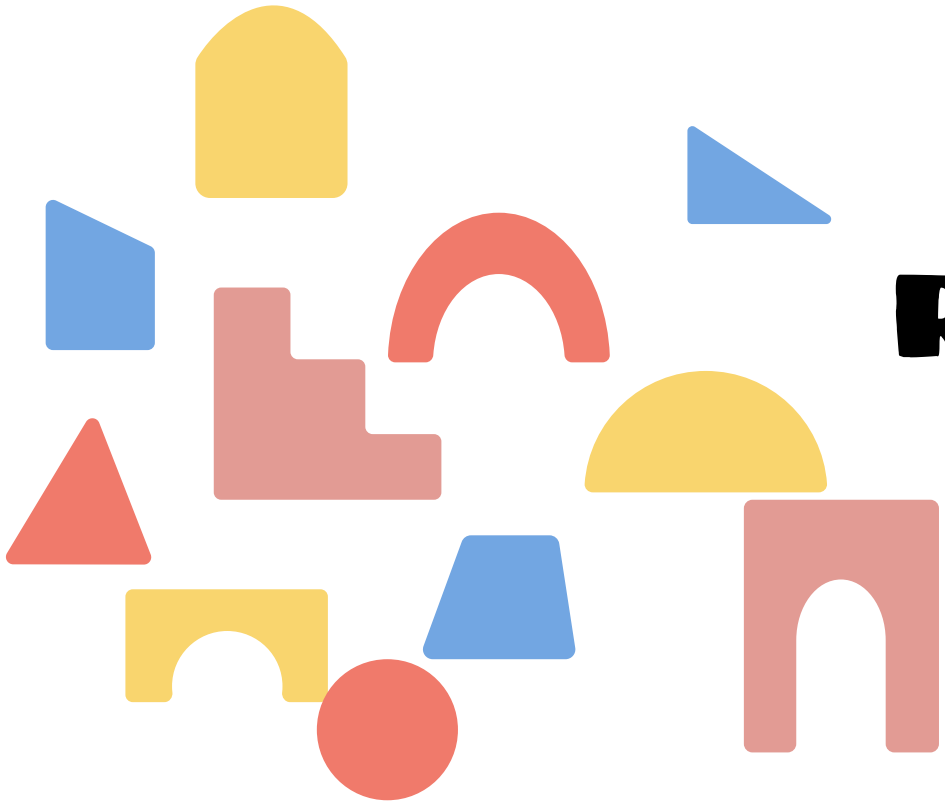

Riverpod 3.0

Spoiler alert! 🎯🔥

Ciao!

Chi sono?

- 🎯 Dart enjoyer
- 🏞️ Riverpod supporter
- ⚗️ Functional programming!

Che si fa oggi?

- 📚 Refresher: cos'è Riverpod?
- 🔥 Riverpod 3.0?
- 🧪 Q/A

Attenzione: NO CODE PERMITTED! 🚫

Business logic, with Riverpod

Providers: *functions with benefits.*

Da ...

```
Future<int> check() {
  return Future.delayed(200.milliseconds, () => ...);
}
```

... a:

```
@riverpod
Future<int> check(CheckRef ref) {
  return Future.delayed(200.milliseconds, () => ...);
}
```

Business logic, with Riverpod

```
@riverpod
class CheckController extends _$CheckController {
  @override
  Future<int> build() {
    return Future.delayed(200.milliseconds, () => 0);
  }

  Future<void> moreSpritz() {
    return update((state) => Future.delayed(200.milliseconds, () => state + 1));
  }
}
```

```
@riverpod
Future<int> fraudCheck(FraudCheckRef ref) async {
  final check = await ref.watch(checkProvider.future);
  return 2 * check;
}
```

Final result

```
class CheckPleaseWidget extends ConsumerWidget {
  Widget build(BuildContext context, WidgetRef ref) {
    final AsyncValue<int> fraud = ref.watch(fraudCheckProvider);
    return Column(
      mainAxisAlignment: MainAxisAlignment.center,
      mainAxisSize: MainAxisSize.min,
      children: [
        switch (fraud) {
          AsyncData(value: 0) => const Text('Cosa ti porto? 👀'),
          AsyncData(:final value) => Text('Sono € $value, grazie 😸'),
          AsyncError() => const Text('Mi spiace, niente spriz oggi 😢'),
          _ => const Image.network('draft-wine-gif-url')
        },
        ElevatedButton(
          onPressed: ref.read(checkControllerProvider.notifier).moreSpritz,
          child: const Text('🍹'),
        ),
      ],
    );
  }
}
```

Spoiler #0: `sealed class AsyncValue<T>`

💈 `AsyncValue` ora è `sealed`

⚠️ `AsyncData` / `AsyncLoading` / `AsyncError` ora sono `final`

```
return Column(
  mainAxisAlignment: MainAxisAlignment.center,
  mainAxisSize: MainAxisSize.min,
  children: [
    switch (fraud) {
      AsyncData(value: 0) => const Text('Cosa ti porto? 👀'),
      AsyncData(:final value) => Text('Sono € $value, grazie 😻'),
      AsyncError() => const Text('Mi spiace, niente spriz oggi 😢'),
      AsyncLoading() => const Image.network('draft-wine-gif-url')
    },
    ElevatedButton(
      onPressed: ref.read(checkControllerProvider.notifier).moreSpritz,
      child: const Text('🍹'),
    ),
  ],
);
```

Spoiler #1: API pulite e semplificate

🗑️ cleanup delle *public api* (~90 tipi rimossi)

⚡ performance improvements

🐛 fixati 50 bug

- "Addio" StateNotifier/StateProvider/...
  - ➡️ Verrà spostato in un package dedicato: `riverpod/legacy.dart`
- Addio `AutoDispose` : ora *tutti* i provider sono `AutoDispose` , non servirà specificarlo più!
  - 👋 `AutoDisposeNotifier` , `AutoDisposeProvider` , etc.
  - Vi serve un `AlwaysAlive` provider / notifier? usate `ref.keepAlive`
- Addio a tante (strane?) sottoclassi di `Ref`
  - 👋 `ProviderRef` , `StreamProviderRef` ; ma anche `AutoDisposeFutureProviderRef` , etc.
  - ... Aspetta, cosa? Perché? 🤔

Spoiler #2: Generics

Sarà possibile definire provider basati su un tipo `T` generico.

```
// warning: pseudo code!!
@riverpod
class MyListNotifier<T> extends _$MyListNotifier<T> {
  @override
  List<T> build() => [];

  void add(T value) {
    state = [...state, value];
  }
}
```

➡️ in arrivo istruzioni per l'uso...!

Spoiler #3: `if (mounted) ...`

🚨 Breaking change!

Riverpod `2.0`: tra un `build` e l'altro `Notifier` non veniva `dispose`.

Riverpod `3.0`: `Notifier` segue lo stesso ciclo di vita di `build`.

```
@riverpod
class SomeNotifier extends _$SomeNotifier {
  @override
  int build() {
    Future.delayed<void>(3.seconds, ref.invalidateSelf);
    // Riverpod 2: tra tre secondi `build` viene re-eseguito
    // Riverpod 3: l'intera classe `SomeNotifier` viene `dispose` e re-allocata, `build` incluso
    return 0;
  }
}
```

Spoiler #3: `if (mounted) ...`

Perché questo cambiamento? 🤔

```dart
@riverpod
class SomeNotifier extends _$SomeNotifier {
  @override
  int build() => ...;

  Future<void> asyncMethod() async {
    await something();

    if (!mounted) return;
    ref.something();
  }

  var _internalState = 0; // look out!
}
```

Spoiler #4: Testing utilities

👀 Tutti noi testiamo il nostro codice, vero?

🧪 E infatti, abbiamo bisogno di test utilities!

```
// Riverpod 2.0
ProviderContainer testContainer({List<ProviderOverride>? overrides}) {
  final container = ProviderContainer(overrides: []);
  addTearDown(container.dispose);

  return container;
}

test('my test', () {
  final container = testContainer();
  // TODO use container to test your code here 😉
})
```

Spoiler #4: Testing utilities

Riverpod 3.0: è in arrivo... `ProviderContainer.test` 😜

```
// Riverpod 3.0
test('my test', () {
  // warn: pseudocode
  final container = ProviderContainer.test();
  // TODO use container to test your code here 😉
})
```

Spoiler #5: Lazy `ref.listen`

```
@riverpod
int printMe(PrintMeRef ref) {
  print('me!');
  return 0;
}
```

```
@riverpod
int another(AnotherRef ref) {
  // warn: pseudocode!!
  ref.listen(printMeProvider, weak: true, (prev, next) {});

  return 42;
}
```

😴 La stringa `me!` non viene stampata.

Spoiler #6: Side effects, ma meglio!

📖 Sicuramente avete letto la documentazione... vero?

```
class MyAsyncNotifier extends _$MyAsyncNotifier {
  @override
  Future<int> build() => ...;

  // Riverpod 2.0: è responsabilità tua gestire la UX per questo effetto collaterale!
  Future<void> sideEffect() async {
    await something();
  }
}
```

Spoiler #6: Side effects, ma meglio!

➡️ In arrivo... query mutations!

```
class MyAsyncNotifier extends _$MyAsyncNotifier {
  @override
  Future<int> build() => ...;

  @mutation  // warn: this is pseudocode.
  Future<void> sideEffect() async {
    await something();
  }
}
```

Spoiler #6: Side effects, ma meglio!

```
// warn: *tons* of pseudocode
... child: switch (ref.watch(sideEffect)) {
  Empty() => ElevatedButton(
    onPressed: () => sideEffect(),
    child: Text('Send 📧'),
  ),
  Loading() => ElevatedButton(
    onPressed: null,
    child: CircularProgressIndicator(),
  ),
  Errored() => ElevatedButton(
    onPressed: query.retry,
    style: ButtonStyle(
      backgroundColor: WidgetStateProperty.all(Colors.redAccent),
      foregroundColor: WidgetStateProperty.all(Colors.white),
    ),
    child: Text('Oof. Try again?'),
  ),
  Success() => ElevatedButton(
    onPressed: null,
    child: Icon(Icons.check),
  ),
};
```

Spoiler #7: Retry w/ exponential backoff

```dart
@riverpod
Future<int> unreliable(UnreliableRef ref) async {
  final someResult = await someRepo.fromNetwork(throws: true);  // fails 100% of the time
  return someResult;
}
```

```dart
Duration? myRetry(int retryCount, Object error) {
  // your logic here, e.g.
  if (retryCount > 10) return null;

  final retryDelay = 1 + retryCount * retryCount;
  return retryDelay.seconds;
}
```

Spoiler #7: Retry w/ exponential backoff

```
// warn: pure imaginative pseudocode
@Riverpod(retry: myRetry)
Future<int> unreliable(UnreliableRef ref) async {
  final someResult = await someRepo.fromNetwork(throws: true);  // fails 100% of the time
  return someResult;  // tries again after 1 sec, 2 secs, 5 secs, 10 secs...
}
```

⚠️ (probably) more assembly required

Spoiler #8: Offline caching (!!)

Reminder: what is riverpod?

> A Reactive Caching and Data-binding Framework

What if...

```
// warn: 110% imaginative example
ProviderScope(
  // TODO: define your offline connector / adapter here (based on your database preference)
  offlineConnector: const SharedAppPreferenceAsJson(),  // example
)
```

Spoiler #8: Offline caching (!!)

Then...

```
// warn: this is my own pure imagination, not even pseudocode at this point
@Riverpod(offline: 'tableName', retry: myRetry)
Future<List<Product>> products(ProductsRef ref) {
  final response = http.get('my-api/products');

  return (response as List).map(Product.fromJson).toList();
}
```

⚠️ (certainly) *way* more assembly required!

Wrap up

☣️ Alcune slide sono *volutamente* "imprecise"

📖 Serve tantissima documentazione...

🩻 C'è ancora molto altro...!!

🎯 Rilascio 3.0 *con* documentazione

🌠 Preview incoming soon-ish...

🆕 Stable release later in 2024/25...?

💭 Q/A... oppure chiacchieriamo

Grazie per l'attenzione