La gestion d’état

Introduction aux Widgets

La gestion d’état - setState mais pas que !

```
Future<void> _incrementCounter() async {
  setState(() {
    _counter++;
  });
}
```

Très bien pour une application simple, mais commence à poser des problèmes quand l’application se complexifie !

🔥 Difficulté de gestion de l'état partagé 🔥

🔥 Manque de séparation des préoccupations 🔥

Gestionnaires d’états

Provider

Riverpod

ValueNotifier & InheritedNotifier

InheritedWidget & InheritedModel

BLoC

GetX

La gestion d’état

Provider

Provider

https://pub.dev/packages/provider

La gestion d’état

Analogie - Avec Provider

La gestion d’état

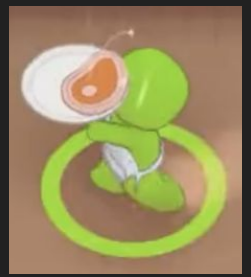
Analogie - Avec Provider

Le Provider

aka Le Chef

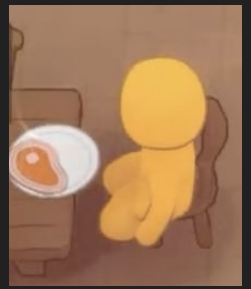
Le Consumer

aka Le Serveur



L’utilisateur

aka Le Client



La gestion d’état

Introduction à Provider

Qu'est-ce que Provider ?

Un pattern architectural pour gérer l'état.

Pourquoi utiliser Provider ?

Séparation des préoccupations, testabilité, meilleure maintenabilité.

Les composants clés :

Provider, Consumer

La gestion d’état

Provider, et en pratique ? Le compteur, encore et toujours !

```
class _MyHomePageState extends State<MyHomePage> {
 int _counter = 0;

 void _incrementCounter() {
   setState(() {
     _counter++;
   });
 }

 @override
 Widget build(BuildContext context) {
   return Scaffold(
     ...
     floatingActionButton: FloatingActionButton(
       onPressed: _incrementCounter,
       tooltip: 'Increment',
       child: const Icon(Icons.add),
     ),
   );
 }
}
```

La gestion d’état

Provider - ChangeNotifierProvider

```
runApp(
    MultiProvider(
      providers: [
        ChangeNotifierProvider(create: (_) => CounterProvider()),
      ],
      child: const MyApp(),
    ),
    //    )
 );
}
```

La gestion d’état

Provider - Provider

```
import 'package:flutter/material.dart' ;

class CounterProvider extends ChangeNotifier {
 int  count = 0;
 int get count => _count;

 void increment() {
    _count++;
   notifyListeners();
 }
}
```

Nous permet de modifier et gérer la valeur du compteur et d’avertir les éléments qui consomment/écoutent le changement !

La gestion d’état

Provider - Consumer

```
Consumer<CounterProvider>(
 builder: (context, counter, child) {
   return Text(
     '$counter',
     style: Theme.of(context).textTheme.headlineMedium,
   );
 },
),
```

Se reconstruit automatique lorsque la donnée du Provider change.

La gestion d’état

Provider - Provider.of

```
Provider.of<CounterProvider>(context, listen: true).count;
```

Permet d'accéder directement à la valeur du Provider.

La gestion d’état

Provider- context.watch

```
context.watch<CounterProvider>().count
```

Similaire à Consumer, mais plus concis. Se reconstruit automatiquement lorsque la donnée du Provider change.

La gestion d’état

Provider - context.read

```
context.read<CounterProvider>().increment()
```

Permet de lire la valeur du provider sans le reconstruire.

La gestion d’état

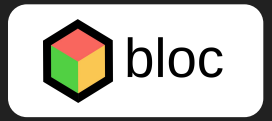
BLoC

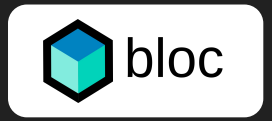
BLoC

https://pub.dev/packages/bloc

https://pub.dev/packages/flutter_bloc

https://pub.dev/packages/bloc_test





La gestion d’état

Introduction à BLoC

Qu'est-ce que BLoC ?

Un pattern architectural pour gérer l'état.

Pourquoi utiliser BLoC ?

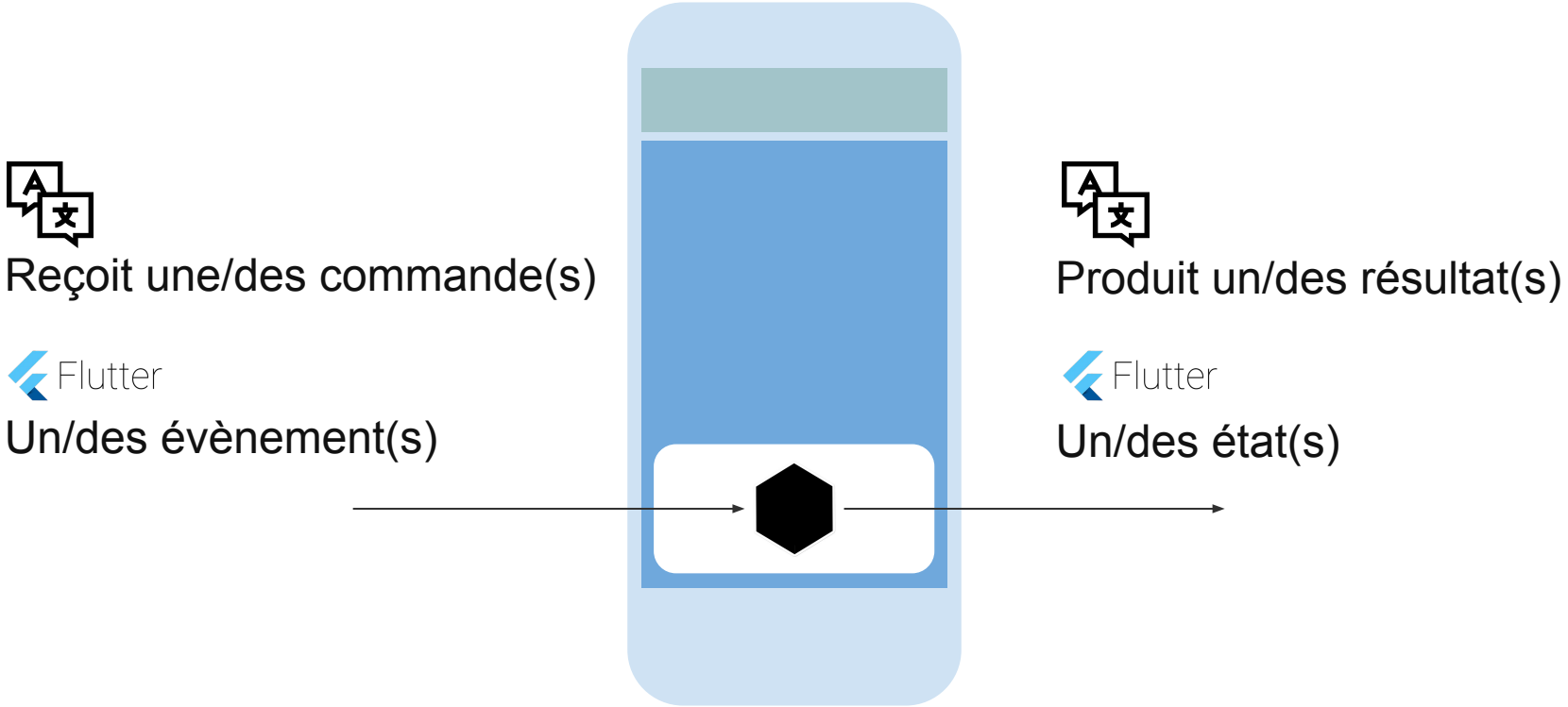
Séparation des préoccupations, testabilité, meilleure maintenabilité.

Les composants clés :

Événements, États, Bloc, Cubit.

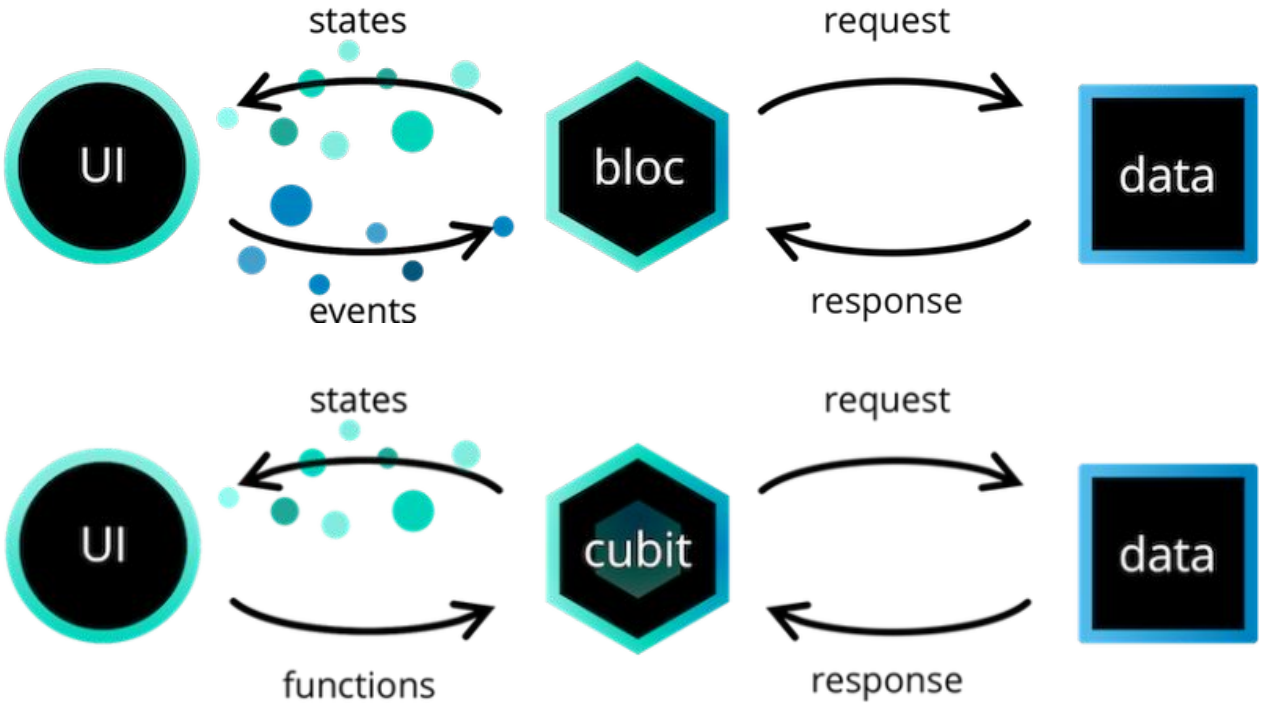
La gestion d’état

Comment définir BLoC ?



La gestion d'état

Un schéma de BLoC !



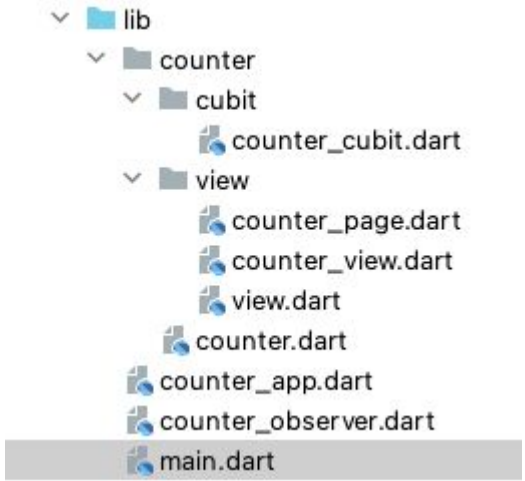
La gestion d'état

BLoC, et en pratique ? Le compteur !

```
class _MyHomePageState extends State<MyHomePage> {
 int _counter = 0;

 void _incrementCounter() {
   setState(() {
     _counter++;
   });
 }

 @override
 Widget build(BuildContext context) {
   return Scaffold(
     ...
     floatingActionButton: FloatingActionButton(
       onPressed: _incrementCounter,
       tooltip: 'Increment',
       child: const Icon(Icons.add),
     ),
   );
 }
}
```

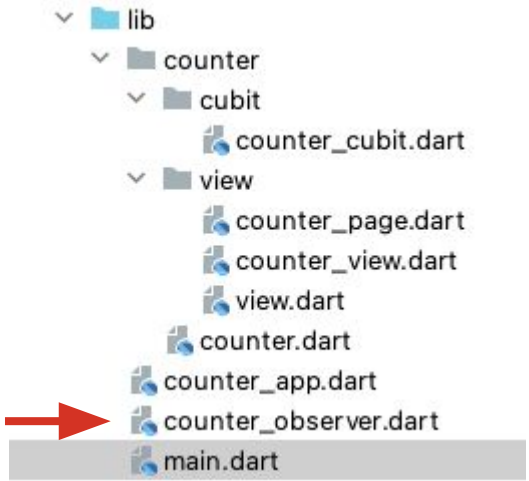
La gestion d’état

Structure du projet - Structure



La gestion d’état

Structure du projet - BlocObserver



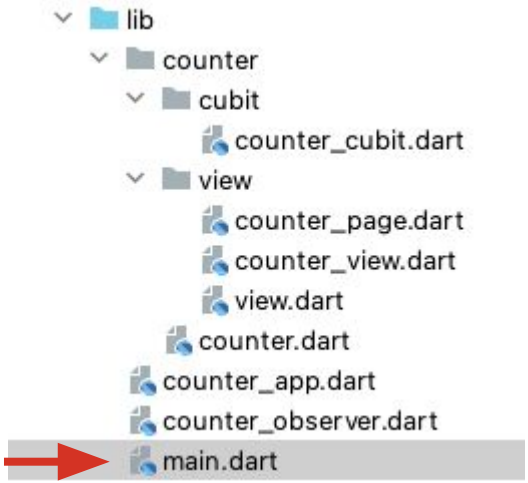
Bloc Observer

```dart
import 'package:bloc/bloc.dart';

class CounterObserver extends BlocObserver {
 const CounterObserver();

 @override
 void onChange(BlocBase<dynamic> bloc, Change<dynamic> change) {
   super.onChange(bloc, change);
   print('${bloc.runtimeType} $change');
 }
}
```

Surveille les modifications d’état et les affiche dans la console pour le debug. Hors prod !

La gestion d’état

Structure du projet - main()



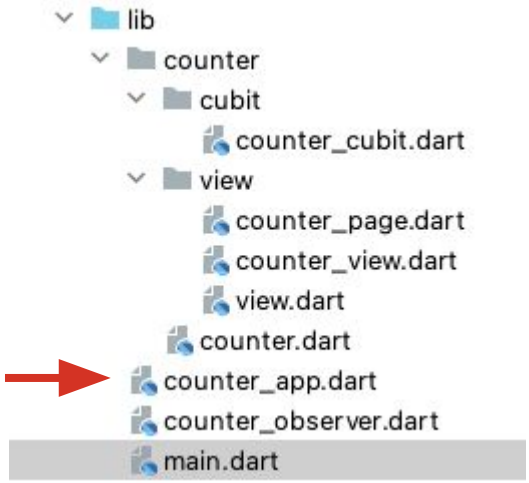
main.dart

```dart
import 'package:bloc/bloc.dart';
import 'package:flutter/widgets.dart';

import 'counter app.dart';
import 'counter_observer.dart';

void main() {
 Bloc.observer = const CounterObserver();
 runApp(const CounterApp());
}
```

Initialise le BlocObserver et on lance l’app via le widget CounterApp

La gestion d’état

Structure du projet - App



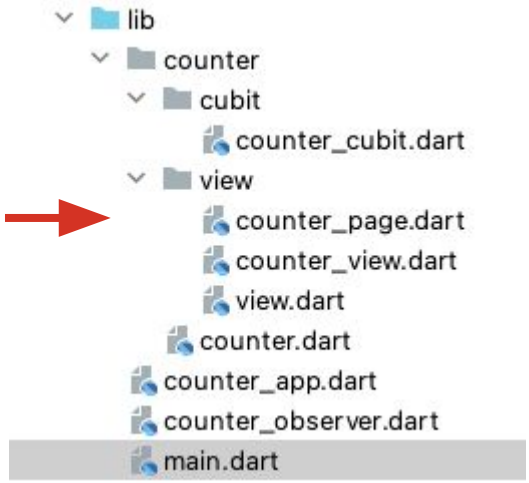
counter_app.dart

```dart
import 'package:flutter/material.dart';

import 'counter/view/counter_page.dart';

class CounterApp extends MaterialApp {
 const CounterApp({super.key}) : super(home: const CounterPage());
}
```

CounterApp est une MaterialApp et CounterPage est notre page principale.

La gestion d'état

Structure du projet - Page



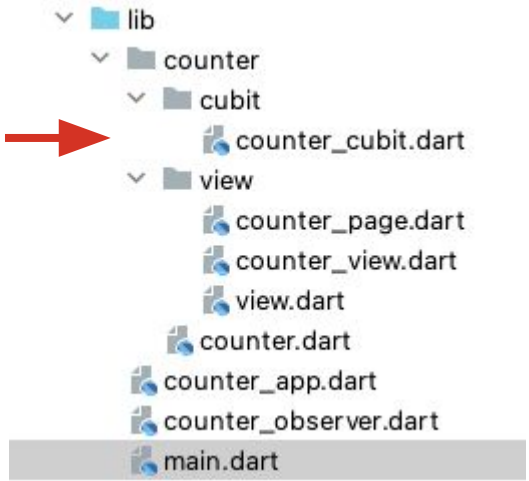
counter_page.dart

```dart
import 'package:flutter/material.dart';
import 'package:flutter_bloc/flutter_bloc.dart';

import '../cubit/counter_cubit.dart';
export 'counter_page.dart';
export 'counter_view.dart';


import 'counter_view.dart';

import '../counter.dart';


class CounterPage extends StatelessWidget {
 const CounterPage({super.key});

 @override
 Widget build(BuildContext context) {
   return BlocProvider(
     create: ( ) => CounterCubit(),
     child: const CounterView(),
   );
 }
}
```

Création d'une instance de CounterCubit pour CounterView

La gestion d’état

Structure du projet - Cubit



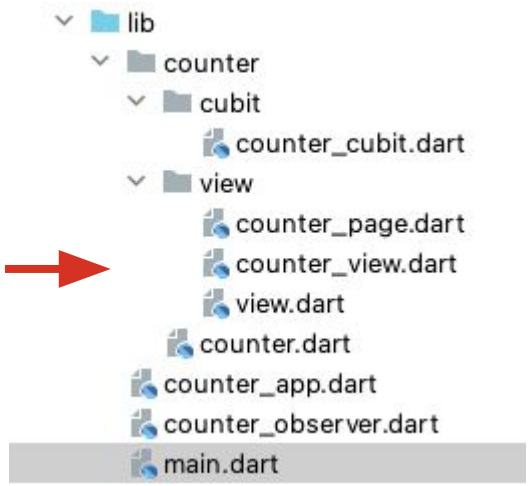
counter_cubit.dart

```dart
import 'package:bloc/bloc.dart';

class CounterCubit extends Cubit<int> {
 CounterCubit() : super(0);

 void increment() => emit(state + 1);

 void decrement() => emit(state - 1);
}
```

Gestion de l’état du compteur en exposant deux méthodes
=> increment : ajoute 1
=> decrement : soustrait 1

La gestion d’état

Structure du projet - View



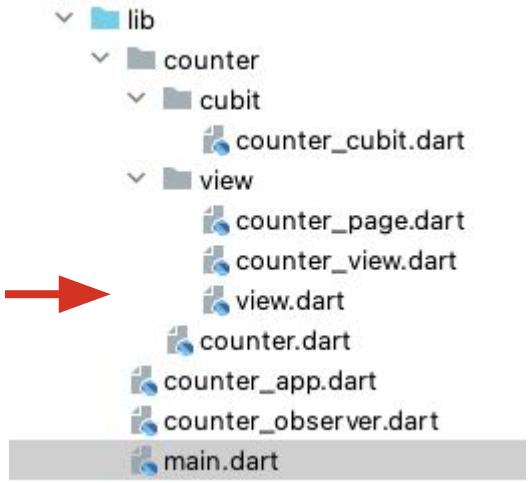
Affichage du compteur et gestion des interactions. Utilise BlocBuilder pour reconstruire le widget à chaque modification de l’état

counter_view.dart

```dart
import 'package:flutter/material.dart';
import 'package:flutter_bloc/flutter_bloc.dart';

import '../cubit/counter_cubit.dart';

class CounterView extends StatelessWidget {
 const CounterView({super.key});

 @override
 Widget build(BuildContext context) {
   final textTheme = Theme.of(context).textTheme;
   return Scaffold(
     body: Center(
       child: BlocBuilder<CounterCubit, int>(
         builder: (context, state) {
           return Text('$state', style: textTheme.displayMedium);
         },
       ),
     ),
     floatingActionButton: Column(
       mainAxisAlignment: MainAxisAlignment.end,
       crossAxisAlignment: CrossAxisAlignment.end,
       children: <Widget>[
         FloatingActionButton(
           key: const Key('counterView_increment_floatingActionButton'),
           child: const Icon(Icons.add),
           onPressed: () => context.read<CounterCubit>().increment(),
         ),
         const SizedBox(height: 8),
         FloatingActionButton(
           key: const Key('counterView_decrement_floatingActionButton'),
           child: const Icon(Icons.remove),
           onPressed: () => context.read<CounterCubit>().decrement(),
         ),
       ],
     ),
   );
 }
}
```

La gestion d’état

Structure du projet - Simplifier les imports - view.dart



view.dart

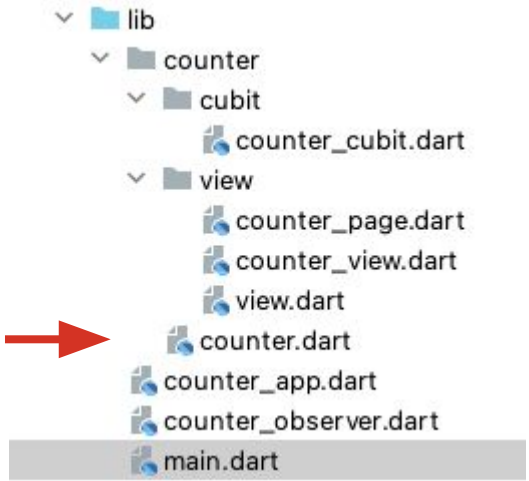
```dart
export 'counter_page.dart';
export 'counter_view.dart';
```

Ré-exporte les fichiers counter_page.dart et counter_view.dart

=> En important view.dart vous accèder aux classes CounterPage et CounterView sans avoir à les importer individuellement !

La gestion d’état

Structure du projet - Simplifier les imports - counter.dart



counter.dart

```dart
export 'cubit/counter_cubit.dart';
export 'view/view.dart';
```

Ré-exporte les fichiers counter_cubit.dart et view.dart

=> En important counter.dart vous accèder aux classes CounterCubit, CounterPage et CounterView sans avoir à les importer individuellement !

La gestion d’état

Riverpod

Riverpod

https://pub.dev/packages/riverpod

La gestion d’état

Introduction à Riverpod

Qu'est-ce que Riverpod ?

Un pattern architectural pour gérer l'état.

Pourquoi utiliser Riverpod ?

Séparation des préoccupations, testabilité, meilleure maintenabilité.

Les composants clés :

@riverpod, ConsumerWidget, WidgetRef, ref.watch, ref.read

La gestion d’état

Riverpod. Compteur, compteur et compteur !

```
class _MyHomePageState extends State<MyHomePage> {
 int _counter = 0;

 void _incrementCounter() {
   setState(() {
     _counter++;
   });
 }

 @override
 Widget build(BuildContext context) {
   return Scaffold(
     ...
     floatingActionButton: FloatingActionButton(
       onPressed: _incrementCounter,
       tooltip: 'Increment',
       child: const Icon(Icons.add),
     ),
   );
 }
}
```

La gestion d'état

Riverpod - ProviderScope

```
void main() {
 runApp(
   const ProviderScope(child: MyApp()),
 );
}
```

Rappel avec Provider

```
runApp(
   MultiProvider(
     providers: [
         ChangeNotifierProvider(create: (_) => CounterProvider()),
         ChangeNotifierProvider(create: (_) => CounterProvider2()),
         ChangeNotifierProvider(create: (_) => CounterProvider3()),
         ChangeNotifierProvider(create: (_) => CounterProvider4()),
         ChangeNotifierProvider(create: (_) => CounterProvider5()),
         ...

     ],
```

La gestion d’état

Riverpod - Counter

```
@riverpod
class Counter extends _$Counter {
 @override
 int build() => 0;

 void increment() => state++;
}
```

La gestion d’état

Riverpod - ConsumerWidget

```
class Home extends ConsumerWidget {
 @override
 Widget build(BuildContext context, WidgetRef ref) {
   return Scaffold(
     appBar: AppBar(title: const Text('Counter example')),
     body: Center(
       child: Text('${ref.watch(counterProvider)}'),
     ),
     floatingActionButton: FloatingActionButton(
       onPressed: () => ref.read(counterProvider.notifier).increment(),
       child: const Icon(Icons.add),
     ),
   );
 }
}
```

La gestion d’état

GetX

GetX

https://pub.dev/packages/get

La gestion d’état

Introduction à GetX

Qu'est-ce que GetX ?

Solution puissante offrant une gestion d’état, l'injection de dépendances, la gestion des routes et la navigation, la gestion de la Locale.

Pourquoi utiliser GetX ?

Séparation des préoccupations, testabilité, simplification de la gestion d’état

Les composants clés :

GetXController, Get.put, Get.find, Obx(()=>), Get.to, Get.snackbar…

La gestion d'état

GetX. Le compteur, encore une fois !

```
class _MyHomePageState extends State<MyHomePage> {
 int _counter = 0;

 void _incrementCounter() {
   setState(() {
     _counter++;
   });
 }

 @override
 Widget build(BuildContext context) {
   return Scaffold(
     ...
     floatingActionButton: FloatingActionButton(
       onPressed: _incrementCounter,
       tooltip: 'Increment',
       child: const Icon(Icons.add),
     ),
   );
 }
}
```

La gestion d’état

GetX - Model

```
class CounterModel {
  final count = 0.obs;
  increment() => count.value++;
}
```

La gestion d’état

GetX - Controller

```
class CounterController extends GetxController {
  final model = CounterModel();

  increment() => model.increment();
}
```

```
@override
void onClose() {
 super.onClose();
}

@override
void onInit() {
 super.onInit();
}
```

La gestion d’état

GetX - View

```
class MyView extends GetWidget<CounterController> {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: Center(
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: <Widget>[
            Text('Count: ${controller.model. count.value}'),
            RaisedButton(
              onPressed: controller.increment,
              child: Text('Increment'),
            ),
          ],
        ),
      ),
    );
  }
}
```

```
final CounterController counterController = Get.put<CounterController>();
Obs(()=>Text('Count: ${controller.model.count.value}'))
```