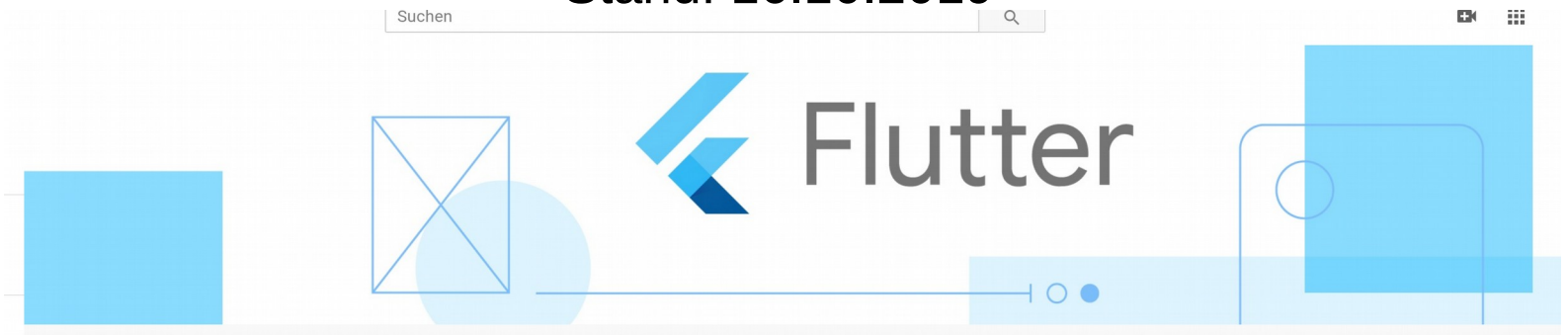
Einführung in Flutter/Dart

Apps entwickeln für Android und iOS (und ja, auch noch fürs Web)

Prof. Dr. Jan-Torsten Milde

Stand: 16.10.2019



Organisatorisches

Ablauf des Kurses

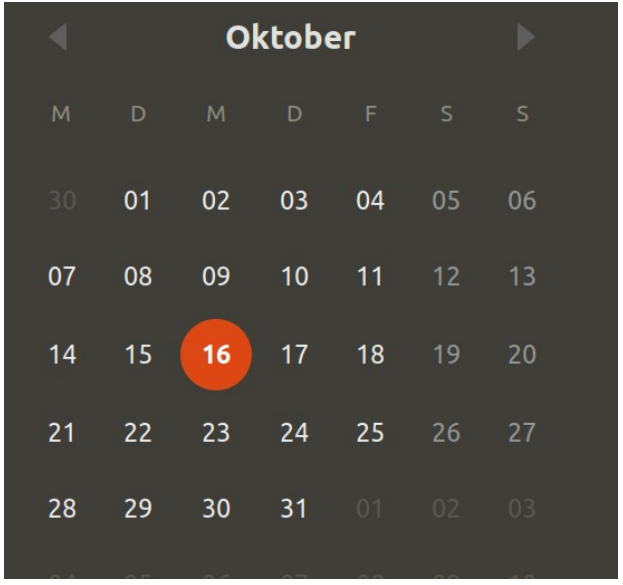
- Einführung
  - Tools: AndroidStudio, VSCode, GitHub, XP Designer, Google Docs und Slack
- Übersicht Widgets
  - Basis Widgets
  - Container Widgets: ListView, SliverApp, Grid, Row, Column
- Styling der App
  - Farbe, Padding, Textstyle, Themes
- Interaktion und Navigation
  - Button, Switch, Slider etc
  - Navigator
- Asynchrone Programmierung und Animation
  - Anbindung von REST API
- State in einer App:
  - Bloc als Ansatz zur Verwaltung des Zustands

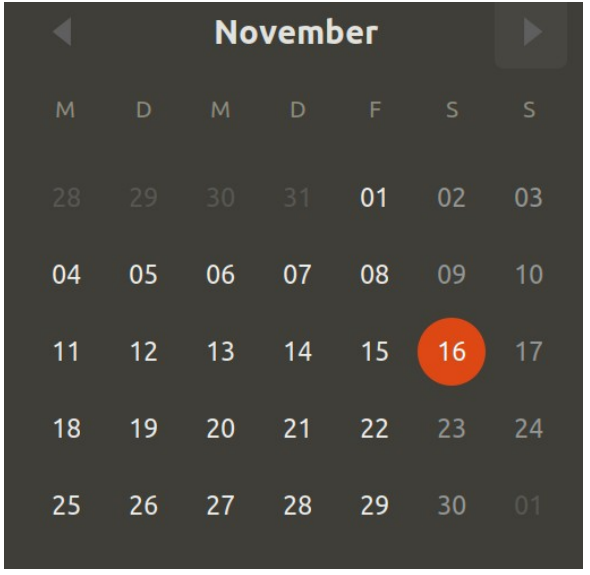
- Vorstellung Projektideen (5 Wochen vor Ende des Semesters)
  - Zielsetzung
  - Skizzierung der Oberfläche und der Interaktion
- Projektphase
  - 4 Sprints a jeweils 1 Woche
- Abschlusspräsentation
  - Vorstellung Ergebnisse und Erläuterung der Eigenanteile

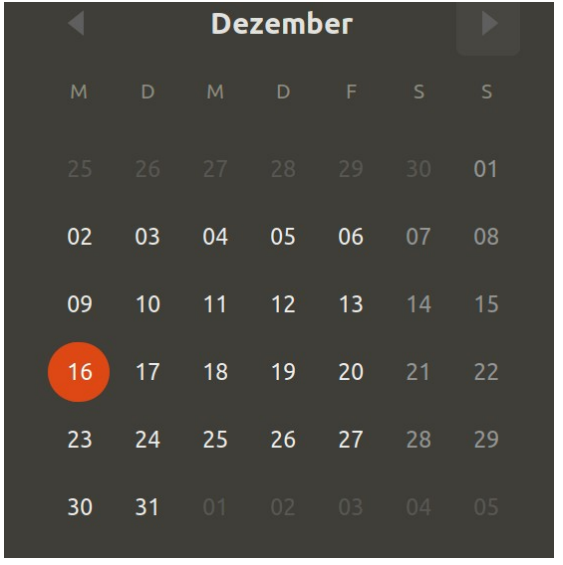
Zeitplan

9 Wochen bis Weihnachten: 6 **Arbeitsblätter bis Anfang der Weihnachtspause**

5 Wochen im neuen Jahr: **Projektphase, beginnend mit dem neuen Jahr**







Arbeitsweise im Kurs

- Projektorientierte Arbeitsweise
  - Jede Woche soll eine App entstehen/weiterentwickelt werden
    - Schrittweise Erhöhung der Komplexität
  - (6) Arbeitsblätter geben die Zielsetzung vor
  - Zusätzlich müssen jede Woche vorgegebene Lehrvideos „erarbeitet“ werden und Fragen zu diesen beantwortet werden
- Teamorientiertes Arbeiten
  - Grundlage sind agile Vorgehensmodelle
  - Pairprogramming
  - Projekte werden hochgeladen, Arbeitsfortschritte sollen sichtbar sein
- Abschließendes Projekt
  - Selbst gewähltes Thema
  - Insgesamt 5 Wochen von der Idee zum Ergebnis
  -

Einstieg: Onlineressourcen

- Flutter Cookbok
  - https://flutter.dev/docs/cookbook
- Flutter Channel
  - https://www.youtube.com/channel/UCwXdFgeE9KYzlDdR7TG9cMw
  - Hier im Besonderen die Playlist: Widget of the week

Übungen

Übungen

- Übung 0: KursApp: private Seite
  - Zielsetzung: eigene Seite mit Namen (Text), Bild (Image), Container, Farbe, Textstyle
  - Einführung von GitHub
  - Kennenlernen von AndroidStudio
  - Generierung der BasisApp

Übungen

- Übung 1: WidgetApp: Widgets vorstellen
  - Jeder Studierende stellt ein Widget aus einer Liste vor
    - Minimale Anwendung: zeigt die Verwendung im Kontext einer kleinen App
    - Zweck des Widgets
    - Live Coding!
    - Max. 5 Folien

Übungen: Liste der Widgets

- Scaffold
- AppBar
- Container
- Material
- Text
  - TextStyle
- Padding
- ListView
  - ListTile
- SingleChildListView
- Row
- Column
- Stack
- Card
- GridView
  - GridList
- RaisedButton
- Switch
- FormField und Textfield
- TabView
  - TabBar
- SnackBar
- NavigationBar
- SizedBox
- Image
- Drawer
- InkWell
- FadeTransition
- SlideTrabstion

Übungen

- Übung 2: StylingApp: Arbeiten mit Farben, Assets, Styles und Themes
  - Es soll eine App entwickelt werden, in der Screens implementiert werden, in der Farbschemata, Textstyles und Images aus unterschiedlichen Quellen angezeigt werden.

Übungen

- Übung 3: UXApp: Oberfläche analysieren und nachimplementieren
  - Aus einer Auswahl existierender Oberflächen wählen die Studierende eine aus, und reimplementieren diese (so akkurat, wie möglich)
- Zu analysierende Apps:
  - TafelApp, WhatsApp ….

Übungen

- Übung 4: StateApp: Asynchrones Programmieren
  - Anbindung eines externen REST API
  - Verarbeitung von JSON
  - Darstellung der Daten in einem entsprechend ausgelegten Widgetbaum

Übungen

- Übung 5: SensorApp: Flutter und Gerätesensoren
  - CameraApp: Einbindung der Camera
  - GyroApp: Einbindung des Gyros
  - SensorApp: Einbindung der Sensoren
  - Einbindung externer Bibliotheken

Übungen

- Übung 6: Abschlussprojekt
  - In dem fünfwöchigen Abschlussprojekt soll eine komplexere Anwendung entstehen
    - Mehrere Screens
    - Durchgängiges Design
    - Eine Animation (können auch mehrere sein, aber die App soll nicht überladen werden)
    - Gegebenenfalls Anbindung an Backend

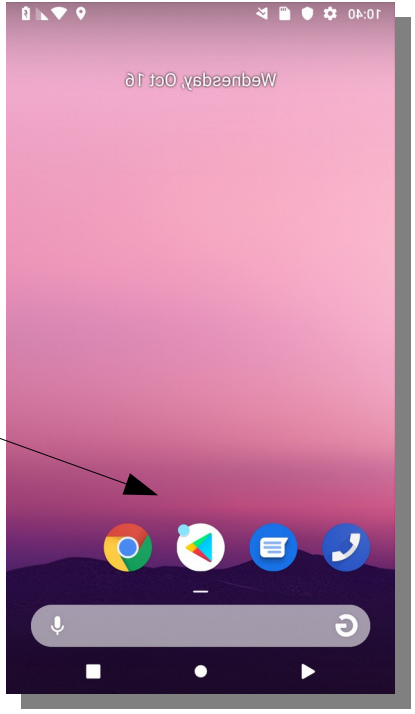
Tools

Sie brauchen

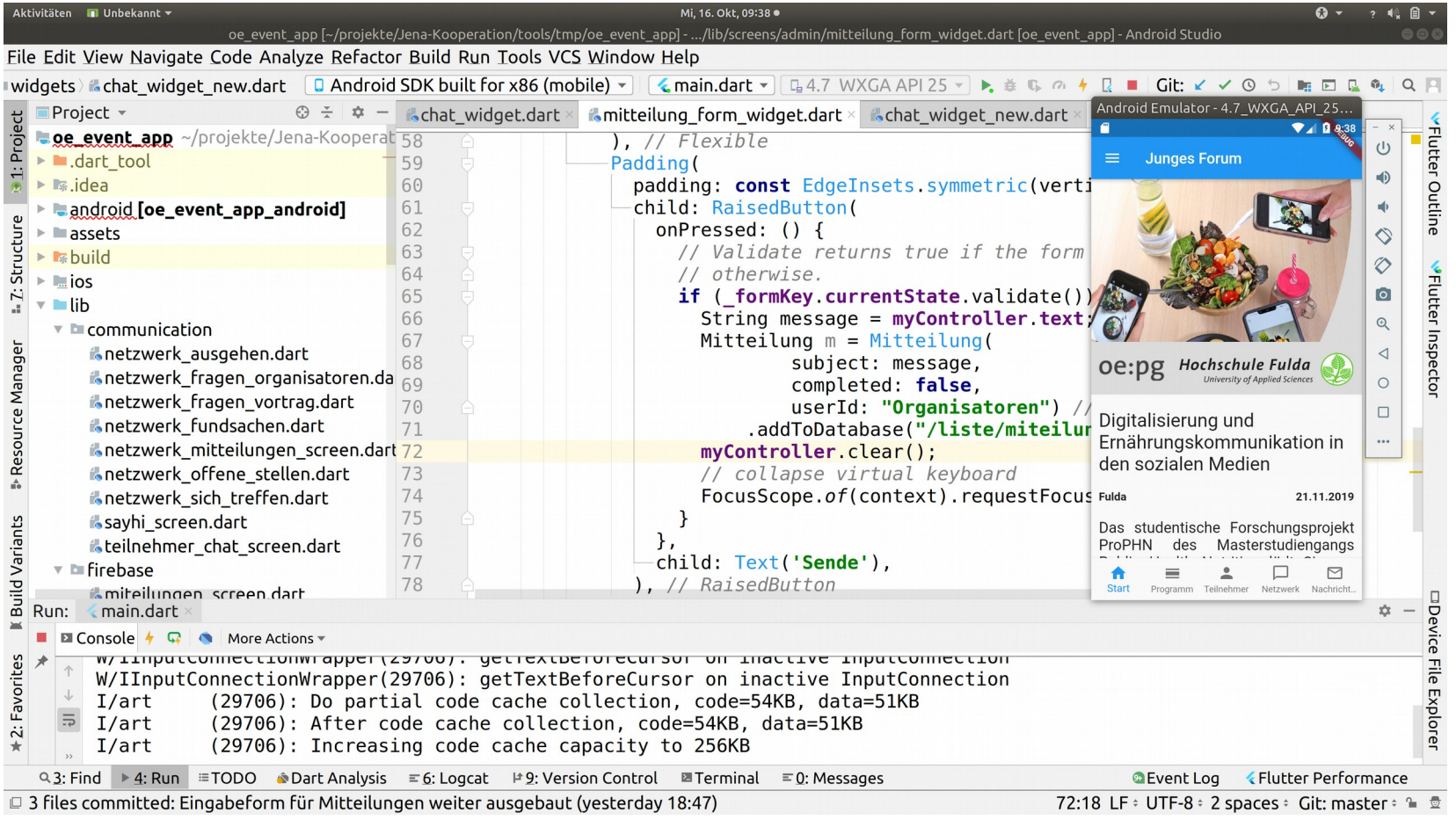
- Software, die unbedingt notwendig ist
  - Flutter mit Dart
  - Java SDK
  - AndroidStudio
  - Git
  - Github (also einen Account)
- Hardware, die Sie unbedingt brauchen
  - Einen vernünftigen Entwicklungsrechner, egal welches Betriebssystem
  - Ich persönlich ziehe ja Linux vor … :)

Sie brauchen

- Software, die hilfreich ist
  - Bildbearbeitung
  - Postman, zum Checken der HTTP Requests
  - Einen Google Account (für Firebase und sowieso: ohne Mutter Google läuft nix)
  - Eine Reihe von Demoprogrammen: im Playstore nach Flutter suchen)
    - Flutter Catalog, Flutter Widget Guide, Flutter Explorer, Flutter Expert, Flutter UIs
- Hardware, die hilfreich ist
  - Ein reales Androidgerät, das für die Entwicklung freigeschaltet ist
  - Ein USB Kabel
  - Eine Halterung für das Gerät, die das Gerät fixiert ohne das Kabel zu verknicken

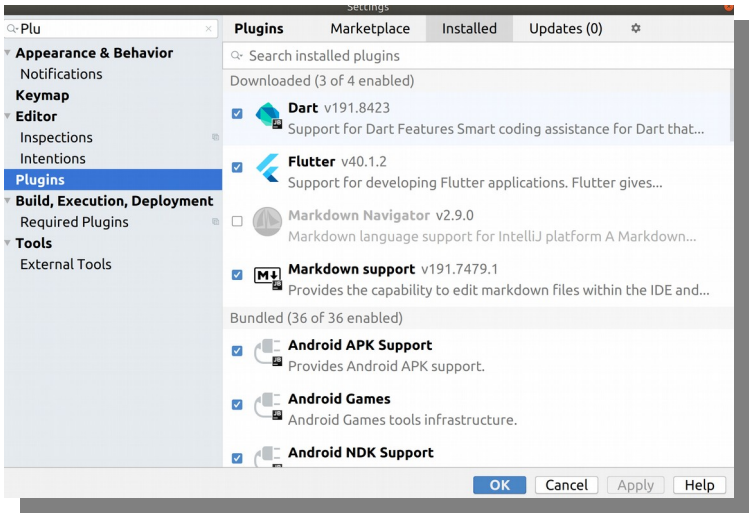


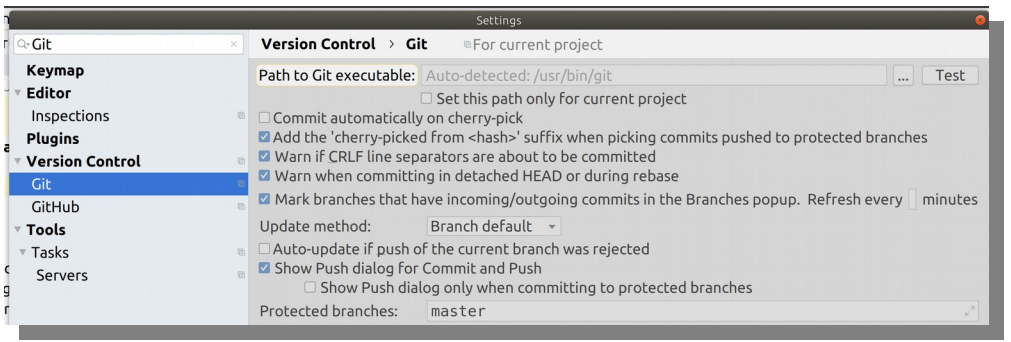
AndroidStudio und Emulator



AndroidStudio

- Die komplette Entwicklung läuft in AndroidStudio ab
  - Nutzen der Plugins für Flutter und Dart
  - Einbindung von Git und Github





Arbeitsumgebung: Einrichtung

- Zunächst muss die Arbeitsumgebung eingerichtet werden
  - Auf dem Rechner muss das Java SDK installiert sein
- Schritt 0: Runterladen und Installieren der aktuellen Version von AndroidStudio (v3.5.1, Stand 16.10.2019)

Arbeitsumgebung: Einrichtung

- Schritt 1: Einrichten eines Verzeichnisses für Flutter
  - C:\flutter
- Schritt 3: Starten von AndroidStudio
- Schritt 4: Erzeugen eines leeren Projektes (nur notwendig für die folgenden Schritte, das Projekt kann später gelöscht werden)
- Schritt 5: Installation der Plugin für Flutter und Dart

Arbeitsumgebung: Einrichtung

- Schritt 6: git auf dem Rechner installieren
- Schritt 7: AndroidStudio neu starten zur Aktivierung der Plugins
- Schritt 8: Testen, ob git gefunden wird und läuft
- Schritt 9: Github Account eintragen
  - Entweder den Account angeben, oder ein Token generieren lassen
- Schritt 10: Ein neues Flutter Projekt erzeugen
  - **Schritt 10a**: beim ersten Mal Flutter herunterladen und in dem vorbereiteten Verzeichnis (c:\flutter) installieren

Arbeitsumgebung: Einrichtung

- Schritt 11: Über den AVD Manager mindestens 1 virtuelles Gerät einrichten
  - Ihr müsst dann ein Android Image runterladen … macht ein Gerät mit AndroidX und noch ein Gerät mit Nougat … und sucht ein Image mit GooglePlay raus … dann kann beliebige Software aus dem AppStore runtergeladen werden.
- Schritt 12: Reales Device für die Programmierung über USB vorbereiten
  - Android: dazu müssen die Entwickleroptionen freigeschaltet werden … 7 mal auf die Versionsnummer tippen :)
  - iOS: keine Ahnung !!!!
- Schritt 13: Emulator mit dem erzeugten virtuellen Gerät starten oder reales Device über USB verbinden
- Schritt 14: Das generierte Flutterprogramm starten und im (virtuellen) Gerät testen

Grundaufbau einer Flutter Anwendung

Zugrundeliegender Ansatz

- Alles ist Widget (jedenfalls in Flutter, und ja, nur fast alles :))
  - Ein Widget bezeichnet eine Komponente, welche in den Widget Tree der Applikation eingebunden wird
  - Viele Widgets repräsentieren Elemente des UI
    - sie können aber u.a. auch visuelle Eigenschaften, Animationen, Übergänge und Interaktionscontroller sein
  - Flutter ist in Dart implementiert und nutzt dessen Syntax zur Beschreibung des Widget Trees
  - Für den Programmierer entsteht so der Eindruck, im Wesentlichen deklarativ zu arbeiten
    - In Flutter beschreibt man in großen Teilen die Struktur der Benutzerschnittstelle
- Flutter unterscheidet grundsätzlich zwei Arten von
  - StatelessWidget
  - StatefulWidget

Flutter auf der Kommandozeile

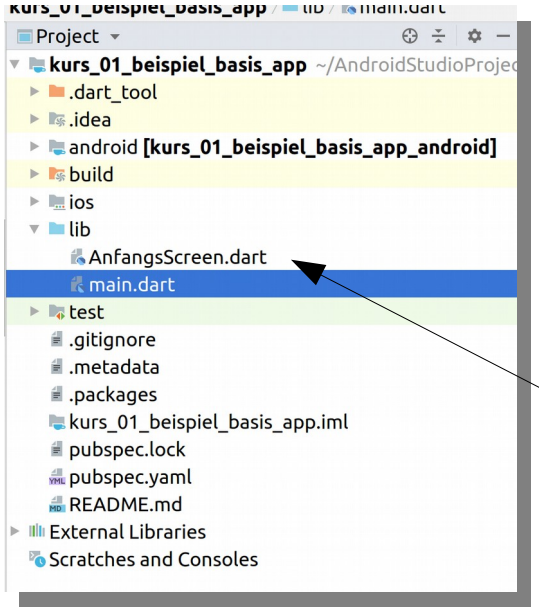
- Weite Teile der Verwaltung von Flutter als Programmierumgebung werden über Kommandozeilenbefehle gesteuert
  - cmd in Windows, bash in Linux und iOS
  - Der Aufbau der Befehle orientiert sich Systemen wie npm oder apt-get, also Paketverwaltungssystemen
- Über die Kommandozeile kann man u.a.
  - Eine Projektgrundstruktur erzeugen
  - Das Programm compilieren und starten (im Emulator oder auf dem Device)
  - Pakete installieren
  - Flutter upgraden und konfigureren

Flutter erklärt am Beispiel

Erzeugen der BasisApp

- AndroidStudio erzeugt eine vollständige Projektstruktur für die Flutter Anwendung
  - Sie ist eine gute Grundlage für Anfänger, um ein erstes Verständnis zu erlangen, aber insgesamt schon zu komplex
- Man löscht daher die komplette Logik bis auf die
  - **`main`** Methode und das
  - **StatelessWidget**, das eine MaterialApp erzeugt
  - Den Namen des Widgets kann man **refaktorieren** (in Android Studio mit **`SHIFT+F6`**)
- In einer neuen Datei erzeugt man den eigentlichen Inhalt der Seite, im folgenden Beispiel als AnfangsScreen bezeichnet
  - Konvention: in diesem Kurs werden Bildschirmseiten als **Screens** bezeichnet

Die BasisApp



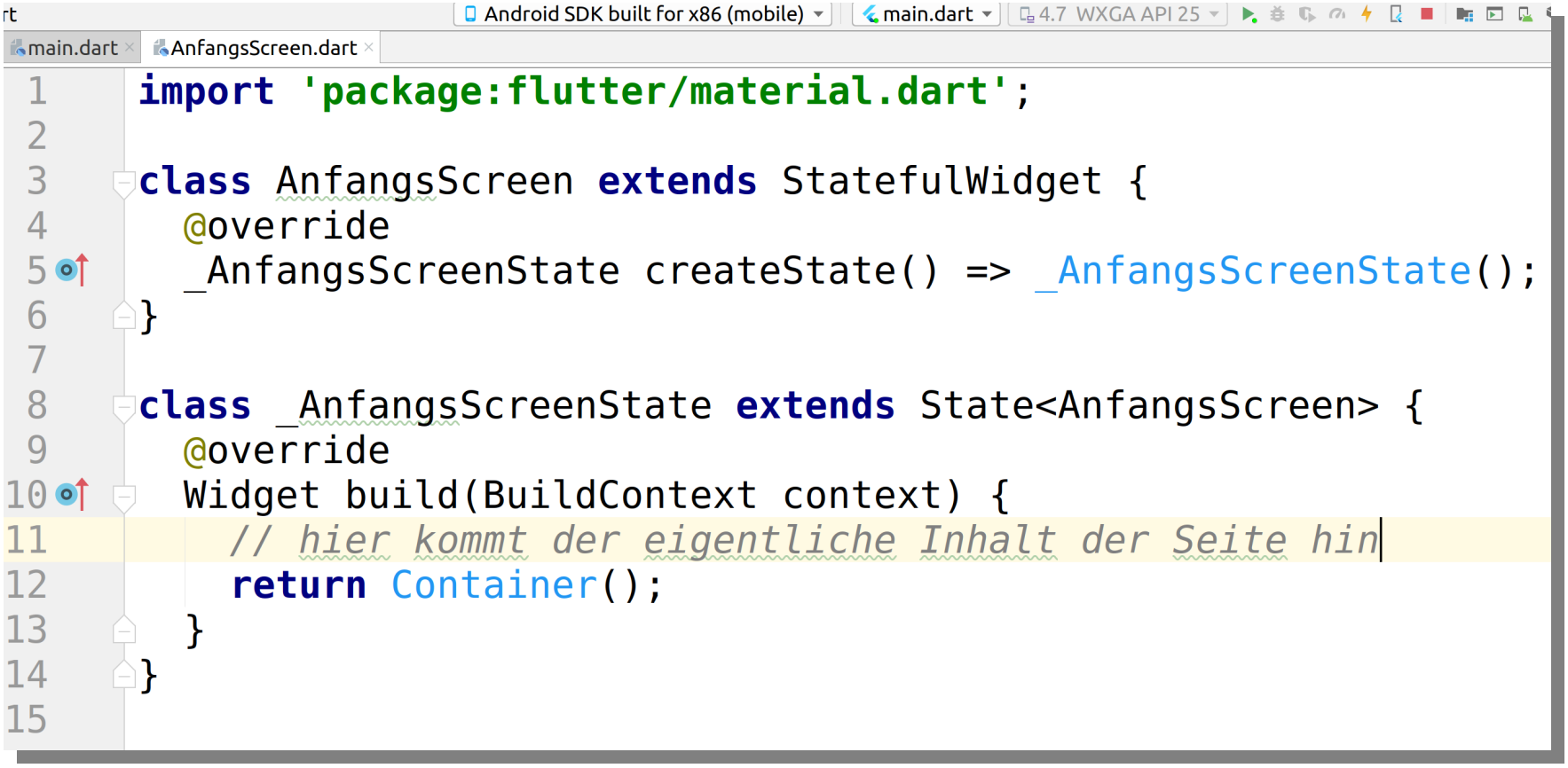
main.dart | AnfangsScreen.dart

```dart
import 'package:flutter/material.dart';
import 'package:kurs_01_beispiel_basis_app/AnfangsScreen.dart';

void main() => runApp(BasisApp());

class BasisApp extends StatelessWidget {
  // This widget is the root of your application.
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Basis App',
      theme: ThemeData(
        primarySwatch: Colors.blue,
      ), // ThemeData
      home: AnfangsScreen(), // Hier kommmt der Startscreen hin
    ); // MaterialApp
  }
}
```

AnfangsScreen

- Der AnfangesScreen ist hier als StatefulWidget umgesetzt
  - Muss nicht so sein, hat aber den Vorteil, dass, wenn man den Zustand zur Laufzeit ändert,
    - der Screen neu gezeichnet wird und sich so aktualisiert
  - dazu muss man die Methode **`setState()`** aufrufen (dazu später mehr)
- Bis jetzt liefert der AnfangsScreen lediglich einen Container() zurück → damit bleibt der Screen einfach nur schwarz

Der (leere) AnfangsScreen



NotYetImplemented Widget

- Um einen Screen als noch nicht Implementiert zu markieren, verwende ich ein (spezielle von mir entwickeltes) NotYetImplemented Widget :)

```dart
class NotYetImplementedWidget extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Column(
      mainAxisAlignment: MainAxisAlignment.center,
      children: <Widget>[
        Padding(
          padding: const EdgeInsets.all(8.0),
          child: Container(
            decoration: BoxDecoration(
                color: Colors.lightGreen,
                borderRadius: BorderRadius.only(
                  topLeft: Radius.circular(2.0),
                  topRight: Radius.circular(20.0),
                  bottomLeft: Radius.circular(20.0),
                  bottomRight: Radius.circular(20.0),
                )), // BorderRadius.only, BoxDecoration
            height: 100,
            width: double.infinity,
            child: Center(
              child: Text(
                "Noch nicht implementiert :)",
                style: Theme.of(context).textTheme.headline,
                textAlign: TextAlign.center,
              ), // Text
            ), // Center
          ), // Container
        ), // Padding
      ], // <Widget>[]
    ); // Column
```

Navigation zwischen Screens

- Um zwischen Screen zu navigieren (also, um sie aufzurufen), verwendet man Routen
- Es gibt verschiedene Möglichkeiten, eine Route anzulegen
  - Named Routes
  - MaterialPageRoute
- Das System legt dazu einen Stack an
  - Der anzuzeigende Screen wird auf den Stack gepushed
  - Zurück geht es dann mit eine pop Operation

MaterialPageRoute

```
38          onTap: () {
39            Navigator.push(
40              context,
41              MaterialPageRoute(
42                builder: (context) => WebviewScreen(
43                  _ausstellerNamen[index].url,
44                  true,
45                  title: _ausstellerNamen[index].name,
46                ), // WebviewScreen
47              ), // MaterialPageRoute
48            );
49          },
```

Named Routes

```
178                 child: Row(
179                   mainAxisAlignment: MainAxisAlignment.spaceEvenly,
180                   children: <Widget>[
181                     _customButton("Umfragen", () {
182                       Navigator.pushNamed(context, "umfragen");
183                     }),
184                     _customButton("Leaderboard", () {
185                       Navigator.pushNamed(context, "leaderboard");
186                     }),
187                   ], // <Widget>[]
188                 ), // Row
```

Übersicht über die Widgets

Scaffold

SafeArea

Expanded

Wrap

Animated Container

Opacity

FutureBuilder

FadeTransition

Floating Action Button (FAB)

PageView

Table

SliverAppBar

SliverList, SliverGrid

FadeInImage

StreamBuilder

InheritedModel

ClipRRect

Hero

CustomPaint

Tooltip

TittedBox

LayoutBuilder

AbsorbPointer

Transform

BackdropFilter

Align

Positioned

AnimatedBuilder

Dismissible

SizedBox

ValueListenableBuilder

Draggable

AnimatedList

Flexible

MediaQuery

Spacer

InheritedWidget

AnimatedIcon

AspectRatio

LimitedBox

PlaceHolder

RichText

ReordableListView

AnimatedPositioned

AnimatedPadding

IndexedStack