

Flutter Architektur

Von einzelnen Files zu einer skalierbaren Architektur



Wer sind wir?

Die Trainer



Markus Kühle

Markus ist Gründer von coodoo, hat langjährige Software- und Architektur-Expertise im Bereich Java EE. Seit 2019 arbeitet er mit Flutter.

[@makueh](#)



Marcel Ploch

Marcel ist Senior Full-Stack-Entwickler mit langjähriger Erfahrung im Front- und Backend mit Javascript. Seit 2019 arbeitet er mit Flutter.

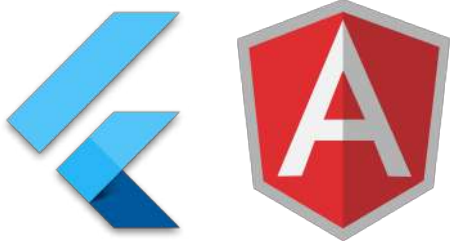


Das coodoo GmbH Team



Unsere Arbeit bei der coodoo GmbH

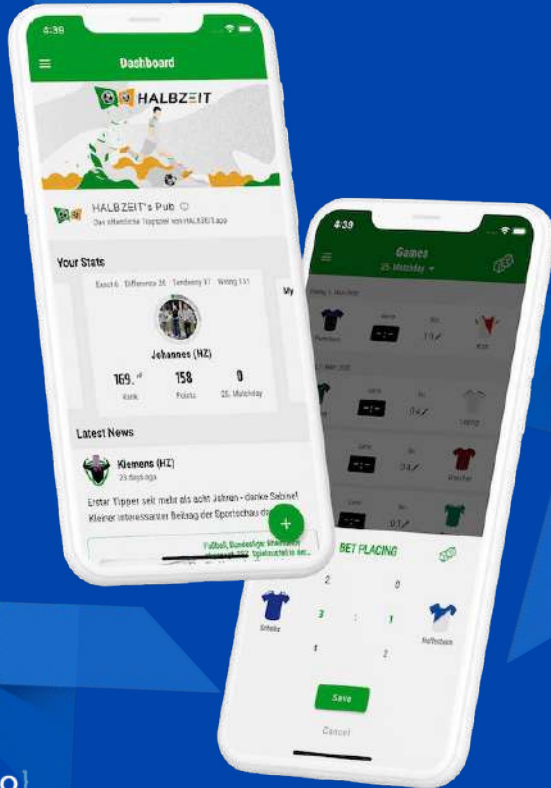
App Entwicklung



Produkte



Unsere Expertise



HALBZEIT Fußball Tippspiel

- Whitelabel Fußball Tippspiel für Firmen
- Über 50.000 Downloads
- iOS, Android und Tablet
- AWS Cloud Service Integration

<https://halbzeit.app>

Unsere Expertise



SCHOTT MIPS

Interne Handscanner-App für
Werksarbeiter zum Untersuchen von
Arbeitszuständen und Analysieren von
technischen Fehlern.

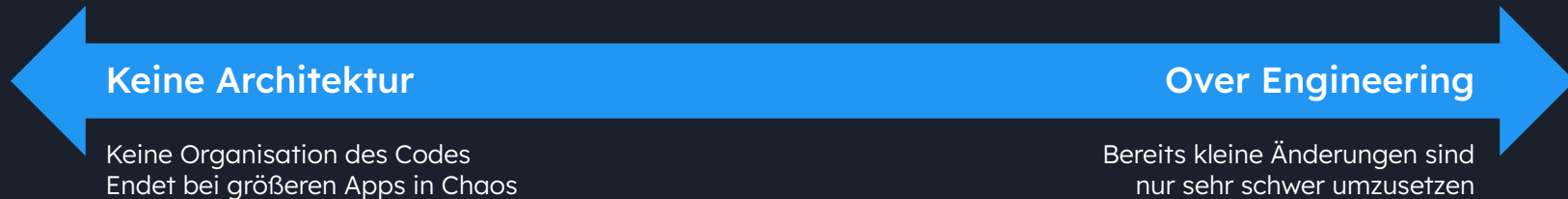
Wird von den Mitarbeitern in der
Industrie verwendet.

Warum braucht man Architektur?

Relevanz einer Architektur

Eine gute **Architektur** zu wählen ist **essentiell**.

Strukturierter Code lässt sich besser **warten**, wenn die App **wächst**.



Wann über Architektur nachdenken?

- **Größe der App**
Wie viele Pages, Navigationstiefe, Komplexität?
- **Größe des Teams**
Wie viele Teammitglieder arbeiten gleichzeitig an der App?
- **Verteilung des Teams**
Wie ist der direkte Austausch des Teams möglich?
- **Verständnis der Sprache und Struktur**
Sprechen alle Mitglieder die gleiche “Flutter-Sprache”?

Separation of Concerns

UI

```
floatingActionButton: Row(  
  mainAxisAlignment: MainAxisAlignment.end,  
  children: [  
    FloatingActionButton(  
      onPressed: () =>  
        ref.read(counterControllerProvider.notifier).decrement(),  
      tooltip: 'Decrement',  
      child: const Icon(Icons.remove),  
    ), // FloatingActionButton  
    const SizedBox(width: 16),  
    FloatingActionButton(  
      onPressed: () =>  
        ref.read(counterControllerProvider.notifier).increment(),  
      tooltip: 'Increment',  
      child: const Icon(Icons.add),  
    ), // FloatingActionButton  
  ],
```

Business Logic

```
class CounterController extends StateNotifier<CounterState> {  
  CounterController({required this.counterRepo})  
    : super(const CounterState()) {  
    init();  
  }  
  
  final CounterRepo counterRepo;  
  
  void init() {  
    state = state.copyWith(value: AsyncValue.data(counterRepo.value))  
  }  
  
  Future<void> increment() async {  
    state = state.copyWith(value: const AsyncValue.loading());  
    state = state.copyWith(  
      value: await AsyncValue.guard(() => counterRepo.increment())  
    );  
  }  
}
```

Flutter Vorgaben & Konventionen

Flutter gibt dem Entwickler **große Freiheiten** bei Architekturentscheidungen.

Flutter hat **keine Vorgabe** zu Ordnerstruktur oder anderen Konventionen bezüglich einer Architektur.



1. Projektaufbau & Grundstruktur

Konventionen

Ordnerstruktur, Dateinamen

Ordner Konventionen

Layers first

- lib
 - src
 - **presentation**
 - feature1
 - feature2
 - **domain**
 - feature1
 - feature2
 - **data**
 - feature1
 - feature2

Feature first

- lib
 - src
 - features
 - **feature1**
 - presentation
 - domain
 - data
 - **feature2**
 - presentation
 - domain
 - data

Es geht bei einem Feature nicht um die UI.
Domain-Driven Design als Startpunkt.

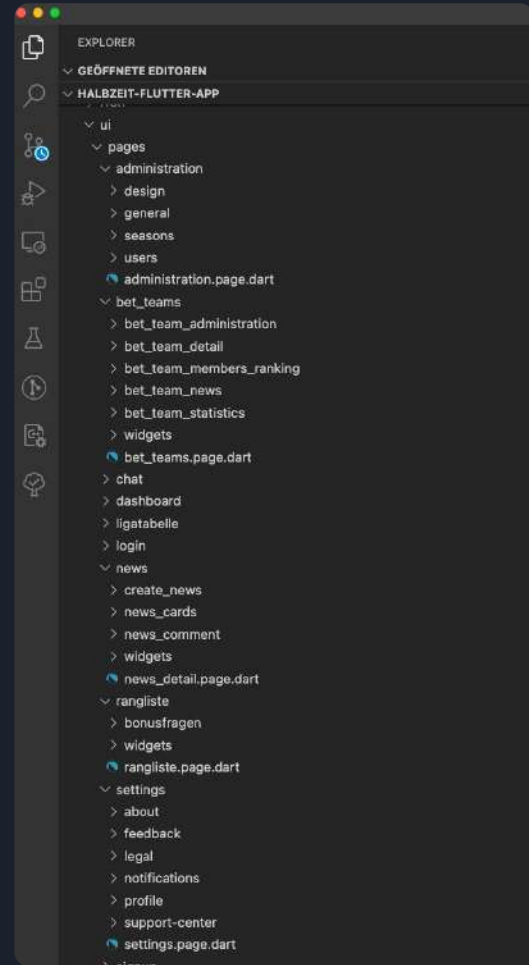
Layers first

Vorteile

- Schneller und einfacher Start
- Geringe Komplexität

Nachteile

- Sehr unübersichtlich, sobald die App wächst
- Zusammengehörende Dateien für ein Feature über das Projekt verteilt



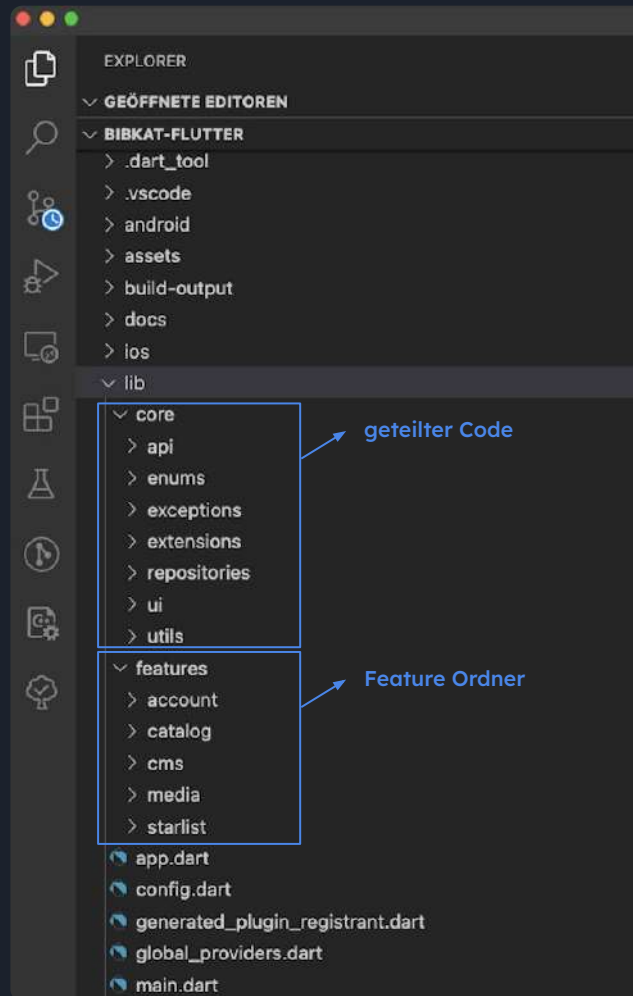
Feature first

Vorteile

- Files, die zu einem Feature gehören, sind an einem Ort zu finden
- Layers in Feature
- Einfache Kommunikation

Nachteile

- Verständnis, was ein Feature ist, muss im Team trainiert werden



Feature first - Startpunkt

Es geht bei einem Feature **nicht** um die UI.

Domain-Driven Design als **Startpunkt**.

Domain-Driven Design ist ein Ansatz für die Softwareentwicklung, der die Entwicklung auf die Programmierung eines Domänenmodells konzentriert, das über ein umfassendes Verständnis der Prozesse und Regeln einer Domäne verfügt.

Martin Fowler über das Buch von Eric Evans von 2003

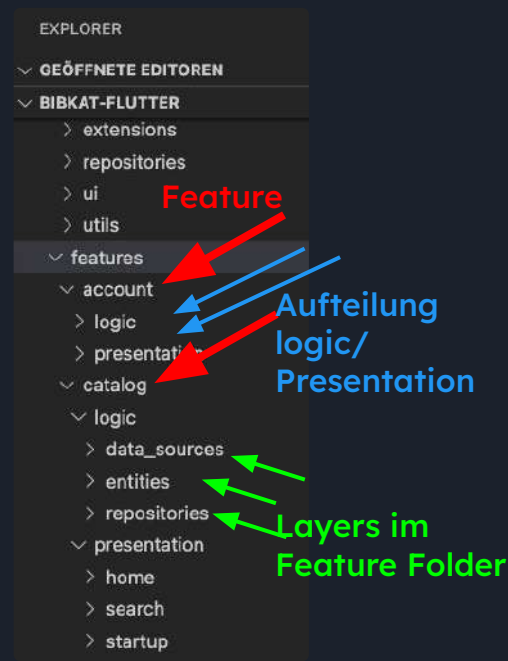
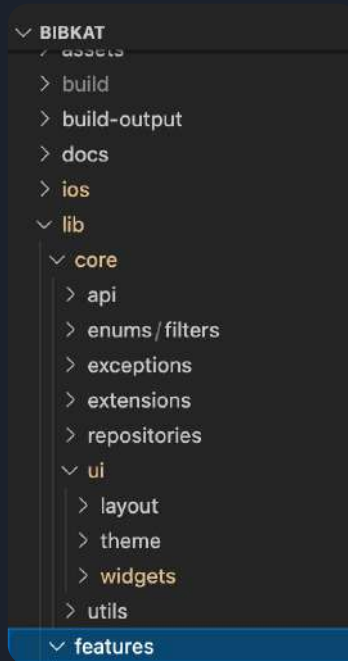
<https://martinfowler.com/bliki/DomainDrivenDesign.html>

Feature first

Layers im Feature

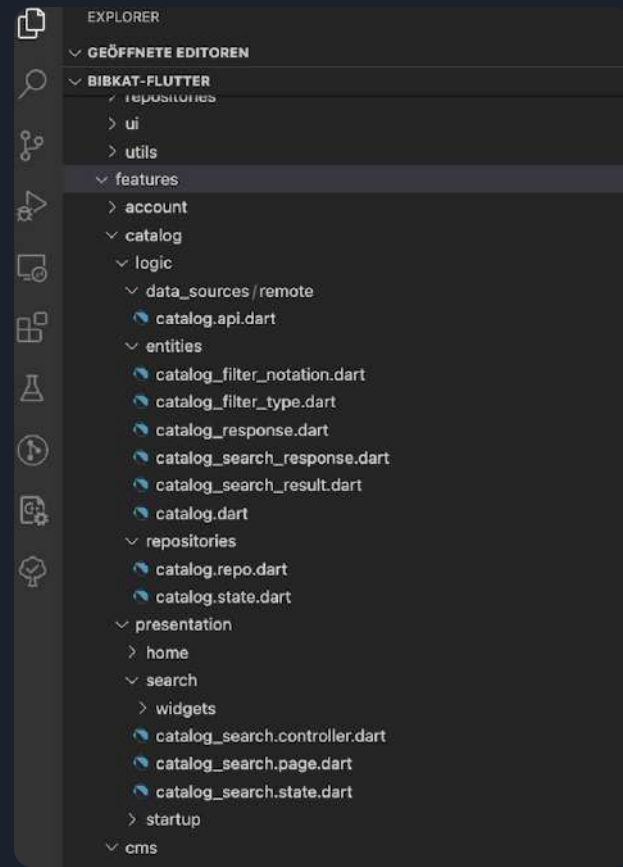
Die **Layer** werden innerhalb des Features als **Ordner** eingerichtet.

Wir unterteilen noch einmal in **logic** und **presentation**.



Konventionen für Dart Files & Klassen

- Filename sagt über die **Art des Inhalts**
aus Bsp.: `catalog_search.page.dart`,
`catalog.repo.dart`,
- **Eine** Klasse pro File
Ausnahme sind eng zusammengehörige
Klassen wie z.B. Stateful Widgets



Analyzer & Lint Regeln

Dart Analyzer & Linter

Der **Analyzer** checkt statisch auf Syntax und Type Error, noch bevor der Code ausgeführt wird.

<https://dart.dev/guides/language/analysis-options>

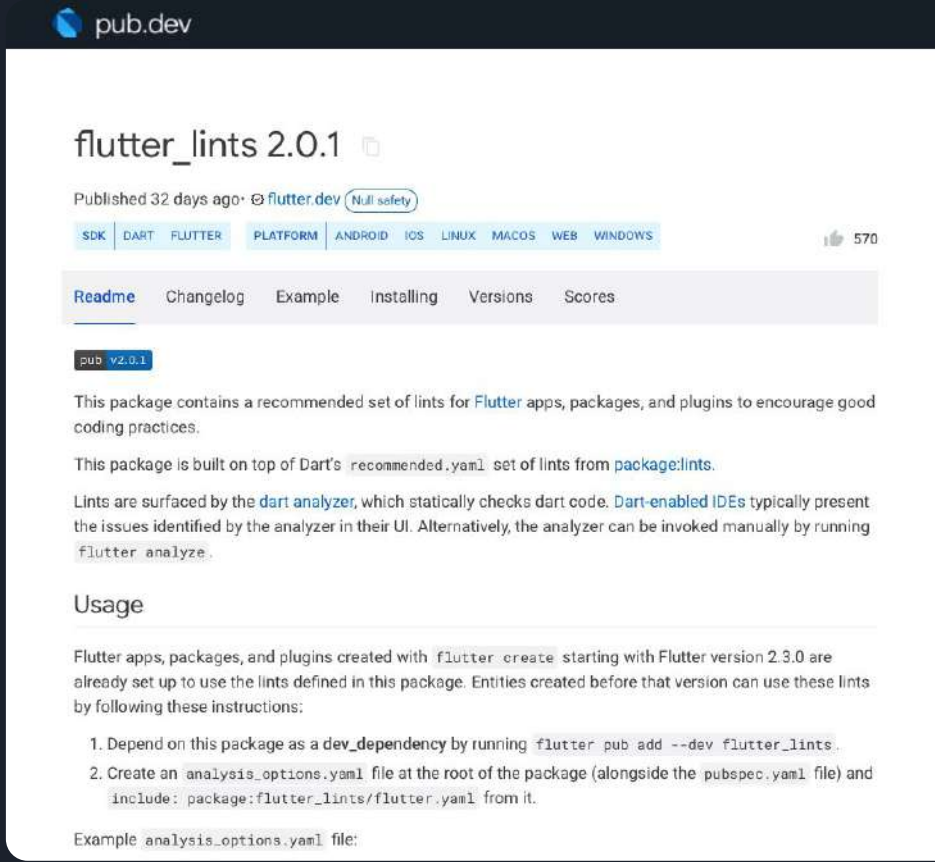
Der **Linter** checkt Style Guide Verletzungen.

<https://dart.dev/tools/linter-rules>

Flutter Lint Regeln

- Linter im Default Projekt bereits konfiguriert
- Flutter Lints basiert auf den empfohlenen Dart Lints
- In `analysis_options.yaml` anpassbar
- Aufrufen mit `flutter analyze`

**Alle konfigurierten Lint
Regeln immer beachten!**



The screenshot shows the pub.dev page for the `flutter_lints` package, version 2.0.1. The page includes a header with the package name and version, a 'Null safety' badge, and a navigation bar with tabs for SDK, Dart, Flutter, Platform, Android, iOS, Linux, MacOS, Web, and Windows. Below the navigation bar are tabs for Readme, Changelog, Example, Installing, Versions, and Scores. The main content area contains a description of the package, its build system (Dart's recommended.yaml), and its usage instructions. The usage instructions include a list of steps to set up the package and an example of the `analysis_options.yaml` file.

flutter_lints 2.0.1

Published 32 days ago · flutter.dev (Null safety)

SDK | DART | FLUTTER | PLATFORM | ANDROID | IOS | LINUX | MACOS | WEB | WINDOWS

570

Readme | Changelog | Example | Installing | Versions | Scores

pub v2.0.1

This package contains a recommended set of lints for Flutter apps, packages, and plugins to encourage good coding practices.

This package is built on top of Dart's recommended.yaml set of lints from package:lints.

Lints are surfaced by the dart analyzer, which statically checks dart code. Dart-enabled IDEs typically present the issues identified by the analyzer in their UI. Alternatively, the analyzer can be invoked manually by running `flutter analyze`.

Usage

Flutter apps, packages, and plugins created with `flutter create` starting with Flutter version 2.3.0 are already set up to use the lints defined in this package. Entities created before that version can use these lints by following these instructions:

1. Depend on this package as a dev_dependency by running `flutter pub add --dev flutter_lints`.
2. Create an `analysis_options.yaml` file at the root of the package (alongside the `pubspec.yaml` file) and include: `package:flutter_lints/flutter.yaml` from it.

Example `analysis_options.yaml` file:

https://pub.dev/packages/flutter_lints

Lint Beispiel - print

You, 38 s

class
int

```
void print(Object? object)
```

Type: `void Function(Object?)` *dart:core*

Prints a string representation of the object to the console.

void
se

Avoid `print` calls in production code. [dart\(avoid_print\)](#)

[View Problem \(⌘.\)](#) [Quick Fix... \(⌘1\)](#)



```
print(_counter);
```

You, 1 second ago • Uncommitted changes

```
});
```

```
}
```

[Lint Regel](#)
[Lint Regel in schön](#)

Error Rules

`always_use_package_imports`

Avoid relative imports for files in **lib/**.

`avoid_dynamic_calls`

Avoid method calls or property accesses on a "dynamic" target.

`avoid_empty_else`

style core

style recommended

style flutter

Avoid empty else statements.

`avoid_print`

style flutter

Avoid **print** calls in production code.

`avoid_relative_lib_imports`

style core

style recommended

style flutter

style pedantic

Avoid relative imports for files in **lib/**.

Die offiziellen "Core" Dart Team Lint Regeln

Für Flutter Projekte empfehlenswert (`flutter create` aktiviert diese automatisch)

Empfohlene Lint Regeln vom Dart Team

analysis_options.yaml

```
include: package:flutter_lints/flutter.yaml

analyzer:
  language:
    strict-casts: true
    strict-raw-types: true
  errors:
    # Bei Fehlenden required params Warnung ausgeben (nicht nur Hinweis)
    missing_required_param: warning
    # Fehlende returns als Warnung ausgeben (nicht als Hinweis)
    missing_return: warning
    # erlaubt TODO Kommentare im code
    todo: ignore

linter:
  rules:
    # Nur einzelne Hochkomma sind erlaubt
    prefer_single_quotes: true
```

[Lint Regeln](#)
[Lint Regel in schön](#)

Makefile

Was ist und bringt eine Makefile?

Eine oder mehrere Befehle werden vereint zu einem einfachen Befehl

- Lange Befehle muss man nicht merken
 - Viele Befehle mit einem ausführen
 - Build Pipeline kann diese Befehle einfach ausführen
-
- Pipeline stellt falsche Formatierungen und Warnungen fest

[Wikipedia](#)

[Ausführliche Einführung](#)

Makefile



Tabs und keine
Leerzeichen verwenden

```
FLUTTER := $(shell which flutter)
```

```
format:
```

```
$(FLUTTER) format . --line-length 120 --set-exit-if-changed
```

```
format-fix:
```

```
$(FLUTTER) format . --line-length 120
```

```
clean:
```

```
$(FLUTTER) clean
```

```
$(FLUTTER) pub get
```

```
...
```

Initial Setup

Aufgabe:

- Neues Flutter Projekt erstellen
- Ordnerstruktur einführen
- Die Regel `prefer_single_quotes` hinzufügen
- Makefile und erste Regeln erstellen:
 - flutter run, analyze & format



2. Architektur Übersicht



Die Wahl der Architektur

MVC, MVC+S, Clean Architecture, State Architecture

Übersicht der Architekturen

- Es **existieren** eine **Menge** Architekturen
- MVC, MVC+S, MVP, MVVM, Clean Architecture, Android App Architecture...
- Flutter gibt **keine** Architektur vor

Populäre State Architekturen

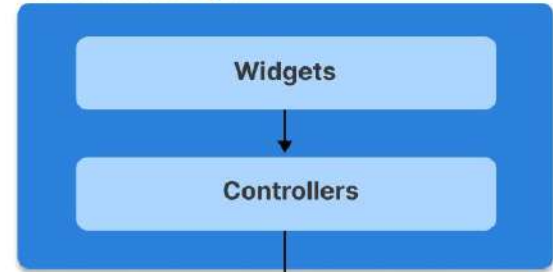
- **Bloc Architektur** (basierend auf der Bloc Library - bloclibrary.dev)
- **Riverpod Architektur** (MVVM State - riverpod Package - pub.dev/packages/riverpod)

Die Architekturen **basieren** auf **Provider**, welches das von Flutter **empfohlene** State Management Package ist.

State Architektur - Übersicht

- Basierend auf State Management
- Drei Layer:
 - Presentation
 - Domain
 - Data
- Jeder Layer hat eine **bestimmte Aufgabe**
- Klarer Vertrag wie die **Kommunikation** stattfindet.

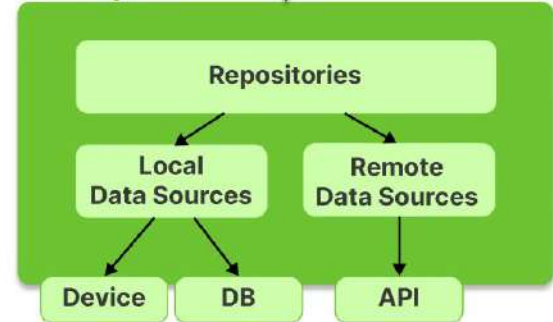
Presentation Layer



Domain Layer



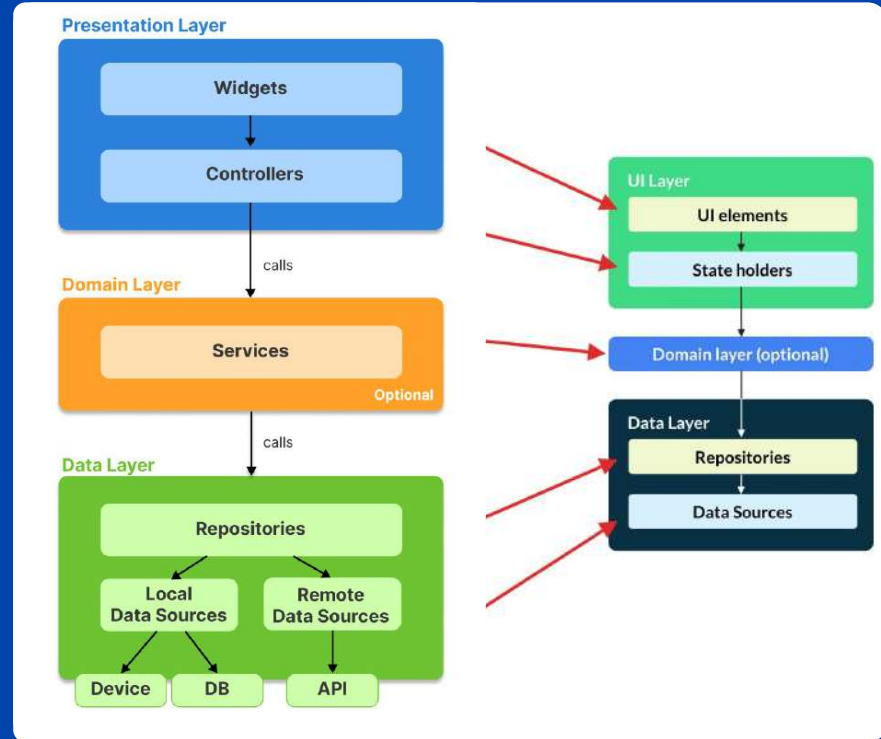
Data Layer



Android Architecture

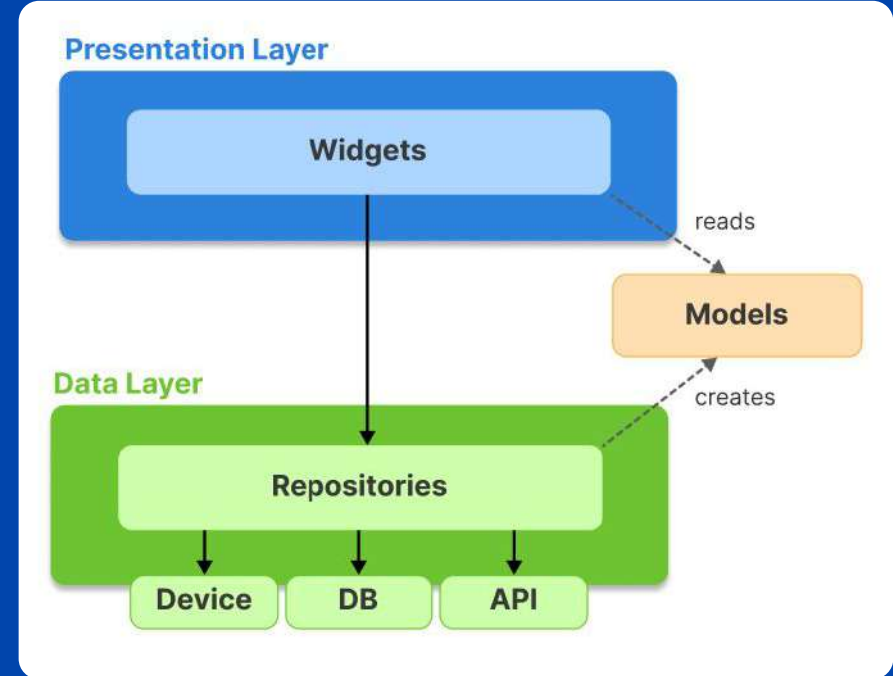
- State Architektur
- 3 Layer
- Kommt unserer Zielarchitektur am nächsten

[Android Guide to app architecture](#)



Minimale State Architektur

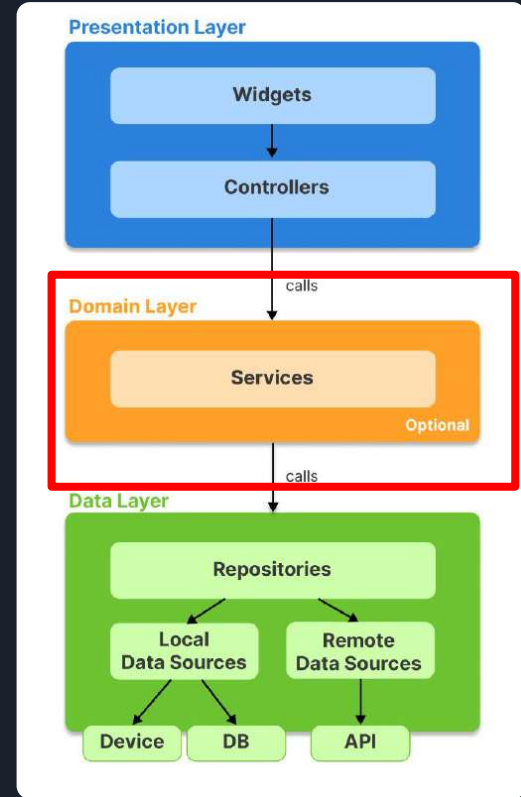
- Kleinste mögliche Architektur
- Zwei Layer:
 - Data
 - Presentation
- Kernbestandteile:
 - State Management
 - Dependency Injection
- Direkt mit **Entities** anstatt DTOs arbeiten



Domain Layer - Models

Domain Layer

Die Domain Layer ist verantwortlich für die Kapselung von Businesslogik, die von mehreren ViewModels wiederverwendet wird.



Model (Entities)

The entities must be our data types or classes that are used in different parts of our software.

Uncle Bob

- Entities beschreiben das **Domain Model**
- Werden in der Domain Layer **erstellt** und **verwaltet**
- Domain Layer wendet über **Business Logik** (Service) CRUD Operationen auf **Entities** an.
- In der **minimale State Architektur** können sie **direkt** im Repository **erstellt** und nach oben gereicht werden.

Immutable Objekte & Values

A class is immutable if all of the instance fields of the class, whether defined directly or inherited, are final.

API Documentation

- Immutable Objekte können ihren **Status** nach der Erstellung **nicht** mehr **ändern**.
- Vorteile in der **Performance**
- Nur eine **Stelle/Layer** kann/darf Entities verändern und den **State** der Anwendung **ändern**.
- In Dart wird mit **@immutable** auf Klassenebene angezeigt, dass diese **Klasse nicht veränderbar** ist.

(<https://api.flutter.dev/flutter/meta/immutable-constant.html>)

Immutable Object

Wir arbeiten mit unveränderbaren Objekten.

Objekt enthält nur final Attribute.

UnmodifiableListView ermöglicht hier eine nicht veränderbare Liste.

Empfänger (Consumer) der Liste sollen die Daten nicht verändern können.

@immutable zeigt dem Analyzer an, dass er das prüfen soll.

```
@immutable
class Birthday {
    final String name;
    final DateTime date;
    final String? profileImage;
    final String? notes;

    const Birthday({required this.name, required
this.date, this.profileImage, this.notes});

    ...
}
```

```
UnmodifiableListView<Birthday> get birthdays =>
UnmodifiableListView(_birthdays);
```

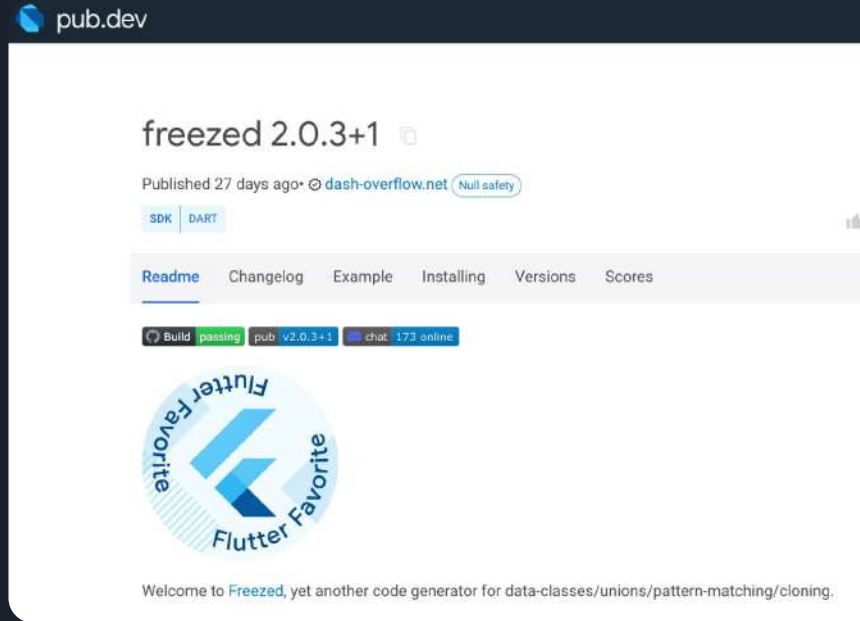


Freezed

Freezed Package

Generiert den notwendigen Code einer unveränderlichen Klasse:

- Konstruktoren
- toString, equals & hashCode Methode
- copyWith Methode
- JSON Serialisierung



[Freezed](#)

Freezed Package

Ohne Freezed
selber
entwickeln

```
@immutable
class Person {
  const Person({
    required this.firstName,
    required this.lastName,
    required this.age,
  });

  factory Person.fromJson(Map<String, Object?> json) {
    return Person(
      firstName: json['firstName'] as String,
      lastName: json['lastName'] as String,
      age: json['age'] as int,
    );
  }

  final String firstName;
  final String lastName;
  final int age;

  Person copyWith({
    String? firstName,
    String? lastName,
    int? age,
  }) {
    return Person(
      firstName: firstName,
      lastName: lastName,
      age: age,
    );
  }

  Map<String, Object?> toJson() {
    return {
      'firstName': firstName,
      'lastName': lastName,
      'age': age,
    };
  }

  @override
  String toString() {
    return 'Person(
      firstName: $firstName,
      lastName: $lastName,
      age: $age
    )';
  }

  @override
  bool operator ==(Object other) {
    return other is Person &&
      person.runtimeType == runtimeType &&
      person.firstName == firstName &&
      person.lastName == lastName &&
      person.age == age;
  }

  @override
  int get hashCode {
    return Object.hash(
      runtimeType,
      firstName,
      lastName,
      age,
    );
  }
}
```

```
@freezed
class Person with _$Person {
  const factory Person({
    required String firstName,
    required String lastName,
    required int age,
  }) = _Person;

  factory Person.fromJson(Map<String, Object?> json)
    => _$PersonFromJson(json);
}
```


Mit
Freezed

Freezed Klasse in VS Code
generieren => zum selber
schreiben

QuickType

Hilft bei API Anbindung

```
1 {
2   "description": {
3     "title": "Contiguous U.S., Average Temperature",
4     "units": "Degrees Fahrenheit",
5     "base_period": "1901-2000"
6   },
7   "data": {
8     "189512": {
9       "value": "50.34",
10      "anomaly": "-1.68"
11    },
12    "189612": {
13      "value": "51.99",
14      "anomaly": "-0.03"
15    },
16    "189712": {
17      "value": "51.56",
18      "anomaly": "-0.46"
19    }
20  }
21 }
```



```
1 @frozen
2 abstract class Temperatures with _$Temperatures {
3   const factory Temperatures({
4     Description description,
5     Map<String, Datum> data,
6   }) = _$Temperatures;
7
8   factory Temperatures.fromJson(Map<String, dynamic> json) => _$TemperaturesFromJson(json);
9 }
10
11 ...
```

Mit Hilfe von QuickType
die Freezed Klasse
generieren lassen

[QuickType](#)

QuickType

QT quicktype

Please Share! Options ?

NameSource type

TemperaturesJSON

```
{
  "description": {
    "title": "Contiguous U.S., Average Temperature",
    "units": "Degrees Fahrenheit",
    "base_period": "1901-2000"
  },
  "data": {
    "189512": {
      "value": "50.34",
      "anomaly": "-1.68"
    },
    "189612": {
      "value": "51.99",
      "anomaly": "-0.83"
    },
    "189712": {
      "value": "51.56",
      "anomaly": "-0.46"
    }
  }
}
```

```
// To parse this JSON data, do
//
//   final temperatures = temperaturesFromJson(jsonString);
import 'package:freezed_annotation/freezed_annotation.dart';
import 'dart:convert';

part 'temperatures.freezed.dart';
part 'temperatures.g.dart';

@freezed
abstract class Temperatures with _$Temperatures {
  const factory Temperatures({
    Description description,
    Map<String, Datum> data,
  }) = _Temperatures;

  factory Temperatures.fromJson(Map<String, dynamic> json) => _$TemperaturesFromJson(json);
}

@freezed
abstract class Datum with _$Datum {
  const factory Datum({
    String value,
    String anomaly,
  }) = _Datum;

  factory Datum.fromJson(Map<String, dynamic> json) => _$DatumFromJson(json);
}

@freezed
abstract class Description with _$Description {
  const factory Description({
    String title,
    String units,
    String basePeriod,
  }) = _Description;

  factory Description.fromJson(Map<String, dynamic> json) => _$DescriptionFromJson(json);
}
```

LanguageOther

Dart

Use this name in 'part' directive

Use this name in 'part' directive

Types only

☒ Put encoder & decoder in Class

☒ Use method names from Map() & toMap()

☐ Make all properties required

☐ Make all properties final

☐ Generate CopyWith method

☒ Generate class definitions with @freezed compatibility

☐ Make all properties optional

Copy Code

QuickType

Dart Build Runner

Build-Runner



build_runner 2.1.11

Published 21 days ago • [tools.dart.dev](#) Null safety

SDK | DART

👍 921

[Readme](#) Changelog Installing Versions Scores

Standalone generator and watcher for Dart using `package:build`.

[open "package: build_runner" issues](#) [38](#) [pub v2.1.11](#) [dartdocs](#) [latest](#) [chat](#) [on gitter](#)

The `build_runner` package provides a concrete way of generating files using Dart code, outside of tools like `pub`. Unlike `pub serve/build`, files are always generated directly on disk, and rebuilds are *incremental* - inspired by tools such as [Bazel](#).

[build_runner](#)

Was macht der Build-Runner?


Mechanismus um Code generieren und diesen auch testen zu lassen.

Im Fall der Freezed Packages:

*.freezed.dart

*.g.dart

Falls JSON Serialisierung generiert werden soll.



Generiert **hashCode()**, **toString()** oder **copyWith()**.

Auf Wunsch auch **toJson()** und **fromJson()** -> dazu muss die build.yaml angepasst werden.

Build-Runner

Package der pubspec hinzufügen

```
dev_dependencies:  
  build_runner: ^2.1.11
```


Make command der Makefile hinzufügen

```
build-runner:  
  $(FLUTTER) pub run build_runner build --delete-conflicting-outputs
```

Make command ausführen

```
make build-runner
```

Wir gehen davon aus, dass Konflikte aus alten Dateien entstanden sind und ignoriert werden können.



[build_runner](#)



Freezed und Build Runner zusammen

Domain Klasse generieren

1. Packages in pubspec.yaml hinzufügen
 - a. `freezed_annotation` unter `dependencies`
 - b. `freezed` und `build_runner` unter `dev_dependencies`
2. Implementierung der `freezed` Klasse
3. Die `freezed` Klassen generieren

```
dependencies:  
  flutter:  
    sdk: flutter  
  freezed_annotation: ^2.0.3  
  
dev_dependencies:  
  flutter_test:  
    sdk: flutter  
  freezed: ^2.0.3+1  
  build_runner: ^2.1.11
```

Ablauf

```
{  
  "counter":2  
}
```

app.quicktype.io

make build-runner

counter.entity.dart

JSON
kopieren



**Freezed
Klasse
generieren
und kopieren**



**Entity
generieren**



**Entiy
verwenden**

Immutable Klasse mit Freezed implementieren

Aufgabe:

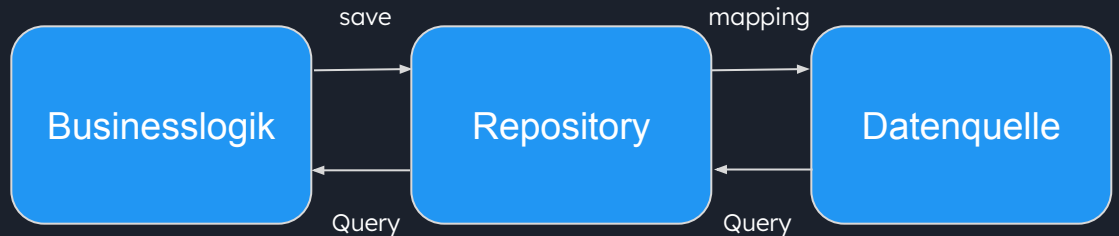
- Konfiguriere Freezed und den Build Runner
- Generiere über Quicktype
- Generiere die Counter Domain Klasse
- Bau die Domain Klasse in das Counter Projekt ein



3. Data Layer - Repository

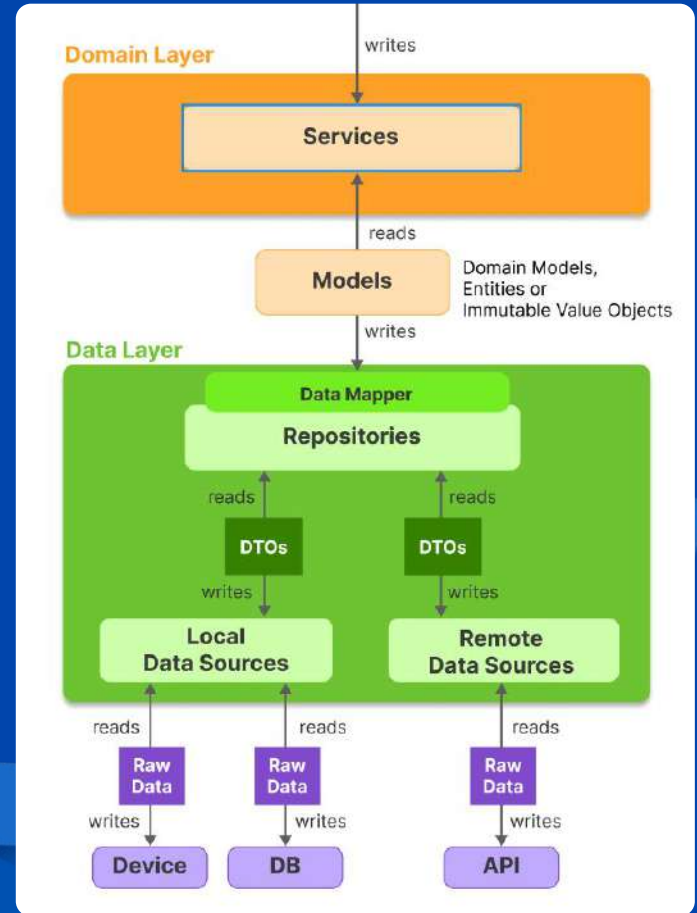
Repository Pattern

- Auftrennung zwischen Businesslogik und Datenbeschaffung
- Zentrale Zuständigkeit, um Daten zu beschaffen
- Ist unabhängig von der Datenquelle
- Lädt Daten und mappt auf Objekte
- Repository lädt selbst nicht die Daten, sondern hilft bei der Datenbeschaffung
- Hilft bei den Tests, denn es lässt sich schnell austauschen

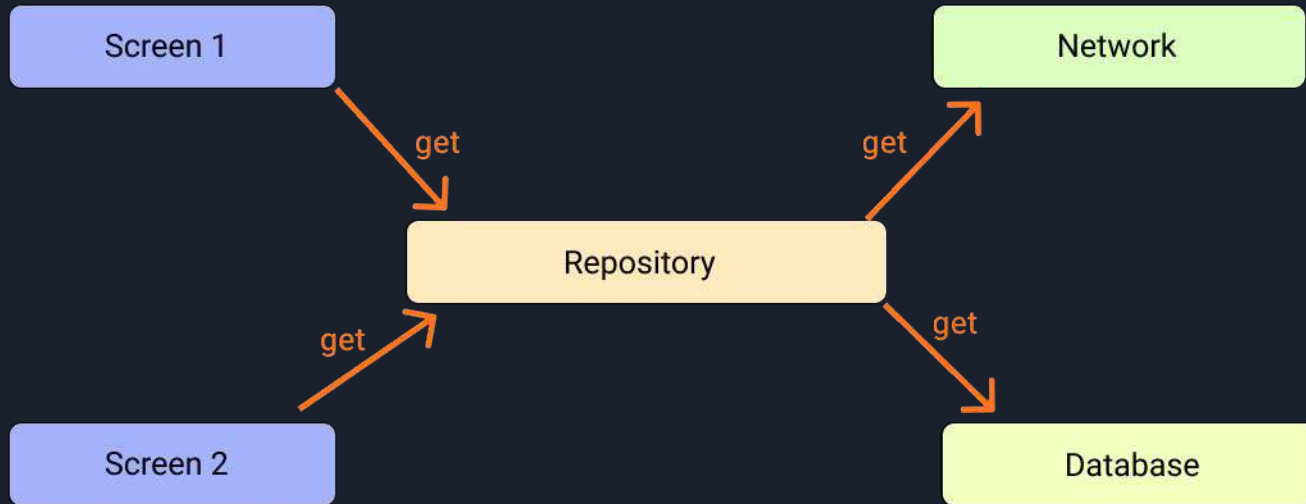


Repository Pattern in der Architektur

- Layer nennt man auch **Infrastructure**
- Unterscheidung zwischen **Local** und **Remote** Data Sources
- Repository managed nur den Aufruf der Data Source
- Wandelt **Datentransferobjekte** (DTO) in **Entities** um



Repository Struktur



Dartpad

Singleton im Repository

- Singleton ist ein Design-Pattern
- Dient dazu, immer nur ein Objekt des Repositories im Projekt zu halten
- Ist global im Projekt zugänglich
- Ermöglicht Zugriffskontrolle im Projekt
- Einmal instanziiert bleiben diese bis zum Beenden der Applikation bestehen
→ negative Performance

```
class CounterRepo {  
    static final CounterRepo _instance = CounterRepo._internal();  
  
    factory CounterRepo() => _instance;  
  
    late Counter counterEntity;  
  
    CounterRepo._internal() {  
        _fetchData();  
    }  
  
    int get counter => counterEntity.counter;  
  
    void increment() {  
        counterEntity = counterEntity.copyWith(counter:  
            counterEntity.counter + 1  
        );  
    }  
  
    void _fetchData() {  
        int receivedData = FakeDataSource.fetchData();  
        counterEntity = Counter(counter: receivedData);  
    }  
}
```

Zugriff über DataSource

Warum benötigt man eine DataSource?

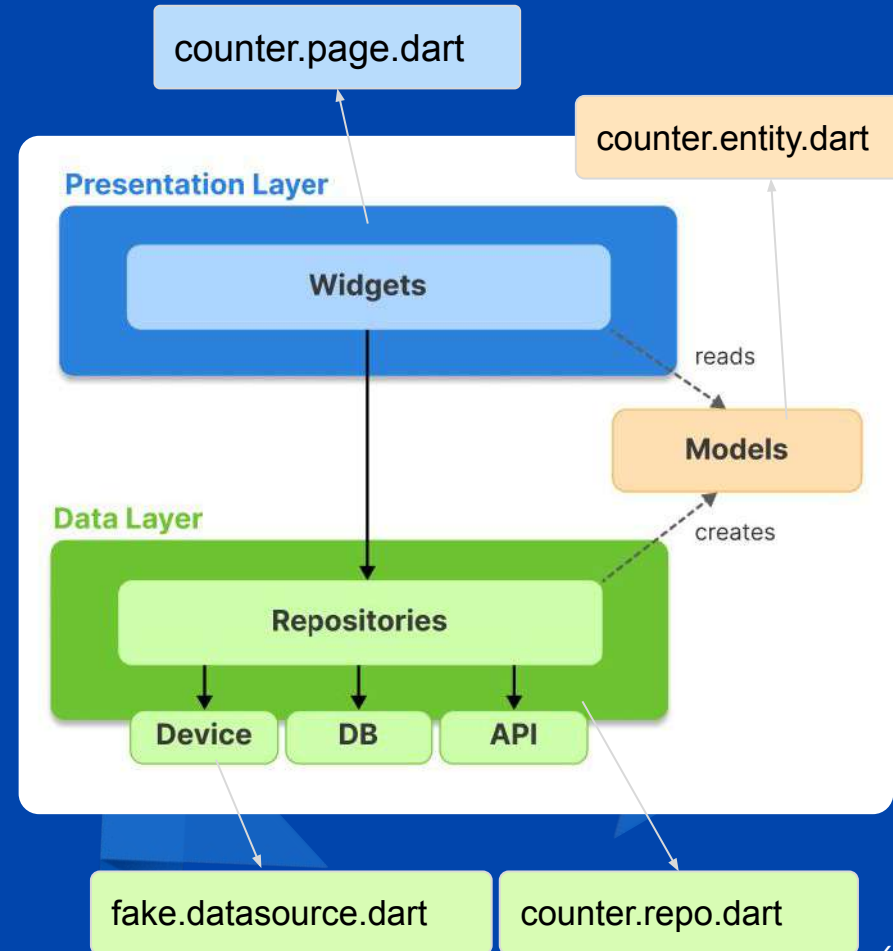
- DataSource kapselt den Zugriff auf die “Raw-Daten” weg.
- Liefert DTOs in der großen Architektur
- In der “minimalen” Architektur dürfen direkt die Domain Models erstellt werden

Idealerweise ist die Datasource generiert oder wird bereitgestellt.

```
class CounterApiDataSource {  
  
    static int getCurrentCount() {  
        // API Call  
    }  
  
    static void resetCounter() {  
        // API Call  
    }  
  
}
```

Minimale Architektur

- Kleinste mögliche Architektur
- Zwei Layer:
 - Data
 - Presentation
- Direkt mit **Entities** arbeiten



Implementiere die Data Layer

Aufgabe:

- Implementiere ein Repository als Singleton
- Lade den Wert initial aus einer Data Source
- Beziehe den Counter Wert über das Repository

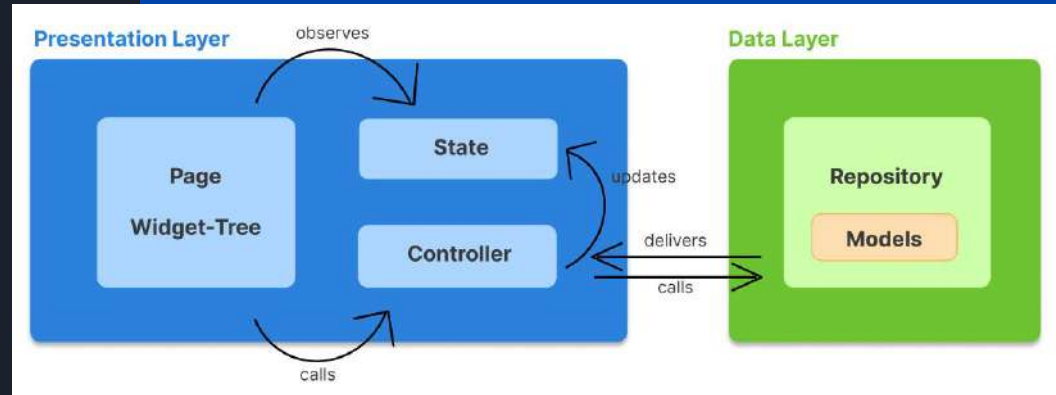


4. Presentation Layer

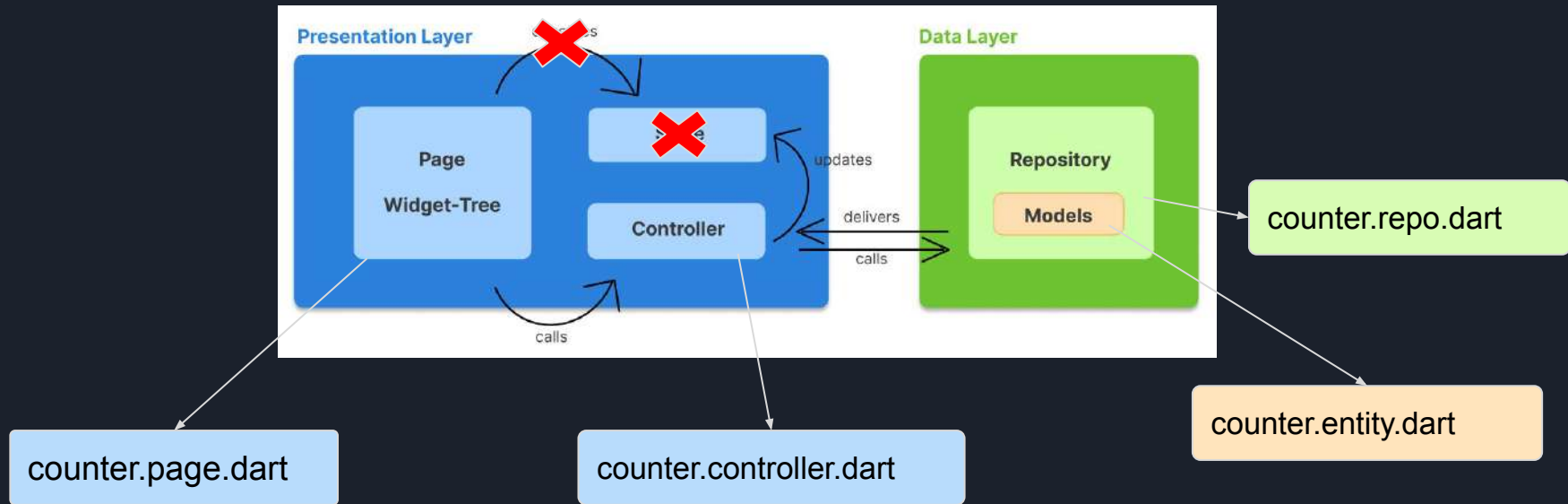
Datenfluss

Presentation Layer

- Page mit Widgets **zeigt** Daten aus dem **State**
- State wird durch den **Controller aktualisiert**
- Page **ruft** Aktionen des **Controllers** auf
- Controller **lädt** aus **Repository** Daten und Updated State



Erste Version mit Controller und ohne State



Controller Klasse

- Separation of concerns -> View / ViewLogic
- Bereitet die Daten für die View vor
- Hält Flags, die für die Darstellung von Teilen der Page notwendig sind
- Bietet Methoden für die View an, um Logik auszuführen oder Aktionen zu starten
- **Jede Page hat seinen eigenen Controller**

```
class CounterController {  
    CounterRepo counterRepo = CounterRepo();  
  
    CounterController();  
  
    int get counter => counterRepo.counter;  
  
    increment() => counterRepo.increment();  
}
```

Implementiere den Controller

Aufgabe:

- Implementiere einen Counter Controller
- Verwende den Counter Controller in der Counter Seite
- Implementiere eine zweite Seite und dessen Controller, worin der Counter Wert auch angezeigt wird
- In der View anzeigen, ob die Zahl gerade oder ungerade ist. Der Controller bereitet vor.

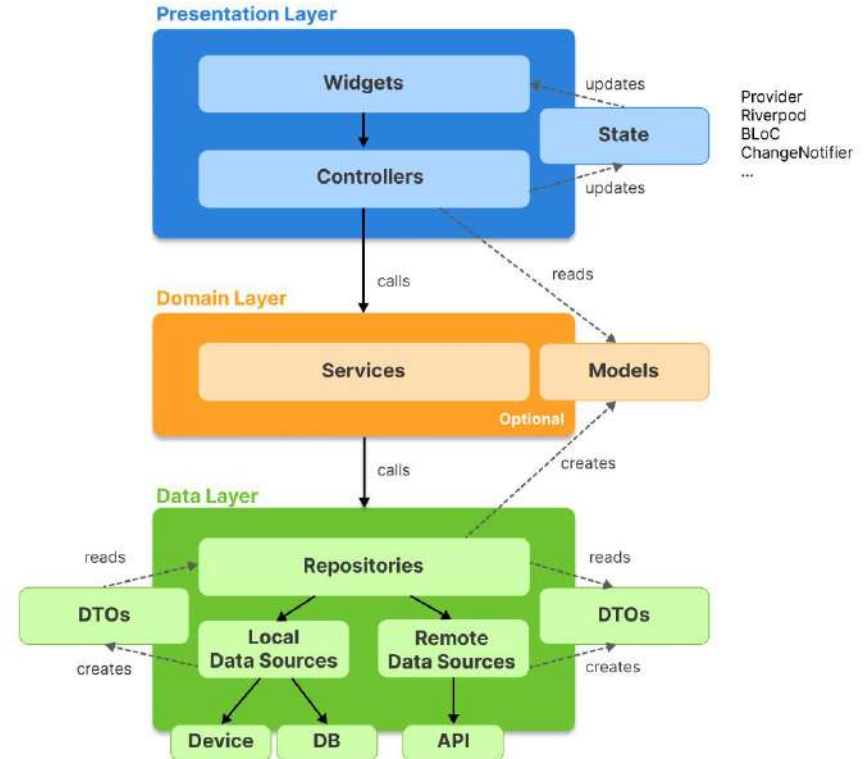


5. State Management mit Provider

State Management Einführung

State Management in der Architektur

- Einordnung des **State Managements** in einer möglichen Architektur
- Controller und State Model als **Presentation Logic Holders**
- In Repositories oder in der Domain Schicht werden Daten manipuliert -> die View muss darauf reagieren



Was ist ein State?

- Flutter ist deklarativ
-> Die UI wird gebaut, um den **aktuellen State** der App darzustellen.
- Wenn der State sich ändert, wird die UI vollständig neu gerendert.
- Unterscheidung zwischen **Ephemeral state** (flüchtig) und **App state** (übergreifend)

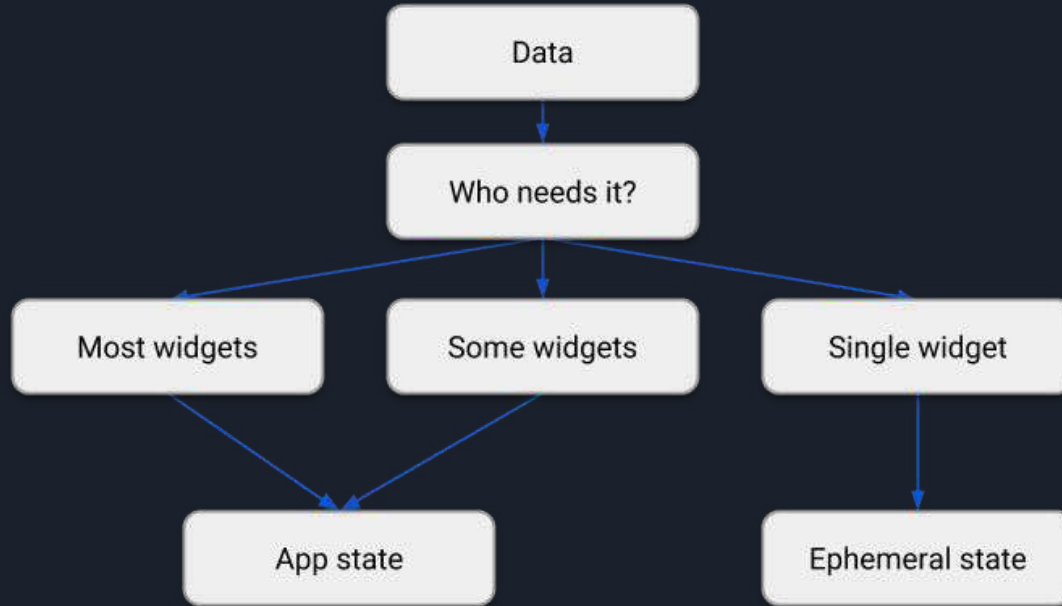
$$\text{UI} = f(\text{state})$$

The layout
on the screen

Your
build
methods

The application state

Ephemeral State & App State



Keine ganz klare Regel,
wann eine Variable zu
dem Ephemeral State
oder App State gehört.



TOP 30 Flutter State

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

Likes and position **Oct 28, 2021**
in **pub.dev** of **all** packages.
(Changes are from issue nr 12, July 6, 2021)

Included info:

- NS = Has Null Safety version
- Test CodeCov % when available
- ApiDoc completeness %
- GitHub stars
- GitHub Issues Open/Closed

Test%	API docs%	[90...100]	Points	[130]	Null safety
-------	-----------	------------	--------	-------	-------------

Test%	API docs%	[80...90[Points	[120...125]
80	80	80	120	120
85	85	85	122.5	122.5
90	90	90	125	125

Test% API docs% [60...80[Points [100...115]

Test% API docs% [7/0...60[Points [0...90] No null safety

Stats summary by @RydMike (Mike Rydstrom)

Package	Author	Rank	Likes	Version	Updated	NS	CodeCov	API docs	Points	Popularity	GitHub ★	Likes Stars	Open Closed	Position		
get (GetX)	Jonaslew	1	6655	+1443	4.3.8	13.08.2021	Yes	43.1%	31.0%	120	-10	99%	5078	1.31	331/1087	1
provider	rrousselGit	2	5343	+743	6.0.1	24.09.2021	Yes	99.3%	90.1%	130		100%	3918	1.36	2/168	2
flutter_bloc	felangel	3	2939	+499	7.3.1	14.10.2021	Yes	100.0%	100.0%	130		100%	+1 7949	0.37	36/1855	7
get_it	escamoteur	4	1595	+268	7.2.0	13.07.2021	Yes	88.6%	76.9%	130		99%	793	2.01	15/171	38
rxdart	ReactiveX	5	1371	+179	0.27.2	03.09.2021	Yes	93.0%	98.3%	125		100%	2967	0.46	29/298	51
bloc	felangel	6	1253	+204	7.2.1	26.09.2021	Yes	100.0%	96.8%	130		99%	7949	0.16	36/1855	58
riverpod	rrousselGit	7	1114	+271	1.0.0-dev-11	07.10.2021	Yes	95.9%	85.5%	130		98%	2049	0.54	57/497	69
stacked	FilledStacks	8	858	+100	2.2.7	11.10.2021	Yes	?	44.9%	110	-10	97%	553	1.55	42/309	91
velocity_x	iampawan	9	+1 763	+124	3.3.0	17.06.2021	Yes	?	32.0%	120	-10	96%	843	0.91	12/79	108
flutter_modular	Flutterando	10	+1 740	+126	4.1.2	08.10.2021	Yes	100.0%	40.8%	130	+10	97%	968	0.76	16/385	110
mobx	mobxjs	11	-2 711	+72	2.0.5	07.10.2021	Yes	99.1%	31.5%	130		98%	2010	0.35	19/404	114
flutter_riverpod	rrousselGit	12	587	+121	1.0.0-dev-11	09.11.2021	Yes	95.9%	93.8%	130		98%	2049	0.29	57/497	147
injectable	Milad-Akari	13	485	+81	1.5.0	15.08.2021	Yes	?	92.7%	130		98%	283	1.71	49/142	188
flutter_mobx	mobxjs	14	417	+57	2.0.2	07.08.2021	Yes	99.1%	91.9%	130		98%	2010	0.21	19/404	207
states_rebuilder	GtatoNTH	15	323	+12	5.0.0	13.10.2021	Yes	94.8%	65.5%	130	+10	95%	-2 397	0.81	17/159	275
flutter_redux	brianegan	16	313	+31	0.8.2	18.03.2021	Yes	98.2%	93.2%	120		97%	1514	0.21	15/171	282
hooks_riverpod	rrousselGit	17	301	+68	1.0.0-dev-11	09.10.2021	Yes	95.9%	91.7%	130		97%	2049	0.15	57/497	295
redux	fluttercommunity	18	249	+29	5.0.0	23.02.2021	Yes	92.1%	88.9%	115		96%	-1 479	0.52	5/34	352
flutter_clean_architecture	ShadyBoukhary	19	new 198	new	5.0.0	05.03.2021	Yes	?	52.8%	110		88%	445	0.44	5/43	438
scoped_model	brianegan	20	-1 180	+16	1.1.0	09.12.2020	No	93.1%	76.9%	110		97%	752	0.24	13/80	482
state_notifier	rrousselGit	21	-1 172	+19	0.7.1	06.09.2021	Yes	?	83.9%	130		98%	242	0.71	4/34	500
mvc_pattern	AndriousSolutions	22	-1 134	+18	7.4.0	08.07.2021	Yes	96.0%	85.7%	120		96%	139	0.96	1/21	610
async_redux	margielasberg	23	100	+7	13.0.6	09.10.2021	Yes	?	39.1%	120		90%	+1 195	0.51	6/80	789
momentum	xamantra	24	87	+5	2.2.1	20.08.2021	Yes	100.0%	95.3%	120	-10	74%	+1 111	0.78	3/39	902
kiwi	vanlooverenkoen	25	84	+19	4.0.1	31.10.2021	Yes	?	91.7%	130		95%	+1 306	0.27	3/35	945
fish_redux	alibaba	26	54	+4	0.3.7	09.03.2021	No	53.2%	31.4%	100		92%	+1 7170	0.01	154/431	1322
get_it_mixin	escamoteur	27	51	+7	3.1.3	21.06.2021	Yes	?	85.0%	130		83%	+2 23	2.22	2/11	1376
binder	letsr	28	+2 48	+7	0.4.0	25.03.2021	Yes	99.7%	94.3%	120		58%	-5 163	0.29	4/14	1440
flutter_command	escamoteur	29	44	+3	2.0.1	07.05.2021	Yes	91.6%	57.1%	130		73%	+1 29	1.52	3/4	1540
rx_command	escamoteur	30	-2 43	+2	6.0.1	13.07.2021	Yes	?	45.2%	115	+10	86%	130	0.33	1/40	1561
mwmm	surfstudio	31	34	+7	7.0.0	07.07.2021	Yes	?	61.7%	120	-10	81%	-1 33	1.03	0/0, 0/1	1881
fountain	aloisdaniel	32	10		0.3.0-beta	14.05.2021	Yes	?	22.4%	120		0%	-47 45	0.22	2/1	4109

<code>flutter_bloc</code> (<code>flutter_bloc + bloc</code>)	3	4192	+703	7949	0.53
<code>riverpod</code> (<code>riverpod + flutter_riverpod + hooks_riverpod</code>)	4	2002	+460	2049	0.98
<code>get_it</code> (<code>get_it + get_it_mixin + flutter_command</code>)	5	1690	+278	845	2.00
<code>mobx</code> (<code>mobx + flutter_mobx</code>)	7	1128	+129	2010	0.56

State Management Lösungen

Stateful Widgets

(Flutter build in)

InheritedWidget & InheritedModel

(Flutter build in)

Provider (von Flutter empfohlen)

 pub.dev <https://pub.dev/packages/provider>

Riverpod

 pub.dev https://pub.dev/packages/flutter_riverpod

BLoC / Rx

 pub.dev https://pub.dev/packages/flutter_bloc

Redux

 pub.dev https://pub.dev/packages/flutter_redux

und noch einige weitere..

<https://docs.flutter.dev/development/data-and-backend/state-mgmt/options>

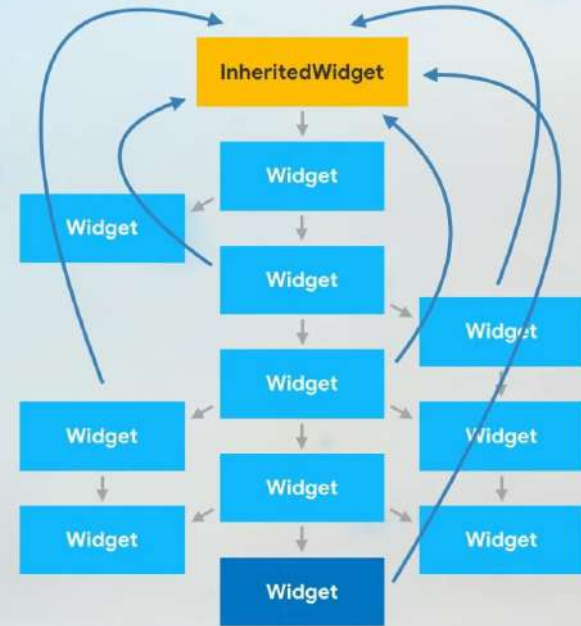
<https://twitter.com/RydMike/status/1528827017172504579>

InheritedWidget Funktion

- Direkter Zugriff auf das Inherited Widget
- Es müssen keine Daten durchgegeben werden.

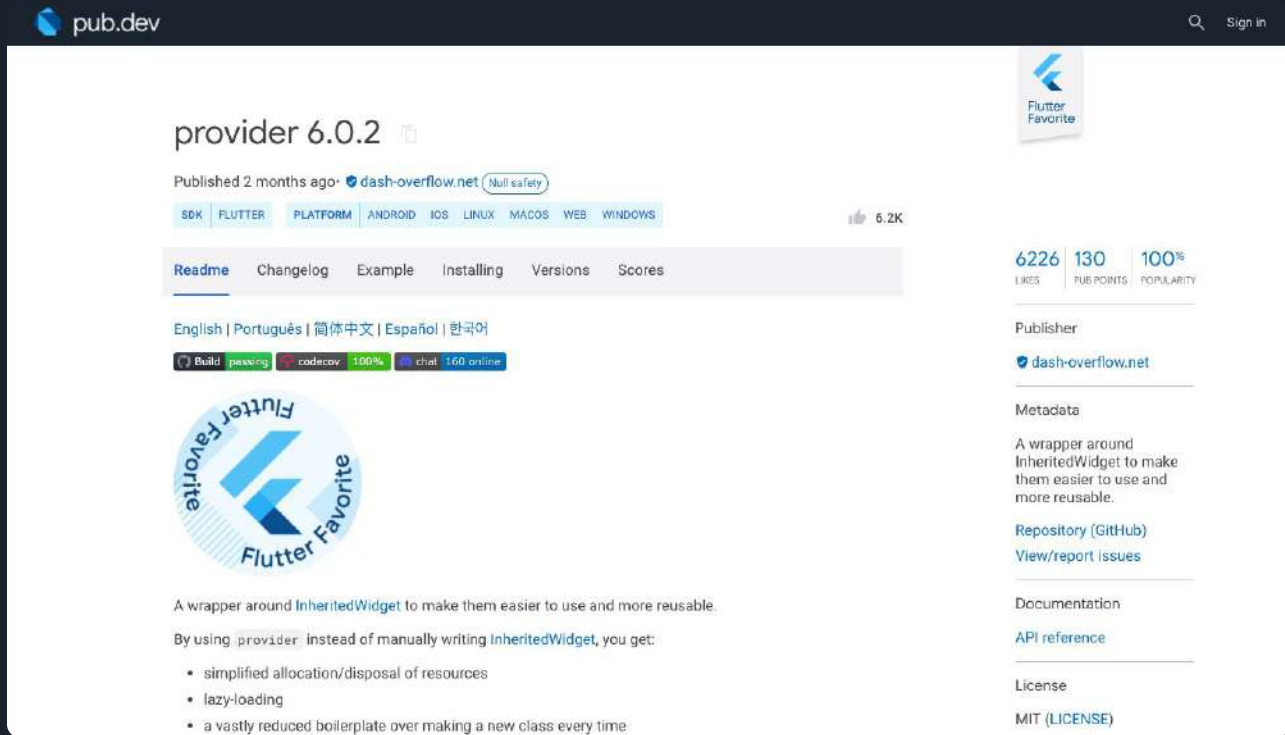
Aus dem Counter Beispiel:

```
children: <Widget>[  
  const Text(  
    'You have pushed the button this many times:',  
  ), // Text  
  Text(  
    '$_counter',  
    style: Theme.of(context).textTheme.headline4,  
  ), // Text  
, // <Widget>[]
```



Einführung Provider

Provider Package



The screenshot shows the pub.dev page for the `provider` package version 6.0.2. The page is titled "provider 6.0.2" and indicates it was published 2 months ago by `dash-overflow.net`. It features a "Flutter Favorite" badge and a "Null safety" label. The package is available for various platforms: SDK, FLUTTER, PLATFORM, ANDROID, IOS, LINUX, MACOS, WEB, and WINDOWS. It has 6.2K likes. The page includes tabs for Readme, Changelog, Example, Installing, Versions, and Scores. The Readme tab is active, showing the package's description: "A wrapper around `InheritedWidget` to make them easier to use and more reusable." It also lists the benefits of using `provider` over manually writing `InheritedWidget`: simplified allocation/disposal of resources, lazy-loading, and a vastly reduced boilerplate. The page also shows the publisher `dash-overflow.net`, metadata, repository (GitHub), documentation, and license (MIT).

pub.dev

provider 6.0.2

Published 2 months ago • `dash-overflow.net` (Null safety)

SDK FLUTTER PLATFORM ANDROID IOS LINUX MACOS WEB WINDOWS

6.2K

Readme Changelog Example Installing Versions Scores

English | Português | 简体中文 | Español | 한국어

Build: passing codecov: 100% chat: 160 online

Flutter Favorite

A wrapper around `InheritedWidget` to make them easier to use and more reusable.

By using `provider` instead of manually writing `InheritedWidget`, you get:

- simplified allocation/disposal of resources
- lazy-loading
- a vastly reduced boilerplate over making a new class every time

Flutter Favorite

6226 130 100%

LIKES PUB POINTS POPULARITY

Publisher

`dash-overflow.net`

Metadata

A wrapper around `InheritedWidget` to make them easier to use and more reusable.

Repository (GitHub)

View/report issues

Documentation

API reference

License

MIT (LICENSE)

<https://pub.dev/packages/provider>

Was wird durch Provider erreicht?

“A wrapper around `InheritedWidget` to make them **easier to use** and more **reusable**.”

Ein Wrapper für `InheritedWidget`, um es **einfacher nutzbar** und **wiederverwendbar** zu machen

Provider

Managed den Lifecycle konfigurierten
Notifier (Models)

Typische Provider

- **MultiProvider**
Wird viel verwendet, weil es ein einfacher und schneller Weg ist, verschiedenste Provider zu initialisieren.
- **ChangeNotifierProvider**
Kann verwendet werden, wenn ein ChangeNotifier genutzt wird.

```
void main() {  
  runApp(  
    MultiProvider(  
      providers: [  
        ChangeNotifierProvider(create: (context) => CounterRepo()),  
        ...  
      ],  
      child: const MyApp(),  
    ),  
  );  
}
```

ChangeNotifier

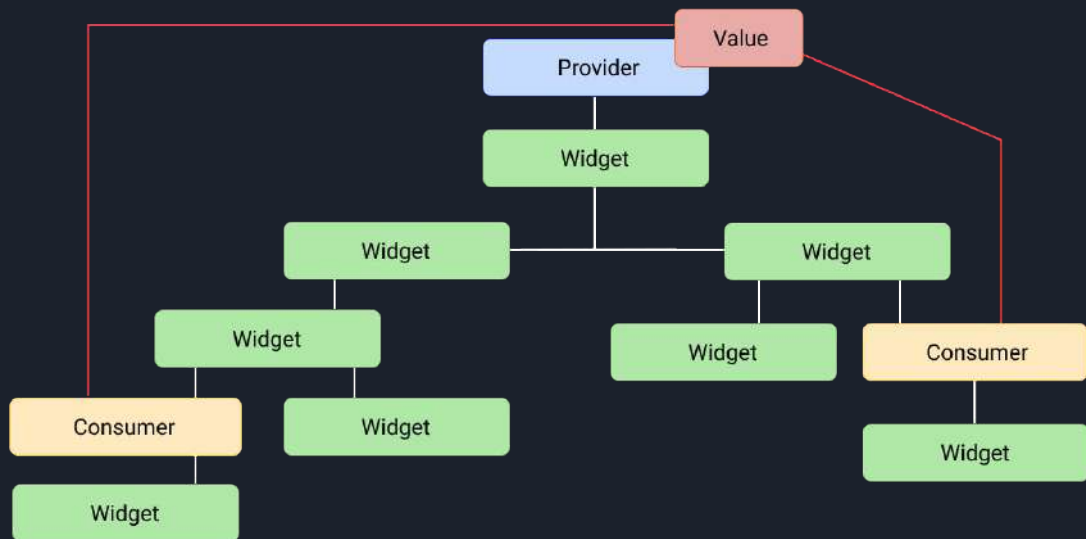
Ein **Notifier** gibt bekannt, wenn sich etwas geändert hat.

Das **Event**, dass sich etwas an den Daten geändert hat, wird durch `notifyListeners()` an alle Beobachter bekannt gemacht.

Alternative für nur ein Value ist der `ValueNotifier`.

```
class CounterRepo extends ChangeNotifier {  
  late Counter _counterEntity;  
  
  CounterRepo() {  
    _fetchData();  
  }  
  
  int get counter => _counterEntity.counter;  
  
  void increment() {  
    _counterEntity = _counterEntity.copyWith(  
      counter: _counterEntity.counter + 1  
    );  
    notifyListeners();  
  }  
}
```


Consumer



Ein Consumer Widget **hört** auf die Änderung des Value und **zeichnet** anschließend **sich selbst** und die **Widgets unterhalb erneut**.

Consumer ist Teil des Provider Package.

[Consumer API-Link](#)

Consumer Widget

Ein Consumer Widget hört auf Änderungen des States.

context.read(): Widget erhält State und ignoriert Änderungen.

context.watch(): Widget erhält State und hört auf Veränderung

```
void _incrementCounter() {  
  setState(() {  
    context.read<CounterController>().increment();  
  });  
}
```

```
@override  
Widget build(BuildContext context) {  
  int counterValue =  
    context.watch<CounterController>().counter.counter;  
  return Scaffold(  
    appBar: AppBar(  
      title: Text(widget.title),  
    ),  
    body: Center(  
      child: Column(  
        ...  
      )  
    )  
  );  
}
```

Provider einführen

Aufgabe:

- Füge das Provider Package zum Projekt hinzu
- Ersetze das Singleton im Repository
- Füge Provider in den Controllern und nutze den neuen Provider des Repositories hinzu.
- Passe die Page entsprechend den Provider Anpassungen an.



6. State Management mit Riverpod

Was ist Riverpod?

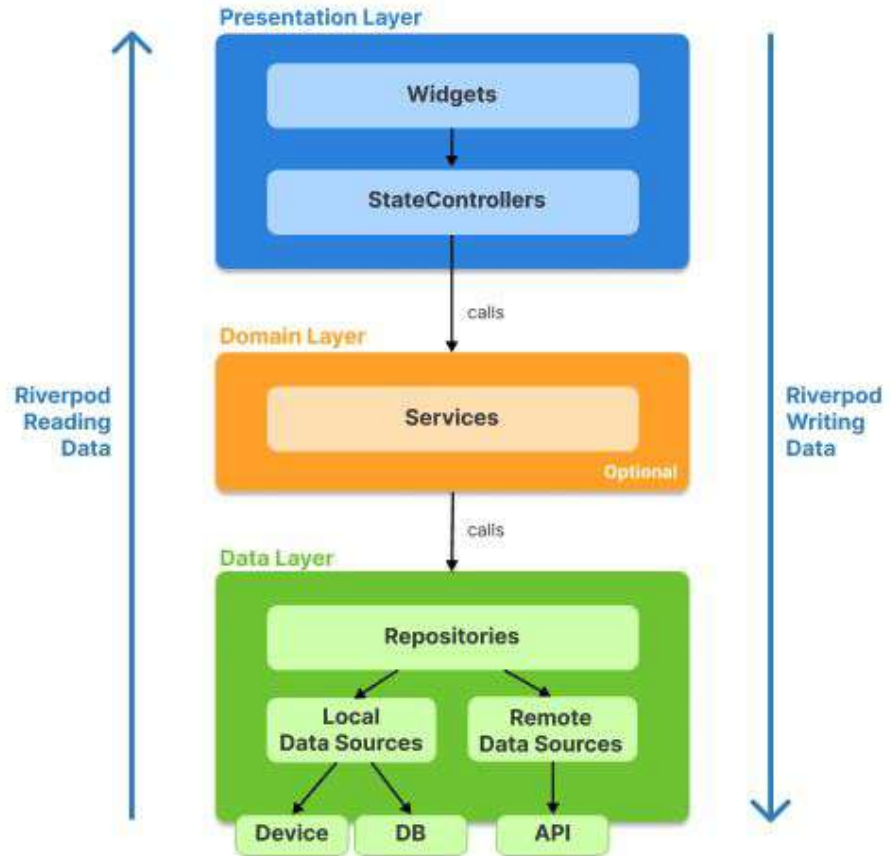
- Verwendet die **Vorteile** von Provider und bringt **neue Erweiterungen** mit
- Ermöglicht **einfach** Provider zu erstellen, zuzugreifen und zu **kombinieren**
- Implementierter Code wird **einfach testbar**
- **Zeigt Fehler** bereits zur **Kompilierzeit**



This project can be considered as a rewrite of **provider** to make improvements that would be otherwise impossible.

Architektur und Datenfluss

Daten über die
verschiedenen Schichten
hinweg **lesen** und **schreiben**
unter Verwendung von
Riverpod.



Was bietet Riverpod?

Riverpod **Provider** sind **global** und leben **außerhalb** des Widget Trees.

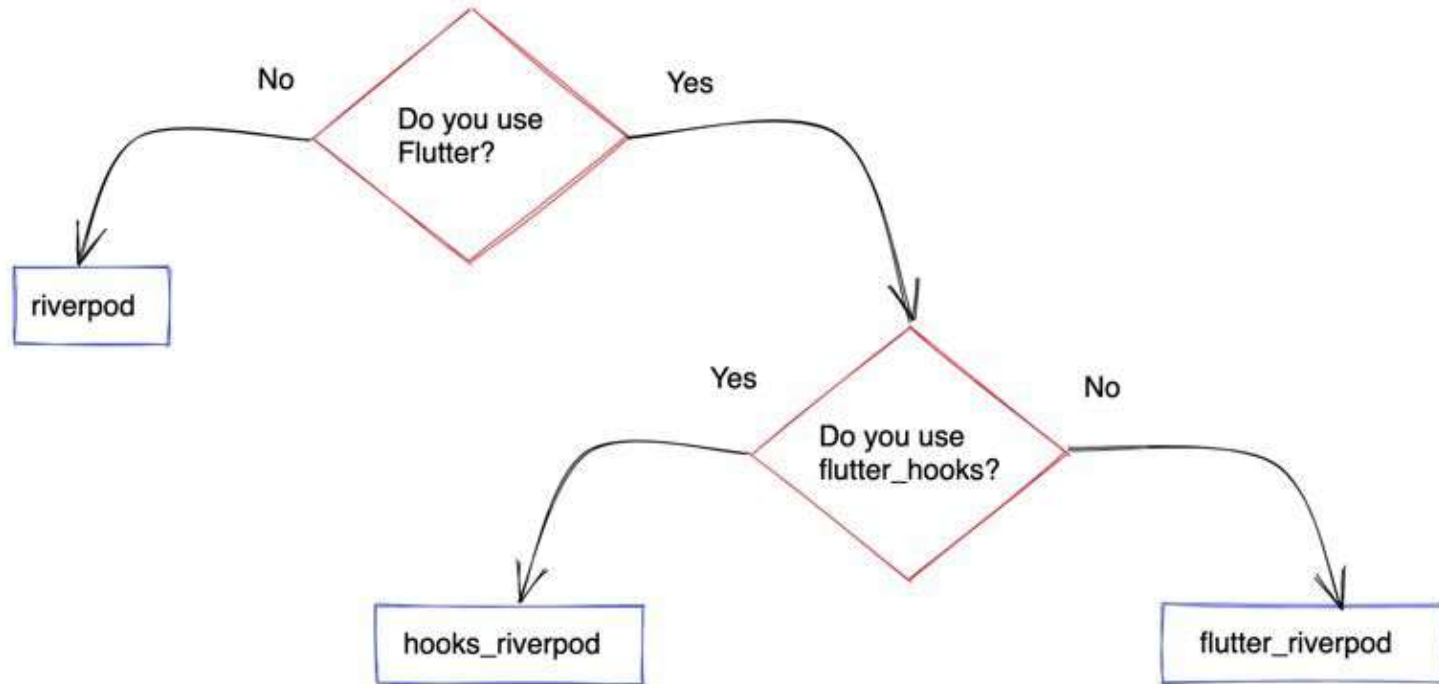
Verschiedene Provider für den jeweiligen **Anwendungsfall**.

Der **Kern** von Riverpod ist **Dependency Injection**.

Bietet State-Management, wie wir es von Provider kennen.

Beschreibt sich selbst als “**A Reactive Caching and Data-binding Framework**”

Welches Riverpod?



Einführung von Riverpod

In der pubspec.yaml

flutter_riverpod konfigurieren

```
dependencies :  
  flutter:  
    sdk: flutter  
  freezed_annotation: ^2.0.3  
  provider: ^6.0.3  
  flutter_riverpod: ^1.0.4
```



Riverpod initialisieren -> ProviderScope

```
void main() {  
  WidgetsFlutterBinding.ensureInitialized();  
  runApp(const ProviderScope(child: App()));  
}
```

ProviderScope muss als Root aller Widgets im Widget Tree vorliegen.

Es speichert die States aller definierten Provider.

Verschiedene Provider

Provider Type	Provider Create Function	Example Use Case
Provider	Returns any type	A service class / computed property (filtered list)
StateProvider	Returns any type	A filter condition / simple state object
FutureProvider	Returns a Future of any type	A result from an API call
StreamProvider	Returns a Stream of any type	A stream of results from an API
StateNotifierProvider	Returns a subclass of StateNotifier	A complex state object that is immutable except through an interface
ChangeNotifierProvider	Returns a subclass of ChangeNotifier	A complex state object that requires mutability

The background of the slide is a solid blue color. Scattered across this background are several abstract, three-dimensional geometric shapes. These shapes are composed of flat, blue polygonal faces, giving them a faceted, crystalline appearance. They are positioned at various angles and locations, some appearing to float or be part of a larger, unseen structure. The lighting is subtle, with some faces appearing slightly darker than others, creating a sense of depth.

Migration von Provider zu Riverpod

Anpassungen um im **Repository** Provider mit Riverpod zu ersetzen

Bisher: Provider

```
class CounterRepo extends ChangeNotifier {  
    CounterRepo() {_fetchData();}  
    late Counter _counter;  
    Counter get counter => _counter;  
    Counter increment() {  
        _counter = _counter.copyWith(  
            counter: _counter.counter + 1  
        );  
        return _counter;  
    }  
    void _fetchData() {  
        int receivedData = FakeDataSource.fetchData();  
        _counter = Counter(counter: receivedData);  
    }  
}
```

Neu: Riverpod

```
final counterRepoProvider = Provider((ref) {  
    return CounterRepo();  
});
```

```
class CounterRepo {  
    CounterRepo() { _fetchData(); }  
    late Counter _counter;  
    Counter get counter => _counter;  
    Counter increment() {  
        _counter = _counter.copyWith(  
            counter: _counter.counter + 1  
        );  
        return _counter;  
    }  
    void _fetchData() {  
        int receivedData = FakeDataSource.fetchData();  
        _counter = Counter(counter: receivedData);  
    }  
}
```

Anpassungen um Controller Provider mit Riverpod zu ersetzen

Bisher: Provider

```
class CounterController extends ChangeNotifier {  
  CounterController(this.context) {  
    _counter = context.read<CounterRepo>().counter;  
  }  
  
  late BuildContext context;  
  late Counter _counter;  
  Counter get counter => _counter;  
  
  increment() {  
    _counter = context.read<CounterRepo>().increment();  
    notifyListeners();  
  }  
}
```

Neu: Riverpod

```
final counterControllerProvider = ChangeNotifierProvider.  
    autoDispose((ref) {  
      return CounterController(ref.read());  
    });
```

```
class CounterController extends ChangeNotifier {  
  CounterController(this.read) {  
    _counter = read(counterRepoProvider).counter;  
  }  
  
  Reader read;  
  late Counter _counter;  
  Counter get counter => _counter;  
  
  increment() {  
    _counter = read(counterRepoProvider).increment();  
    notifyListeners();  
  }  
}
```

Anpassungen um Page Provider mit Riverpod zu ersetzen

Bisher: Provider

```
class MyHomePage extends StatefulWidget {  
  ...  
  Widget build(BuildContext context) {  
    int counterValue = context.watch<CounterController>()  
      .counter.counter;  
    ...  
    void _incrementCounter() {  
      setState(() {  
        context.read<CounterController>().increment();  
      });  
    }  
  }  
}
```

Neu: Riverpod

```
class MyHomePage extends ConsumerWidget {  
  ...  
  @override  
  Widget build(BuildContext context, WidgetRef ref) {  
    // Auf Änderungen hören und anschließend neu bauen  
    int counterValue =  
      ref.watch(counterControllerProvider).counter.counter;  
    return Scaffold( ...  
      body: Center(...  
        Text(  
          '$counterValue',  
          ...  
        floatingActionButton: FloatingActionButton(  
          onPressed: () => ref.read(  
            counterControllerProvider.notifier).increment(),  
            ...  
          )  
        )  
      )  
    )  
  }  
}
```

Zugriff und Modifier

Über ref mit den Providern interagieren

- ref.watch()** Lädt die Daten des Providers und hört auf Änderungen (**observe**).
Ändern sich die Daten, wird das Widget neu gebaut (**rebuild**).
Verwendung: z.B. reaktiver Controller
- ref.listen()** Fügt einen Listener hinzu und führt eine bestimmte Aktion (**Callback**) aus,
wenn sich die Daten im Provider ändern.
Verwendung: z.B. Snackbar anzeigen
- ref.read()** Liest die Daten eines Providers nur einmal und reagiert nicht auf
Veränderung der Daten im Provider.
Verwendung: Standard Dependency Injection

Über ref mit den Providern interagieren

Wenn immer möglich **besser `ref.watch()` verwenden** anstatt `ref.read()` oder `ref.listen()`. Wenn man die App so implementiert, dass sie sich auf `ref.watch()` verlässt, wird sie wartungsfreundlicher.

`ref.watch(counterControllerProvider)`

auf Änderungen “hören” und reagieren



`ref.watch(counterControllerProvider.notifier)`

nur lesend zugreifen



Provider Modifiers

Standard

Globaler Provider, der als `Singleton` agiert - kein Modifier.

```
final counterRepoProvider = Provider<int>((ref) => 8);
```

AutoDispose

Die Provider `Instanz` aus dem Speicher `entfernen`, wenn sie nicht mehr benötigt wird - `AutoDispose` Modifier.

```
final counterRepoProvider = Provider.autoDispose<int>((ref) => 8);
```

Family

Einen `eindeutigen` Provider anhand einer `ID` erhalten - `Family` Modifier.

```
final counterRepoProvider = Provider.family<int, String>((_, ref) { return 8; });
```

Code-Beispiel mit Modifiers (autoDispose)

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

final counterControllerProvider = ChangeNotifierProvider.autoDispose((ref) {
  return CounterController(ref.read);
});

class CounterController extends ChangeNotifier {
  CounterController(this.read) {
    _counter = read(counterRepoProvider).counter;
  }
  ...
}
```



Provider wird aus dem Speicher entfernt, sobald kein `watch()` mehr darauf zeigt.

Solange mit `watch()` ein Provider innerhalb eines aktiven `BuildContext` beobachtet wird, solange wird der `autoDispose` nicht ausgeführt.

Riverpod implementieren

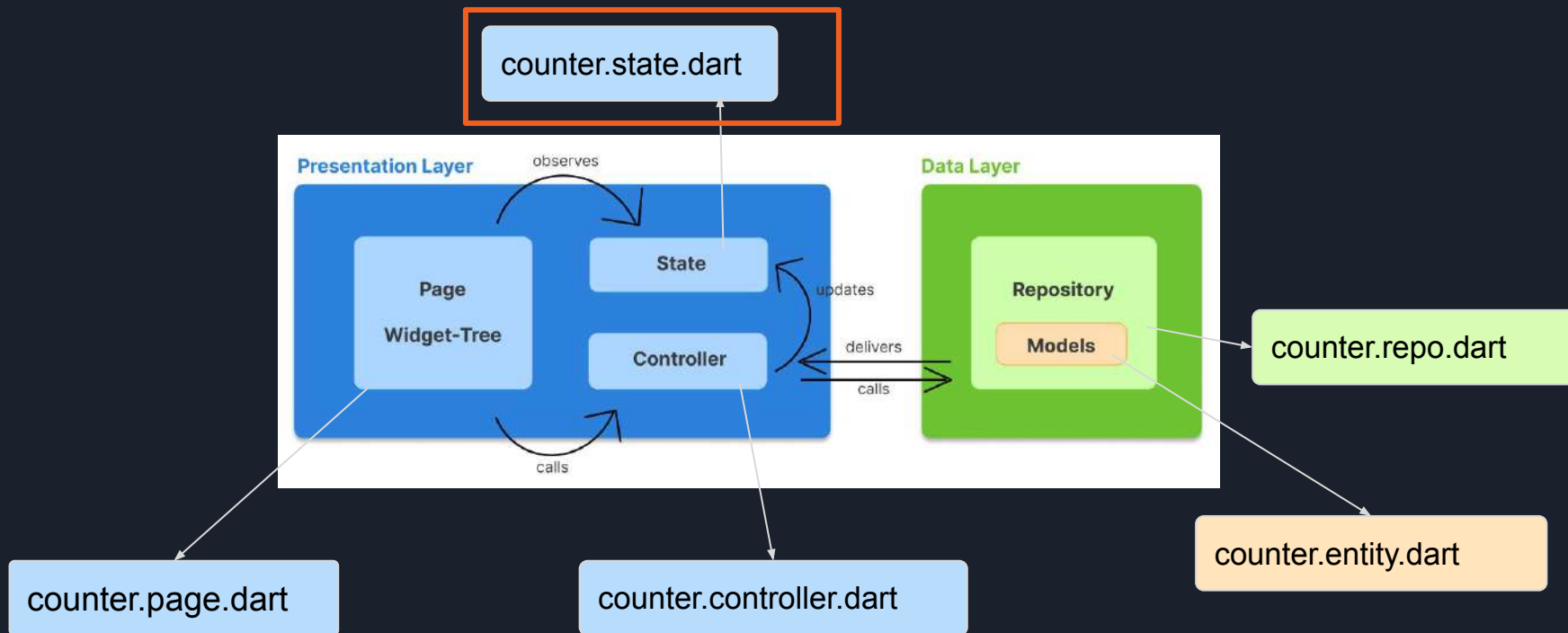
Aufgabe:

- Führe Riverpod in das Projekt ein
- Ersetze im Repository den Provider mit Riverpod
- Ersetze in den Controllern Provider mit Riverpod
- Führe Riverpod in der Counter Seite ein.



6.1 Riverpod StateNotifierProvider

Vollständige Presentation State Architektur




Immutable State Klasse

```
import 'package:flutter_architektur_workshop/src/features/counter/domain/counter.entity.dart';
import 'package:freezed_annotation/freezed_annotation.dart';

part 'counter.state.freezed.dart';

@freezed
class CounterState with _$CounterState {
  const factory CounterState({
    @Default(Counter(counter: 0)) Counter counter,
  }) = _CounterState;
}
```



Hält das gesamte Objekt.
Auch einzelne Attribute sind möglich.

CounterController als StateNotifierProvider

Konfiguration Controller & State Objekt

```
final counterControllerProvider = StateNotifierProvider.autoDispose<CounterController, CounterState>((ref) {  
  return CounterController(const CounterState(), ref.read);  
});
```

```
class CounterController extends StateNotifier<CounterState> {  
  CounterController(CounterState state, this.read) : super(state) {  
    Counter _counter = read(counterRepoProvider).counter;  
    state = state.copyWith(counter: _counter);  
  }  
}
```

Initialisierung des States.
State ist immutable -> copyWith()

CounterController als StateNotifierProvider

```
class MyHomePage extends ConsumerWidget {  
  ...  
  Widget build(BuildContext context, WidgetRef ref) {  
    CounterState counterState = ref.watch(counterControllerProvider);  
    int counterValue = counterState.counter.counter;  
    return Scaffold(  
      body: Center(  
        child: Column(  
          mainAxisAlignment: MainAxisAlignment.center,  
          children: <Widget>[  
            ...  
            Text(  
              '$counterValue',  
              style: Theme.of(context).textTheme.headline4,  
            ),  
          ],  
        ),  
      ),  
    );  
  }  
}
```

State-Objekt watchen

Provider liefert den immutable State aus

Wert ausgeben

Es wird alles in der build-Methode neu gerendert.

Bei größeren build-Methoden das einzelne Widget als neues Consumer Widget auslagern.

Riverpod Erweiterung um Controller State Klasse

Aufgabe:

- Implementiere State als Freezed Klassen für Counter und Dashboard
- Führe die jeweiligen States in den jeweiligen Controllern ein
- Verwende den primitiven Counter Typ als State



Navigation



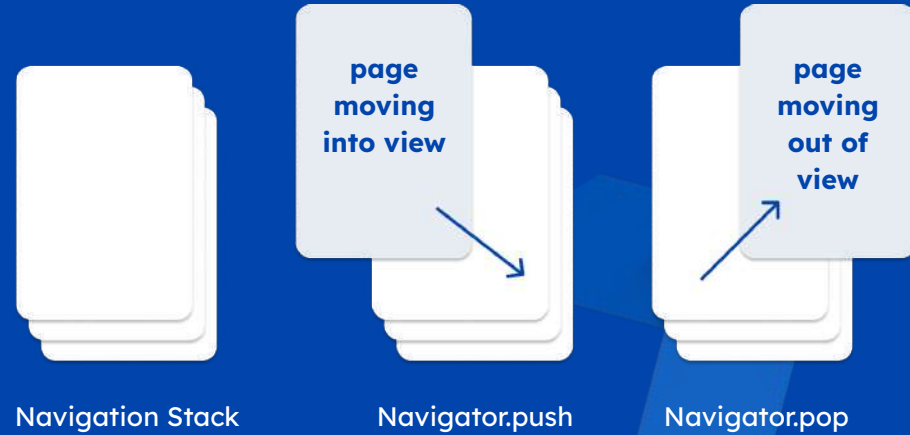
Routing 1.0

Navigation Stack

- Konzept des Stacks in Flutter implementiert
- LIFO (Last In First Out)
- Über Navigator API

Zwei weitere existierende Konzepte:

- **Deep Linking**
Öffnet über eine URL direkt eine Seite (für Android, iOS und Web verfügbar)
- **URL Strategien**
Für Flutter Web Hash- und Path-Strategie verfügbar



ListView

Push neue Page

State Objekte
bleiben vorhanden

ListView

DetailView

Vorteile

- einfache API
 - push und pop
- anonyme Routen sind möglich
- Routing mit Named routes vereinfacht die Handhabung
 - Zentrales Register aller Routen
 - besserer Weg über
 - `onGenerateRoute`

Nachteile

- kein guter Weg, um die Seiten zentral zu verwalten
- keine Möglichkeit, um auf den Route Stack zuzugreifen
- Updated nicht den Web URL Pfad
- es ist möglich, dass der Back Button unter Android nicht funktioniert

The background of the slide is a solid blue color. Scattered across this background are several abstract, three-dimensional geometric shapes. These shapes are composed of flat, triangular and quadrilateral faces, giving them a faceted, crystalline appearance. They are rendered in a lighter shade of blue than the background, with some internal lines suggesting depth and structure. The shapes are positioned in the upper half of the slide, with one near the top left, another near the top right, and a few smaller ones towards the bottom left and bottom right.

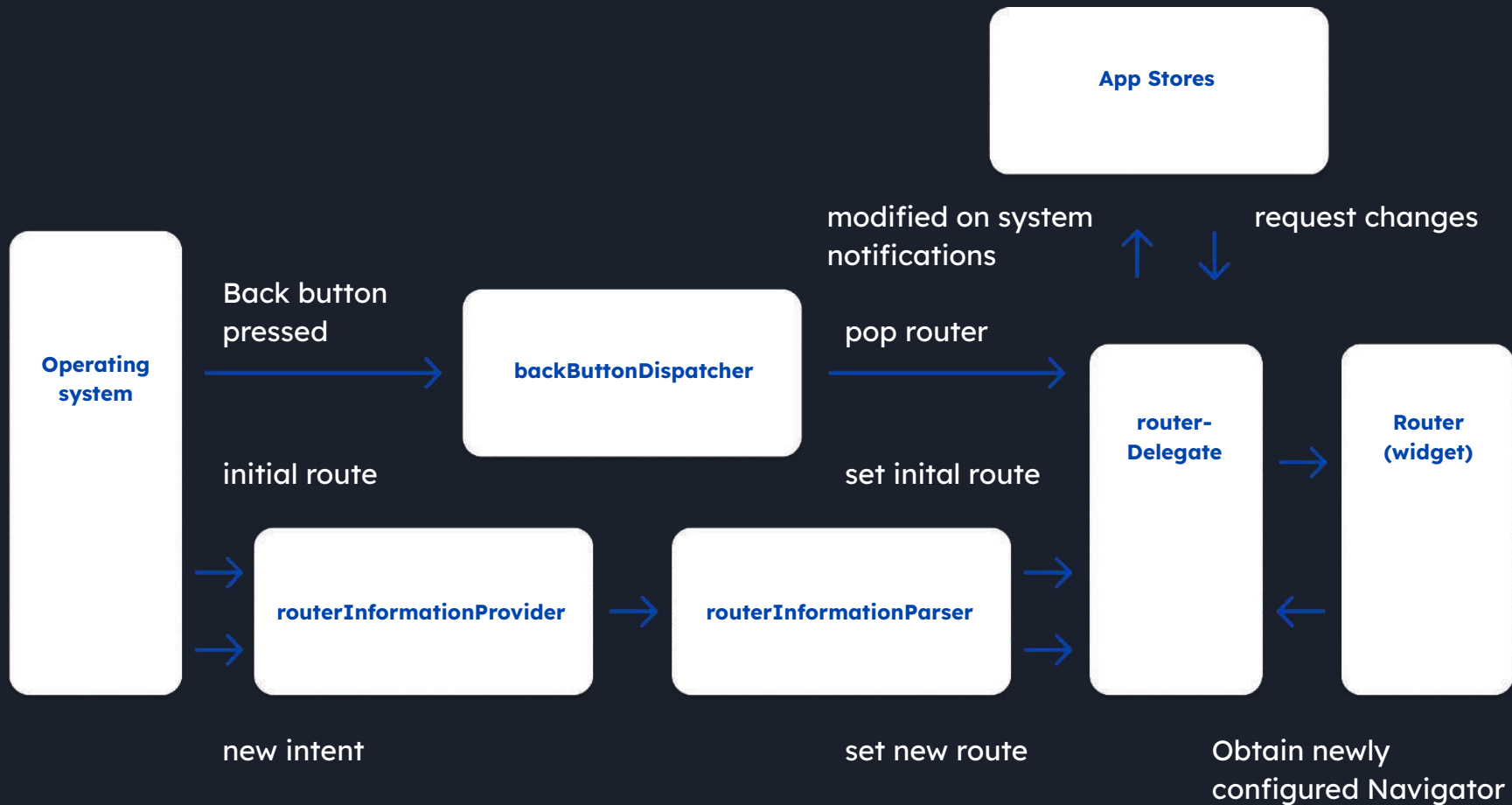
Routing 2.0

Vorteile

- Kontrolle über den Routing Stack
- Managen von verschachtelten Routen
- Managen von aktuellen Routen und deep linking
- weiterhin möglich: Nutzung der Routing 1.0 API
- bessere Integration in System Events

Nachteile

- komplex im Aufbau



Bestandteile zur Nutzung von Routing 2.0

- **Page** beschreibt die Konfiguration einer Seite
- **RouterDelegate** beinhaltet das Navigationsverhalten anhand der Konfiguration
- **RouteInformationProvider** übergibt RouteInformation an den Router
- **RouteInformationParser** parsed Parameter aus der URL, setzt die URL im Browser
- **BackButtonDispatcher** beschreibt das Verhalten des Back Buttons
- **TransitionDelegate** beschreibt die Seitenübergänge

Ablauf

Neue Route
wird übermittelt

RouterDelegate
wird aufgerufen

Aufruf von
NotifyListeners um
Router neu zu bauen

neuer Navigator mit
den Änderungen an
den Seiten verfügbar

detail/2



**update
App state**



**rebuild
RouterDe
legate**



**new
Navigator**

Zusammenfassung

- Routing 1.0 nutzen wenn:
 - reine Mobile App
 - einfacher Navigationsbaum
- Routing 2.0 nutzen wenn:
 - neben Mobile auch Web aus der gleichen Code Basis entstehen soll
 - Komplexer Navigationsbaum mit verschachtelten Navigations-Pfaden
 - Routen anhand von bestimmten Zuständen abzusichern sind
 - Kontrolle über den Navigations Stack benötigt wird
 - Best Practice hier:
 - Nutzung von bestehenden Packages -> `go_router`





Routing mit dem Flutter Favorite go_router

go_router

- go_router wurde als Erleichterung für die Handhabung vom Routing 2.0 geschaffen
- einfache API, trotz mächtigen Router 2.0
- Zentrales Verwalten der Routen
- Als Flutter Favorite aus verschiedenen packages ausgewählt worden



TOP30 Navigation Issue nr. 6

LIKE based ranking of navigation packages

Likes and position **June 5, 2022**
in **pub.dev** of **all** packages.
(Changes are from issue nr 5, Dec 17, 2021)

Included info:

- NS = Has Null Safety version
 - Test CodeCov % when available
 - ApiDoc completeness %
 - GitHub stars
 - GitHub Issues Open/Closed
 - Navigation API Level
- R = API-2 Uses MaterialApp.router
P = API-2 Uses MaterialApp with Pages
C = API-1 Uses Classic Navigator

Test% API docs% [90...100] Points [130]	NAV2 Router
Test% API docs% [80...90] Points [120...125]	Null safe
Test% API docs% [60...80] Points [100...115]	NAV2 Pages
Test% API docs% [0...60] Points [0...90]	NAV1 Classic
Not null safe Last update > 1 year	

Stats summary by @RydMike (Mike Rydstrom)

Navigation Package	NAV API	Package author	Rank	Likes	Version	Updated	NS	CodeCov	API docs	Points	Popularity	GitHub ★	Likes Stars	Open Closed	Position
auto_route	R	Milad-Akari	1	1612	+418	4.0.1 12.05.2022	Yes	Not given	13.9%	110	99%	+1	978	1.65 265/683	64 +8
go_router (Flutter Favorite)	R	flutter.dev	2	+2 1030	+691	3.1.1 27.05.2022	Yes	Not given	98.6%	120	98%	+4	N/A	N/A 48/not given	105 +135
beamer (Flutter Favorite)	R	beamer.dev slowicki	3	727	+256	1.4.1+1 13.04.2022	Yes	96.4%	96.5%	130	96%	+2	404	1.80 49/381	165 +36
fluro	C	lukepiggett.com	4	-2 686	+84	2.0.3 01.08.2021	Yes	Not given	26.9%	120	98%		3472	0.20 21/171	181 -21
routermaster (Flutter Favorite)	R	tom.gilder.dev tomgilder	5	374	+140	1.0.0 24.04.2022	Yes	98.9%	94.5%	110	95%	+3	259	1.44 55/109	326 +79
vrouter	R	vrouter.dev lulupointe	6	263	+58	1.2.2 12.05.2022	Yes	Not given	77.8%	120	94%		180	1.46 16/169	453 +5
flow_builder	P	felangel	7	255	+60	0.0.9 31.05.2022	Yes	100.0%	90.3%	130	96%	+2	294	0.87 17/26	465 +21
sailor	C	guriensethi	8	126	+5	0.7.1 10.04.2020	No	Not given	38.2%	100	85%	-2	146	0.86 29/14	867 -144
yeet	R	ident.app HosseinVosefi	9	80	+12	0.4.10 21.04.2022	Yes	71.8%	44.9%	120	74%	+9	39	2.05 8/12	1249 -64
glevar_router	R	glevar.de SchabanBo	10	76	+27	1.5.11 24.03.2022	Yes	77.8%	52.9%	130	86%	+1	47	1.62 5/46	1304 +205
navigator	C	nubank.dev nubank	11	43	+8	1.6.2 05.05.2022	No	Not given	fail	80	92%		220	0.20 1/15	2109 -171
riverpod_navigation	R	alioisdeniel	12	28	+1	0.2.1-pre 21.06.2021	Yes	Not given	18.0%	90	48%	-14	24	1.17 5/0	2748 -427
deep_link_navigation	C	Dennis-Krasnov	13	+2 25	2	1.3.1 30.12.2019	No	Not given	67.4%	70	21%	+19	61	0.41 2/0	3004 -360
flutter_deep_linking	C	wanke.dev JonasWanke	14	24	+1	0.2.1 21.04.2021	Yes	Not given	16.7%	100	80%	+1	8	3.00 0/0	3057 -441
voyager	R	vishma.dev vishma	15	-2 24		3.0.0 04.03.2021	Yes	100.0%	95.0%	110	62%	+15	63	0.38 3/50	3065 -524
i_navigation	C	kamalsayed98	16	21		1.0.1 26.06.2021	No	Not given	60.5%	110	0%	-11	0	inf 1/0	3322 -549
page_router	R	johnpryan	17	18	-1	0.0.3 25.02.2021	No	Not given	17.5%	50	40%	+12	51	0.35 3/1	3716 -741
navi	R	zenonline.com zenonline	18	16	+2	0.2.2 03.05.2022	Yes	90.7%	19.7%	100	53%		19	0.84 3/6	3988 -427
flouter	R	kreak.dev kreak	19	15	1	0.2.0 04.03.2021	Yes	Not given	64.9%	110	51%	+25	41	0.37 5/3	4190 -570
navme	R	zfx.com	20	12	-1	1.0.1 24.06.2021	Yes	Not given	89.8%	130	46%		2	6.00 0/0	4732 -996
aps_navigator	R	guthiermex	21	+1 12	+1	0.0.3-dev.1 03.05.2021	Yes	91.0%	60.9%	120	37%	+7	13	0.92 1/0	4748 -597
nav_router	C	fluttercandies	22	-1 12		0.0.7 31.07.2020	No	Not given	43.2%	100	72%	-2	107	0.11 3/1	4792 -789
wouter	R	shovelmm12	23	new 11	+3	0.1.8 13.05.2022	Yes	Not given	25.7%	90	31%		19	0.58 4/3	4849
nav_stack	R	gskinnerTeam	24	-1 11	+3	0.0.3+3 16.04.2021	Yes	Not given	39.6%	110	48%		19	0.58 4/3	4967 -8
advanced_navigator	P	LucasAchenbach	25	-3 10	+2	0.2.0+3 14.05.2022	Yes	Not given	38.9%	120	52%	+48	8	1.25 0/3	5097 -302
pragmatic_navigation	R	mucoc.it	26	7		0.1.2 07.03.2022	Yes	Not given	83.9%	120	22%	-28	12	0.58 0/0	6214 -909
nav	C	BansookNam	27	-2 6	+1	1.1.3 31.05.2022	Yes	Not given	28.8%	120	53%	+1	2	3.00 0/0	6592 -1346
routeborn	R	KristianBalaj	28	-1 6	+1	0.4.2 11.03.2022	Yes	81.9%	37.2%	110	47%		5	1.20 0/0	6701 -510
theseus_navigator	R	echedev	29	-5 6	-1	0.0.14 06.03.2022	Yes	63.0%	96.9%	120	38%	+12	4	1.50 0/3	6706 -1587
implicit_navigator	P	caseycrogers.dev	30	-1 6	+2	1.0.3 03.02.2022	Yes	Not given	82.1%	110	0%		1	6.00 0/0	6746 +175
Includes Navigation	NAV API	Package author	Rank	Likes	Version	Updated	NS	CodeCov	API docs	Points	Popularity	GitHub ★	Likes Stars	Open Closed	Position
get (GetX)	R	getx.site jonataslaw	1	9154	+1871	4.6.5 22.05.2022	Yes	Not given	30.8%	130	100%		6767	1.35 555/1181	1
velocity_x	R	mtechrival.com lampawan	2	951	+154	3.5.1 28.05.2022	Yes	Not given	31.9%	130	97%		1037	0.92 5/93	114 -5
flutter_modular	R	flutterando.com.br	3	903	+119	5.0.3 03.06.2022	Yes	100.0%	38.8%	130	98%	+1	1071	0.84 15/471	123 -15

Eintragen der Lib in die pubspec.yaml

```
dependencies:  
  flutter:  
    sdk: flutter  
  freezed_annotation: ^2.0.3  
  provider: ^6.0.3  
  flutter_riverpod: ^1.0.3  
  go_router: ^3.1.1  
  cupertino_icons: ^1.0.2
```

```
dev_dependencies:  
  flutter_test:  
    sdk: flutter  
  freezed: ^2.0.3+1  
  build_runner: ^2.1.11
```

GoRouter Setup

- MaterialApp wird mit MaterialApp.router ergänzt
- routeInformationParser und routerDelegate aus GoRouter in MaterialApp.router übergeben

```
Widget build(BuildContext context, _) {  
    final GoRouter _router =  
    ref.read(routerProvider);  
    return MaterialApp.router(  
        title: 'Flutter Demo',  
        theme: ThemeData(  
            primarySwatch: Colors.blue,  
        ),  
        routeInformationParser:  
        _router.routeInformationParser,  
        routerDelegate: _router.routerDelegate,  
    );  
}
```



GoRouter Setup


- Erstellen einer router Klasse
`router.dart`
- **Definiere** alle Seiten mit
`GoRoute` unter `routes`
 - Hinweis: `'/'` definiert die
Startseite
 - Diese kann mit
`initialLocation`
manipuliert werden

```
final routerProvider = Provider<GoRouter>((ref) {  
  return GoRouter(  
    routes: [  
      GoRoute(path: '/', redirect: (_) => '/home',),  
      GoRoute(  
        path: '/home',  
        builder: (_, __) =>  
          MyHomePage(title: 'Flutter Demo Home Page'),  
      ),  
      GoRoute(  
        path: '/dashboard',  
        builder: (_, __) => const Dashboard(),  
      ),  
    ],  
    initialLocation: '/dashboard'  
  );  
});
```

Nested Routes

- **Nested Routes** werden innerhalb der route als subrouten deklariert
 - Haben automatisch einen Back Button

```
final router = GoRouter(  
  routes: [  
    GoRoute(path: '/', redirect: (_) => '/home',),  
    GoRoute(  
      path: '/',  
      builder: (_, __) =>  
        MyHomePage(title: 'Flutter Demo Home Page'),  
      routes: [  
        GoRoute(  
          path: ':id',  
          builder: (_, GoRouterState s) => MyHomePage(  
            title: 'Flutter Demo Home Page with ID ',  
            id: int.tryParse(s.params['id']!),  
          ),  
        ),  
      ],  
    ),  
    GoRoute(  
      path: '/dashboard',  
      builder: (_, __) => const Dashboard(),  
    ),  
  ],  
);
```



Parameter

- **Pfad Parameter** können in der route definiert werden
- **Query Parameter** sind optional über den **GoRouterState** können diese ausgelesen werden
`state.queryParams['id']`
- **Extra Parameter** können ein eigenes Objekt entgegen nehmen
`state.extra as Counter`

```
final router = GoRouter(  
  routes: [  
    GoRoute(path: '/', redirect: (_) => '/home',),  
    GoRoute(  
      path: '/',  
      builder: (_, __) =>  
        MyHomePage(title: 'Flutter Demo Home Page'),  
      routes: [  
        GoRoute(  
          path: ':id',  
          builder: (_, GoRouterState s) => MyHomePage(  
            title: 'Flutter Demo Home Page with ID ',  
            id: int.tryParse(s.params['id']!),  
          ),  
        ],  
      ),  
    GoRoute(  
      path: '/dashboard',  
      builder: (_, __) => const Dashboard(),  
    ),  
  ],  
);
```


GoRouter nutzen

- Importiere `go_router`
- Nutze `context.go` oder `context.push` um zur entsprechenden Seite zu navigieren
- Beim `push` werden die Seiten **gestapelt**, wodurch eine Navigation zur **vorherigen Seite** möglich ist.

```
ElevatedButton(  
  onPressed: () => Navigator.push(  
    context,  
    MaterialPageRoute(  
      builder: (build) => const Dashboard(),  
    ), // MaterialPageRoute  
  ),  
  child: const Text('Dashboard'),  
) // ElevatedButton
```


```
ElevatedButton(  
  onPressed: () =>  
    context.go('/dashboard'),  
  child: const Text('Dashboard'),  
)
```

```
ElevatedButton(  
  onPressed: () =>  
    context.push('/dashboard'),  
  child: const Text('Dashboard'),  
)
```

Navigieren mit Parametern

- Query Parameter mit `?name=value` direkt an die Route
- Path-Parameter mit `/` anhängen oder
- als `params` Map bei `go / push - Named` übergeben
- extra: Ganze Objekte an eine andere Route übergeben

```
ElevatedButton(  
  onPressed: () =>  
    context.go('/dashboard?query=1'),  
  child: const Text('Dashboard'),  
)
```



```
ElevatedButton(  
  onPressed: () =>  
    context.go('/dashboard/1'),  
  child: const Text('Dashboard'),  
)
```



```
ElevatedButton(  
  onPressed: () =>  
    onPressed: () =>  
      context.goNamed('home_id',  
        params: {'id': '1'}),  
)
```



```
ElevatedButton(  
  onPressed: () =>  
    context.go('/dashboard', extra: <MyObject>),  
  child: const Text('Dashboard'),  
)
```



initialen Wert setzen

- methoden zum setzen des initialen wertes im Repo und CounterController hinzufügen
- Im Controller das setzten mit einem Future.delayed wrappen
- im **builder** der Route den Wert setzen

```
class CounterRepo {  
  Counter setInitialCounterValue(int initialValue) {  
    return _counter.copyWith(counter: initialValue);  
  }  
}
```

```
Future<void> initialValue(int initialValue) async {  
  Future.delayed(Duration.zero, () {  
    state = state.copyWith(  
      counter:  
        read(counterRepoProvider).setInitialCounterValue(initialValue));  
  });  
}
```

```
GoRoute(  
  name: 'home_id',  
  path: ':id',  
  builder: (_, GoRouterState state) {  
    int? value =  
      int.tryParse(state.params['id']!);  
    ref.read(counterControllerProvider.notifier)  
      .initialValue(value ?? 0);  
  },  
)
```

GoRouter implementieren

Aufgabe:

- Führe GoRouter in das Projekt ein
- Ersetze alle bestehenden Navigationen mit GoRouter
- Erstelle eine Nested Route, die einen initialen Wert für den Counter setzt

go_router advanced

The background of the slide is a solid blue color. It is decorated with several 3D geometric shapes, specifically rectangular prisms or cubes, that are tilted at various angles. These shapes are rendered in a lighter shade of blue, creating a sense of depth and modern design. They are scattered across the upper half of the slide.

go_router advanced

guards, states mit riverpod, transitions, error
Page

GoRouter als Riverpod Provider

```
final routerProvider = Provider<GoRouter>((ref) {
```

← Umbau auf Provider

```
  return GoRouter(  
    routes: [  
      GoRoute(  
        name: 'login',  
        path: '/login',  
        builder: (_, __) => const LoginPage(),  
      ),  
      ...  
    ],  
  );  
}
```

Mit Riverpod verbinden

```
Widget build(BuildContext context, WidgetRef ref) {  
  final GoRouter _router = ref.read(routerProvider);  
  return MaterialApp.router(  
    title: 'Flutter Demo',  
    theme: ThemeData(  
      primarySwatch: Colors.blue,  
    ),  
    routeInformationParser: _router.routeInformationParser,  
    routerDelegate: _router.routerDelegate,  
  );  
}
```

- Alle Daten werden über Riverpod Dependency Injection geladen
- Router wird zu einem Riverpod Provider

Routing Redirects mit State

- Absichern von Routen
 - Login
 - Rechtevergabe
 - Routing anhand von State Veränderungen



Routing Redirects mit State

```
final routerProvider = Provider<GoRouter>((ref) {  
  return GoRouter(  
    refreshListenable: RouterNotifier(ref),  
    redirect: (state) {  
      final user = ref.read(loginControllerProvider);  
      final areWeLoggingIn = state.location == '/login';  
      if (user == null) {  
        return areWeLoggingIn ? null : '/login';  
      }  
      if (areWeLoggingIn) return '/';  
      return null;  
    },  
    routes: [  
      GoRoute(  
        name: 'login',  
        path: '/login',  
        builder: (_, __) => const LoginPage(),  
      ),  
      ...  
    ],  
  );  
});
```

Ermöglicht das Reagieren auf State Changes - Für Riverpod unumgänglich

Hier redirects anhand eigener Logik definieren


RouterNotifier implementieren

- Wenn sich etwas an der UserEntity ändert, reagiert der Router.
- So kann man bei Änderungen von State-Objekten automatisch routen.

Typische Anwendungsfälle:

- Authentifizierung des Users
- Alerting

```
final routerProvider = Provider<GoRouter>((ref) {  
  return GoRouter(  
    refreshListenable: RouterNotifier(ref),  
    ...  
  });  
}
```



```
class RouterNotifier extends ChangeNotifier {  
  final Ref _ref;  
  RouterNotifier(this._ref) {  
    _ref.listen<UserEntity?>(  
      loginControllerProvider,  
      (_, __) => notifyListeners(),  
    );  
  }  
}
```

LoginController

- Übernimmt den Login und Logout eines Users
- Bei Änderungen am State wird die redirect Klasse aufgerufen

```
class LoginController extends StateNotifier<UserEntity?> {  
    LoginController() : super(null);  
  
    Future<void> login(String email, String password) async {  
        state = const UserEntity(  
            name: 'My Name',  
            email: 'My Email',  
        );  
    }  
  
    Future<void> logout() async {  
        state = null;  
    }  
}  
  
final loginControllerProvider =  
    StateNotifierProvider<LoginController,  
    UserEntity?>((ref) {  
        return LoginController();  
    }));
```

LoginPage

- Zeigt ein Loginformular
- Ruft die Login Methode auf

```
class LoginPage extends ConsumerWidget {  
  const LoginPage({Key? key}) : super(key: key);  
  
  @override  
  Widget build(BuildContext context, WidgetRef ref) {  
    return Scaffold(  
      appBar: null,  
      body: Center(  
        child: Column(  
          ...  
          ElevatedButton(  
            onPressed: () {  
  
              ref.read(loginControllerProvider.notifier).login(  
                "myEmail",  
                "myPassword",  
              );  
            },  
            child: const Text("Login"),  
          ),  
        ),  
      );  
    }  
  }  
}
```

GoRouter Advanced implementieren

Aufgabe:

- Erstelle eine neue Page 'Login' mit Login Button
- Erstelle die Logik zum Setzen eines User States
- Erstelle den Redirect mit Prüfung auf den User State
- Leite auf das Dashboard weiter





go_router

404 Error Page

- zentrales Einrichten einer 404 Seite

```
final routerProvider = Provider<GoRouter>((ref) {  
  return GoRouter(  
    // Wrapper um Listenable zum Notifien des state changes  
    refreshListenable: RouterNotifier(ref),  
    routes: [  
      GoRoute(  
        name: 'login',  
        path: '/login',  
        builder: (_, __) => const LoginPage(),  
      ),  
      1.  
      errorBuilder: (context, state) => ErrorPage(text:  
        state.error.toString()),  
    );  
  }]);
```


Eigene Transitions

- Transitions erben von CustomTransitionPage
- im pageBuilder der Route wird die Transition ausgeführt

```
class FadeTransitionPage extends CustomTransitionPage<void> {  
  FadeTransitionPage({  
    required LocalKey key,  
    required Widget child,  
  }) : super(  
    key: key,  
    transitionsBuilder:  
      (context, animation, secondaryAnimation, child) {  
        const begin = Offset(0.0, -1.0);  
        const end = Offset(0.0, 0.0);  
        return SlideTransition(  
          position: animation.drive(Tween(begin: begin, end: end)),  
          child: child,  
        );  
      },  
    child: child);  
}  
  
GoRoute(  
  name: 'home',  
  path: '/home',  
  pageBuilder: (context, state) => FadeTransitionPage(  
    key: const ValueKey<String>('login'),  
    child: MyHomePage(title: 'Flutter Demo Home Page'),  
  ),  
)
```

Wichtigste Punkte

- Nicht jedes Projekt hat die gleiche Architektur
- Im Team eine **einheitliche Projektsprache** sprechen
- Das Team auf **Konventionen** und Strukturen trainieren
- **State Architektur** einführen und Team schulen
- Je größer das Projekt wird, umso strikter die Architektur anwenden



Vielen Dank



@makueh



markus-kuehle



@marcel.ploch

<https://coodoo.de>

<https://flutter.de>

