📝 Cheatsheet – Navegación, Estado y Responsive Design en Flutter

Unidad 07 - Autor: Sergi García Barea

🧱 Arquitectura MVVM en Flutter

- **Model:** Gestiona la lógica de negocio y datos (consultas a base de datos, APIs, etc.).
- **View:** Todo lo que ve el usuario (widgets que componen la interfaz gráfica).
- **ViewModel:** Actúa como puente entre View y Model. Contiene la lógica de presentación y maneja el estado.

🧩 Ventajas:

- Separa claramente responsabilidades.
- Facilita pruebas automáticas y mantenimiento.
- Mejora la escalabilidad del proyecto.

🧭 Navegación en Flutter

1. Navegación con `Navigator` (Imperativa)

- **`Navigator.push(context, ...)`**: añade una nueva pantalla a la pila.
- **`Navigator.pop(context)`**: elimina la pantalla actual de la pila y vuelve a la anterior.

🧪 Ejemplo:

```
Navigator.push(
  context,
  MaterialPageRoute(builder: (context) => PantallaDetalle()),
);
```

📌 Recomendado para:

- Aplicaciones pequeñas.
- Navegación básica sin paso complejo de datos.

2. Navegación con GoRouter (Declarativa)

- **GoRouter** permite definir rutas de forma centralizada y navegar como si fuera una web.
- Usa rutas nombradas y parámetros dinámicos.

🧪 Ejemplo:

```
context.go('/perfil');
context.push('/producto/123'); // con parámetro dinámico
```

CC BY NC SA

Unidad 07 - Autor: Sergi García Barea

🎯 Ventajas:

- Ideal para Flutter Web.
- Facilita el deep linking (enlace directo a páginas internas).
- Compatible con arquitecturas modernas.

📌 Recomendado para:

- Aplicaciones medianas y grandes.
- Cuando se requiere mantener historial de navegación o usar URLs.

📦 Paso de datos entre pantallas

➤ Mediante constructor (forma directa):

```
Navigator.push(
  context,
  MaterialPageRoute(builder: (_) => Detalle(producto: miProducto)),
);
```

➤ Con argumentos en rutas nombradas:

```
onGenerateRoute: (settings) {
  if (settings.name == '/detalle') {
    final producto = settings.arguments as Producto;
    return MaterialPageRoute(builder: (_) => Detalle(producto: producto));
  }
}
```

🔑 Consejo: no sobrecargues la navegación con muchos datos → usa gestores de estado.

🔄 Gestión de Estado

1. `setState()` (local y sencillo)

- Actualiza el estado de un widget puntual. No mantiene el estado entre pantallas.

📌 Ideal para: prototipos y apps pequeñas.

2. `Provider` (global y recomendado)

- Usa `ChangeNotifier` para escuchar y notificar cambios.
- Permite compartir datos entre widgets sin tener que pasarlos manualmente.

🧪 Ejemplo básico:

```
Provider.of<UsuarioProvider>(context).cambiarNombre('María');
```

CC BY NC SA

Unidad 07 - Autor: Sergi García Barea

🎯 Beneficios:

- Fácil de usar.
- Integración oficial con Flutter.
- Escalable para apps medianas.

3. `Riverpod` (moderno y avanzado)

- Separación completa entre lógica y UI.
- Mejor testabilidad y sin dependencias del contexto.

📌 Recomendado para apps grandes o proyectos a largo plazo.

📱 Diseño Responsive en Flutter

📐 El diseño responsive permite que la interfaz se adapte automáticamente al tamaño de pantalla del dispositivo: móvil, tablet o escritorio.

🧰 Herramientas clave

- **MediaQuery**: obtiene información del dispositivo (ancho, alto, orientación).

```
final ancho = MediaQuery.of(context).size.width;
```

- **LayoutBuilder**: permite diseñar basándose en el tamaño del widget padre.
- **OrientationBuilder**: detecta si el dispositivo está en modo retrato o apaisado.

📏 Breakpoints comunes

Dispositivo	Ancho
Móvil	< 600px
Tablet	600 – 1200px
Escritorio	> 1200px

🧪 Diseño adaptado:

```
LayoutBuilder(
  builder: (context, constraints) {
    if (constraints.maxWidth < 600) return VistaMovil();
    if (constraints.maxWidth < 1200) return VistaTablet();
    return VistaEscritorio();
  },
);
```