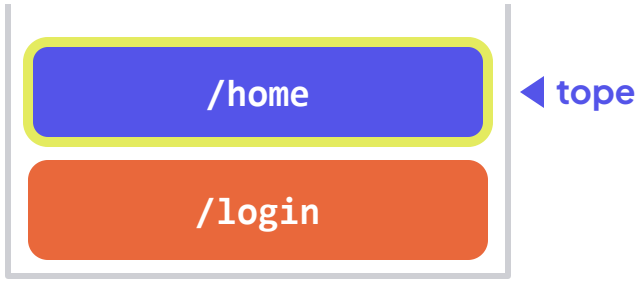
```dart
// la app arranca en el login
initialRoute: '/login',
```

La pila empieza con el login.

Login con pushNamed: el problema



PILA DE RUTAS



login_screen.dart

```dart
onPressed: () {
  Navigator.pushNamed(context, '/home');
}
```

Con pushNamed, el login queda debajo de Home.

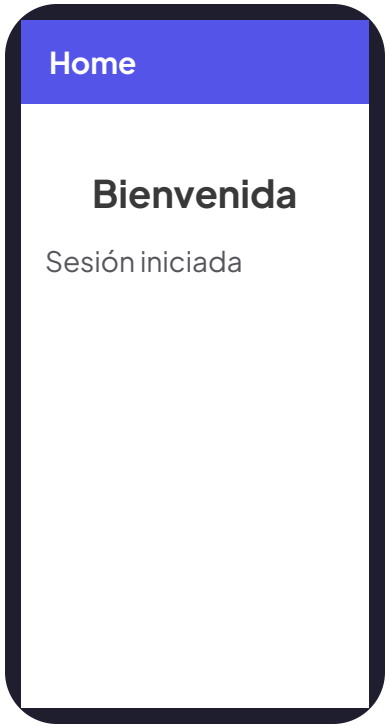
Login con pushNamed: el problema

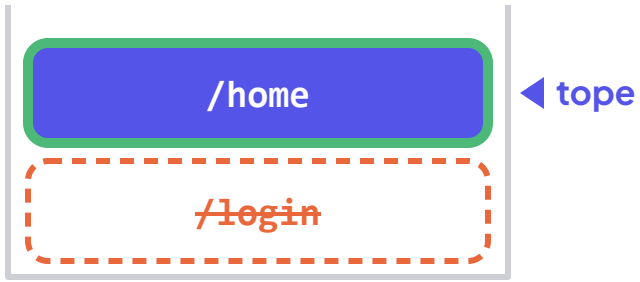
home_screen.dart

```
// el usuario toca Atrás
Navigator.pop(context);
```

Error de flujo: Atrás devuelve al formulario de login.

Login con pushNamed: el problema



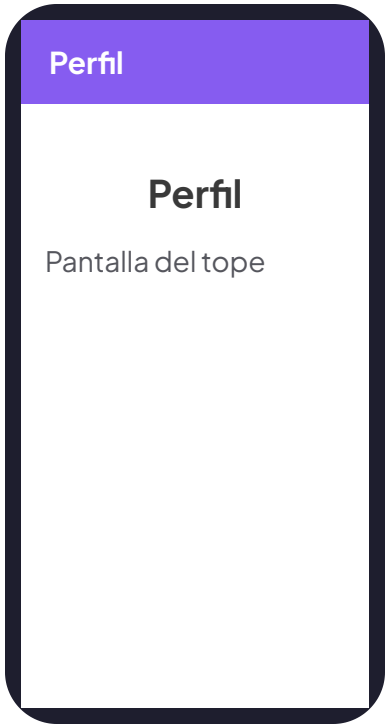


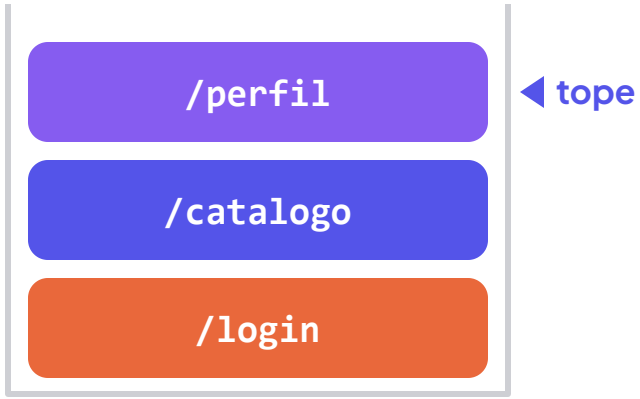
login_screen.dart

```dart
Navigator.pushNamedAndRemoveUntil(
  context,
  '/home',
  (route) => false,
);
```

El login se elimina. Atrás ya no vuelve a él.

(route) => false vacía la pila



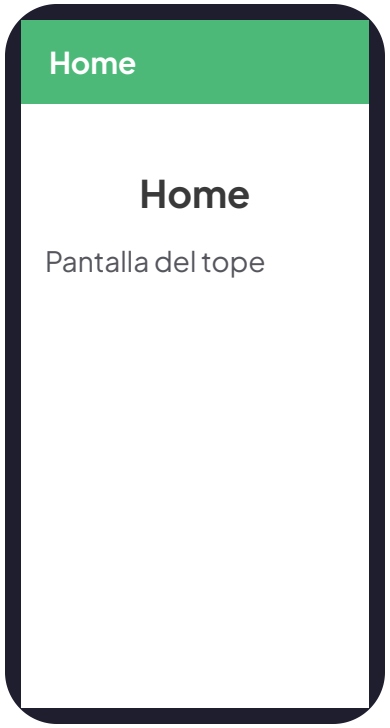


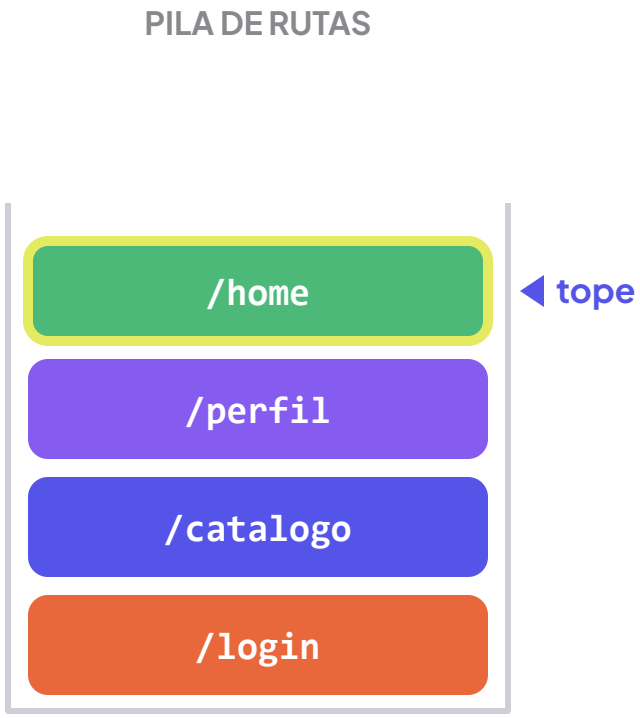
pushNamedAndRemoveUntil

```
Navigator.pushNamedAndRemoveUntil(
  context,
  '/home',
  (route) => false,
);
```

Tres pantallas en la pila.

(route) => false vacía la pila



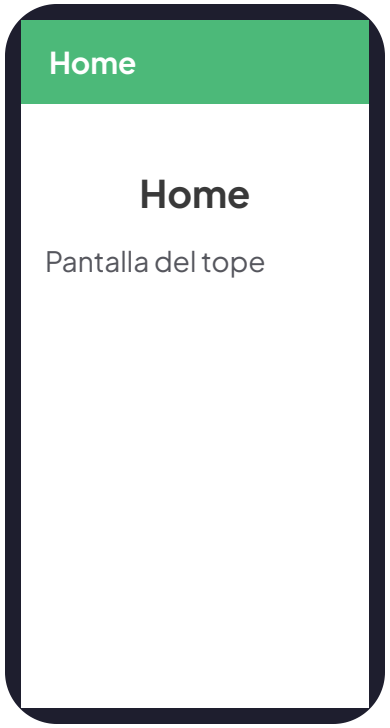


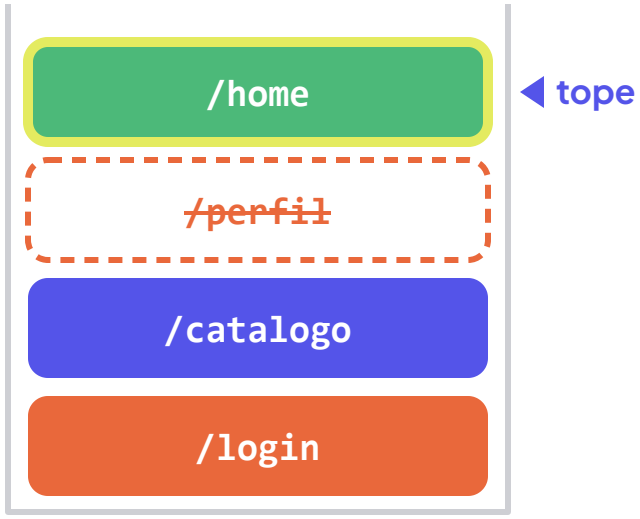
pushNamedAndRemoveUntil

```
Navigator.pushNamedAndRemoveUntil(
  context,
  '/home',
  (route) => false,
);
```

1. Se apila primero la ruta nueva.

(route) => false vacía la pila



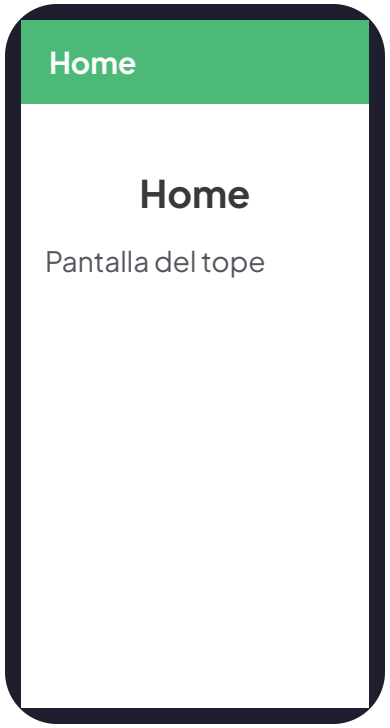


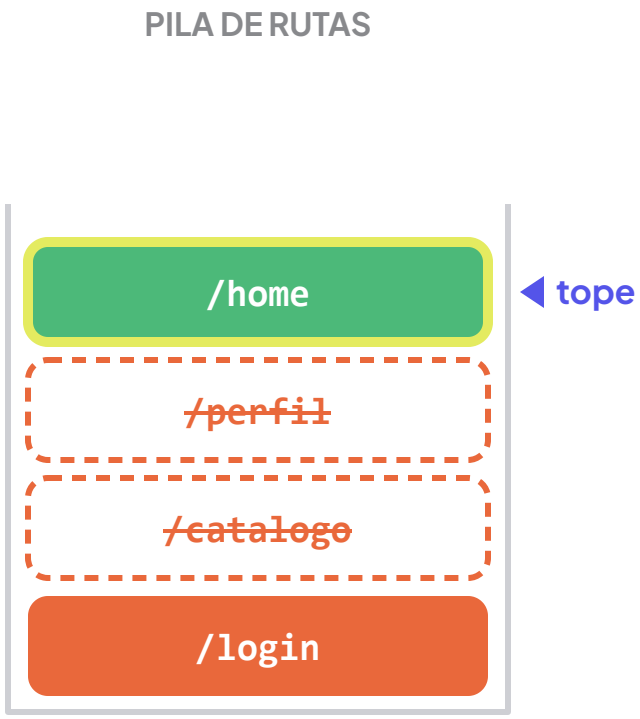
pushNamedAndRemoveUntil

```
Navigator.pushNamedAndRemoveUntil(
  context,
  '/home',
  (route) => false,
);
```

2. El predicado revisa de arriba abajo. Perfil: false, se elimina.

(route) => false vacía la pila



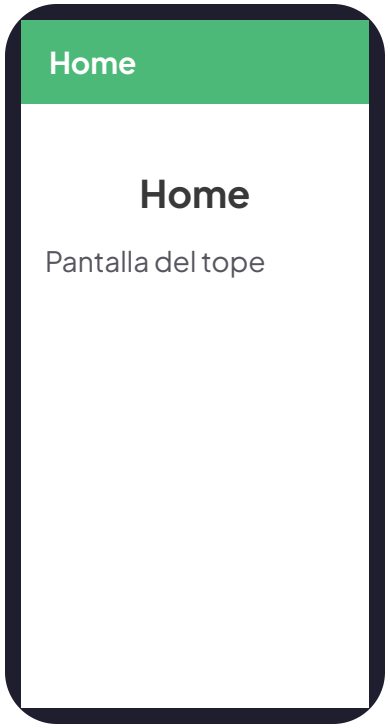


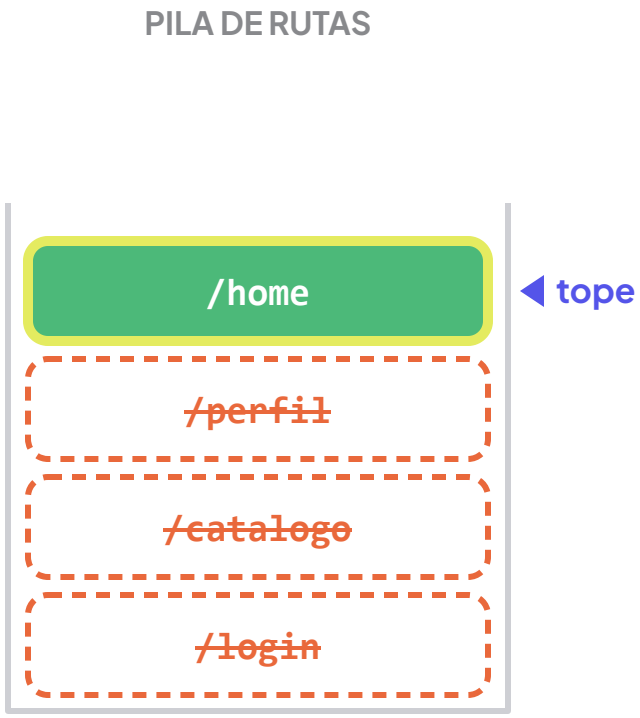
pushNamedAndRemoveUntil

```
Navigator.pushNamedAndRemoveUntil(
  context,
  '/home',
  (route) => false,
);
```

Catálogo: false, se elimina.

(route) => false vacía la pila



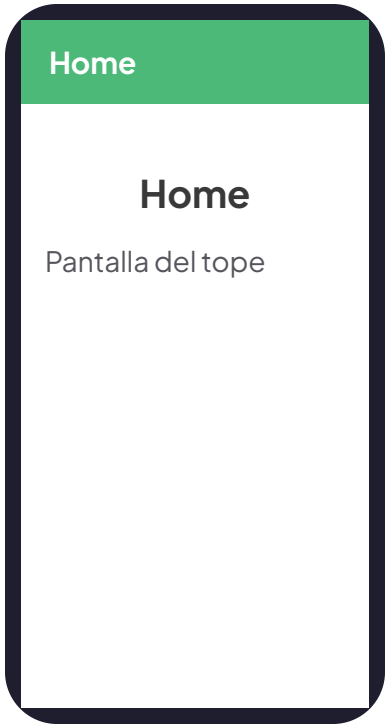


pushNamedAndRemoveUntil

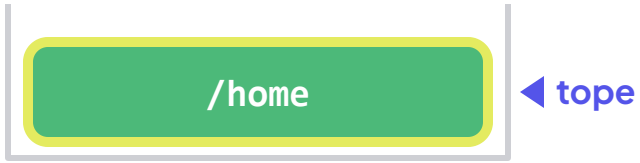
```
Navigator.pushNamedAndRemoveUntil(
  context,
  '/home',
  (route) => false,
);
```

Login: false, se elimina. Nada se conserva.

(route) => false vacía la pila



PILA DE RUTAS

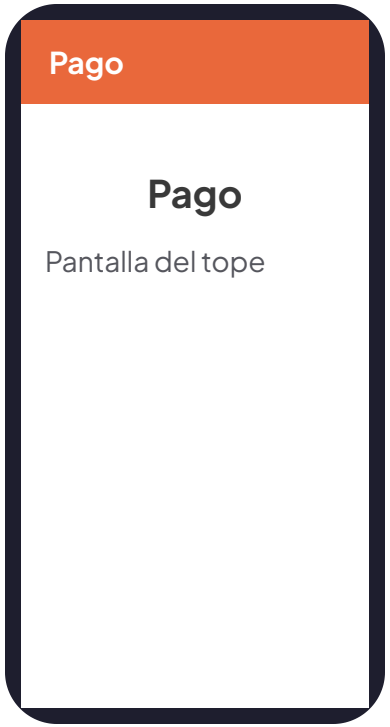


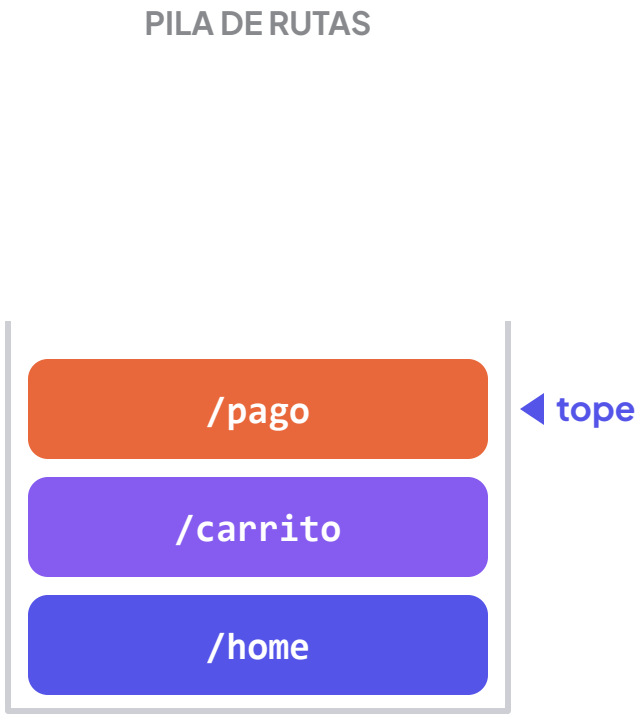
pushNamedAndRemoveUntil

```
Navigator.pushNamedAndRemoveUntil(
  context,
  '/home',
  (route) => false,
);
```

Resultado: Home es la única ruta. Atrás cierra la app.

route.isFirst conserva la raíz



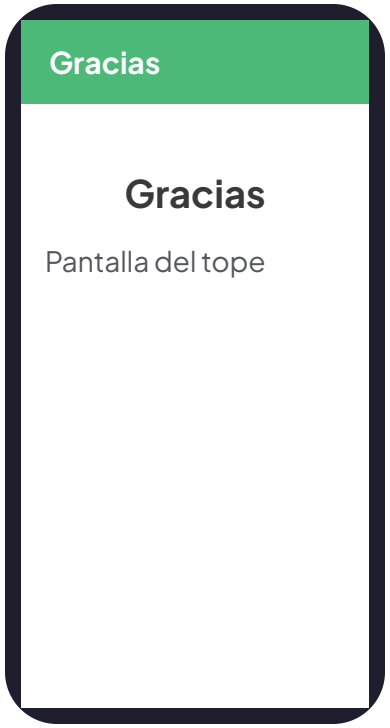


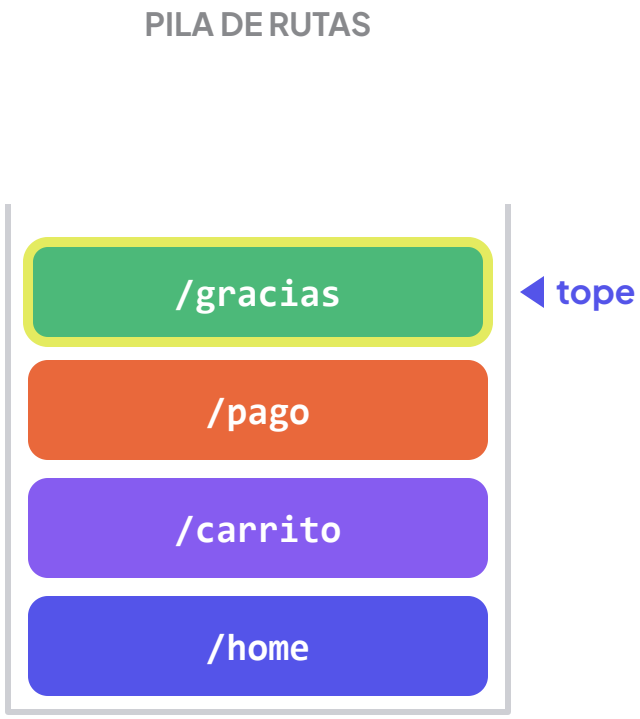
pushNamedAndRemoveUntil

```
Navigator.pushNamedAndRemoveUntil(
  context,
  '/gracias',
  (route) => route.isFirst,
);
```

Flujo de compra: Home, Carrito y Pago.

route.isFirst conserva la raíz



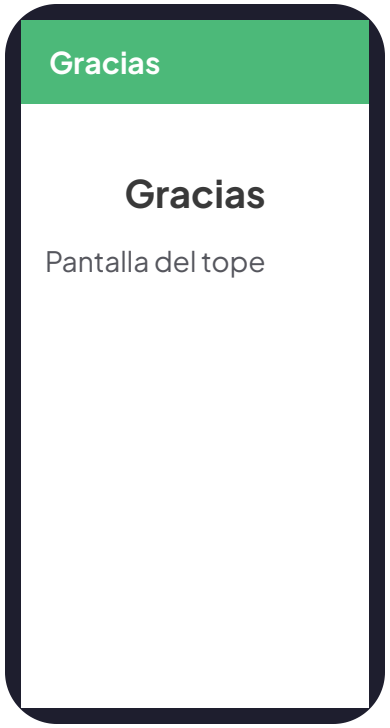


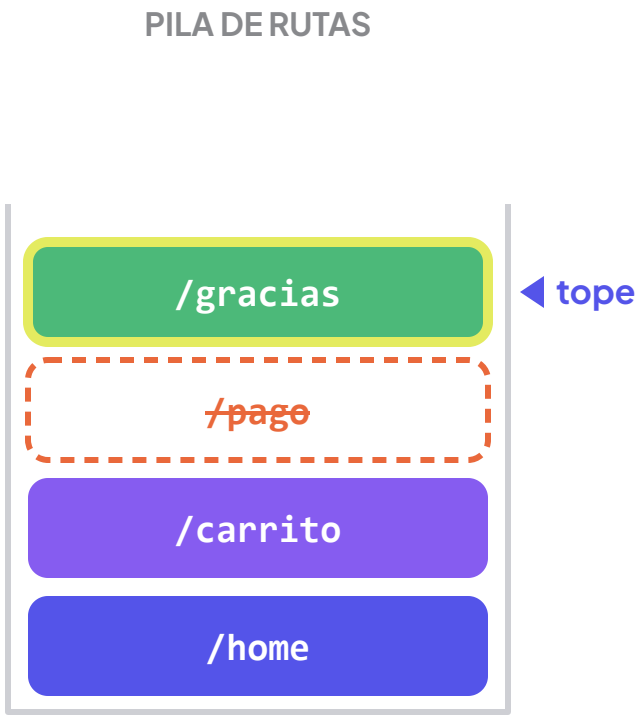
pushNamedAndRemoveUntil

```
Navigator.pushNamedAndRemoveUntil(
  context,
  '/gracias',
  (route) => route.isFirst,
);
```

Se apila Gracias.

route.isFirst conserva la raíz



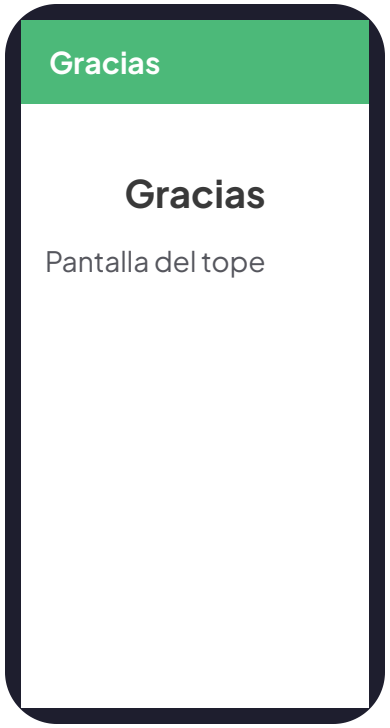


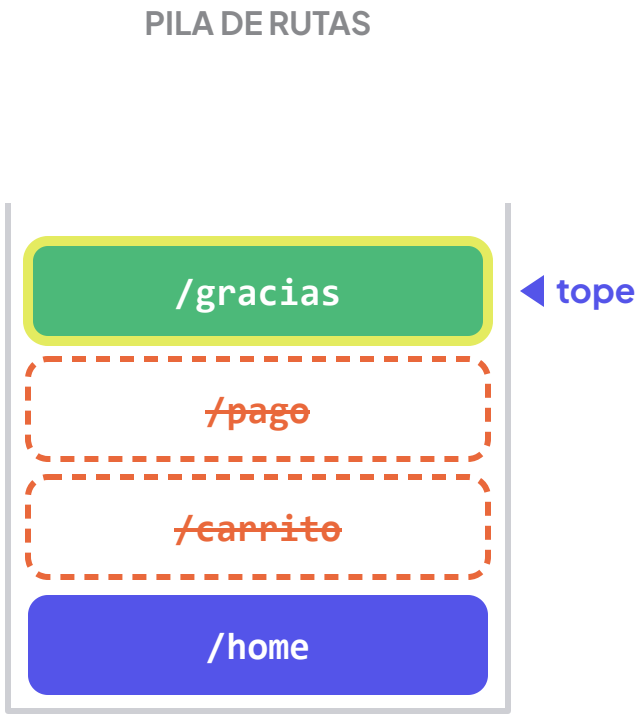
pushNamedAndRemoveUntil

```
Navigator.pushNamedAndRemoveUntil(
  context,
  '/gracias',
  (route) => route.isFirst,
);
```

Pago: isFirst es false, se elimina.

route.isFirst conserva la raíz



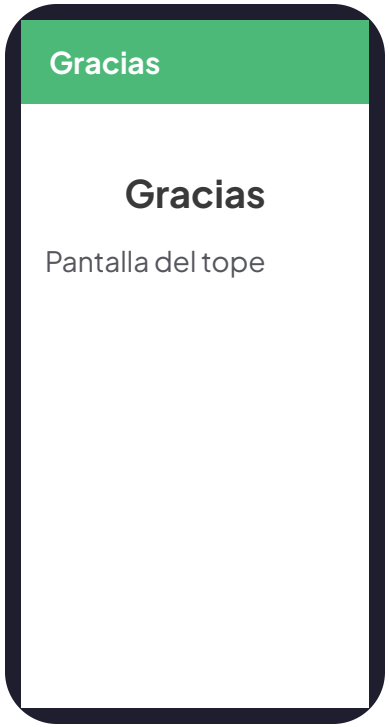


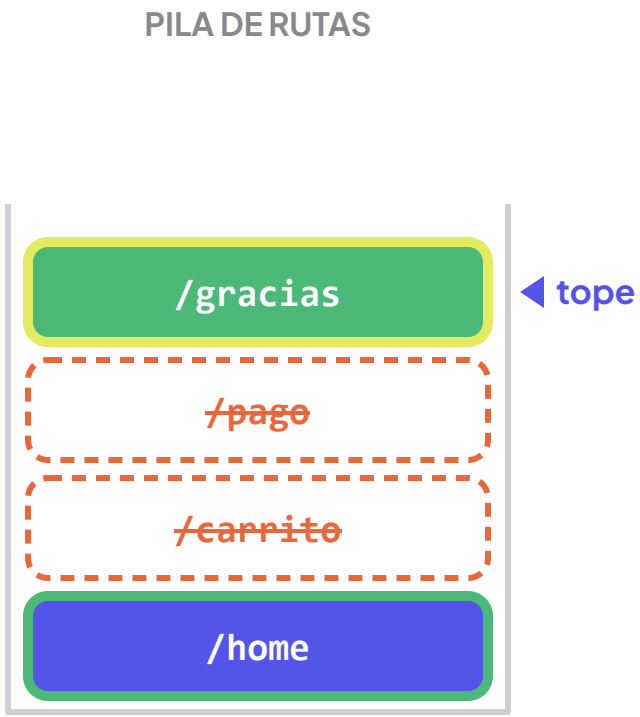
pushNamedAndRemoveUntil

```
Navigator.pushNamedAndRemoveUntil(
  context,
  '/gracias',
  (route) => route.isFirst,
);
```

Carrito: false, se elimina.

route.isFirst conserva la raíz



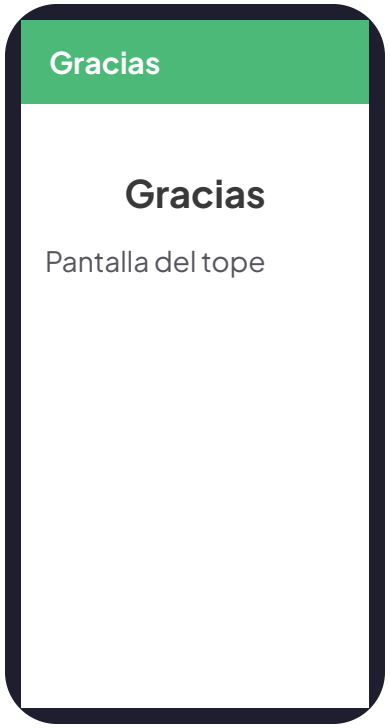


pushNamedAndRemoveUntil

```
Navigator.pushNamedAndRemoveUntil(
  context,
  '/gracias',
  (route) => route.isFirst,
);
```

Home: isFirst es true. La purga se detiene.

route.isFirst conserva la raíz



pushNamedAndRemoveUntil

```
Navigator.pushNamedAndRemoveUntil(
  context,
  '/gracias',
  (route) => route.isFirst,
);
```

Pila final: Home y Gracias. Atrás lleva a Home.

Predicados más usados

Predicado	Queda en la pila	Caso típico
`(route) => false`	Solo la ruta nueva	Login, logout, token vencido
`(route) => route.isFirst`	[raíz, nueva]	Fin de una compra
`ModalRoute.withName('/home')`	Desde esa ruta hacia abajo, y la nueva	Volver a una pantalla concreta

isFirst conserva la primera ruta de la pila, no a Home por nombre. Para conservar por nombre usa withName.

Cuadro resumen de métodos

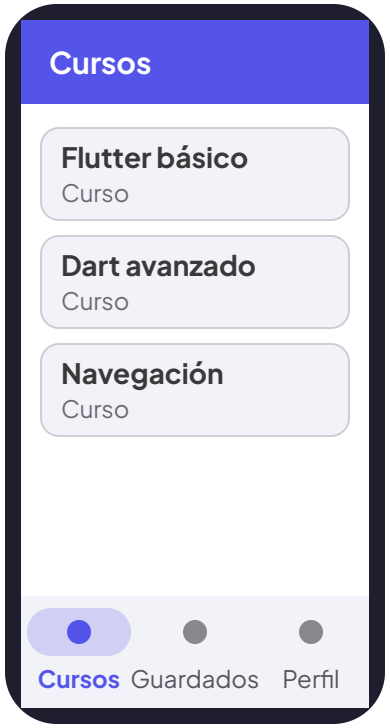
Método	Acción en la pila	Cuándo
`pushNamed`	Agrega al tope	Navegación jerárquica
`pop`	Quita el tope y devuelve un valor	Atrás, Cancelar, Guardar
`canPop`	Consulta si hay más de una ruta	Decidir si mostrar salida
`popUntil`	Desapila hasta cumplir el predicado	Volver a la raíz
`pushReplacementNamed`	Reemplaza el tope	Splash a Home
`pushNamedAndRemoveUntil`	Apila y purga con un predicado	Login, logout, compra

Universidad icesi

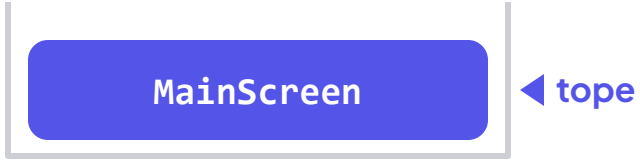
Navegación horizontal

Pestañas con NavigationBar

Pestaña o detalle: dos movimientos



PILA DE RUTAS

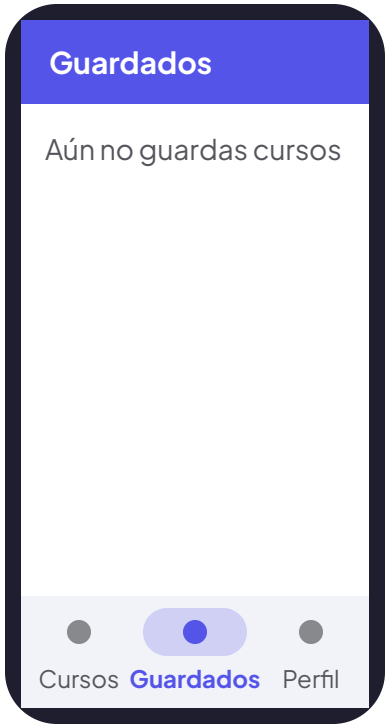


main_screen.dart

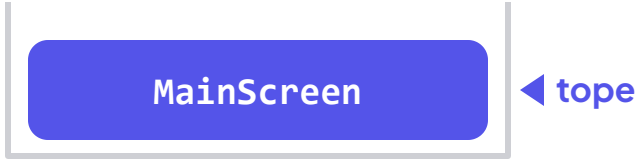
```dart
int _selectedIndex = 0;
```

Arranca en la pestaña 0. La pila tiene una ruta.

Pestaña o detalle: dos movimientos



PILA DE RUTAS

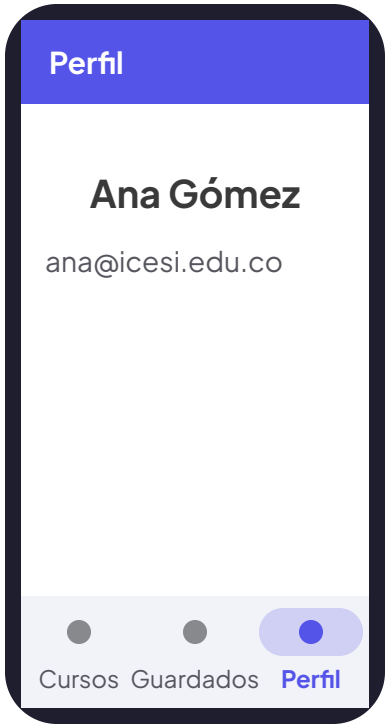


main_screen.dart

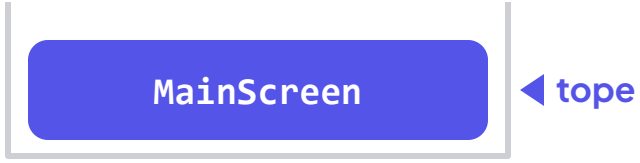
```
onDestinationSelected: (i) =>
  setState(() => _selectedIndex = i),
```

Cambiar de pestaña es setState. La pila no cambia.

Pestaña o detalle: dos movimientos



PILA DE RUTAS

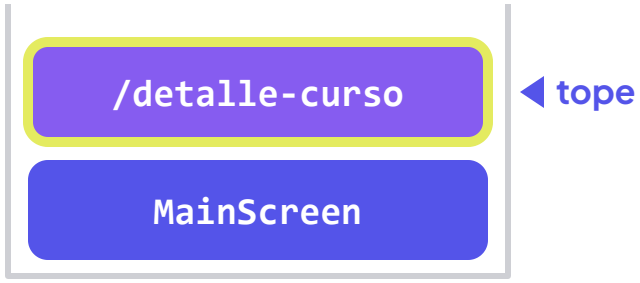


main_screen.dart

```
onDestinationSelected: (i) =>
  setState(() => _selectedIndex = i),
```

Otra pestaña: sigue una sola ruta.

Pestaña o detalle: dos movimientos

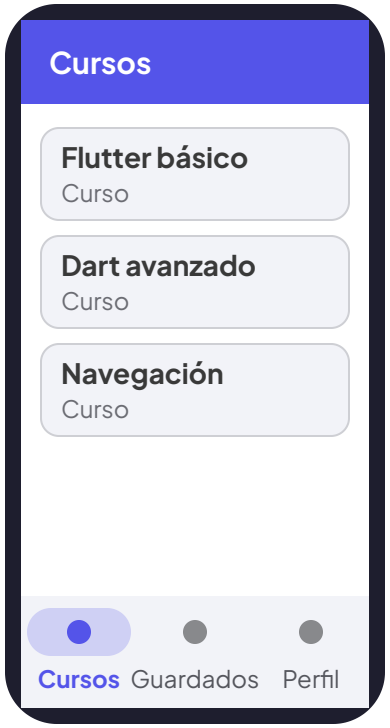


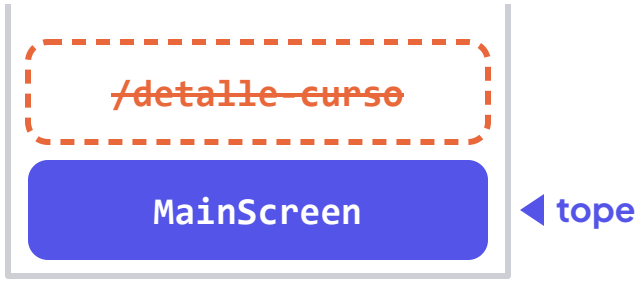
main_screen.dart

```dart
Navigator.pushNamed(
  context, '/detalle-curso');
```

Abrir un detalle es push: la pila crece y cubre la barra.

Pestaña o detalle: dos movimientos





main_screen.dart

```dart
Navigator.pop(context);
```

Atrás hace pop y reaparece la barra.

Universidad ICESI

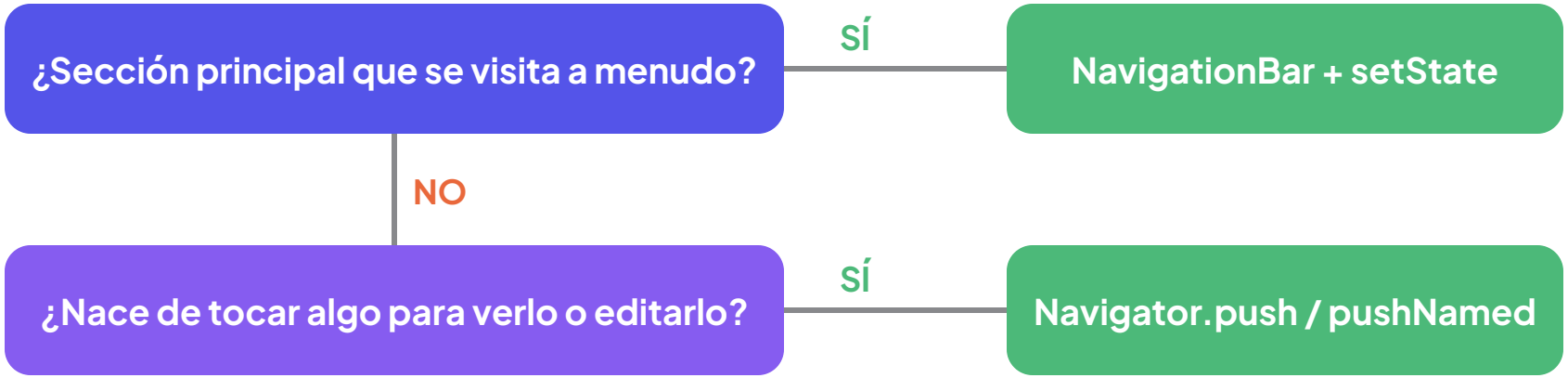
Pestaña frente a detalle

	Cambiar de pestaña	Abrir un detalle
Mecanismo	`setState`	`Navigator.push`
Dirección	Horizontal, entre hermanas	Jerárquica, hacia adentro
Pila de rutas	No cambia	Crece
Barra inferior	Siempre visible	Cubierta
Botón Atrás	No existe	Automático (pop)

¿Qué uso en cada caso?

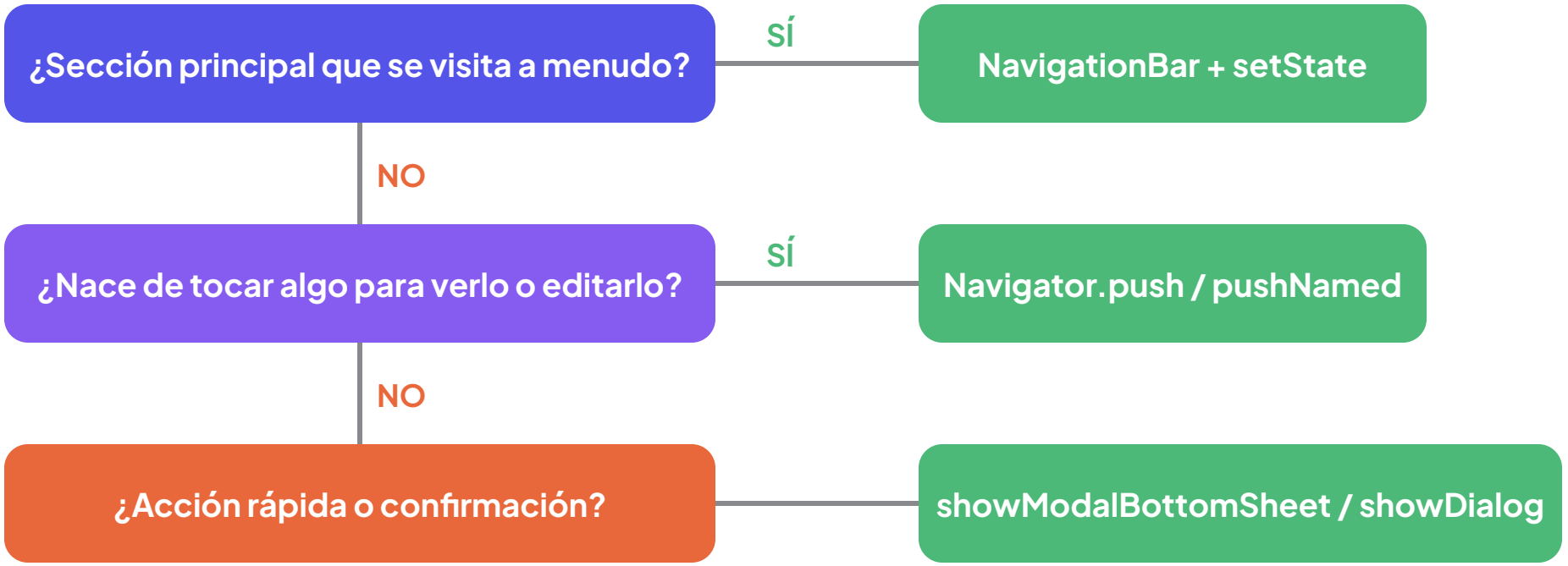
Pregunta 1: ¿es una sección principal?

¿Qué uso en cada caso?



Pregunta 2: ¿abre un elemento concreto?

¿Qué uso en cada caso?



Pregunta 3: ¿es una acción rápida?

Screen anfitriona y Page

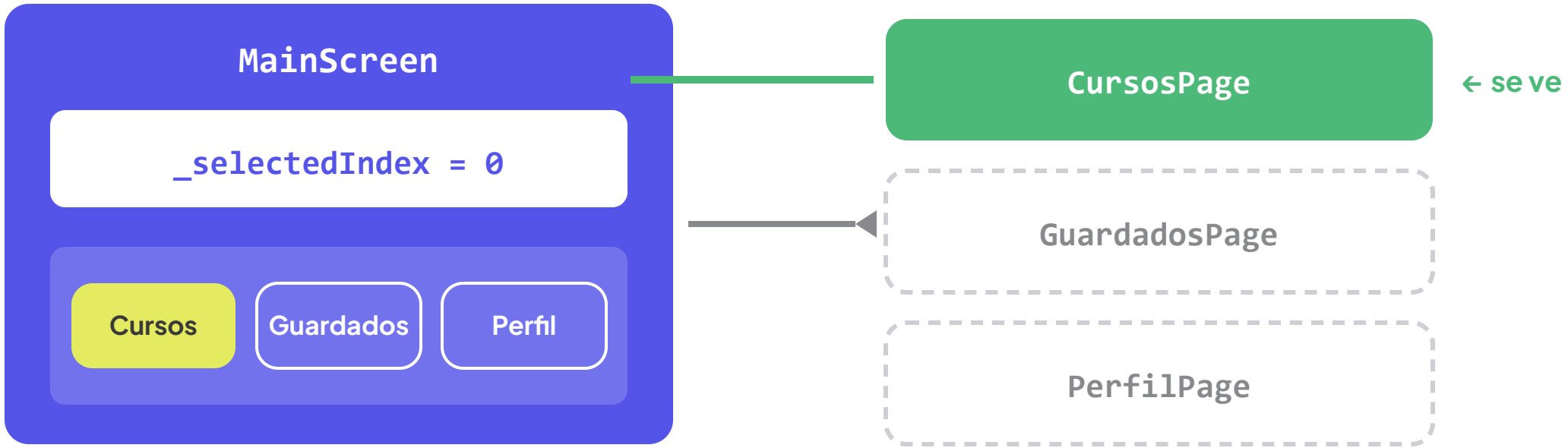
Screen anfitriona

Tiene el Scaffold, la barra inferior y un entero: _selectedIndex.

Page de contenido

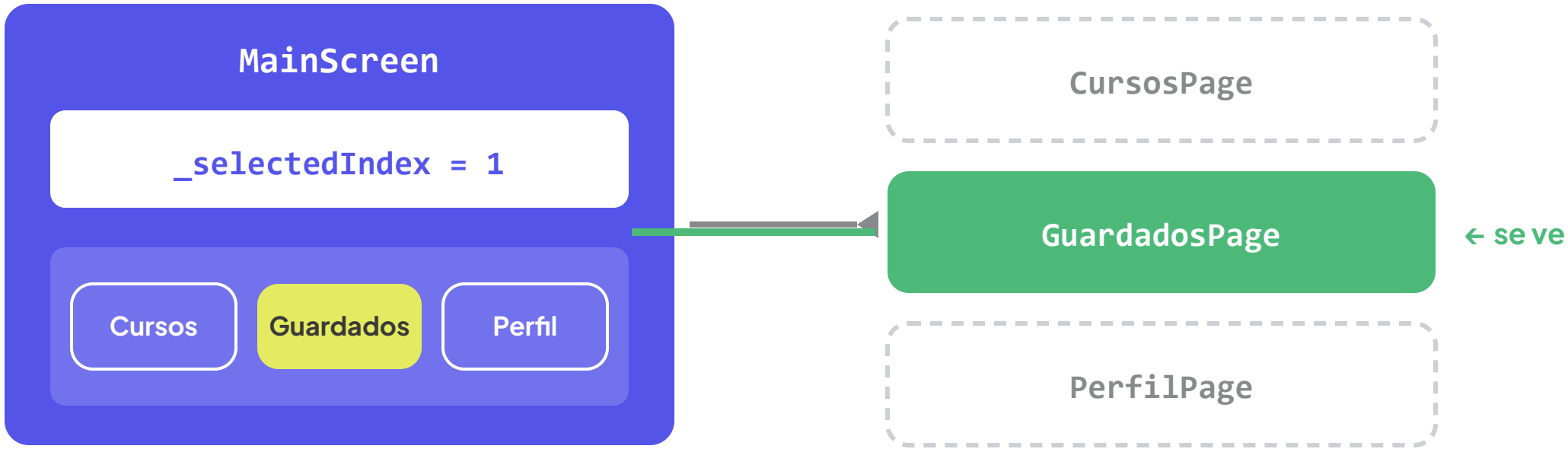
Dibuja solo el cuerpo de una sección. No declara su propio Scaffold.

Un índice decide qué se ve



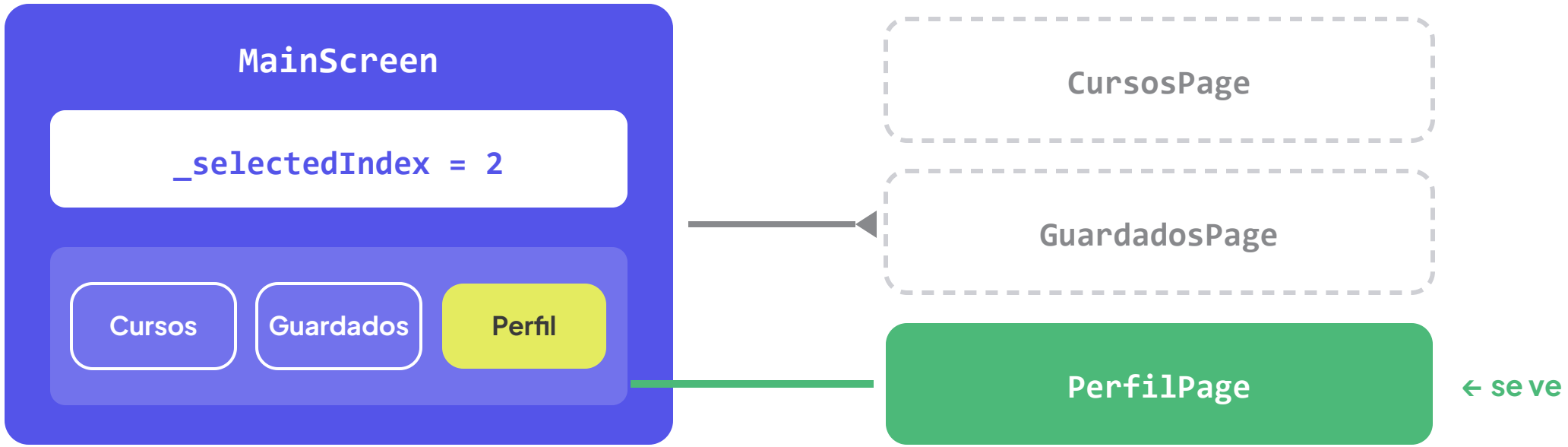
Índice 0: se muestra CursosPage.

Un índice decide qué se ve



Se toca Guardados: setState cambia el índice a 1.

Un índice decide qué se ve



Índice 2: se muestra PerfilPage.

IndexedStack conserva el estado

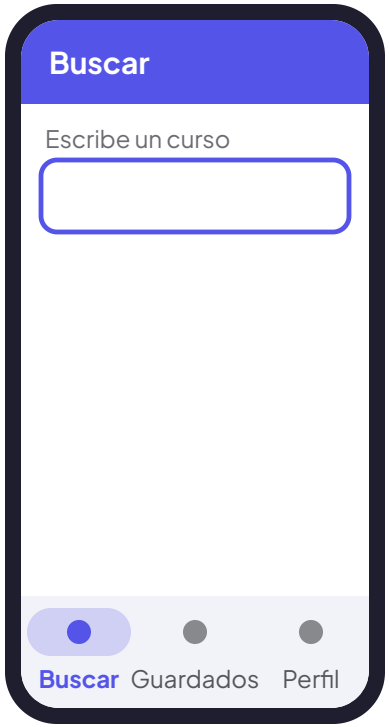
`_paginas[_selectedIndex]`

BuscarPage
montada, visible

GuardadosPage
destruida

PerfilPage
destruida

`IndexedStack`



BuscarPage
montada, visible

GuardadosPage
montada, oculta

PerfilPage
montada, oculta

Las dos arrancan en Buscar. IndexedStack monta todas las páginas.

IndexedStack conserva el estado

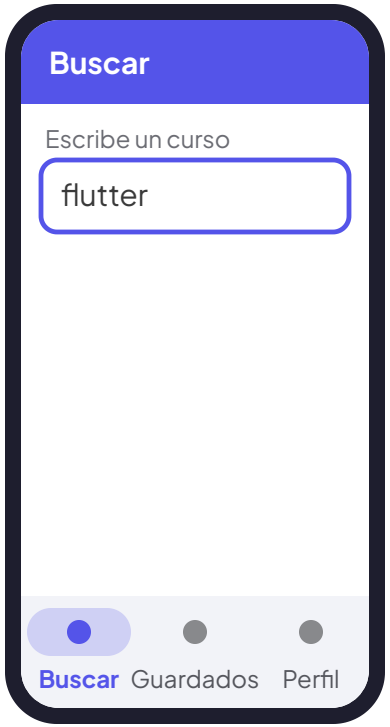
`_paginas[_selectedIndex]`

BuscarPage
montada, visible

GuardadosPage
destruida

PerfilPage
destruida

`IndexedStack`



BuscarPage
montada, visible

GuardadosPage
montada, oculta

PerfilPage
montada, oculta

El usuario escribe «flutter» en Buscar.

IndexedStack conserva el estado

Pasa a Guardados. A la izquierda, Buscar se destruye; a la derecha sigue montada.

IndexedStack conserva el estado

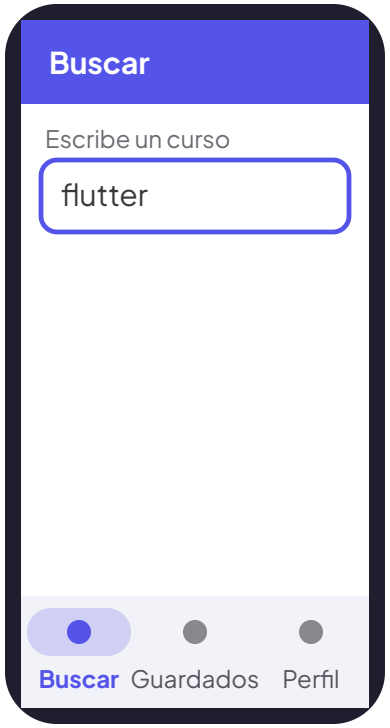
`_paginas[_selectedIndex]`

BuscarPage
montada, visible

GuardadosPage
destruida

PerfilPage
destruida

`IndexedStack`



BuscarPage
montada, visible

GuardadosPage
montada, oculta

PerfilPage
montada, oculta

Vuelve a Buscar: la lista directa perdió el texto; IndexedStack lo conserva.

screens/ y pages/

Estructura del proyecto

```
lib/
├── main.dart
├── screens/
│   ├── main_screen.dart
│   └── detalle_curso_screen.dart
└── pages/
    ├── cursos_page.dart
    ├── guardados_page.dart
    └── perfil_page.dart
```

screens/

Pantallas completas con Scaffold: la anfitriona y el detalle, que se abre con push.

screens/ y pages/

Estructura del proyecto

```
lib/
├── main.dart
├── screens/
│   ├── main_screen.dart
│   └── detalle_curso_screen.dart
└── pages/
    ├── cursos_page.dart
    ├── guardados_page.dart
    └── perfil_page.dart
```

pages/

Solo el cuerpo de cada pestaña. La anfitriona los coloca en su body.

screens/ y pages/

Estructura del proyecto

```
lib/
├── main.dart
├── screens/
│   ├── main_screen.dart
│   └── detalle_curso_screen.dart
└── pages/
    ├── cursos_page.dart
    ├── guardados_page.dart
    └── perfil_page.dart
```

main.dart

Arranca la app: MaterialApp(home: MainScreen()).

main_screen.dart, por partes

screens/main_screen.dart

```dart
class _MainScreenState extends State<MainScreen> {
  int _selectedIndex = 0;
  final _paginas = const [
    CursosPage(),
    GuardadosPage(),
    PerfilPage(),
  ];
}
```

El estado

Una sola variable: el índice de la pestaña elegida.

main_screen.dart, por partes

screens/main_screen.dart

```dart
@override
Widget build(BuildContext context) {
  return Scaffold(
    body: IndexedStack(
      index: _selectedIndex,
      children: _paginas,
    ),
    bottomNavigationBar: NavigationBar(
      selectedIndex: _selectedIndex,
      onDestinationSelected: (i) =>
          setState(() => _selectedIndex = i),
      destinations: const [ ... ],
    ),
  );
}
```

El cuerpo

IndexedStack muestra una página y mantiene montadas todas.

main_screen.dart, por partes

screens/main_screen.dart

```dart
@override
Widget build(BuildContext context) {
  return Scaffold(
    body: IndexedStack(
      index: _selectedIndex,
      children: _paginas,
    ),
    bottomNavigationBar: NavigationBar(
      selectedIndex: _selectedIndex,
      onDestinationSelected: (i) =>
          setState(() => _selectedIndex = i),
      destinations: const [ ... ],
    ),
  );
}
```

La barra

Al tocar una pestaña, setState cambia el índice y build corre otra vez.

Universidad ICESI

Buenas prácticas

Un solo Scaffold

Las Page no repiten Scaffold ni barra.

IndexedStack

Úsalo si la pestaña tiene formularios, scroll o paginación.

El detalle tapa la barra

Navigator.push lo cubre: no compite por atención.

Pestaña sin pushNamed

Apilaría una copia de la anfitriona en cada toque.

Universidad ICESI

Lo que te llevas

Pila

pushNamed apila, pop desapila. Solo se ve el tope.

Datos

arguments al ir; await y pop(resultado) al volver.

Purga

pushNamedAndRemoveUntil con un predicado.

Pestañas

setState cambia de pestaña; push abre un detalle.

Universidad Icesi

Preguntas

Semana 13

Seminario de I+D+I — Universidad Icesi