Riverpod for your next Flutter project

Guillaume Roux

Mobile Engineer at Shipfix
Co-organizer of Flutter Lyon

@TesteurManiak

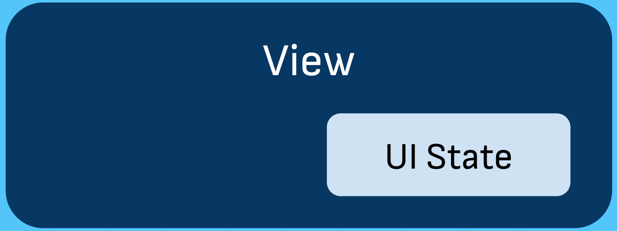
Disclaimer

- There will be personal opinions in this presentation
- Everything presented is merely observations coming from my own experience
- Riverpod is not a “silver bullet”
- This presentation might not age quite so well

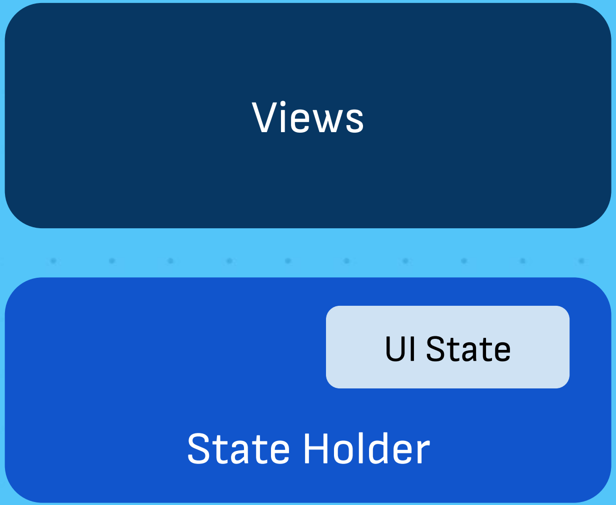
Wake Up call!
Do you know what “State management” is?

What is a State ?

The State is the object holding the info needed to build your UI.



Ephemeral state (e.g: StatefulWidget)



Application State (e.g: State management)

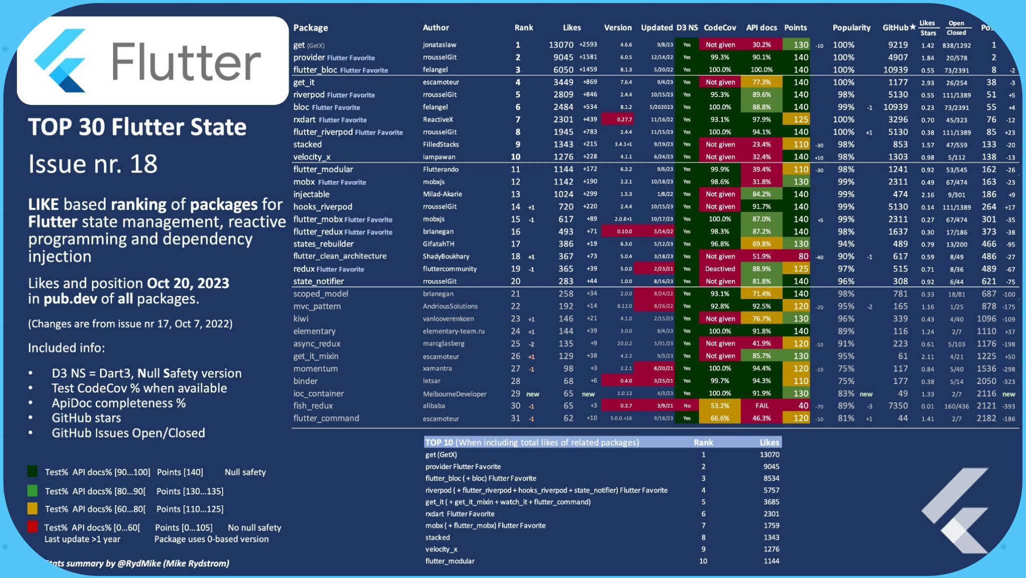
How to do State Management?

Lot of options available

- Provider
- Riverpod
- BLoC
- Stacked
- GetX
- etc.

Want to learn more?

OSLO | flutter vikings.com
ALESSIO SALVADORINI
Creative Technologist at Nokia
FLUTTER ARCHITECTURE COMPONENTS
FLUTTER VIKINGS



TOP 30 Flutter State

Issue nr. 18

LIKE based **ranking** of **packages** for **Flutter** state management, reactive programming and dependency injection

Likes and position **Oct 20, 2023** in **pub.dev** of **all** packages.

(Changes are from issue nr 17, Oct 7, 2022)

Included info:

- D3 NS = Dart3, **N**ull **S**afety version
- Test CodeCov % when available
- ApiDoc completeness %
- GitHub stars
- GitHub Issues Open/Closed

Package	Author	Rank		Likes		Version	Updated	D3 NS	CodeCov	API docs	Points		Popularity		GitHub★	Likes/Stars	Open/Closed	Po	
get (GetX)	jonataslaw	1		13070	+2593	4.6.6	9/8/23	Yes	Not given	30.2%	130	-10	100%		9219	1.42	838/1292	1	
provider Flutter Favorite	rrousselGit	2		9045	+1581	6.0.5	12/14/22	Yes	99.3%	90.1%	140		100%		4907	1.84	20/578	2	
flutter_bloc Flutter Favorite	felangel	3		6050	+1459	8.1.3	5/20/22	Yes	100.0%	100.0%	140		100%		10939	0.55	73/2391	8	-2
get_it	escamoteur	4		3449	+869	7.6.4	9/4/23	Yes	Not given	77.3%	140		100%		1177	2.93	26/254	38	-3
riverpod Flutter Favorite	rrousselGit	5		2809	+846	2.4.4	10/15/23	Yes	95.3%	89.6%	140		98%		5130	0.55	111/1389	51	+5
bloc Flutter Favorite	felangel	6		2484	+534	8.1.2	5/202023	Yes	100.0%	88.8%	140		99%	-1	10939	0.23	73/2391	55	+4
rxdart Flutter Favorite	ReactiveX	7		2301	+439	0.27.7	11/16/22	Yes	93.1%	97.9%	125		100%		3296	0.70	45/323	76	-12
flutter_riverpod Flutter Favorite	rrousselGit	8		1945	+783	2.4.4	11/15/23	Yes	100.0%	94.1%	140		100%	+1	5130	0.38	111/1389	85	+23
stacked	FilledStacks	9		1343	+215	3.4.1+1	9/19/23	Yes	Not given	23.4%	110	-30	98%		853	1.57	47/559	133	-20
velocity_x	iampawan	10		1276	+228	4.1.1	6/24/23	Yes	Not given	32.4%	140	+10	98%		1303	0.98	5/112	138	-13
flutter_modular	Flutterando	11		1144	+172	6.3.2	9/6/23	Yes	99.9%	39.4%	110	-30	98%		1241	0.92	53/545	162	-26
mobx Flutter Favorite	mobxjs	12		1142	+190	2.2.1	10/18/23	Yes	98.6%	31.8%	130		99%		2311	0.49	67/474	163	-23
injectable	Milad-Akarie	13		1024	+299	1.5.3	1/8/22	Yes	Not given	84.2%	140		99%		474	2.16	9/301	186	+9
hooks_riverpod	rrousselGit	14	+1	720	+220	2.4.4	10/15/23	Yes	Not given	91.7%	140		99%		5130	0.14	111/1389	264	+17
flutter_mobx Flutter Favorite	mobxjs	15	-1	617	+89	2.0.6+1	10/17/23	Yes	100.0%	87.0%	140	+5	99%		2311	0.27	67/474	301	-35
flutter_redux Flutter Favorite	brianegan	16		493	+71	0.10.0	5/14/22	Yes	98.3%	87.2%	140		98%		1637	0.30	17/186	373	-38
states_rebuilder	GIfatahTH	17		386	+19	6.3.0	5/12/23	Yes	96.8%	69.8%	130		94%		489	0.79	13/200	466	-95
flutter_clean_architecture	ShadyBoukhary	18	+1	367	+73	5.0.4	3/18/23	Yes	Not given	51.9%	80	-40	90%	-1	617	0.59	8/49	486	-27
redux Flutter Favorite	fluttercommunity	19	-1	365	+39	5.0.0	2/23/21	Yes	Deactived	88.9%	125		97%		515	0.71	8/36	489	-67
state_notifier	rrousselGit	20		283	+44	1.0.0	8/16/23	Yes	Not given	81.8%	140		96%		308	0.92	6/44	621	-75
scoped_model	brianegan	21		258	+34	2.0.0	8/24/22	Yes	93.1%	71.4%	140		98%		781	0.33	18/81	687	-100
mvc_pattern	AndriousSolutions	22		192	+14	8.12.0	8/26/22	Yes	92.8%	92.5%	120	-20	95%	-2	165	1.16	1/25	878	-175
kiwi	vanlooverenkoen	23	+1	146	+21	4.1.0	2/15/23	Yes	Not given	76.7%	130		96%		339	0.43	4/40	1096	-109
elementary	elementary-team.ru	24	+1	144	+39	3.0.0	8/4/23	Yes	100.0%	91.8%	140		89%		116	1.24	2/7	1110	+37
async_redux	marcglasberg	25	-2	135	+9	20.0.2	5/31/23	Yes	Not given	41.9%	120	-10	91%		223	0.61	5/103	1176	-198
get_it_mixin	escamoteur	26	+1	129	+38	4.2.2	9/3/23	Yes	Not given	85.7%	130		95%		61	2.11	4/21	1225	+50
momentum	xamantra	27	-1	98	+3	2.2.1	8/20/21	Yes	100.0%	94.4%	120	-10	75%		117	0.84	5/40	1536	-298
binder	letsar	28		68	+6	0.4.0	3/25/21	Yes	99.7%	94.3%	110		75%		177	0.38	5/14	2050	-323
ioc_container	MelbourneDeveloper	29	new	65	new	1.0.12	4/3/23	Yes	100.0%	91.9%	130		83%	new	49	1.33	2/7	2116	new
fish_redux	alibaba	30	-1	65	+3	0.3.7	3/9/21	No	53.2%	FAIL	40	-70	89%	-3	7350	0.01	160/436	2121	-393
flutter_command	escamoteur	31	-1	62	+10	5.0.0.+16	9/18/23	Yes	66.6%	46.3%	120	-10	81%	+1	44	1.41	2/7	2182	-186

TOP 10 (When including total likes of related packages)	Rank	Likes
get (GetX)	1	13070
provider Flutter Favorite	2	9045
flutter_bloc (+ bloc) Flutter Favorite	3	8534
riverpod (+ flutter_riverpod + hooks_riverpod + state_notifier) Flutter Favorite	4	5757
get_it (+ get_it_mixin + watch_it + flutter_command)	5	3685
rxdart Flutter Favorite	6	2301
mobx (+ flutter_mobx) Flutter Favorite	7	1759
stacked	8	1343
velocity_x	9	1276
flutter_modular	10	1144

🟩	Test% API docs% [90...100]	Points [140]	Null safety
🟩	Test% API docs% [80...90[	Points [130...135]	
🟨	Test% API docs% [60...80[	Points [110...125]	
🟥	Test% API docs% [0...60[Last update >1 year	Points [0...105] Package uses 0-based version	No null safety

ats summary by @RydMike (Mike Rydstrom)

And Riverpod in all of that?

“A Reactive Caching and Data-binding Framework”

source: https://riverpod.dev/

How is data handled with riverpod?

- Data are exposed through providers
- Providers are top-level references (e.g: `final myProvider`)
- `ref.watch(myProvider)` to observe a value
- `ref.read(myProvider)` to get a value
- `ref.watch(myProvider.select)` to observe and filter a value
- `ref.listen(myProvider, listener)` to trigger a callback when the value changes

All providers

- Provider
- (Async)NotifierProvider
- ~~StateNotifierProvider~~ (prefer NotifierProvider)
- FutureProvider
- StreamProvider
- StateProvider
- ~~ChangeNotifierProvider~~ (prefer NotifierProvider)

Usage for dependency injection

How to use Riverpod?

For dependency injection

```dart
typedef Json = Map<String, dynamic>;

final dioProvider = Provider<Dio>((ref) => Dio());

final itemsApiProvider = FutureProvider<List<Item>>(
  (ref) async {
    final dio = ref.watch(dioProvider);
    final result = await dio.get<List>('my-api');
    final parsed = result.data.map((e) {
      return Item.fromJson(e as Json);
    });
    return parsed.toList();
  },
);
```

How to use Riverpod?

For dependency injection

```dart
typedef Json = Map<String, dynamic>;

final dioProvider = Provider<Dio>((ref) => Dio());

final itemsApiProvider = FutureProvider<List<Item>>(
  (ref) async {
    final dio = ref.watch(dioProvider);
    final result = await dio.get<List>('my-api');
    final parsed = result.data.map((e) {
      return Item.fromJson(e as Json);
    });
    return parsed.toList();
  },
);
```

How to use Riverpod?

For dependency injection

```dart
typedef Json = Map<String, dynamic>;

final dioProvider = Provider<Dio>((ref) => Dio());

final itemsApiProvider = FutureProvider<List<Item>>(
  (ref) async {
    final dio = ref.watch(dioProvider);
    final result = await dio.get<List>('my-api');
    final parsed = result.data.map((e) {
      return Item.fromJson(e as Json);
    });
    return parsed.toList();
  },
);
```

Usage for state management

How to access the `ref` object from a widget?

- `Consumer` widget - Similar to the `Builder` widget from Flutter, takes a callback that exposes the `ref` object and return a widget.
- `ConsumerWidget` - Drop-in replacement to `StatelessWidget`, exposes the `ref` object in the `build` method.
- `ConsumerStatefulWidget` & `ConsumerState` - Drop-in replacement to `StatefulWidget` & `State`, exposes the `ref` as a getter in the `ConsumerState`.

How to use Riverpod?

For state management

```dart
final itemsApiProvider = /* … */;

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return ProviderScope(
      child: Home(),
    );
  }
}

class Home extends ConsumerWidget {
  @override
  Widget build(BuildContext context, WidgetRef ref) {
    return ref.watch(itemsApiProvider).when(
      data: (items) => _ItemList(items),
      loading: () => _Loading(),
      error: (error, stackTrace) => _Error(error),
    );
  }
}

class _ItemList extends StatelessWidget {}
class _Loading extends StatelessWidget {}
class _Error extends StatelessWidget {}
```

How to use Riverpod?

For state management

```dart
final itemsApiProvider = /* … */;

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return ProviderScope(
      child: Home(),
    );
  }
}

class Home extends ConsumerWidget {
  @override
  Widget build(BuildContext context, WidgetRef ref) {
    return ref.watch(itemsApiProvider).when(
      data: (items) => _ItemList(items),
      loading: () => _Loading(),
      error: (error, stackTrace) => _Error(error),
    );
  }
}

class _ItemList extends StatelessWidget {}
class _Loading extends StatelessWidget {}
class _Error extends StatelessWidget {}
```

How to use Riverpod?

For state management

```dart
final itemsApiProvider = /* … */;

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return ProviderScope(
      child: Home(),
    );
  }
}

class Home extends ConsumerWidget {
  @override
  Widget build(BuildContext context, WidgetRef ref) {
    return ref.watch(itemsApiProvider).when(
      data: (items) => _ItemList(items),
      loading: () => _Loading(),
      error: (error, stackTrace) => _Error(error),
    );
  }
}

class _ItemList extends StatelessWidget {}
class _Loading extends StatelessWidget {}
class _Error extends StatelessWidget {}
```

More complex state management

How to use Riverpod?

More complex state management

```dart
class HomeController extends Notifier<HomeState> {
  HomeState build() => const HomeInitialState();

  Future<void> fetchItems() async {
    state = const HomeLoadingState();
    try {
      final items = await ref.read(
        itemsApiProvider.future,
      );
      state = switch (items.isEmpty) {
        true => const HomeEmptyState(),
        false => HomeLoadedState(items),
      };
    } catch (e) {
      state = HomeErrorState(e);
    }
  }
}

final homeProvider =
  NotifierProvider<HomeController, HomeState>(
    () => HomeController(),
  );

/// Initial, Loading, Loaded, Empty, Error
sealed class HomeState {}
```

How to use Riverpod?

More complex state management

```dart
class HomeController extends Notifier<HomeState> {
  HomeState build() => const HomeInitialState();

  Future<void> fetchItems() async {
    state = const HomeLoadingState();
    try {
      final items = await ref.read(
        itemsApiProvider.future,
      );
      state = switch (items.isEmpty) {
        true => const HomeEmptyState(),
        false => HomeLoadedState(items),
      };
    } catch (e) {
      state = HomeErrorState(e);
    }
  }
}

final homeProvider =
  NotifierProvider<HomeController, HomeState>(
    () => HomeController(),
  );

/// Initial, Loading, Loaded, Empty, Error
sealed class HomeState {}
```

How to use Riverpod?

More complex state management

```dart
class HomeController extends Notifier<HomeState> {
  HomeState build() => const HomeInitialState();

  Future<void> fetchItems() async {
    state = const HomeLoadingState();
    try {
      final items = await ref.read(
        itemsApiProvider.future,
      );
      state = switch (items.isEmpty) {
        true => const HomeEmptyState(),
        false => HomeLoadedState(items),
      };
    } catch (e) {
      state = HomeErrorState(e);
    }
  }
}

final homeProvider =
  NotifierProvider<HomeController, HomeState>(
    () => HomeController(),
  );

/// Initial, Loading, Loaded, Empty, Error
sealed class HomeState {}
```

How to use Riverpod?

More complex state management

```dart
final homeProvider = /* … */;

class Home extends ConsumerWidget {
  @override
  Widget build(BuildContext context, WidgetRef ref) {
    final state = ref.watch(homeProvider);
    return Scaffold(
      floatingActionButton: FloatingActionButton(
        onPressed: () {
          ref.read(homeProvider.notifier).fetchItems();
        },
        // …
      ),
      body: switch (state) {
        HomeInitialState() => _Initial(),
        HomeLoadingState() => _Loading(),
        HomeEmptyState() => _Empty(),
        HomeLoadedState(:final items) => _Loaded(items),
        HomeErrorState(:final error) => _Error(error),
      },
    );
  }
}
```

How to use Riverpod?

More complex state management

```dart
final homeProvider = /* … */;

class Home extends ConsumerWidget {
  @override
  Widget build(BuildContext context, WidgetRef ref) {
    final state = ref.watch(homeProvider);
    return Scaffold(
      floatingActionButton: FloatingActionButton(
        onPressed: () {
          ref.read(homeProvider.notifier).fetchItems();
        },
        // …
      ),
      body: switch (state) {
        HomeInitialState() => _Initial(),
        HomeLoadingState() => _Loading(),
        HomeEmptyState() => _Empty(),
        HomeLoadedState(:final items) => _Loaded(items),
        HomeErrorState(:final error) => _Error(error),
      },
    );
  }
}
```

How to use Riverpod?

More complex state management

```dart
final homeProvider = /* … */;

class Home extends ConsumerWidget {
  @override
  Widget build(BuildContext context, WidgetRef ref) {
    final state = ref.watch(homeProvider);
    return Scaffold(
      floatingActionButton: FloatingActionButton(
        onPressed: () {
          ref.read(homeProvider.notifier).fetchItems();
        },
        // …
      ),
      body: switch (state) {
        HomeInitialState() => _Initial(),
        HomeLoadingState() => _Loading(),
        HomeEmptyState() => _Empty(),
        HomeLoadedState(:final items) => _Loaded(items),
        HomeErrorState(:final error) => _Error(error),
      },
    );
  }
}
```

But wait, there’s more!

Caching

Riverpod will keep values of asynchronous providers (e.g: FutureProvider, StreamProvider & AsyncNotifier) until disposed or invalidated.

```
final itemsApiProvider = FutureProvider(/* … */);

final itemStreamProvider = StreamProvider(/* … */);

final itemsNotifierProvider = AsyncNotifierProvider(
  // …
);
```

Testing

Easy to override = easy to mock

- UncontrolledProviderScope exposes a ProviderContainer to the widget tree
- ProviderContainer stores the state of the providers and allows overriding

```dart
class TestableWidget extends StatelessWidget {
  const TestableWidget(
    required this.container,
    required this.child,
  );

  final ProviderContainer container;
  final Widget child;

  @override
  Widget build(BuildContext context) {
    return UncontrolledProviderScope(
      container: container,
      child: child,
    );
  }
}
```

Testing

Easy to override = easy to mock

Create a `ProviderContainer` to mock your providers.

```
ProviderContainer makeContainer(List<Override> overrides) {
  final container = ProviderContainer(
    overrides: overrides,
  );
  addTearDown(container.dispose);
  return container;
}

testWidgets('Display error state', (tester) async {
  final controllerMock = MyMock();
  when(() => controllerMock.state)
    .thenReturn(HomeErrorState('Test error'));

  await tester.pumpWidget(
    TestableWidget(
      container: makeContainer([
        homeProvider.overrideWithValue(controllerMock),
      ]),
      child: Home(),
    ),
  );
});

class MyMock extends Mock implements HomeController {}
```

Tooling

- riverpod_lint – *"Riverpod_lint is a developer tool for users of Riverpod, designed to help stop common issues and simplify repetitive tasks."*
- riverpod_generator – *"A code generator for Riverpod. This both simplifies the syntax empowers it, such as allowing stateful hot-reload."*
- riverpod_graph – *"A command line tool that analyzes a Riverpod project and generates a graph of the interactions between providers/widgets."*
- And much more…

New codegen syntax

Without codegen

```
class HomeController extends Notifier<HomeState> {
  HomeState build() => const HomeInitialState();

  Future<void> fetchItems() async {
    // …
  }
}

final homeProvider =
  NotifierProvider<HomeController, HomeState>(
    (ref) => HomeController(),
  );
```

With codegen

```
@riverpod
class HomeController extends _$HomeController {
  HomeState build() => const HomeInitialState();

  Future<void> fetchItems() async {
    // …
  }
}
```

Advantages of codegen

- No need to specify the provider anymore
- Can pass multiple parameters to your provider (Can only use 1 with the `.family` constructor)
- Allows support for hot-reload
- Better debugging through the generation of extra metadata
- Will become the only way to use riverpod once Static Metaprogramming is available in Dart

Wrap up

Why use Riverpod?

- Combine state management and dependency injection
- Less boilerplate
- Type-safety
- Reactivity
- Caching
- Tested and testable
- Toolings
- Starts to be used more and more

Find more at

https://riverpod.dev/

Thank You!

Questions?

Guillaume Roux

Mobile Engineer at Shipfix
Co-organizer of Flutter Lyon

@TesteurManiak