

Flutter

Saban Ünlü

Zwei Worte zu mir

Saban Ünlü

- Software Architekt und Programmierer
- Berater und Dozent seit 2000
- Autor
- Adobe Influencer
- LinkedIn IoT & Google Expert
- Gründer von netTrek



Was wir behandeln werden

- Einführung
 - Technologie Stack
 - Über Flutter
 - Setup
- Dart
 - Was ist Dart?
 - Syntax
 - Datentypen
 - Objektorientierung
 - CRUD



Was wir behandeln werden

- Flutter
 - Funktionsweise
 - "Everything is a widget"
 - Widgets nutzen
 - StatelessWidget Widgets erstellen
 - Styling von Widgets
 - StatefulWidget Widgets erstellen und welche Alternativen es gibt
 - Gestenerkennung
 - Animation
 - Bloc
 - Navigation und Routing



Einführung

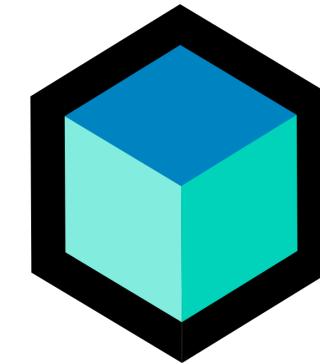
Technologie Stack – Übersicht



android



git



bloc

Technologie Stack – Dart



- Programmiersprache nach ECMA Standard
- Programmiersprache für Clientseitige Entwicklung
- Basierend auf mehreren Paradigmen: Funktionale, Imperative, Objektorientierte und Reflektive Entwicklung
- Garbage-collected
- C-style Syntax
- Entwickelt durch Google
- Umfassende Entwickertools

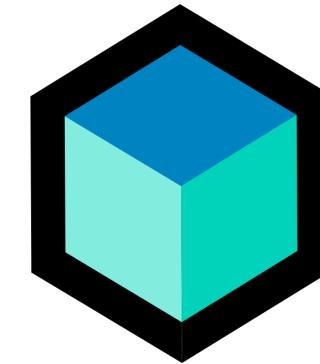
Technologie Stack – Übersicht



android



git



bloc

Technologie Stack – Flutter



- Dart UI-Framework
- Von Google
- Schnelles Aufstreben
- Rendering mit Skia engine
- Hohe Leistung durch erneutes verwenden von Elementen
- => Schnelleres Rendering bei Animationen als native Bibliotheken
- "Everything is a Widget"

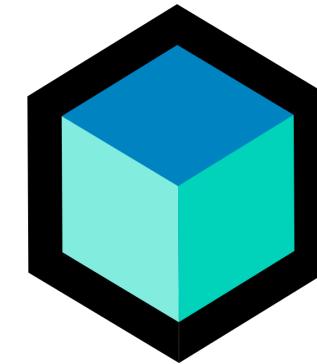
Technologie Stack – Übersicht



android

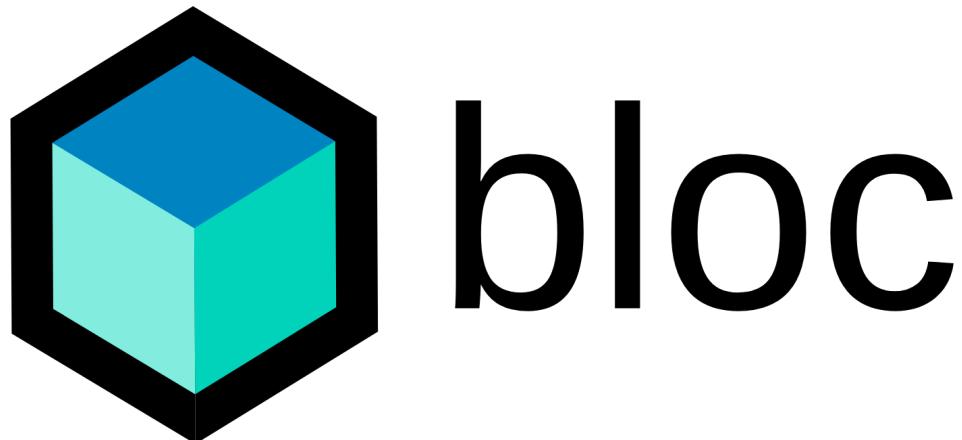


git



bloc

Technologie Stack – BLoC Pattern



- Kurz für Business Logic Controller
- State Management in Flutter
- Nutzt Streams
- Allgemeines Design Pattern, in Flutter umgesetzt mit flutter_bloc

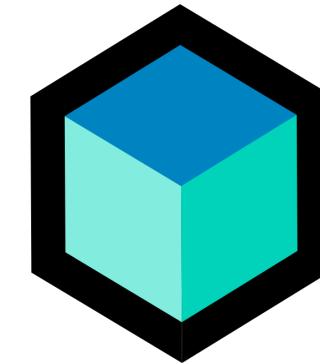
Technologie Stack – Übersicht



android



git



bloc

Technologie Stack – Git



- Versionierungssystem für Software
- GitHub – Filehoster
- Ermöglicht, unterschiedliche Zustände einer Software zu verwalten
- Optimiert Teamwork

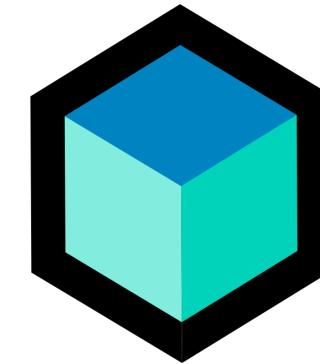
Technologie Stack – Übersicht



android

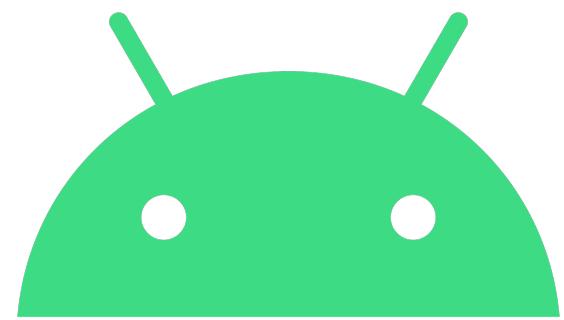


git



bloc

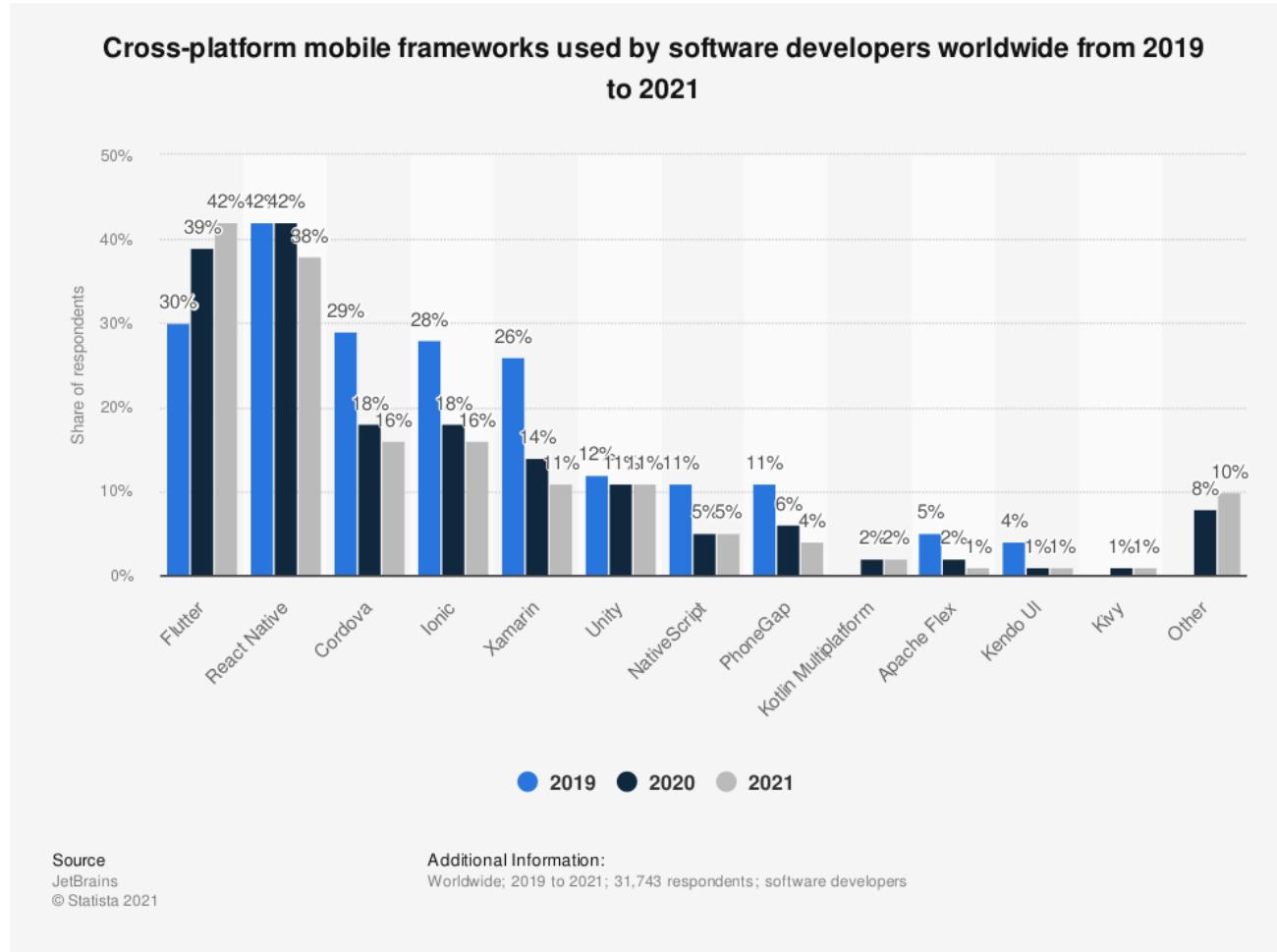
Technologie Stack – Native Technologien



android



Wieso Flutter



<https://www.statista.com/statistics/869224/worldwide-software-developer-working-hours/>

Wieso Flutter – Grobe Unterteilung

Webtechnologien

- Einfache HTML/CSS Seiten innerhalb einer nativen WebView
- Brücke zwischen der nativen Schicht und der WebView

=> Cordova, Ionic,

Web/Hybrid

- Logik über die JavaScript V8 Engine
- Darstellung über native Renderingschichten mit nativen Komponenten

=> ReactNative, NativeScript, Xamarin

Nativ/Hybrid

- Flutter (Google) via SKIA
- Darstellung über eigene Komponenten, die in nativer Performance gerendert werden
- Eigene Programmiersprachen
- Nativer Anwendungscode

=> Flutter, Adobe AIR, ..

Wieso Flutter – Vor- & Nachteile

Vorteile

- AOT kompiliert
 - Ahead-of-time (AOT) compilation
- Hot Reload
 - Über Just-in-time compilation (JIT)
- Einfache Umsetzung eigener Komponenten
- Hohe Performance, teilweise besser als Native Performance
- Detaillierte Dokumentation
- Große Community

Nachteile

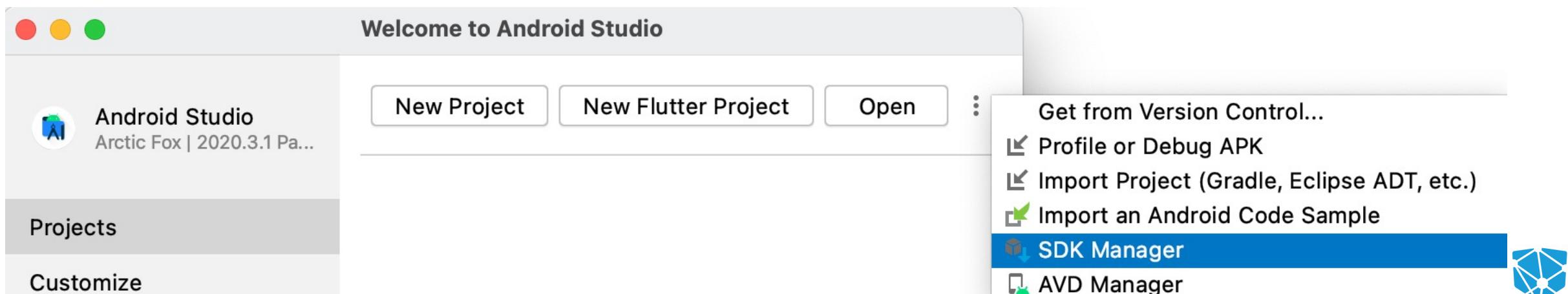
- Plattformspezifische Designs werden simuliert
 - Nicht immer identisch
- Dart nicht prominent
 - Im OOP Kontext leicht zu lernen

Setup – Was wir brauchen

- Android Studio
- XCODE
 - Unter Mac um zusätzlich iOS Anwendungen zu erzeugen
- Flutter SDK
- Flutter Plugins in Android Studio

Setup – Android Studio

- Herunterladen von <https://developer.android.com/studio>
- Ausführen von EXE auf Windows oder DMG auf Apple Geräten
 - Oder über JetBrains Toolbox
- Installation durch den Wizard
 - Android SDK & SDK Command-line Tools installieren



Setup – SDK & SDK Command-line Tools

SDK Platforms SDK Tools SDK Update Sites

Each Android SDK Platform package includes the Android platform and sources pertaining to an API level by default. Once installed, the IDE will automatically check for updates. Check "show package details" to display individual SDK components.

Name	API Level	Revision	Status
<input type="checkbox"/> Android Sv2 Preview			
<input checked="" type="checkbox"/> Android 12.0 (S)			
<input type="checkbox"/> Android 11.0 (R)			
<input type="checkbox"/> Android 10.0 (Q)			
<input type="checkbox"/> Android 9.0 (Pie)			
<input type="checkbox"/> Android 8.1 (Oreo)			

SDK Platforms SDK Tools SDK Update Sites

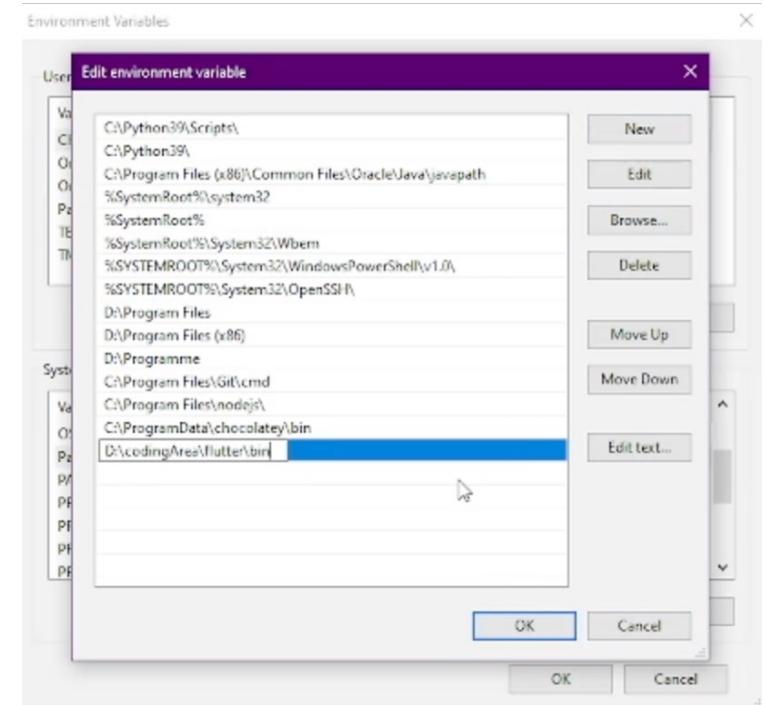
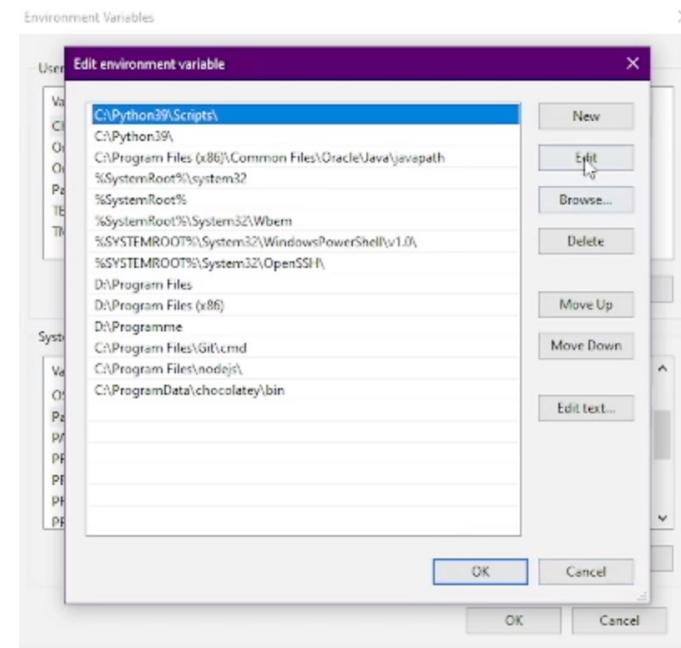
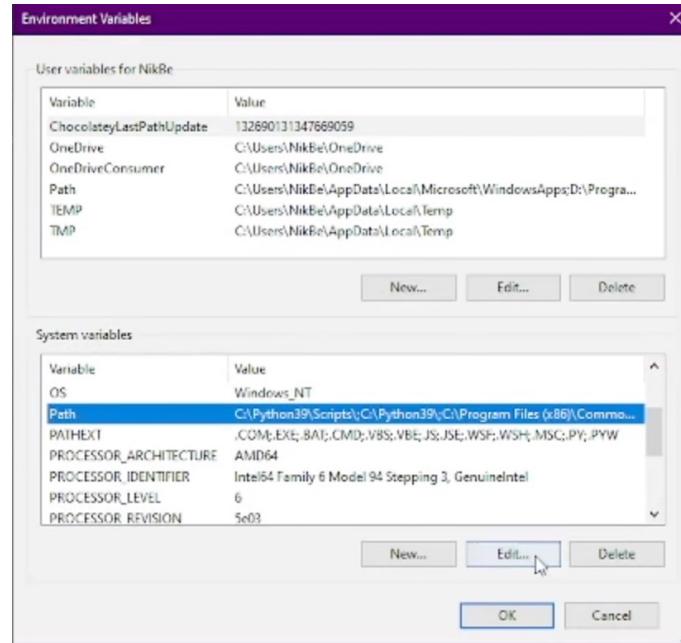
Below are the available SDK developer tools. Once installed, the IDE will automatically check for updates. Check "show package details" to display available versions of an SDK Tool.

Name	Version	Status
<input type="checkbox"/> - Android SDK Build-Tools 32-rc1		Update Available: 32.0.0 rc1
<input type="checkbox"/> NDK (Side by side)		Not Installed
<input checked="" type="checkbox"/> Android SDK Command-line Tools (latest)		Installed
<input type="checkbox"/> CMake		Not Installed
<input type="checkbox"/> Android Auto API Simulators	1	Not installed
<input type="checkbox"/> Android Auto Desktop Head Unit Emulator	1.1	Not installed
<input checked="" type="checkbox"/> Android Emulator	30.8.4	Installed
<input checked="" type="checkbox"/> Android SDK Platform-Tools	31.0.3	Installed
<input type="checkbox"/> Google Play APK Expansion library	1	Not installed

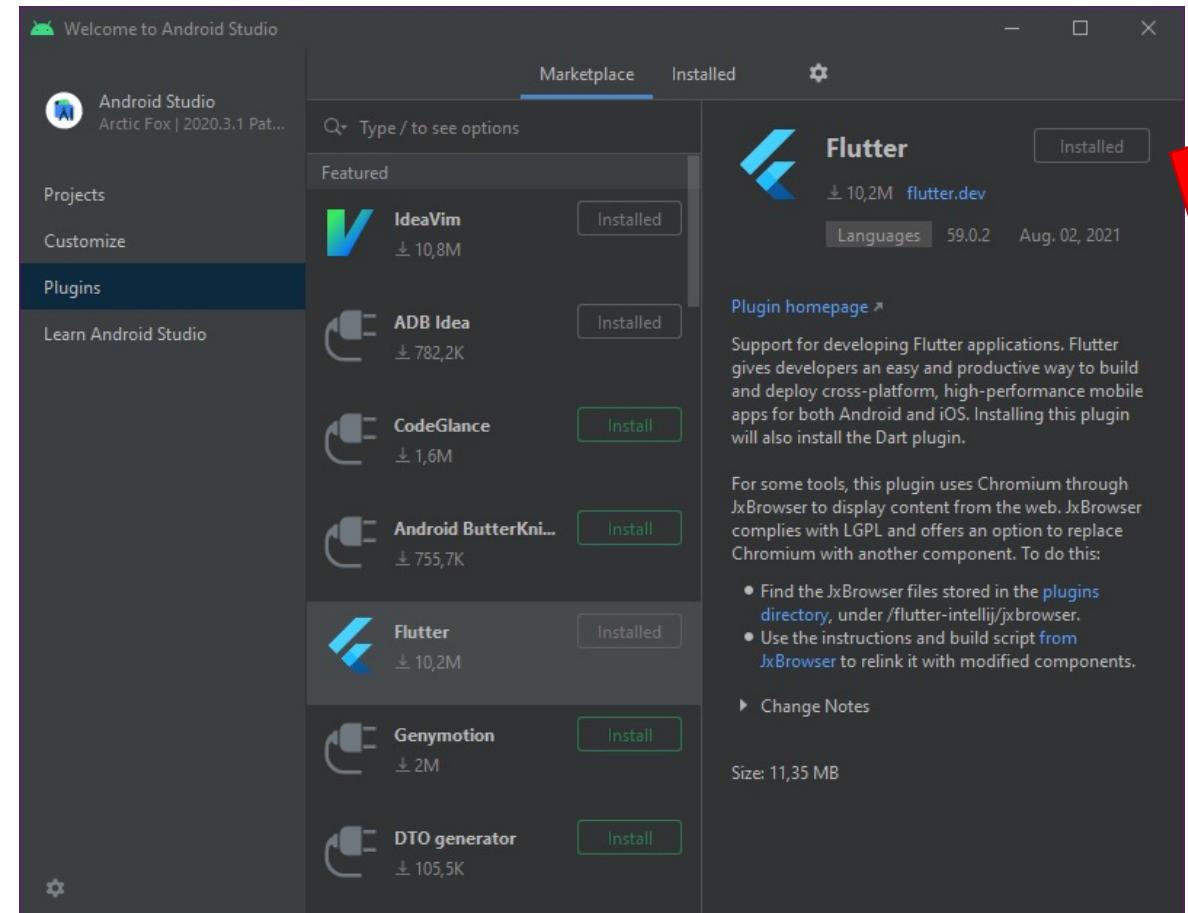
Setup – Flutter SDK

- Herunterladen der ZIP von <https://flutter.dev/docs/get-started/install>
- Entpacken der heruntergeladenen Datei
- Den Ordner flutter/bin zur PATH variable des Betriebssystems hinzufügen
 - Mac - .zshrc erweitern um Pfad
 - `export PATH="$PATH:/Users/[USERNAME]/[PATH2SDK]/flutter/bin"`
 - Windows - In Systemeinstellungen – Umgebungsvariablen für das System - Pfad erweitern

Setup – Flutter SDK – Pfad unter Windows



Setup – Plugins in Android Studio



Hier Plugin installieren

Setup – flutter doctor

- Installation prüfen
- Terminal starten und flutter doctor ausführen - Anleitungen folgen!

```
[suenlue@MBP-von-Saban ~ % flutter doctor
Doctor summary (to see all details, run flutter doctor -v):
[✓] Flutter (Channel stable, 2.5.3, on macOS 12.0.1 21A559 darwin-arm, locale
    de-DE)
[✓] Android toolchain - develop for Android devices (Android SDK version 31.0.0)
[✓] Xcode - develop for iOS and macOS
[✓] Chrome - develop for the web
[✓] Android Studio (version 2020.3)
[✓] Android Studio (version 2020.3)
[✓] Connected device (1 available)

• No issues found!
suenlue@MBP-von-Saban ~ %
```

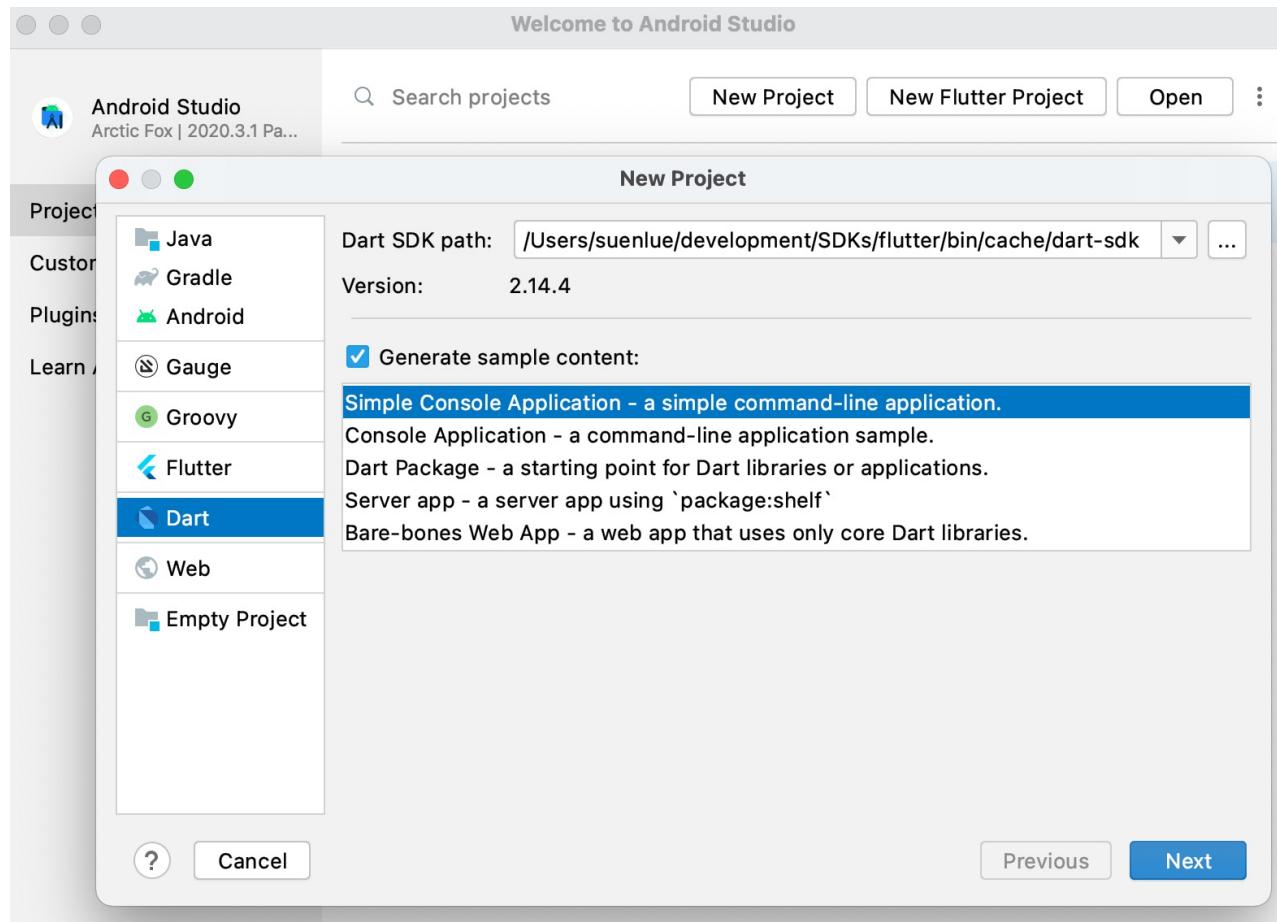
Dart für Flutter Entwickler



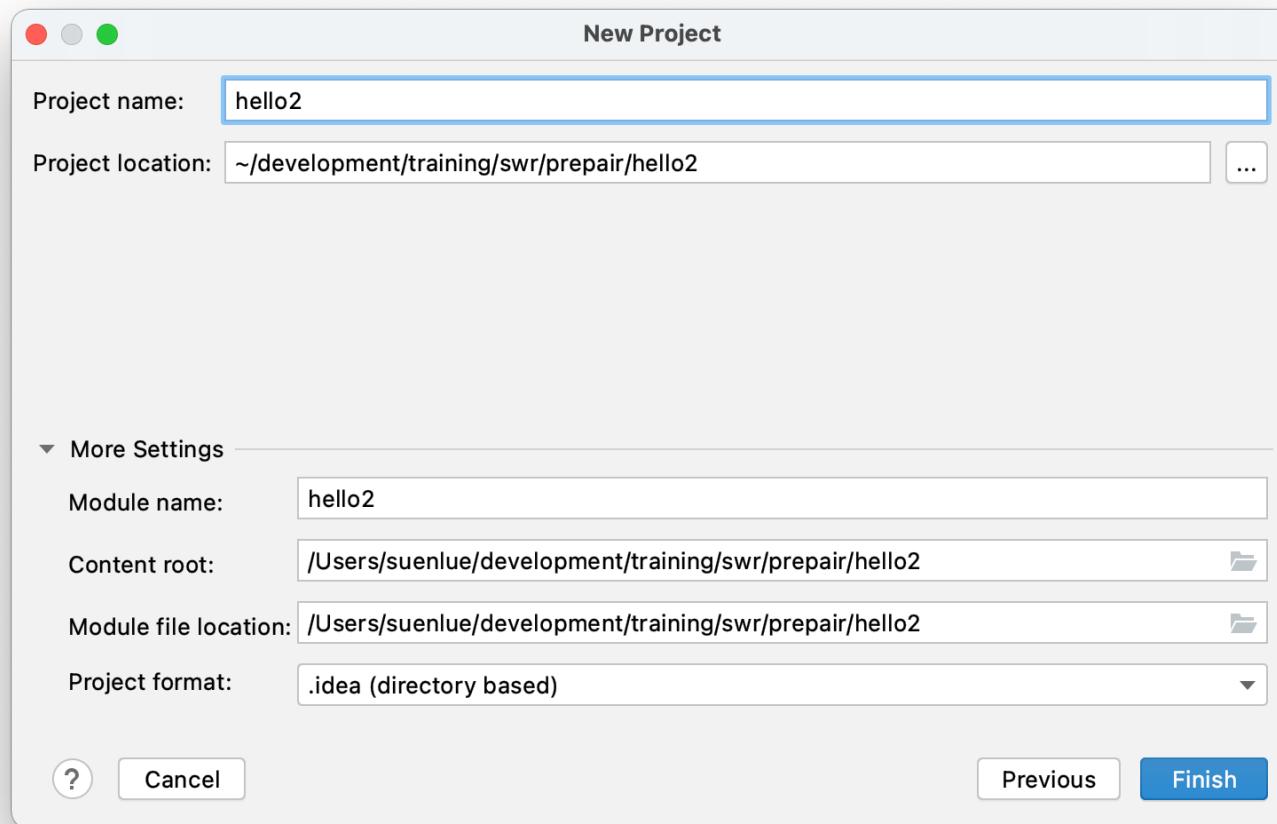
Anlegen eines Dart Projekts

- Über Terminal
 - dart create hello
 - cd hello
 - dart run

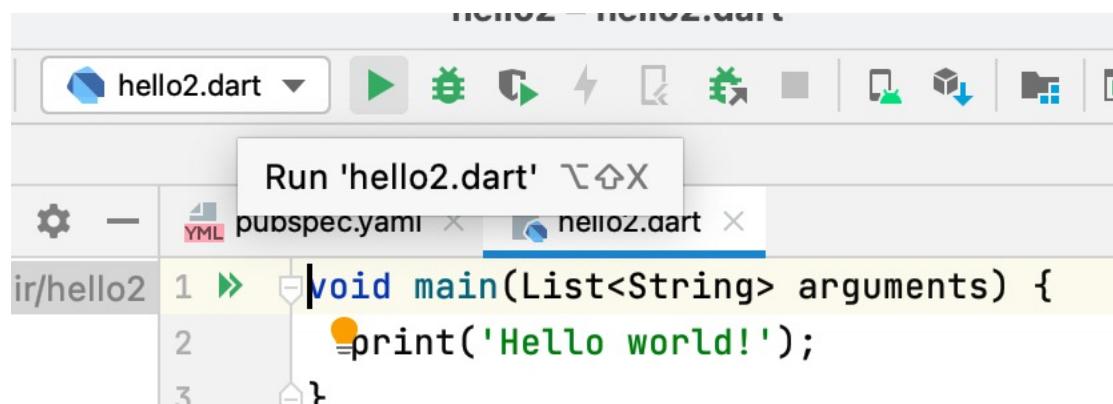
Anlegen eines Dart Projekts - Studio



Anlegen eines Dart Projekts - Studio



Anlegen eines Dart Projekts - Studio



The screenshot shows the Dart Editor interface. At the top, there's a toolbar with various icons. Below it is a 'Run' dialog box containing the text 'Run 'hello2.dart''. The main workspace has tabs for 'pubspec.yaml' and 'hello2.dart'. The 'hello2.dart' tab is active, showing the following code:

```
void main(List<String> arguments) {  
  print('Hello world!');  
}
```

Das Dart CLI

- `dart run path/to/file.dart` => Führt main() Funktion aus der Datei aus
- `dart pub get` => Abhängigkeiten herunterladen
- `dart pub add package_name [-dev]` => Fügt Paket von pub.dev den Dependencies in der pubspec.yaml hinzufügen
- `dart pub upgrade` => Aktualisiert Pakete

Was ist die pubspec.yaml

- Ähnlich zur package.json in node.js
- Beinhaltet grundlegende Informationen zum Projekt
 - Name & Beschreibung
 - Dependencies & Dev Dependencies
 - Environment
 - Version & Publishing

Einstiegspunkt der Dart Anwendung

- `void main(List<String> arguments)`
 - Keine Rückgabe
 - Optional mit Liste von Argumenten

Exkurs: Schreibweisen

- Typdefinitionen vor dem Bezeichner
- snake_case für Dateien
- PascalCase für Klassen
- camelCase für Methoden, Funktionen & Variablen
- _underscore Präfix für private Attribute

Datentypen

Primitive Datentypen

- Klein geschrieben
- Beinhalten einen Datensatz
- Von Sprache vorgegeben

Referenz Datentypen

- Groß geschrieben
- Beinhalten einen Pointer
- Durch Klassen definierbar

Primitive Datentypen

- Boolean (bool) => Wahrheitswert
- Integer (int) => Ganzahlwert
- Double (double) => Dezimalwert

Referenz Datentypen

- String => Buchstabenkette
- List => Liste (Keine Arrays)
- Map => Schlüssel-Wert Paare
- Set => Liste eindeutiger Werte

Lokale Variablen

- Deklaration beginnt über
 - var - Schlüsselwort
 - Datentyp z. B. int, String
 - int? – nullable – ohne Initialisierung / Zuweisung
 - final // kann nur einmal gesetzt werden
 - const // muss initialisiert werden - Wertzuweisung

Variablen

- Definiert durch: 'typ bezeichner = wert;'
- 'var' für impliziten Typen
- 'dynamic' für veränderbaren Typen
- 'typ?', um null zu erlauben

Lokale Variablen

Typ
Bezeichner
Zuweisung

```
var num = 0; // automatische Typisierung durch Wertzuweisung
int realNum = 11;
// int realNum = 11;
final finalNum = 33; // automatische Typisierung durch Wertzuweisung
const constNum = 44; // automatische Typisierung durch Wertzuweisung
// const int constNum2; // constant must be initialized.
final int finalNum2;
// print ( finalNum2 ); // can't be read - potentially unassigned at this point.
finalNum2 = 4711;
//
String name = 'Saban';
String? nullableName;
//
print ('$num - $realNum - $constNum - $finalNum - $finalNum2 - $nullableName??"is null"');
// finalNum = 3; // final variable can only be set once
// constNum = 3; // constant variables can't be assigned a value.
```

Def.Value

Funktionen

- Aufbau
 - Bezeichner
 - Parameter Definitionen
 - Funktionsblock
- Arrow
 - => statt Funktionsblock zur Rückgabe eines Wertes

Iterative Programmierung – Funktionen

The screenshot shows a Dart code editor with the following code:

```
timesTwo(x) {
    return x * 2;
}

timesFour(x) => timesTwo(timesTwo(x));

runTwice(x, f) {
    for (var i = 0; i < 2; i++) {
        x = f(x);
    }
    return x;
}

main() {
    //String Extrapolation durch $var oder ${code}
    print("4 times two is ${timesTwo(4)}"); // 8
    print("4 times four is ${timesFour(4)}");
    print("2 x 2 x 2 is ${runTwice(2, timesTwo)}"); // 8
    print("2 x 3 x 3 is ${runTwice(2, (x) => x * 3)}"); // 18
    print("2 x 3 x 3 is ${runTwice(2, (x) {
        return x * 4;
    })}"); // 32
}
```

Annotations with red arrows and text:

- A red arrow labeled "Bezeichner" points to the identifier "timesTwo".
- A red arrow labeled "Parameter" points to the parameter "x" in the function definition.
- A red arrow labeled "Arrow Functions" points to the declaration of "timesFour".
- A red arrow labeled "Einstiegspunkt der App" points to the "main()" function.
- A red arrow labeled "Funktion als Parameter" points to the call "runTwice(2, timesTwo)".
- A red arrow labeled "Anonyme Functions" points to the lambda expression "(x) => x * 3".

Typsichere Funktionen

- Aufbau
 - Rückgabetyp
 - Bezeichner
 - Parameter Definitionen mit
 - Funktionsblock

Primitive Datentypen

```
//Integer als Rückgabe- und Parametertyp
int timesTwo(int x) {
    return x * 2;
}

int timesFour(int x) => timesTwo(timesTwo(x));

//Funktion als Parameter
int runTwice(int x, int Function(int) f) {
    //Impliziter Typ Integer
    for (var i = 0; i < 2; i++) {
        x = f(x);
    }
    return x;
}

//Kein Rückgabewert
void main() {
    print("4 times two is ${timesTwo(4)}");
    print("4 times four is ${timesFour(4)}");
    print("2 x 2 x 2 is ${runTwice(2, timesTwo)}");
}
```

Rückgabetyp

Typisierte Parameter

Funktionen – Parameter

- Positionelle Parameter
 - Notwendige
 - Optionale => []
- Benannte Parameter => {}
 - required oder optional bzw. mit Standardwert

Bedingungen und Schleifen

if(bool) {
}

for(x in
collection) {
}

do {
} while(bool)

while(bool){
}

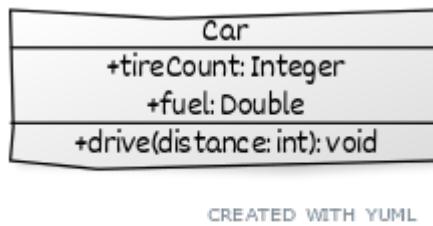
for(int i = 0;
i < 10; i++) {
}

bool? a:b

OOP – Object Oriented Programming

- Gliederung des Codes in Klassen
- Klasse als abgeschlossenes System (Datenkapselung)
- Eine Klasse, eine Verantwortung (SRP=Single Responsibility Principle)
- Vererbung oder Dekoration zur Erweiterung von Funktionen

OOP – Was ist eine Klasse?



- Klassen als Vorlagen für Objekte
- Klassen werden über den Konstruktor instanziert
- Klassen haben Attribute & Methoden
- Jede Klasse bildet ihren eigenen Datentypen
 - Polymorphismus durch Vererbung

OOP – Instanziierung einer Klasse

- Zur Instanziierung sind benötigt
 - Eine Klassen Definition über
 - class
 - Ein Konstruktor ist dabei zunächst Optional
- Instanziierung erfolgt (mit optionalem new Schlüsselwort) durch den Aufruf der Klasse
- Die Instanz hat den Typ der Klasse

OOP – Attribute

- Variablen einer Klasse
- Spezifisch für das resultierende Objekt
- In Dart normalerweise 'public', 'private' wenn der Name mit _ (Unterstrich) beginnt

OOP – Attribut Schlüsselwörter

- Initialisierung ähnlich Variablen.
- static => spezifisch für die Klasse, nicht das Objekt
 - const => bekannt zur Compilierzeit
- final => unveränderbar nach Initialisierung
 - Wert Zuweisung nur über den Konstruktor

OOP – Attribut schlüsselworte

- late => Initialisierung geschieht später
- Unterstrich als Präfix des Bezeichners
 - Privat im Kontext der Bibliothek

OOP – Methoden

- Funktionen einer Klasse
- Agieren auf dem resultierenden Objekt (Instanz)
 - static => spezifisch für die Klasse, nicht das Objekt
- Unterstrich als Präfix des Bezeichners
 - Privat im Kontext der Bibliothek

OOP – Getter & Setter

- Methoden, die mit 'get' oder 'set' gekennzeichnet sind Getter oder Setter
- Sind wie Attribute zugänglich
- Machen private Attribute außerhalb der Klasse zugänglich
- Können Werte vorab verändern oder nur Kopien von Referenzdatentypen weiterreichen

OOP – Konstruktoren und Factories

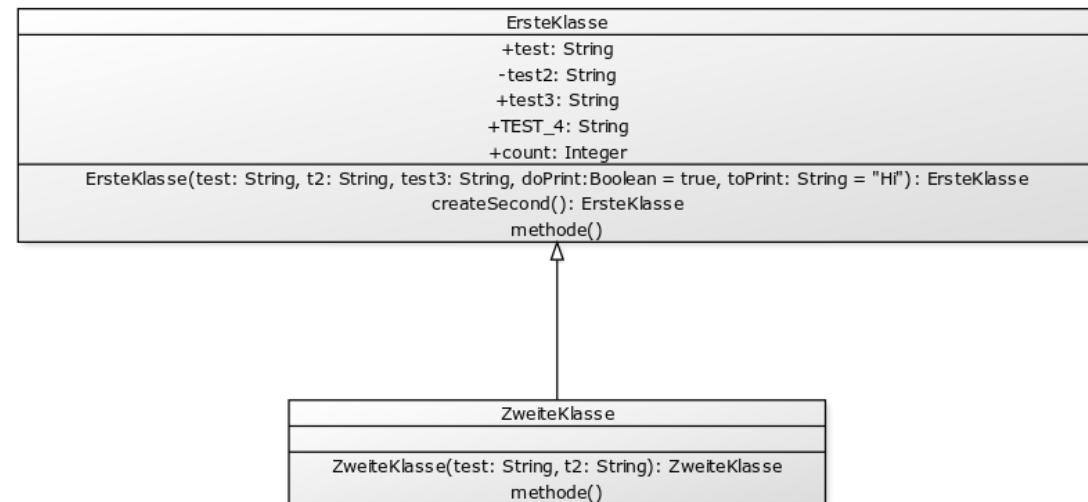
- Sind spezielle Methoden zur Instanziierung
- Nur einer kann den Namen der Klasse tragen
 - Factories müssen mit dem Klassennamen beginnen und nutzen statt static -> factory als Schlüsselwort
 - Konstruktor initialisiert Objekt, Factory muss Objekt returnieren

OOP – Konstruktoren und Factories

- Initialisierungsliste
 - Instanziierung finaler Variablen so wie Aufruf von super()
 - Wird vor dem Konstruktorrumpf nach einen Doppelpunkt ,:, Kommasepariert aufgeführt
 - kein Zugriff auf this – da Rumpf nicht abgeschlossen.
 - D.h. Initialisierungslisten aller Superklasse werden vorab erzeugt

OOP – Vererbung

- Attribute, Methoden und der Typ werden vererbt
- Keine mehrfach Vererbung
- Größte Abstraktion zu spezifischer Implementation (Spezifisch erbt von Allgemein)



CREATED WITH YUML

OOP – Abstrakte Klassen

- Können nicht instanziert werden
- 'abstract' Schlüsselwort zwingt Kinder zum implementieren
- Interface als rein abstrakte Klasse
 - In Dart implizite Definition durch Klassen
 - Mit 'implements' können mehrere Interfaces implementiert werden

OOP – Mixins

- Zur Vermeidung von Redundanzen bei nicht linearen Vererbungen
- Werden mit 'with' zu einer Klasse hinzugefügt
- Implizit erstellt mit jeder Klassendefinition oder mit [mixin Schlüsselwort \(on\)](#) stellt erbende Klasse bei Kind sicher)
- Alle Attribute und Eigenschaften werden übernommen, nach den Eigenschaften der eigentlichen Klasse

Operatorn

- Kaskaden-operator: .. => der folgende Ausdruck returniert den letzten
- Spread-operator: ... => Liste als einzelne Elemente
- Null-safety-operator: ?. oder !. => macht nur weiter, wenn nicht null
(? => null, ! => error)
- Replace-null-operator: ?? => wenn wert null nutze folgendes
- operator -> eigene Operatoren

REST

Json mit Dart

- JsonEncode & JsonDecode konvertieren zwischen String & Map
- Modelle sollten mit fromJson Konstruktor & toJson serialisieren
- Modelle können im Web oder durch Plugins automatisch generiert werden -> jsonToDart

REST – Get über id

- Packet Installieren: <https://pub.dev/packages/http>
 - dart pub add http & dart pub upgrade
 - import 'package:http/http.dart' as http;
 - Asynchroner Aufruf
 - http.get(Uri.parse('https://jsonplaceholder.typicode.com/posts/\$id'));
 - Optional mit Header Definitionen: {Map<String, String>? headers}

REST – CRUD

- Analog zu GET gibt es alle bekannten CRUD Methoden
 - Post
 - Put
 - Delete
- Bei Put und Post kann ein Body übergeben werden

Flutter

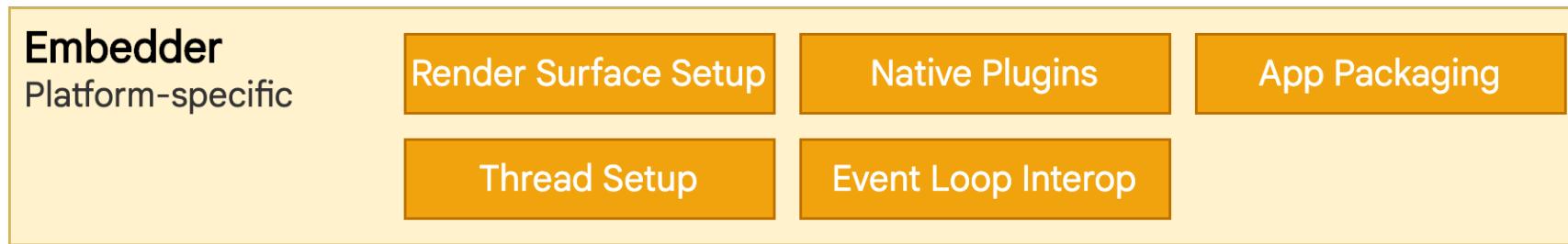


Funktionsweise

Die Technologie hinter Flutter

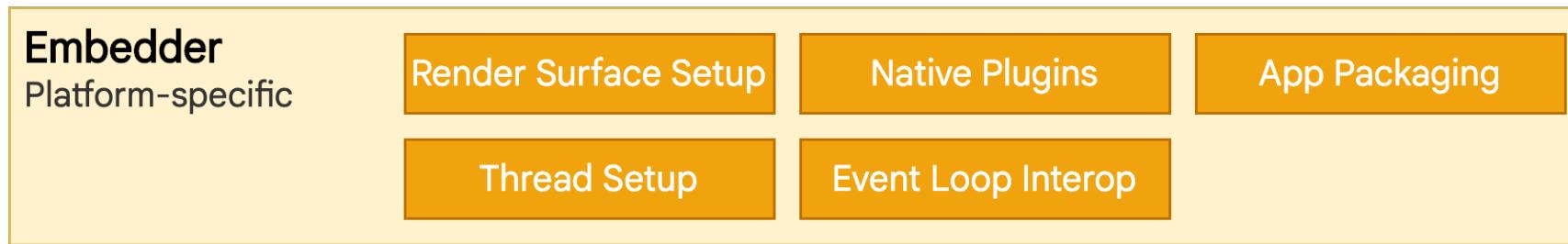
Schichten der Flutter Welt - Embedder

- Flutter-Anwendungen werden plattformspezifisch gepackt und als native Anwendung ausgegeben.
- Einstiegspunkt plattformspezifischer Embedder
 - Koordiniert Zugriff auf Dienste
 - Rendering-Oberflächen
 - Zugang auf Eingaben und Nachrichtenreignisschleife.



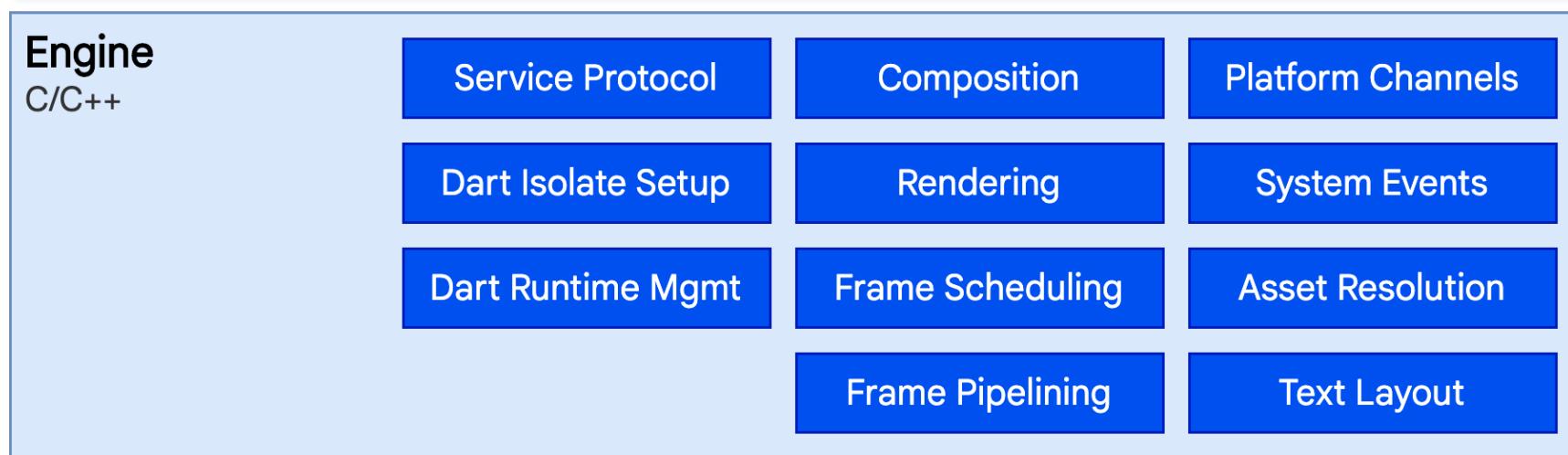
Schichten der Flutter Welt - Embedder

- Der Embedder ist in einer für die Plattform geeigneten Sprache geschrieben
- Embedder kann Flutter-Code im Host einbinden als:
 - Modul in eine bestehende Anwendung
 - Gesamter Inhalt einer Anwendung



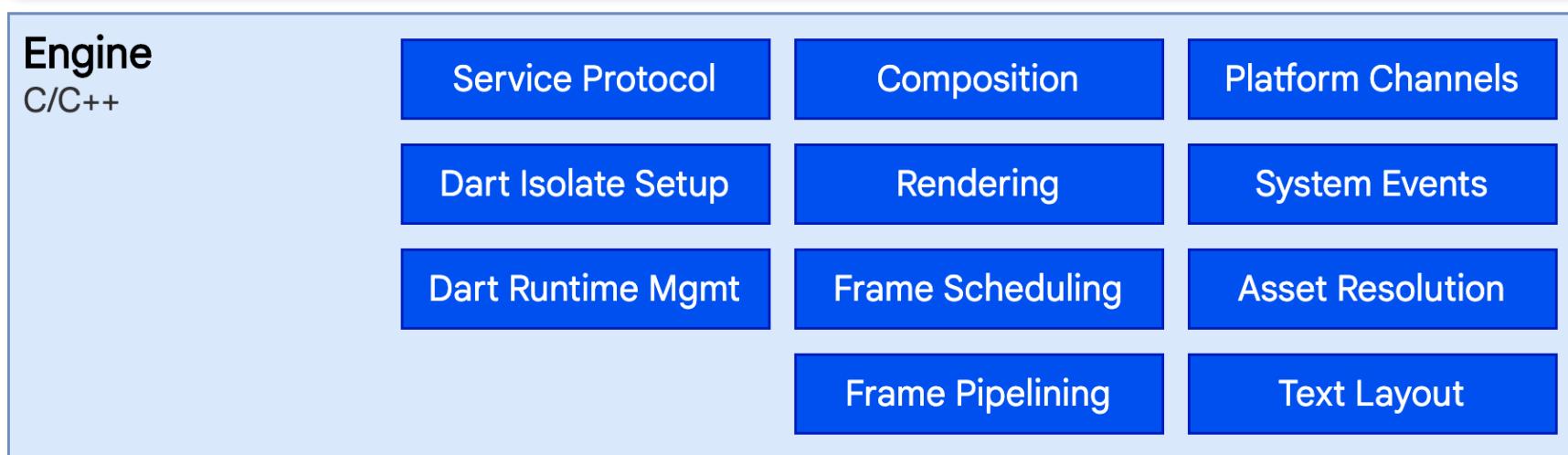
Schichten der Flutter Welt - Engine

- Kern: Flutter-Engine
 - größtenteils in C++ geschrieben (Low-Level-Implementierung)
 - Raster Szenen die gezeichnet werden muss beim Framewechsel



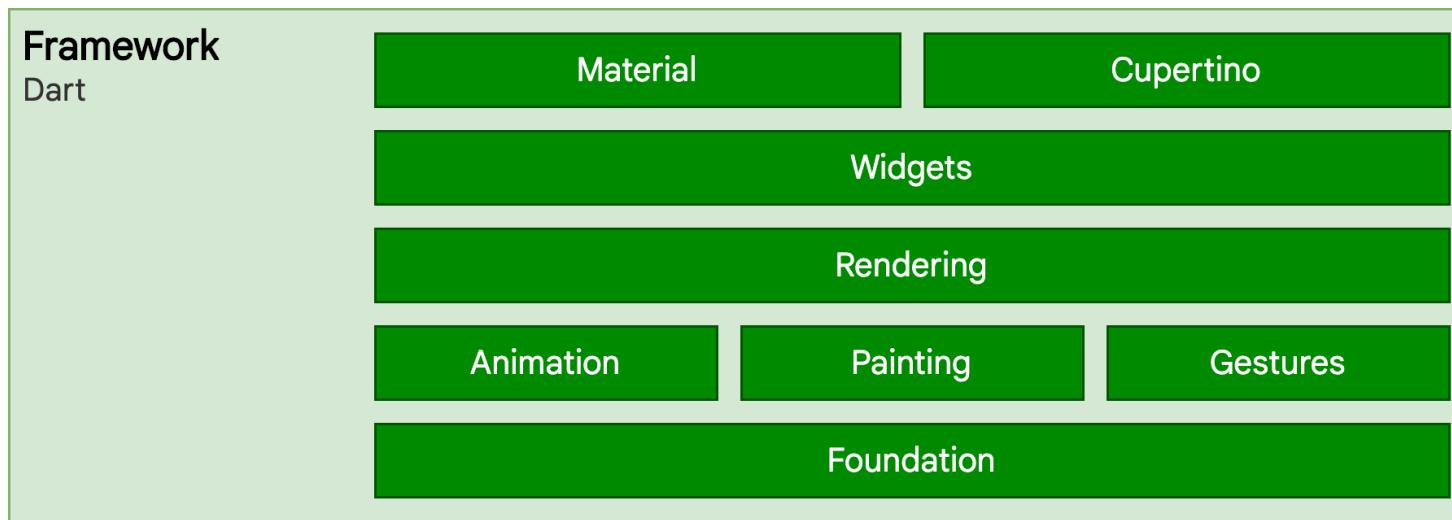
Schichten der Flutter Welt - Engine

- Zeichnet Grafiken (über Skia)
- Textlayout, Datei- und Netzwerk-I/O, Plugin-Architektur
- Dart-Laufzeit- und Kompilierungs-Toolchain.



Schichten der Flutter Welt - Engine

- Flutter-Framework nutzt Engine über dart:ui
- Dart-Klassen sprechen direkt C++-Code an
- Framework stellt Ebene zur Steuerung von Eingabe-, Grafik- und Text-Rendering-Subsystemen bereit



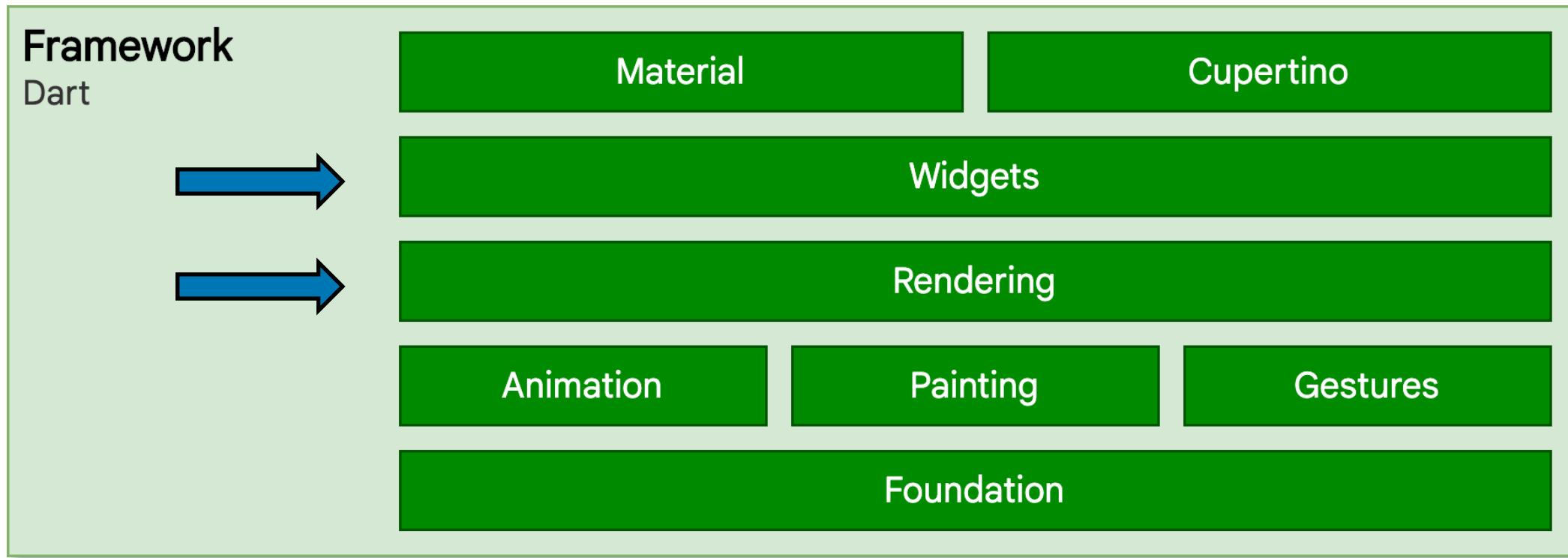
Schichten der Flutter Welt - Framework

- Flutter-Framework ist relativ klein
- Funktionen auf höherer Ebene werden als Pakete implementiert
 - einschließlich Plattform-Plugins (camera und webview)
 - sowie plattformunabhängige Funktionen (http)

Widgets und Rendering

Das erstellen einer App

Schichten der Flutter Welt - Engine



Zusammenspiel

Widget

- Unveränderbar Konfiguration der Benutzeroberfläche
- Konfiguration der Elemente

Element

- Repräsentation von Widgets

RenderObject

- Größen und Darstellung des Elements
- Benutzerinteraktionen durchreichen

Flutter - Rendering

Konfiguration

- Eigenschaften verwalten
- Öffentliche Schnittstellen bereitstellen

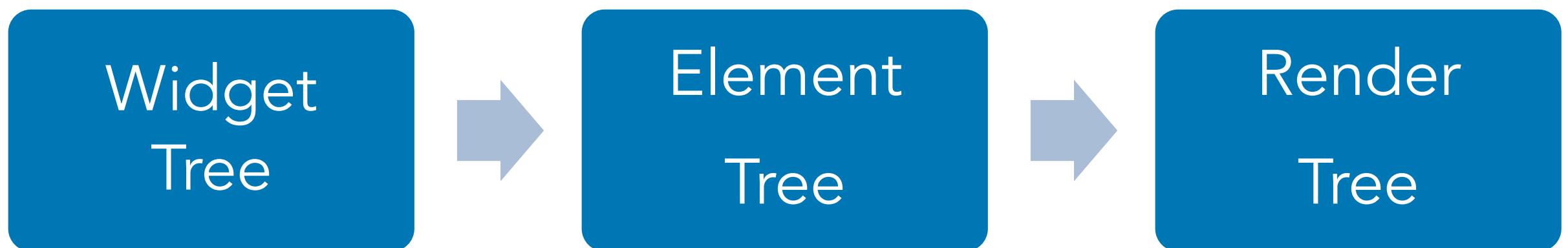
Lebenszyklus

- Fixer Bestandteil der Benutzer-oberfläche
- Verwaltung der Eltern-Kind Beziehungen

Darstellung

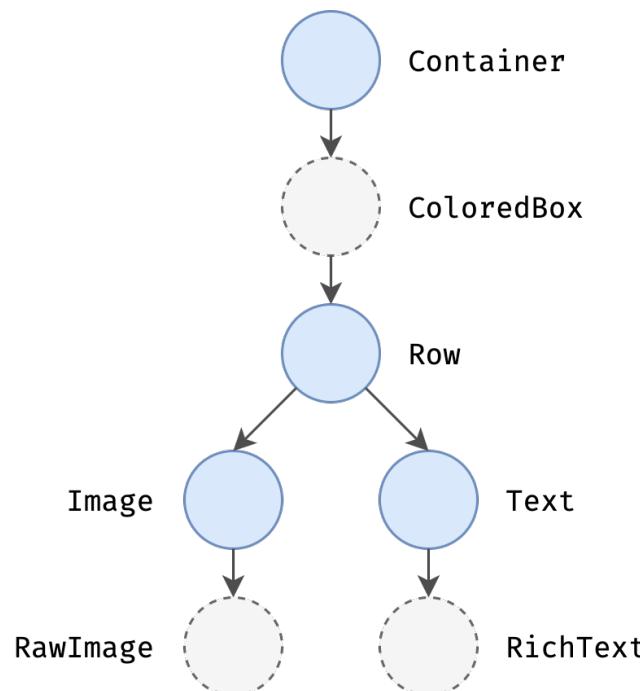
- Größen und Darstellung des Elements
- Kindselemente aufstellen
- Benutzerinteraktionen durchreichen

Die drei Flutter Bäume



Die drei Flutter Baume.

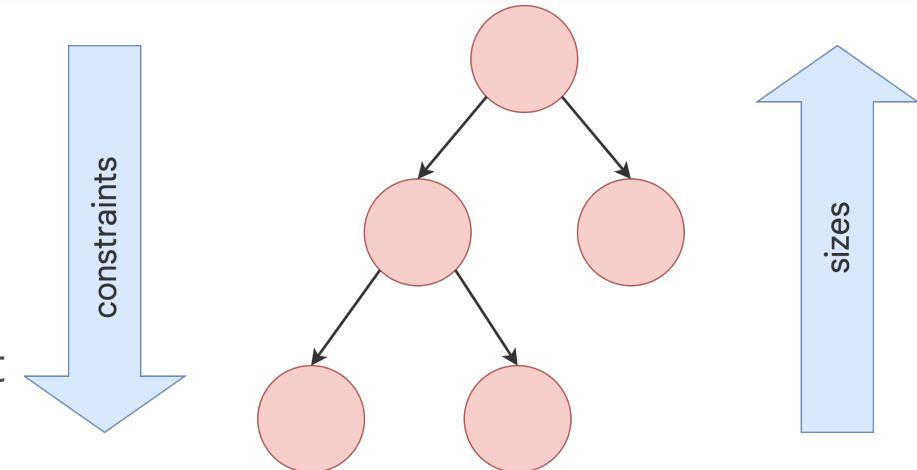
Widgets



https://dartpad.dev/?null_safety=true&id=c9a683cced98966fafc7689e629eb60e

Lauf durch den Baum

- Layout umsetzen
 - Renderbaum in die Tiefe Durchquerung
 - Größenbeschränkungen vom übergeordneten Objekt übergeben
- Kinder reagieren
 - Stellt Größe ein und Sagt dem Elternteil bescheid
 - Am Ende wird gezeichnet (paint)



Beschränkungsmodell für Layout Prozesse

- Sehr performant
- Fixe Beschränkung und Größendefinition wenn maximale und minimale Einschränkungen identisch
 - Root Objekt bekommt die Aufmaße des Displays
 - Breite des Kindes diktieren und Höhe flexibel gestalten (oder umgekehrt)
 - Beispiel: Fließtext
 - Entscheidung, ob ein- oder zweispaltiges Layout

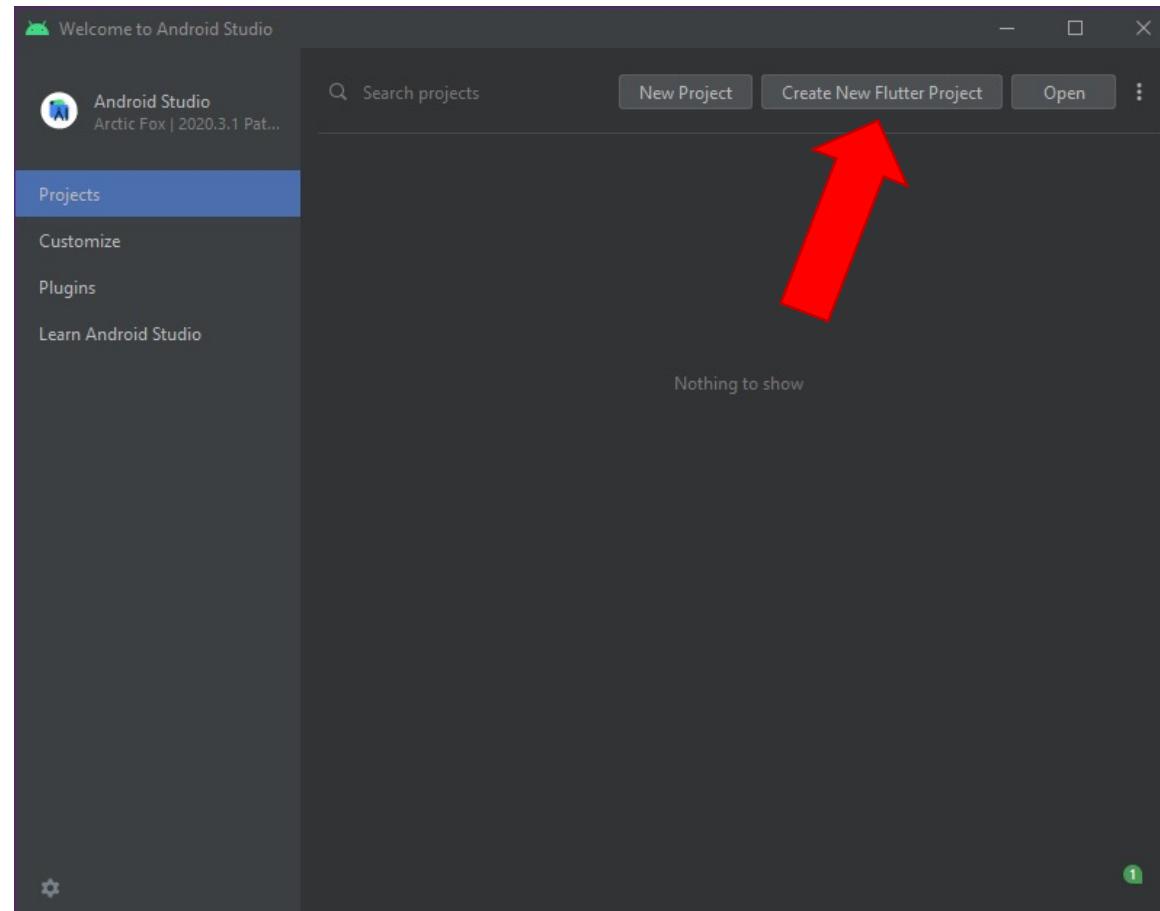
Zusammenfassung

- Widget ist Grundklasse, von der alle UI Vorlagen erben
- Widget-Tree stellt Konfigurationsebene der Oberfläche dar
- Widgets erzeugen (build) RenderObjectElementes
 - Elemente sind Bühnen Aktoren, die wieder verwendet werden, solange sich nichts am Widget ändert
 - Elemente Werden gerendert (RenderObject)
- Widgets lassen sich verschachteln und selber schreiben

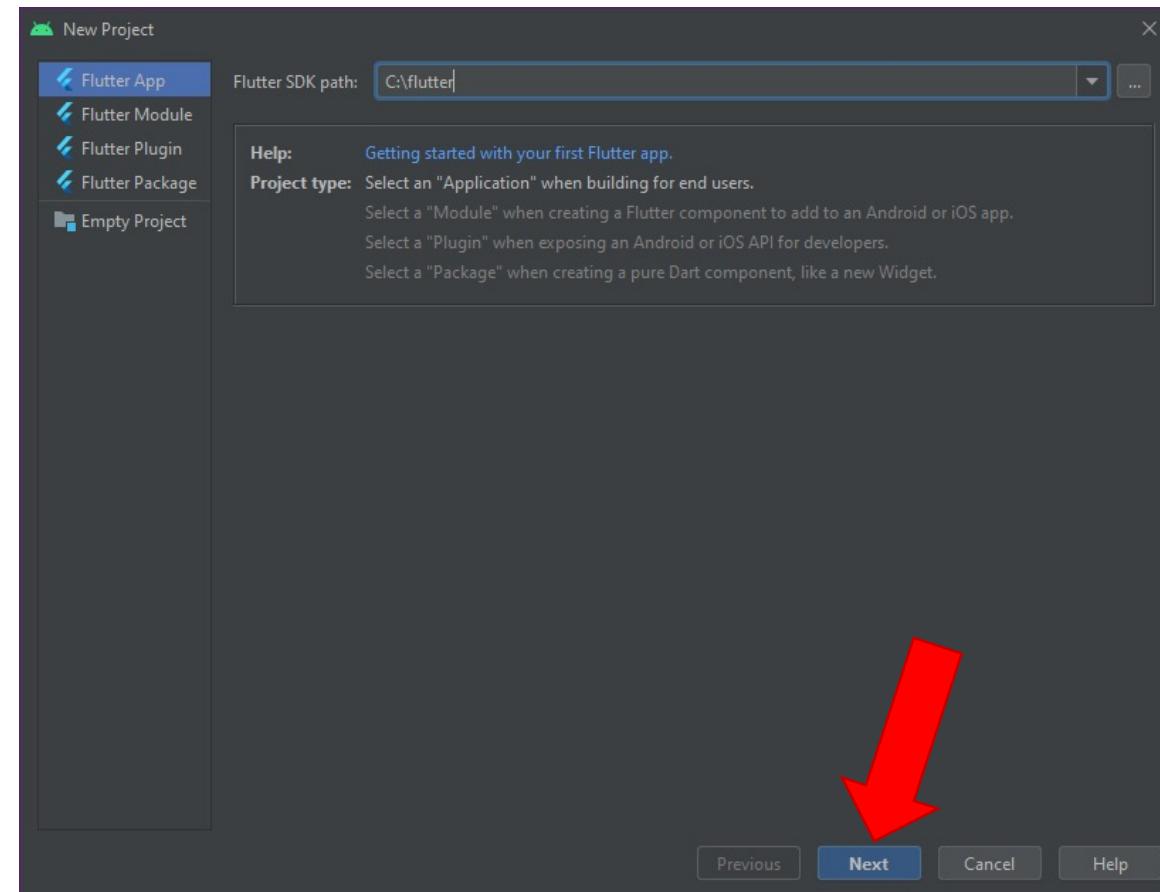
Anlegen eines Projektes

Das erstellen einer App

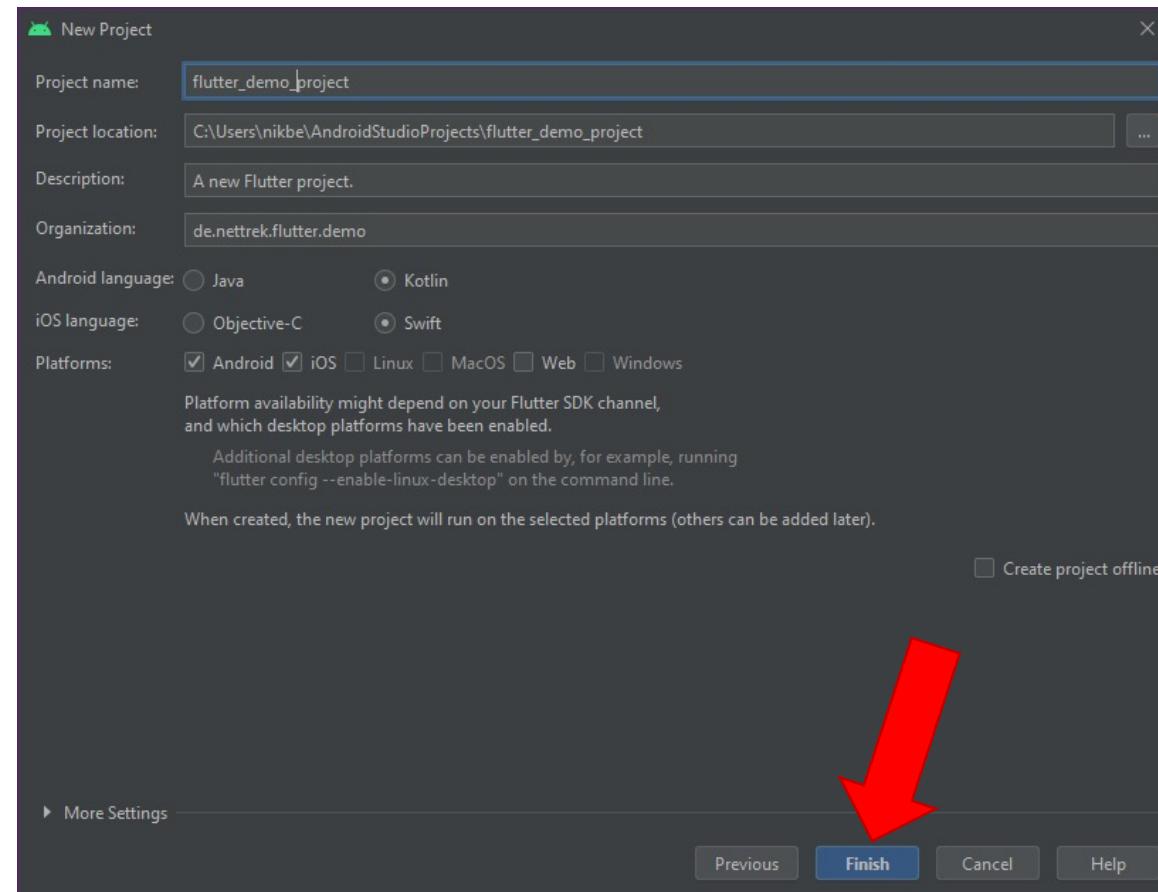
Erstellen einer App



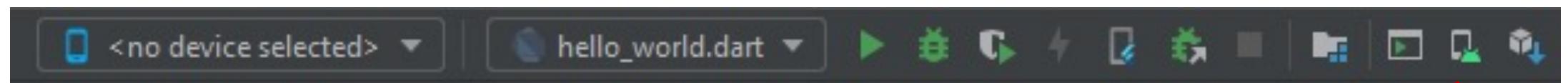
Erstellen einer App



Erstellen einer App

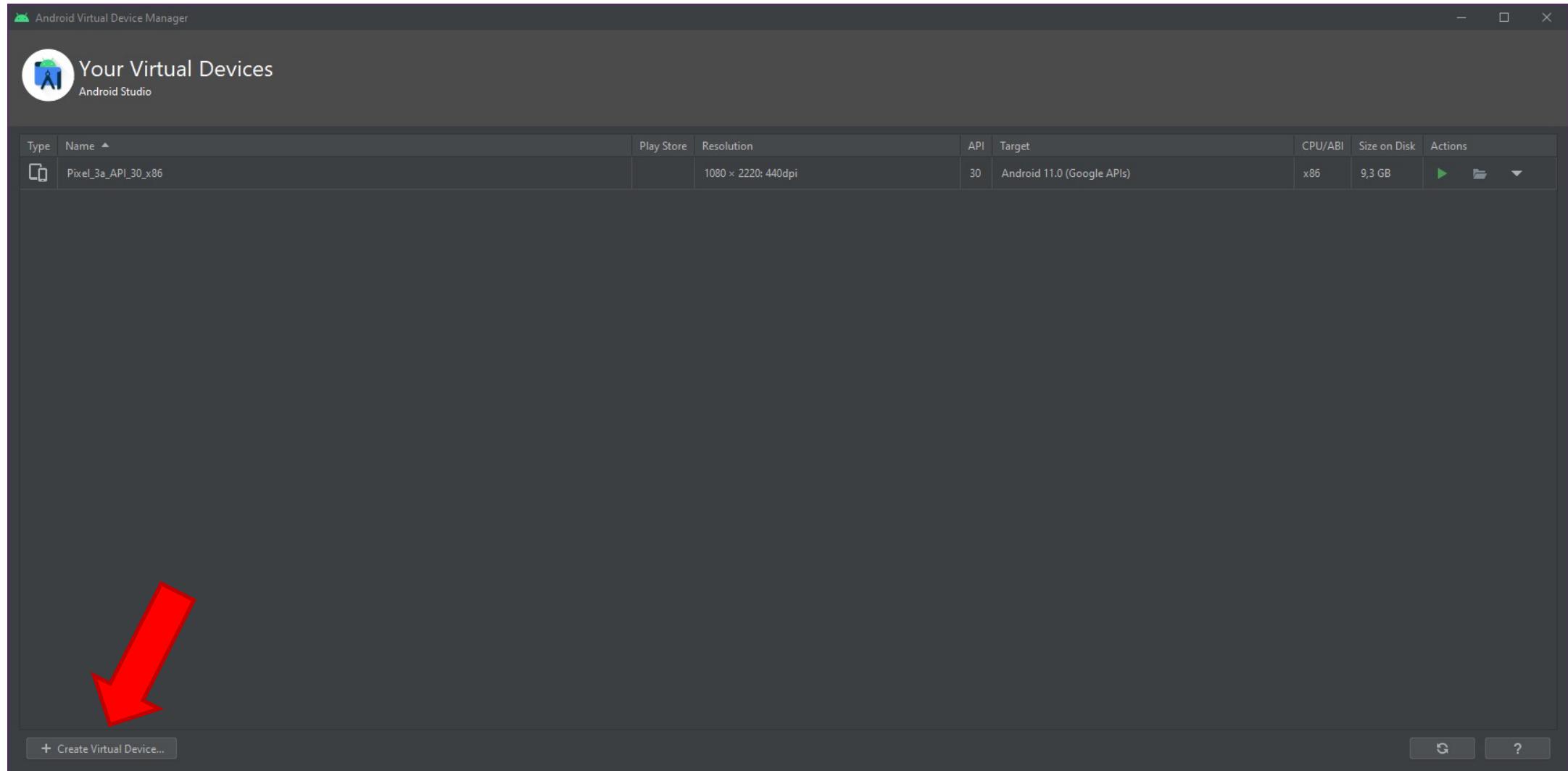


Exkurs: Emulatoren in Android Studio

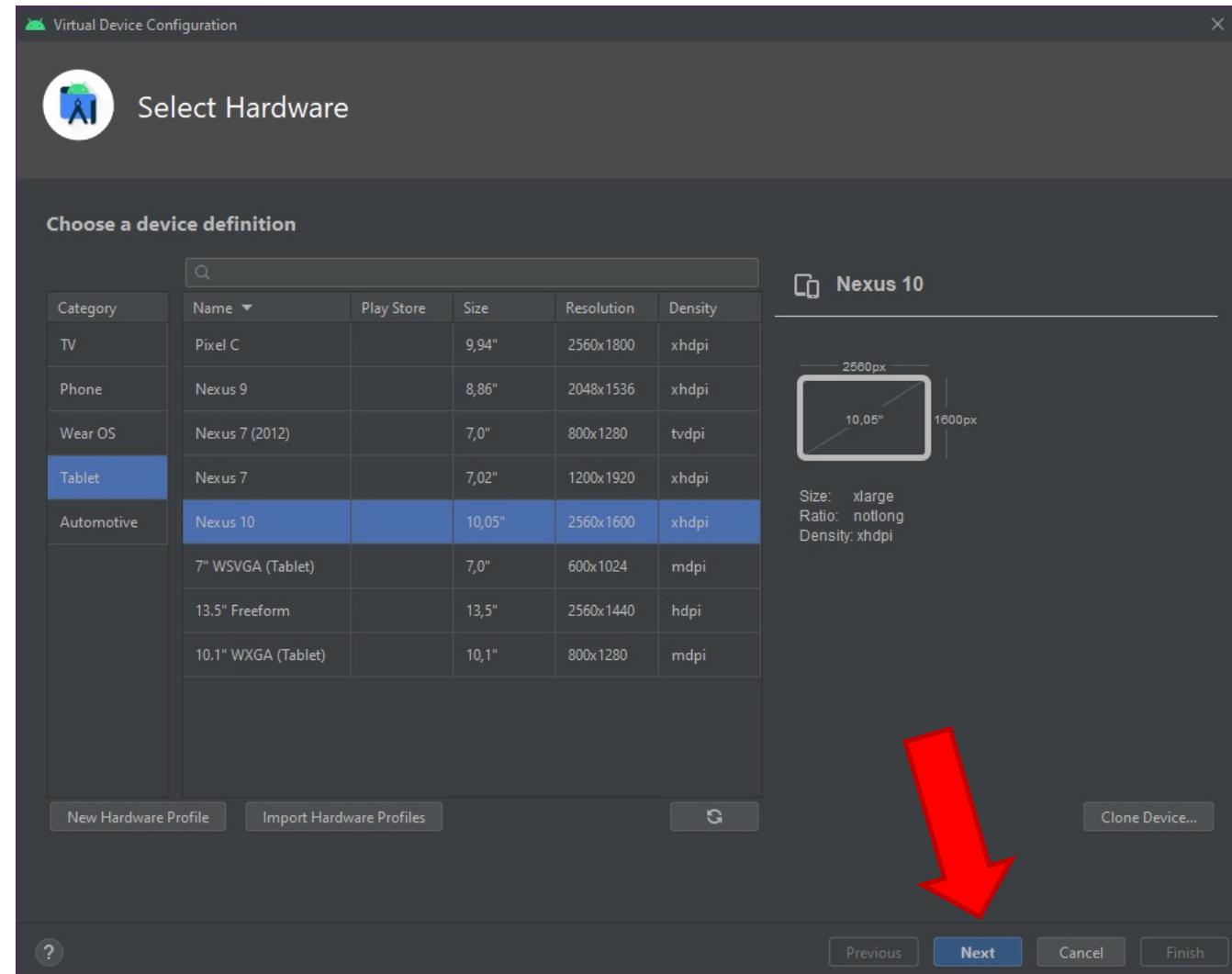


Android Virtual Device
Manager öffnen

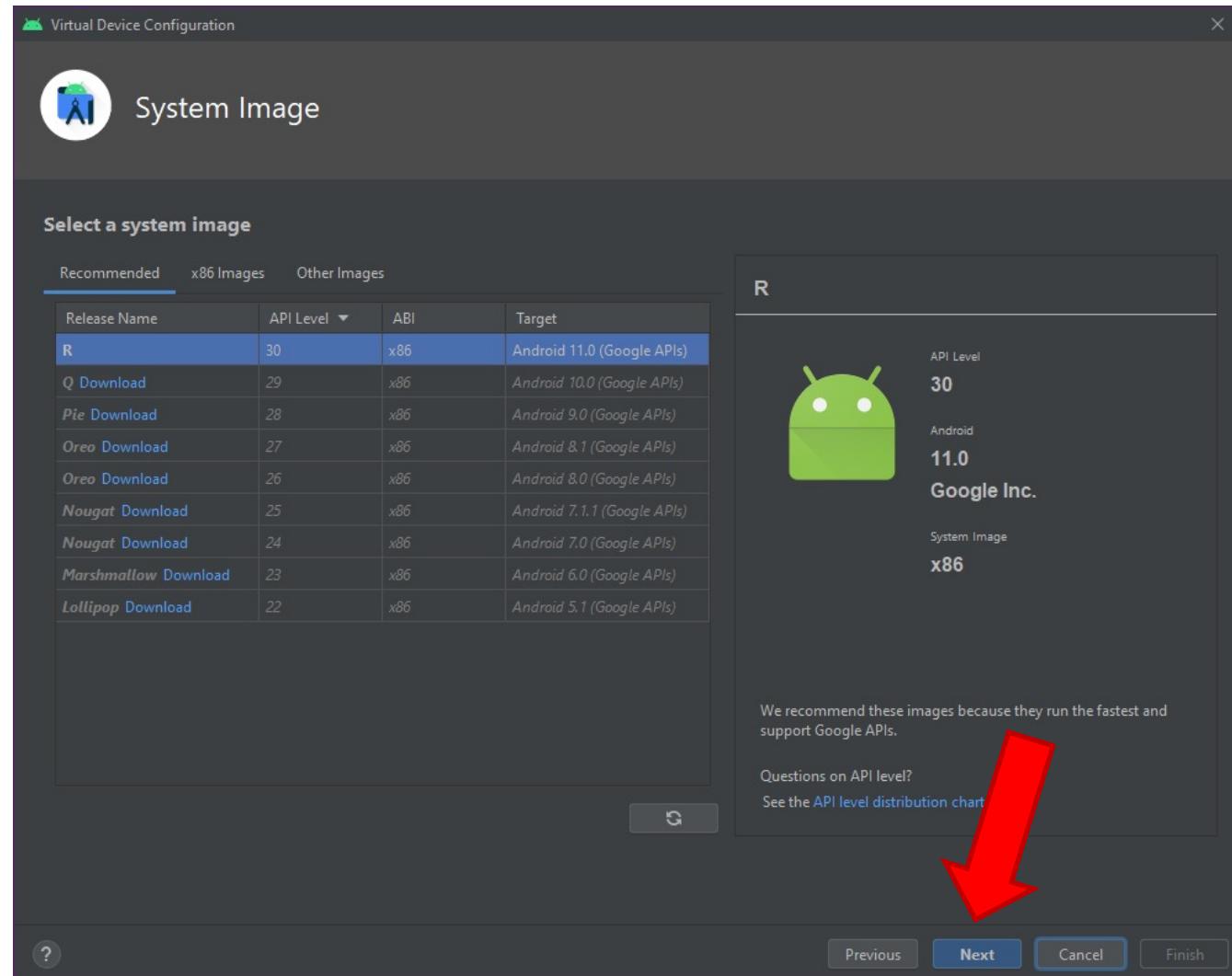
Exkurs: Emulatoren in Android Studio



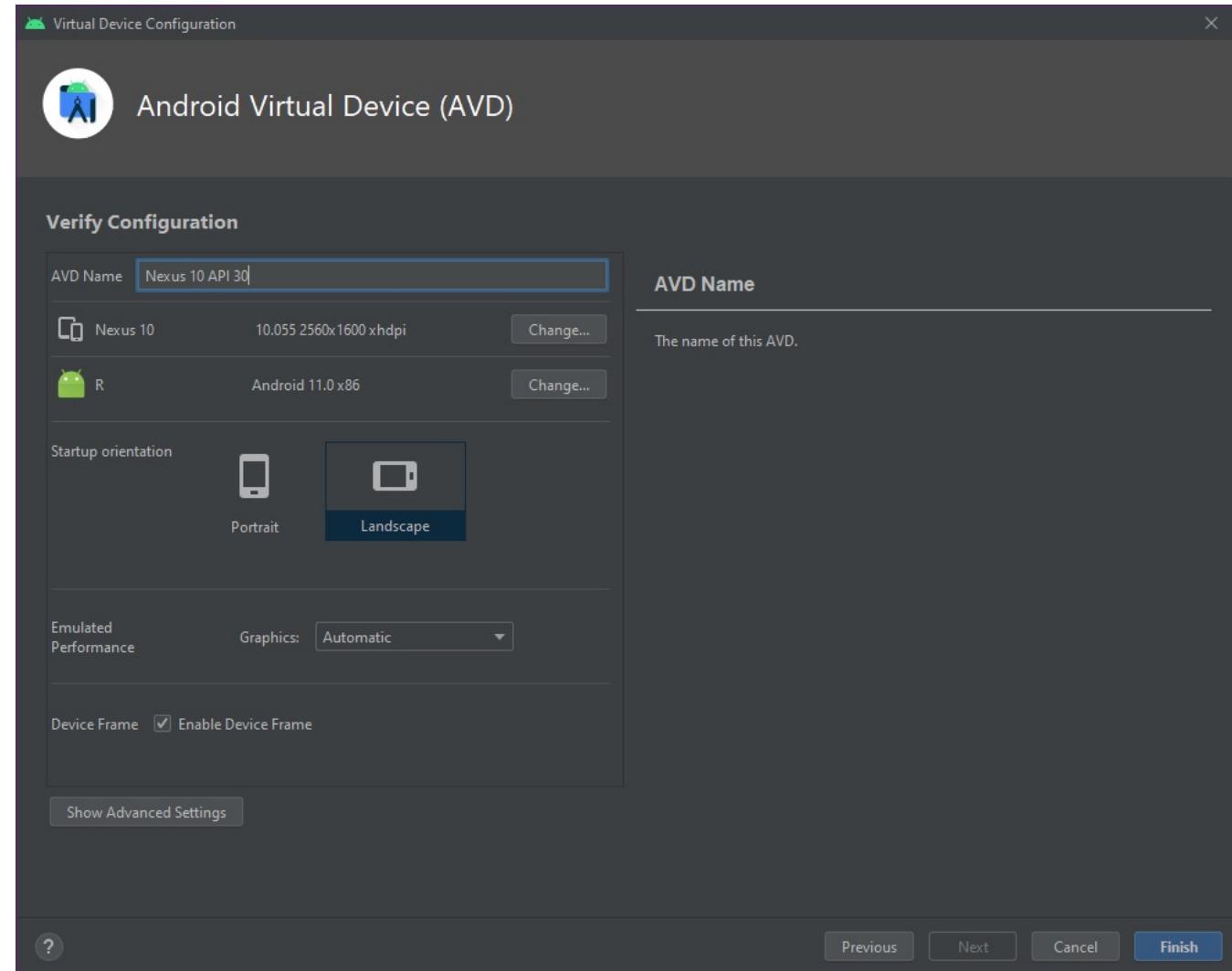
Exkurs: Emulatoren in Android Studio



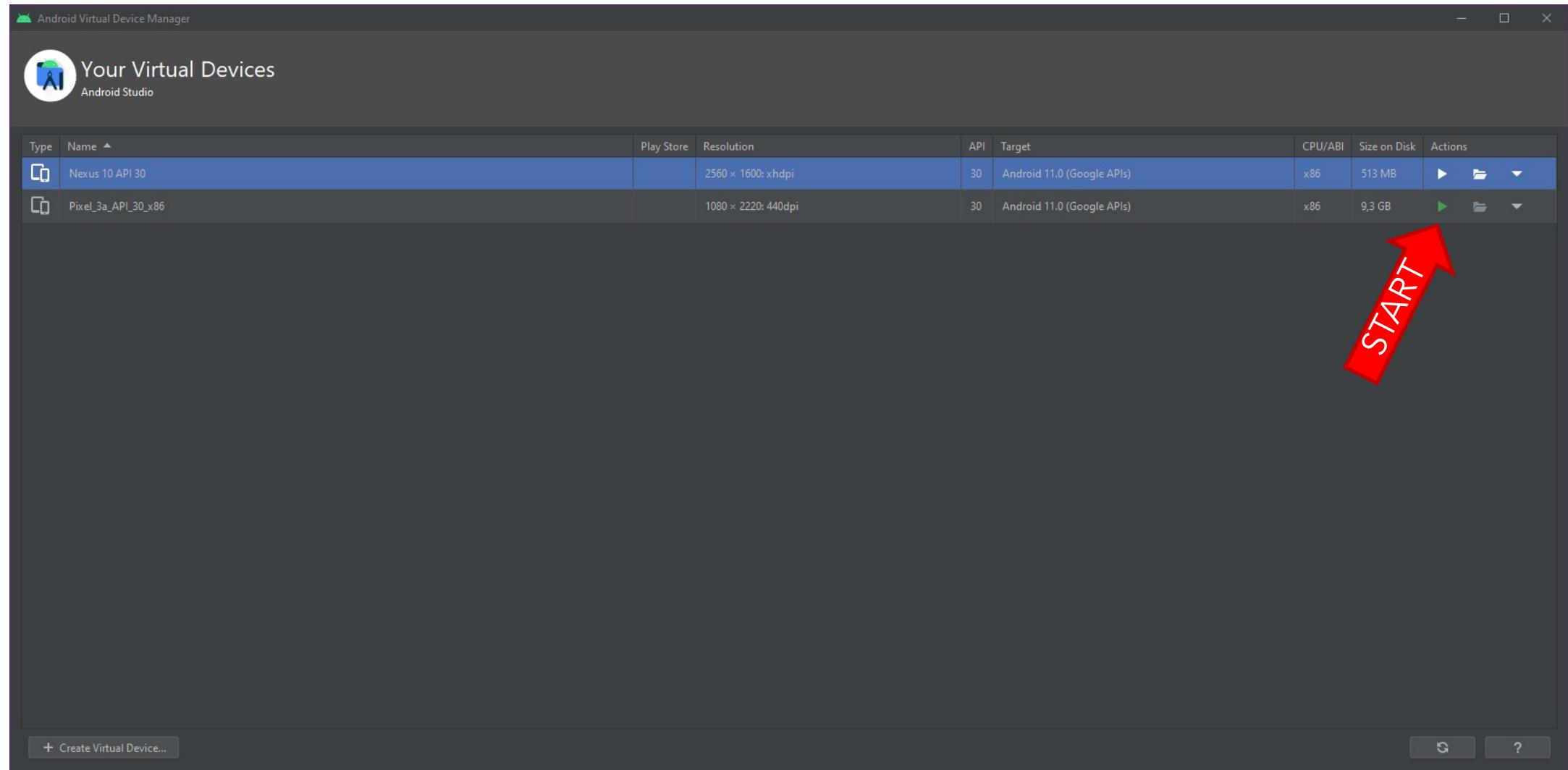
Exkurs: Emulatoren in Android Studio



Exkurs: Emulatoren in Android Studio



Exkurs: Emulatoren in Android Studio



Projektaufbau

- runApp(Widget)
 - Erstellt und initialisiert die App aus einem Widget
 - Typischerweise in der main() Funktion aufgerufen
 - Notwendig, für eine Flutter App

Exkurs: pubspec.yaml in Flutter

- Die Konfigurationsdatei eines Flutter Projekts
- In Flutter werden hier auch Assets und Fonts konfiguriert

Widgets

"Everything is a widget" - Google

Zwei Arten von Widgets

Layouting Widgets

- Container
- Center
- Stack
- ...

Content Widgets

- Text
- Buttons
- Image
- ...



Zwei Arten von Widgets

Layouting Widgets

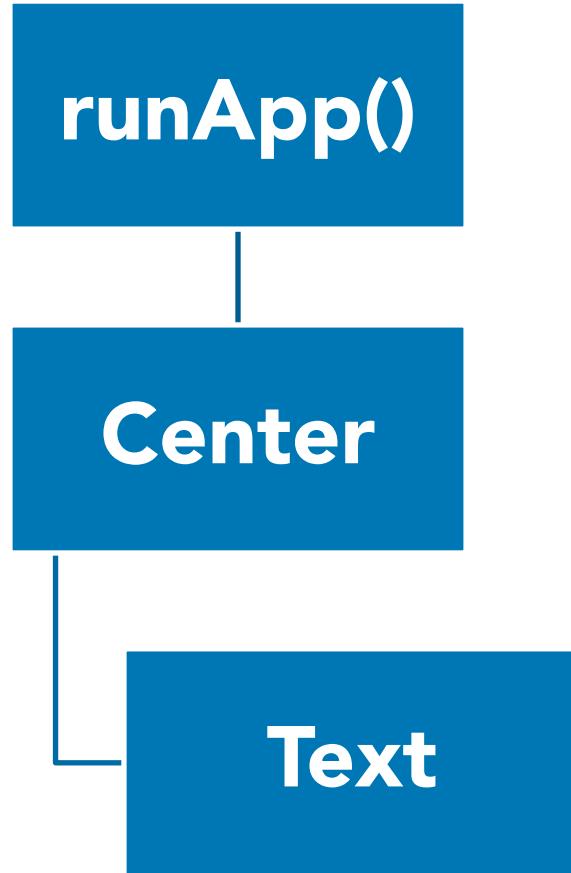
=> Akzeptieren ein (oder mehrere) 'child' Widget(s) (meist über den gleichnamigen benannten Parameter) und positionieren es eine Ebene tiefer im 'WidgetTree'.

Content Widgets

=> Zeigen einen Inhalt an.

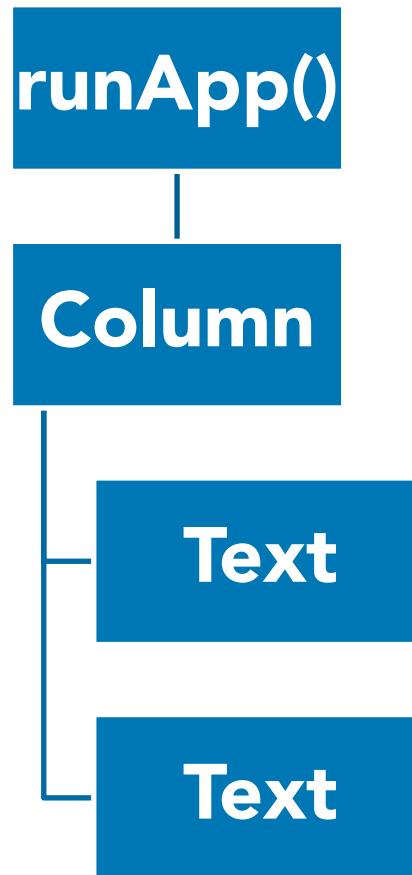


Aufbau eines WidgetTree



- runApp() startet die App und bildet damit den Ursprung der App (root)
- Center ist der einzige Ast vom Ursprung und zentriert sein Kind
- Text ist unser einziges Blatt und zeigt den Inhalt "Hello World!"

Aufbau eines WidgetTree



- runApp() startet die App und bildet damit den Ursprung der App (root)
- Column ist der einzige Ast vom Ursprung und zeigt seine Kinder untereinander an
- Text & Text sind unser Blätter und zeigt den Inhalt "Hello World!"

Themes

MaterialApp

- Android-like Theme
- Zusammen mit Material Package und entsprechenden Widgets

CupertinoApp

- iOS-like Theme
- Zusammen mit Cupertino Package und entsprechenden Widgets

=> Setzen einen Kontext für das App Theme

https://github.com/netTrek-Flutter/demo/tree/master/lib/hello_world_material.dart &
https://github.com/netTrek-Flutter/demo/tree/master/lib/hello_world/cupertino.dart



Was sind Material & Cupertino?

- Repräsentieren Designrichtlinien von Google & Apple
- Ansammlung vieler Widgets und anderer Klassen
- Werden nicht automatisch an die Plattform angepasst
=> iOS Designs auf Android und umgekehrt entstehen,
wenn kein neutrales Theme verwendet wird!

https://dartpad.dev/?null_safety=true&id=ac8be69c4e27689bc513b224b362d5e3 &
https://dartpad.dev/?null_safety=true&id=5ed1a336847646b6fc071f9a7bde5e37

Widgets nutzen - Scaffold

- Gibt die Grundstruktur des Layouts vor
- Ist die Material Variante, Cupertino hat CupertinoPageScaffold
- Setzt Googles Material Design Richtlinien durch

Widgets nutzen - Scaffold

- Wichtige Attribute wie appBar, bottomNavigationBar oder body
- body -> Hauptinhalt der Anwendung
- appBar -> Anwendungsleiste (Titel, Drawer, etc.)
- floatingActionButton -> Aktionsschaltfläche
- bottomNavigationBar -> Reiter Navigationen

Widgets nutzen – CupertinoPageScaffold

- Setzt Apples Design Richtlinien durch
- Deckt nur zusammen mit CupertinoTabScaffold alle Funktionen vom material Scaffold ab, da Apple ein anderes Navigationsmodell verwendet
- Hat Attribute wie child, backgroundColor oder navigationBar

Farben

- Auch Farben sind Designrichtlinienspezifisch
- Material => Colors
- Cupertino => CupertinoColors
- Allgemein mit Color(0xAARRGGBB) definierbar (0x Präfix für Integer in hexadezimalschreibweise)
- Nur RGB resultiert in durchsichtiger Farbe => nicht sichtbar

Styling – Ein Theme erstellen

- Theme ist unabhängig einer Designrichtlinie
- Bereitgestellt durch Theme Widget
- Alle gewünschten Eigenschaften in unserem Theme definiert, Rest hat Standard Werte
- Einige Widgets nutzen es von alleine
- Theme.of(BuildContext) um auf werte zuzugreifen

ColorScheme – In Theme nutzen

- **primary** Die Farbe, die auf den Bildschirmen und Komponenten Ihrer App am häufigsten angezeigt wird.
- **primaryVariant** Eine dunklere Version der Primärfarbe.
- **onPrimary** Eine Farbe, die beim Zeichnen auf der Primärfarbe deutlich lesbar ist.

ColorScheme – In Theme nutzen

- **secondary** Eine Akzentfarbe, die bei sparsamer Verwendung auf Teile Ihrer App aufmerksam macht.
- **secondaryVariant** Eine dunklere Version der Sekundärfarbe.
- **onSecondary** Eine Farbe, die beim Zeichnen auf der Sekundärfarbe deutlich lesbar ist.

ColorScheme – In Theme nutzen

- **background** Eine Farbe, die normalerweise hinter scrollbaren Inhalten angezeigt wird.
- **brightness** Die Gesamthelligkeit dieses Farbschemas.
- **error** Die für Eingabeverifikationsfehler zu verwendende Farbe,
- **onError** Eine Farbe, die beim Zeichnen auf der Fehlerfarbe deutlich lesbar ist.

ColorScheme – In Theme nutzen

- `surface` Die Hintergrundfarbe für Widgets wie Card. Finale.
- `onSurface` Eine Farbe, die beim Zeichnen auf der Oberfläche deutlich lesbar ist

Material – Linklist

- <https://material.io/design/material-theming/overview.html#material-theming>
- <https://material.io/resources/color/#!/?view.left=0&view.right=0&primary.color=9FA8DA&secondary.color=1E88E5>
- <http://mcg.mbitson.com/>

Widgets nutzen – Container

- Platziert ein Kind und berücksichtigt dabei
 - Innere-Leerraum (Padding) -> padding: EdgeInsets
 - Rand (zusätzlicher Leerraum) -> decoration: BoxDecoration
 - border: Border.all
 - Beschränkungen
 - height, width, constraints

Widgets nutzen – Container

- Ohne Einschränkung und Kind, versucht ein Container so groß wie möglich zu sein.
- Mit Kind -> passen Ihre Größe dem Kind an
- Mittels der `margin` Eigenschaft lassen sich Äußere Abstände definieren.
- Mittels der `alignment` Eigenschaft lassen sich Ausrichtungen der Kinder Steuern

Widgets nutzen – Container

- Mittels der `transform` Eigenschaft lassen sich Transformationen (z.B. Rotation) umsetzen

Widgets nutzen – Column / Row

- Platziert alle Kinder in einer Spalte / Zeile
- Kann in Richtung der mainAxis mehr Platz einnehmen
 - Im ‚debug mode‘ gekennzeichnet durch gelbe Balken
- Ausdehnung der crossAxis ist die, des größten Kindes
- Ausdehnung der mainAxis abhängig von mainAxisSize

Widgets nutzen – Column / Row

- mainAxisAlignment -> Ausrichtung der Hauptachse
 - start
 - Platziert die Kinder so nah wie möglich am Anfang der Hauptachse.
 - Horizontal wird TextDirection benötigt, um zu entscheiden ob links oder rechts
 - Analog VerticalDirection

Widgets nutzen – Column / Row

- end -> analog wie start
 - Platziert die Kinder so nah wie möglich ans Ende der Hauptachse.
- center
 - Platziert die Kinder so nah wie möglich in der Mitte der Hauptachse.

Widgets nutzen – Column / Row

- spaceBetween
 - Platziert den freien Platz gleichmäßig zwischen den Kindern.
- spaceAround
 - Platziert den freien Platz gleichmäßig zwischen den Kindern sowie die Hälfte davon vor und nach dem ersten und letzten Kind.

Widgets nutzen – Column / Row

- spaceEvenly
 - Platziert den Freiraum gleichmäßig zwischen den Kindern sowie davor und nach dem ersten und letzten Kind

Widgets nutzen – Column / Row

- `crossAxisAlignment` -> Ausrichtung der Querachse
 - `start`
 - Platziert die Kinder so, dass ihre Startkante mit der Startseite der Querachse ausgerichtet ist.
 - `TextDirection` bzw. `VerticalDirection` wird benötigt, um zu entscheiden, ob `start` rechts || links ist bzw. oben || unten

Widgets nutzen – Column / Row

- end -> analog wie start
 - Platziert die Kinder so nah wie möglich ans Ende der Querachse.
- center
 - Platziert die Kinder so, dass ihre Mittelpunkte mit der Mitte der Querachse übereinstimmen. (default)

Widgets nutzen – Column / Row

- stretch
 - Fordern die Kinder auf, die Querachse zu füllen.
 - Dies führt dazu, dass die an die Kinder übergebenen Beschränkungen in der Querachse spezifiziert ist.
- baseline
 - Platziert die Kinder so entlang der Querachse, dass ihre Grundlinien übereinstimmen.

Widgets nutzen – Column / Row

- baseline
 - Für horizontale Hauptachsen vorgesehen. Wenn die Hauptachse vertikal ist, wird dieser Wert wie [start] behandelt.

Widgets nutzen – Text

- Inhalt als einziger positioneller Parameter
 - RichText unter der Haube
 - Normaler Style über Theme
 - Style detailliert anpassbar

Styling – TextStyle

- Als style Parameter eines Text Widgets oder global im Theme
 - fontStyle -> FontStyle.normal / italic
 - fontWeight -> FontWeight.normal / bold [w100...w900]
 - color / backgroundColor
 - fontFamily

Styling – TextStyle

- `fontSize` -> Größe der Glyphen (in logischen Pixeln)
- `textScaleFactor` -> Skalierung [1.5 -> 50% größer]
- `maxLines`
 - `overflow: TextOverflow.clip / ellipsis`

Styling – TextStyle

- decoration
 - TextDecoration.none / underline / overline / lineThrough
- decorationStyle
 - TextDecorationStyle.solid / double / dotted / dashed / wavy
- decorationThickness
- decorationColor

Widgets nutzen – Icons

- Über Icon Widget
- Material Icons über Icons, Cupertino über Cupertinolcons, adaptive Icons mit Icons.adaptive
- Eigene Icons können über Font importiert werden
- Farbe und Größe spezifizierbar

Widgets nutzen – Buttons

- Reagieren auf einen Click
- Benötigen den Parameter onPressed (null := disabled)
 - TextButton & IconButton
 - FloatingActionButton
 - ElevatedButton & OutlinedButton
- .icon (icon, label, onPressed) als ShortCut

Widgets nutzen – Buttons

- Styles setzen über [type]Button. *styleFrom*
 - primary & backgroundColor
 - textStyle
 - alignment & padding
 - maximumSize & minimumSize

Widgets nutzen – Buttons

- ButtonStyle - analog wie zuvor aber Zustandsbasiert
 - MaterialStateProperty
 - .all<T>(...) //
 - resolveWith<T>(MaterialPropertyResolver<T> callback)

Widgets nutzen – Buttons

- Themeing
 - ThemeData
 - [type]ButtonTheme
 - [type]ButtonThemeData
 - [type]Button.styleFrom

Widgets nutzen – Image

- Zeigt ein Bild von einer URL, einer Datei, den Assets oder aus dem RAM an
 - `Image.asset (String assetPath)`
 - Asset muss in die pubspec (PlugIn) - Upgrade
 - `Image.network (String url)`
 - Verarbeitet keine SVGs

Widgets nutzen – Image

- Zeigt ein Bild von einer URL, einer Datei, den Assets oder aus dem RAM an
 - `Image.asset (String assetPath)`
 - Asset muss in die pubspec (PlugIn) - Upgrade
 - `Image.network (String url)`
 - Verarbeitet keine SVGs

Images aus Assets – Assets

- Assets werden in der pubspec.yaml aufgelistet
 - Unter flutter > assets
 - Assets einzeln oder als ganzer Ordner
- Pfad relativ zur pubspec.yaml

Widgets nutzen – Image

BoxFit.cover -> So klein wie möglich, aber dennoch die gesamte Zielbox abdecken.



BoxFit.fill -> Füllt das Zielfeld, indem Sie das Seitenverhältnis der Quelle verzerren.



BoxFit.fitWidth -> Stellt sicher, dass die volle Breite der Quelle angezeigt wird, unabhängig davon, ob die Quelle die Zielbox vertikal überläuft.



BoxFit.fitHeight -> Stellt sicher, dass die volle Höhe der Quelle angezeigt wird, unabhängig davon, ob die Quelle die Zielbox horizontal überläuft.



Widgets nutzen – Image

BoxFit.contain -> So groß wie möglich, während die Quelle immer noch vollständig in der Zielbox enthalten ist.



BoxFit.none -> Richtet die Quelle innerhalb der Zielbox aus (standardmäßig zentriert) und verwirft alle Teile der Quelle, die außerhalb der Beschränkung liegen.



BoxFit.scaleDown -> Dies ist dasselbe wie "contain", wenn das Bild dadurch verkleinert würde, andernfalls ist es dasselbe wie "none"



SVG aus Assets – flutter_svg

- Wir machen uns ein Projekt der Flutter Community zu nutze: flutter_svg
- Zu finden auf https://pub.dev/packages/flutter_svg
- Importieren über cli: dart pub add flutter_svg
- Wird aufgelistet unter dependencies

SVG aus Assets – Assets

- Assets werden in der pubspec.yaml aufgelistet
 - Unter flutter > assets
 - Assets einzeln oder als ganzer Ordner
- Pfad relativ zur pubspec.yaml

SVG aus Assets – Anwendung

- Importieren über Package Name
- Konstruktor SvgPicture.asset('path/to/asset') aufrufen

Assets – Icons importieren

- Wird über Font eingebunden
- Wrapper über Dart Klasse

Assets – Icons importieren

- Generierung über <https://www.fluttericon.com/>
 - Icons selektieren und hochladen
 - .zip herunterladen
 - .ttf in Assets & .dart in lib einbinden
 - Font in Pubspec einbinden (Kopieren von Kommentar in .dart)

Widgets nutzen – CustomPaint

- Zeichnet mit CustomPaint Widget auf einem Canvas
- Kann im Vordergrund & Hintergrund zeichnen
 - CustomPainter painter
 - CustomPainter foregroundPainter

Widgets nutzen – CustomPaint

- Passt Dimension, falls vorhanden, dem Kind an, andernfalls size Parameter
 - Benötigt CustomPainter Instanz
 - Painter muss implementieren:
 - paint(Canvas, Size)
 - shouldRepaint(CustomPainter): bool

Widgets nutzen – CustomPaint

- Canvas hat draw Methoden wie z. B.
 - `drawRect(Rect rect, Paint paint)`
 - `drawOval(Rect rect, Paint paint)`
 - `drawCircle(Offset c, double radius, Paint paint)`

Widgets nutzen – CustomPaint

- Canvas hat draw Methoden wie z. B.
 - `drawRect(Rect rect, Paint paint)`
 - `drawOval(Rect rect, Paint paint)`
 - `drawCircle(Offset c, double radius, Paint paint)`

Widgets nutzen – Center

- Zentriert Kinder
 - Versucht normalerweise den maximalen Platz ein zu nehmen
 - Mit heightFactor & widthFactor nimmt es den entsprechenden Faktor des Platzes des Kindes in Anspruch

Widgets nutzen – Transform

- Transformiert Kind vor dem Zeichnen durch Rotation, Skalierung oder Repositionierung
- Wird vor dem Rendering, allerdings nach dem Layouten vollzogen
 - `Transform.translate({required Offset offset ...})`
 - `Transform.rotate({required double angle ...})`

Widgets nutzen – Transform

- `Transform.scale({required double scale ...})`
- `Matrix4 transform`
 - Beispiel
 - `Matrix4.rotationX(30 * pi / 180)`
 - `..rotateZ(30 * pi / 180)`
 - `..setTranslationRaw(100, -0, 0),`

Widgets nutzen – Stack

- Zeigt mehrere Widgets übereinander
- Kinder werden ohne eingreifen in der oberen linken Ecke dargestellt
- Positioned Widget ist in den meisten Fällen empfohlen zu verwenden,
 - da bei fit: Expand ansonsten ein Widget tieferer Ebenen dominieren kann
- Passt sich zuerst an die Größe von nicht Positioned Kindern an

Widgets nutzen – Positioned

- Positioniert Kinder in einem Stack
- Positionierung durch abstand zum Rand oder Dimension
 - Positioned({ left, top, right, bottom, width, height, })
 - Positioned.fill({ left, top, right, bottom })
 - Positioned.directional({ required TextDirection textDirection, start, top, end, bottom, width, height})

StatelessWidgets – Was ist das?

- Eine Immutable (Unveränderbare) Klasse, die von StatelessWidget und damit von Widget erbt
- Organisiert Widget Tree in wiederverwendbare Klassen
- Stellt BuildContext Objekt bereit

Das BuildContext Objekt

- Ist ein Element
 - Beinhaltet das Element des Widgets als Kind
- Gibt einen Kontext für das Widget
- Kann genutzt werden, um an Informationen über die Position im Widget Tree zu kommen (Theme, MediaQuerys)

Widgets nutzen – Builder

- Stellen ein BuildContext Objekt bereit
- Lassen auf Änderungen reagieren
- Builder spezifisch für bestimmte Änderungen (Orientation, Layout, BLoC, ...)

MediaQuerys

- Liefern Daten über das Gerät
- Benötigen den BuildContext
 - Class.of(BuildContext) Methode bei allen statischen Klassen von Flutter, die den Kontext benötigen

InheritedWidgets – Wieso .of(context)

- Bereitstellen von Informationen für alle Widgets am eigenen Teilbaum im WidgetTree
- Bereitstellung erfolgt über BuildContext Objekt
- Widget muss von InheritedWidget erben

InheritedWidgets – Wieso .of(context)

- .of(context) als Konvention, um context.dependOnInheritedWidgetOfExactType() auf zu rufen
 - Gibt nächstgelegene Instanz im WidgetTree des InheritedWidget zurück
 - null, wenn nicht vorhanden

Exkurs: Hot Reload

- Änderungen der Anwendungen zur Laufzeit
- Funktioniert nicht in main oder init Methoden
(siehe <https://flutter.dev/docs/development/tools/hot-reload#special-cases>)
- Benötigt entsprechend Stateless- oder Statefull- Widgets

Widgets nutzen – FutureBuilder

- Builder, der nach abgeschlossenem asynchronen Ereignis erneut die Builderfunktion ausführt
- Nimmt generischen Typen (als nullable)
- Typischerweise um Lade Animationen zu zeigen, während auf eine Webrequest gewartet wird
- Snapshot Objekt weiß, ob es Daten hat, mit einem Fehler beendet wurde, oder noch am laden ist

Widgets nutzen – ListView

- Zeigt eine Liste von Widgets, die gescrollt werden kann
- ListView.builder() Konstruktor baut Kinder aus Index
 - Braucht 'builder' Parameter als Widget
 - Funktion(context, index)
 - Retuniert generierte Elemente
 - Ist lazy-loaded

Widgets nutzen – ListView

- Es muss definiert werden wie viele Kinder erzeugt werden sollen
- Kinder sind lazy-loaded
 - Wiederverwendung

Widgets nutzen – ListView

- ListView.seperated() ist analog zu .builder()
- Allerdings mit einem zusätzlichen builder,
 - Der Seperatoren (Kindes Elemente)
 - zwischen den jeweiligen Kindes Elementen baut

Widgets nutzen – ListView

- scrollDirection => Ausrichtung der Liste, Also die Richtung, in die gegeben falls gescrollt werden kann
- physics => Scrollverhalten
 - AlwaysScrollableScrollPhysics() => Kann gescrollt werden
 - NeverScrollableScrollPhysics() => Kann nicht gescrollt werden
 - ...

Widgets nutzen – ListTile

- Material für ein Item in einer Liste
- Ist unabhängig von Liste
- Hat Positionen wie ‚leading‘, ‚trailing‘, ‚title‘, ‚subtitle‘
- Hat color, padding, ... als styling attribute

Widgets nutzen – GridView

- Zeigt ein Grid aus Widgets
- Kann gescrollt werden
- Scrollverhalten analog zu ListView

Widgets nutzen – GridView

- GridView.count()
 - crossAxisCount als Spaltenanzahl
 - Children für die darzustellenden Kinder

Widgets nutzen – GridView

- GridView()
 - Benötigt gridDelegate & Liste von Kindern
 - gridDelegate ordnet Grid an, in unserem Fall bleiben wir bei SliverGridDelegateWithFixedCrossAxisCount

Widgets nutzen – GridView

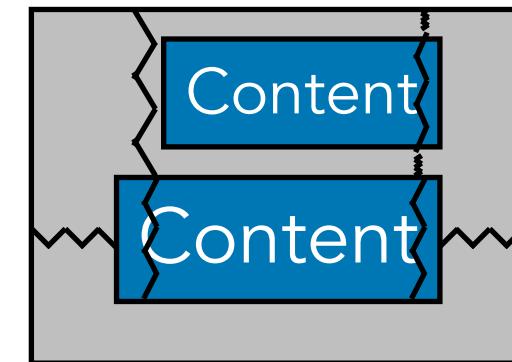
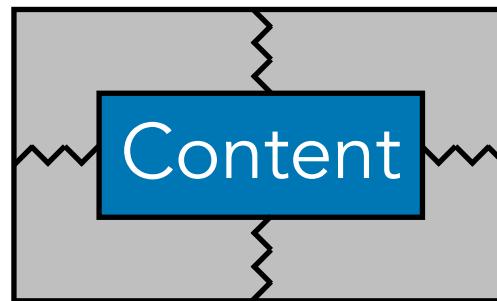
- `GridView.builder()`
 - Benötigen `gridDelegate`, `itemBuilder` & `itemCount`
 - `lazy loading`
 - `itemBuilder` nutzt `SliverDelegate`, um Kinder zu bauen
 - `Sliver` folgen auf den nächsten Seiten

Sliver - Einstieg

- Sliver sind spezielle Widgets, die mit [CustomScrollView] kombiniert werden können , um benutzerdefinierte Bildlaufeffekte zu erstellen.
- RenderSliver anstatt RenderBox
- Sind lazy loaded, wenn Builder benutzt werden
 - Instanziierung erfolgt wenn View benötigt
- Low level Rendering Engine

Render Box vs. Render Sliver

- Positionierung durch Größe und Platz
- Positionierung durch Größe, gescrollte Distanz, verbrauchte Distanz vorheriger Sliver, ...



Sliver – versteckt in ListView

- Kann über den
 - ListView.custom Konstruktor direkt Sliver anzeigen
- ListView ist also ein Wrapper um SliverList in CustomScrollView zu erzeugen (austauschbar)

Sliver – versteckt in GridView

- Nutzt immer Sliver im Hintergrund
- Wrapper um SliverGrid in CustomScrollView (austauschbar)

Sliver – CustomScrollView

- Erstellt eine [ScrollView], die benutzerdefinierte Bildlaufeffekte mit Sliver erstellt.
- 'Viewport' für Sliver in RenderBox Umgebung
- Nimmt Sliver im slivers Array (im sliver Kontext nicht mehr children) und rendert diese mit dem SliverProtocoll
- Notwendig, um Sliver zu benutzen (Wenn auch manchmal nur im Hintergrund)

Sliver – SliverAppBar

- Erstellt eine Materialdesign-App-Leiste, die in einer [CustomScrollView] platziert werden kann.
- Floating, Pinned oder Verschwinden sind möglich

Sliver – SliverPadding

- Erstellt einen Sliver, der auf jeder Seite eines anderen Sliver angewendet werden kann.
- Ist ein Sliver, der einen anderen Sliver als Argument nimmt
- Setzt einen Padding um sein Kind

Sliver – SliverGrid

- Equivalent zu GridView (GridView ist Wrapper um SliverGrid)
- Zeigt ein Grid im Scroll Container
- Nutzt GridDelegates, um Kinder zu rendern

Sliver – SliverToBoxAdapter

- Sliver sind spezielle Widgets, die mit [CustomScrollView] kombiniert werden können , um benutzerdefinierte Bildlaufeffekte zu erstellen.
- Ein [SliverToBoxAdapter] ist ein einfacher Sliver, der eine Brücke zurück zu einem der üblichen boxbasierten Widget (z.B. Container) herstellt.

State Management

- Ein System, um den Zustand eines Widgets zu beschreiben
- Widgets sollen nicht unveränderbar bleiben
- Elemente werden basierend auf neuem Zustand angepasst oder neu erstellt

State Management – StatefulWidgets

- Flutters native Lösung
- Ähnlich, wie StatelessWidget, besitzen allerdings ein State Objekt
 - Widget verfügt über Keine `build` Methode
 - Erzeugt ein State über `createState()`

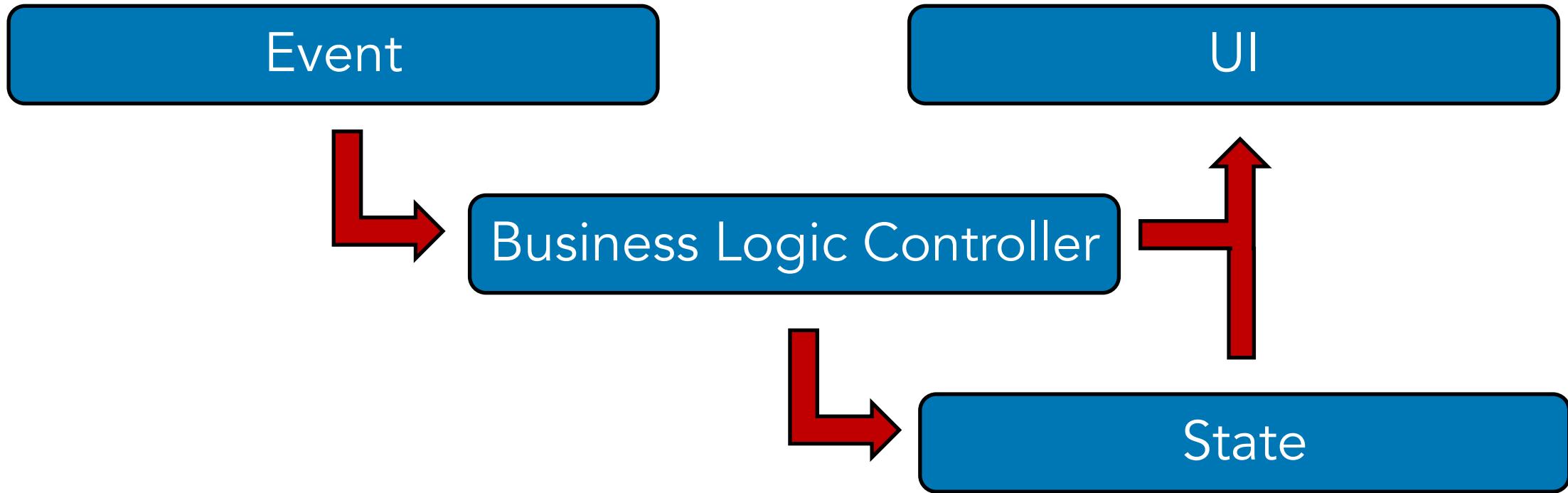
State Management – StatefulWidget

- Ein Zustand erbt von `State<T>`
 - Verfügt über änderbare Attribute
- `setState` markiert den Frame als dirty
 - Kann in einen Handler-Funktion die veränderbaren Attribute anpassen
- `widget` – referenziert (late) auf das Widget, das den State nutzt.

State Management – StatefulWidgets

- State wird durch äußere oder innere Einflüsse verändert
 - Benutzerinteraktionen
 - Animationen
 - I/O

State Management – BLoC



BLoC – Bloc

- Erben der Klasse Bloc<Ereignisse, Zustand>
- Bildet Events auf States ab
 - Beide sind generische Parameter von Bloc
 - <Ereignisse, Zustand>
 - Keine Bedingungen für die Klassen, erben von Equatable ist empfohlen (Vergleichbar)

BLoC – Reagieren auf Ereignisse

- Im Konstruktor-Body „Listener“ registreren
- Aufrufen von on<Event>(fn)
 - Funktion erhält (event, emit)
 - Event – beinhaltete Daten des Ereignis
 - emit kann neuen Zustand setzen

BLoC – Provider

- Ist ein InheritedWidget
- Stellt einen Bloc bereit
 - BlocProvider(create, child)
 - Ist lazy loaded
- BlocProvider.of<MyBloc>(context), um auf Bloc zu greifen

BLoC – Builder

- BlocBuilder analog zu Builder
 - Liefert Zustand (State) eines Blocs
 - Und den Context
 - Benötigt Bloc & State als generische Parameter

BLoC – Ereignisse ausführen

- BlocProvider.of<MyBloc>(context)
 - .add (
 - Ereignis)

BLoC – Cubit

- Cubit ist eine BloC-Form die ohne Ereignisse auskommt
- Stattdessen werden Methoden auf dem Cubit angestoßen
- Welche via `emit` den neuen Zustand sendet!

Gestenerkennung

- GestureDetector Widget
- Callback-Methoden für verschiedenste Gesten mit Payload des Ereignisses als Parameter
- Gesten können jederzeit Sieg oder Niederlage über eine andere Geste erklären
- Gesten werden immer vom GestureDetector, der im Pfad am weitesten entfernt von root ist gehandhabt

Gestenerkennung

- Überblick der vorhandenen Handler:
 - onTap, onDoubleTap ->
 - onForcePressPeak: -> (ForcePressDetails details) =>
 - details.pressure 0.0 -> 1.0

Gestenerkennung

- onHorizontalDragEnd
 - details.primaryVelocity
- onHorizontalDragUpdate
 - details.primaryDelta
- Analog Vertical
- ...

Animation – Animierte Widgets

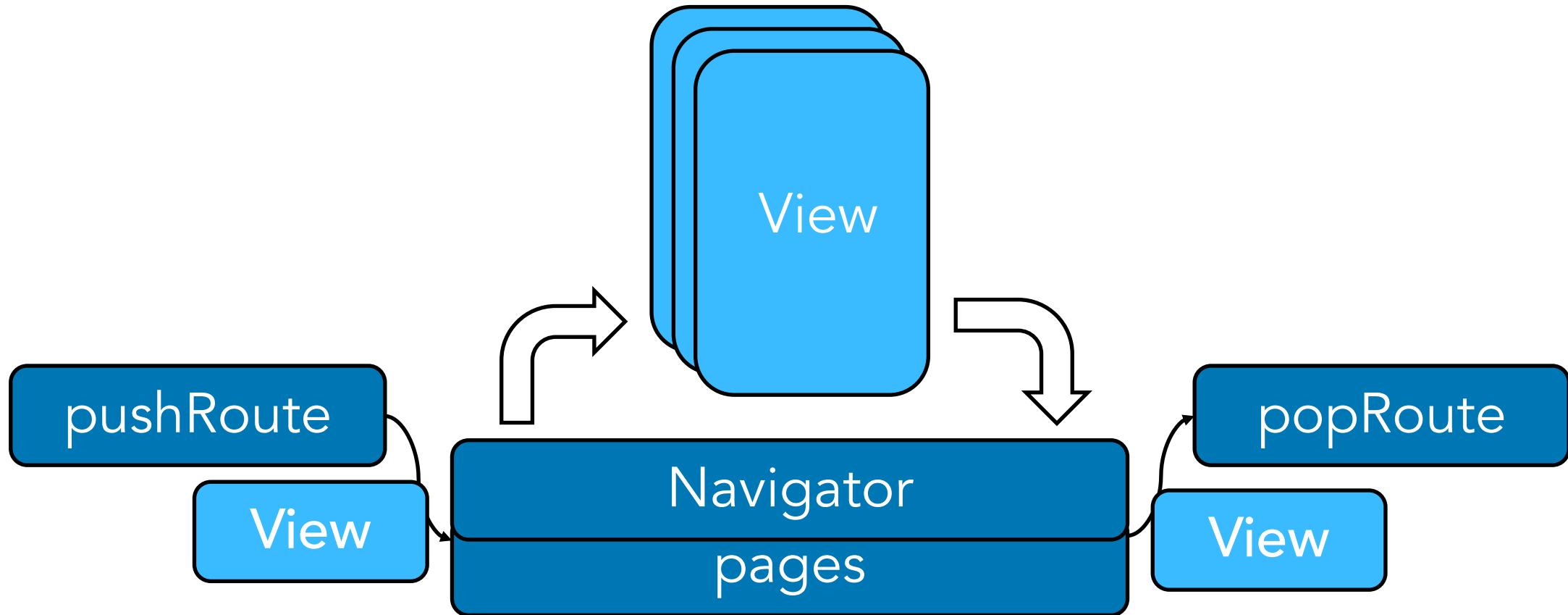
- Animieren ihre Werte automatisch
- Benötigen duration Parameter
 - duration: const Duration(milliseconds: 500)
- Animation passt sich dynamisch neuen Werten an
- Die meisten Widgets haben eine animierte Variante

Animations - AnimationController

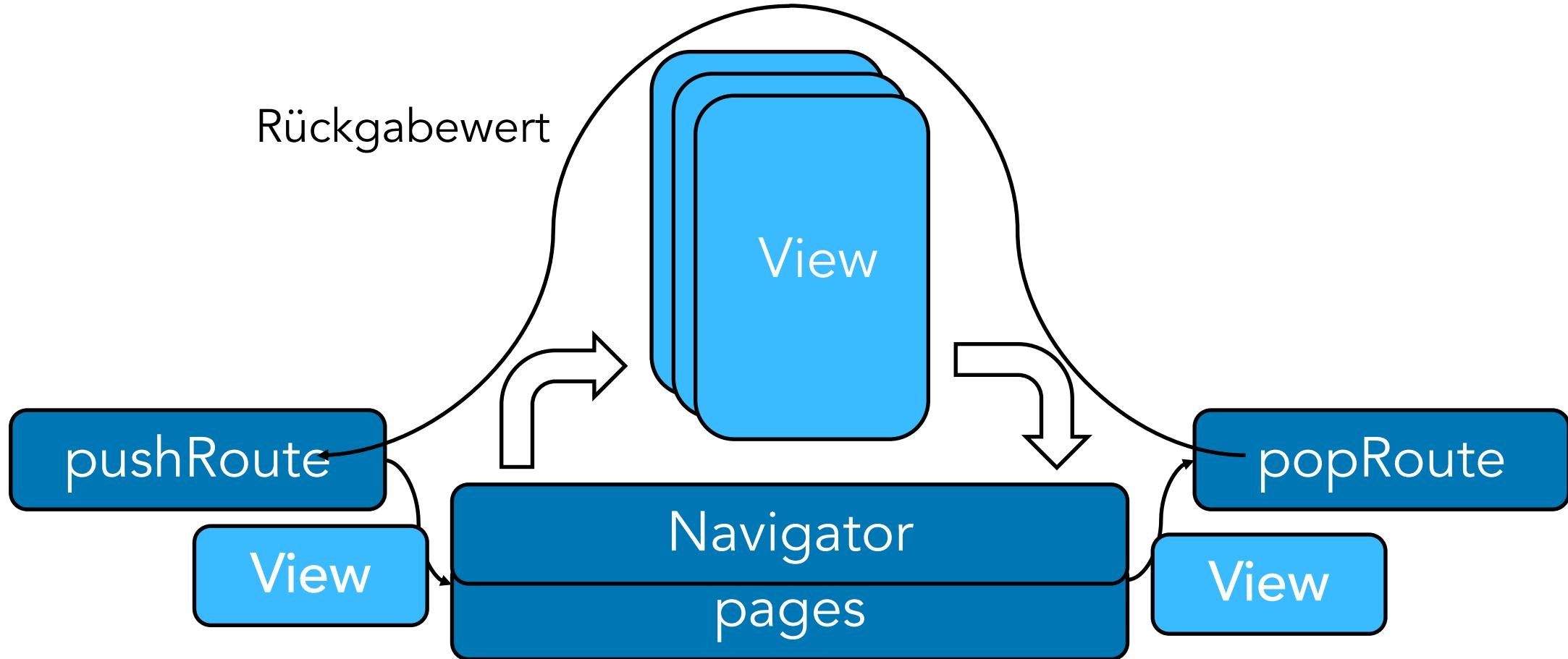
- Generiert Werte in einer definierbaren Reichweite (normalerweise zwischen 0 & 1) über einen gegebenen Zeitraum
- Kann in beide Richtungen animieren
- Sollte nach Gebrauch disposed werden

Navigation & Routing

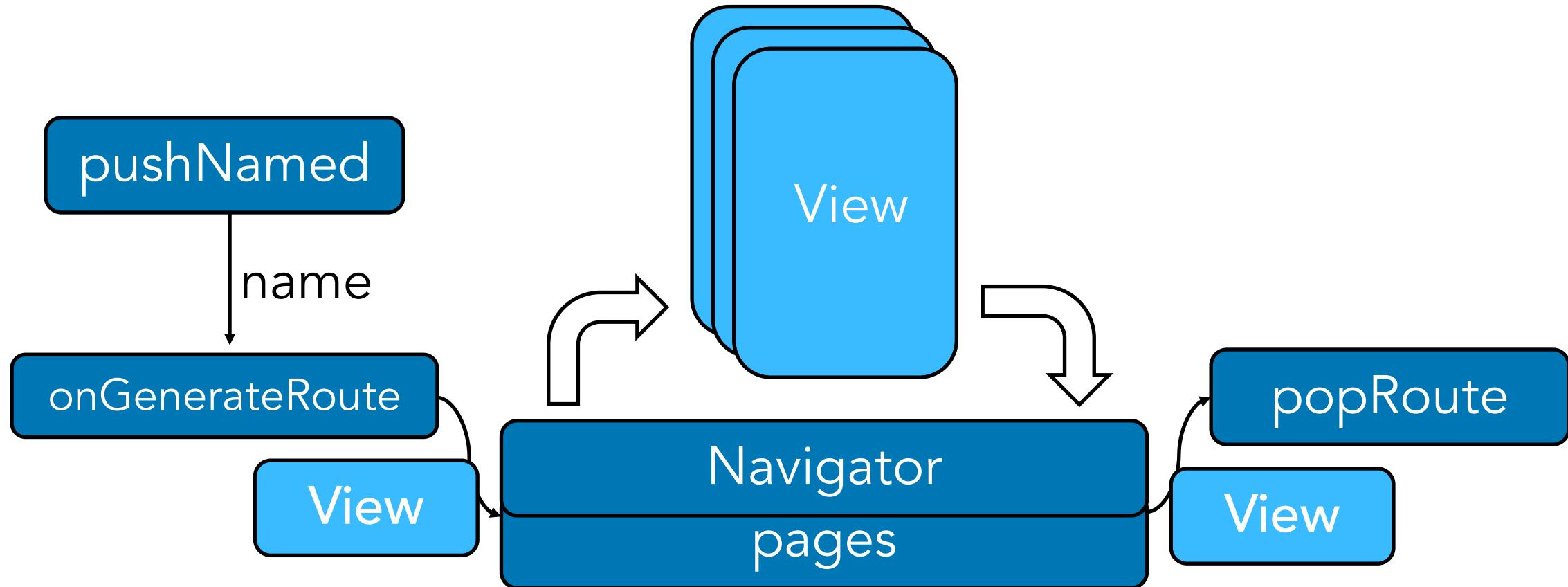
Navigator 1.0



Navigator mit Rückgabewert



GenerateRoute



Navigator – als Home der MaterialApp

- MaterialApp { home: Navigator }
 - Navigator
 - initialRoute -> Name der initialen Route
 - onGenerateRoute: RouterFactory
 - Route<dynamic>? Function(RouteSettings settings)
 - Gibt eine Route [MaterialPageRoute] abh. Der RouteSettings zurück

Drawer

- Icon in App Bar oder Drag Geste zum öffnen
- Container mit Navigationselementen
- Klassisches Material Design zur Navigation

Tabs

- Tab bar am unteren Rand des Bildschirms
- Klassisches Material oder Cupertino Design zur Navigation

Dialog

- Nur für wichtige Informationen / Entscheidungen
- Mit bedacht verwenden
- Erstellen eigenes Element in den Pages des Navigators

Heros

- Automatische Animation bei Navigation
- 'Fliegen' von altem Widget zum neuen
- Widgets benötigen selben tag

Testing

1

Unit

Testet eine abgeschlossene Einheit, also Klasse, Methode oder Funktion

2

Widget

Testet ein Widget durch simulierte Interaktionen, ohne großen Overhead

3

Integration

Testet die gesamte Interaktion der App in sich selbst

Unit Testing

- Testen abgeschlossenes System
- Grundlage des TDD (Test Driven Development)
- Enden mit "_test.dart"
- Rufen etwas auf und erwarten ein bestimmtes Ergebnis
- Lassen sich in Gruppen aufrufen

Mockito

- Ersetzt eine Klasse durch eine automatisch generierte, gemockte Version der Klasse
- Speichert alle Interaktionen auf der Mock Klasse
- Sinnvoll fürs testen mit einer externen, asynchronen, und / oder Fehleranfälligen Schnittstelle

Widget testing

- In anderem Kontext auch Component testing genannt
- Unterliegen den selben regeln wie Unit Tests
- Bauen ein Widget, ohne den Overhead der ganzen App
- Simulieren UI interaktionen

Integration testing

- Startet die App mit driver Extension
- Tests im "test_driver" Ordner
- Aktionen werden über 'FlutterDriver' in der App ausgeführt
- Profiling ist möglich
- "**flutter drive**", um Tests zu starten