

# **Entwicklung einer Cross-Plattform-App zur Datenerfassung würzig belegter Fladenbrote**

Bearbeiter:	Tony Spiegel Stiftsgasse 32 07407 Rudolstadt
Betreuer:	Prof. Herr Stepping
Matrikel-Nr.:	639872
Fachsemester:	8
Studiengang:	Wirtschaftsingenieurwesen / E-Commerce
Modul:	Mobile App Entwicklung II
Eingereicht am:	07.06.2019

# Inhaltsverzeichnis

<b>1</b>	<b>Motivation</b>	<b>1</b>
<b>2</b>	<b>Ziele</b>	<b>1</b>
<b>3</b>	<b>Umsetzung</b>	<b>2</b>
3.1	Technologie . . . . .	2
3.2	Besonderheiten von Flutter . . . . .	2
3.2.1	Performance . . . . .	2
3.2.2	Dart . . . . .	3
3.2.3	Deklaratives UI . . . . .	3
3.3	Herausforderungen . . . . .	5
3.3.1	Date-Library . . . . .	5
3.4	Widgets . . . . .	5
3.4.1	Pizzaltem . . . . .	5
<b>4</b>	<b>Fazit</b>	<b>5</b>

## **Abkürzungsverzeichnis**

<b>CSS</b>	Cascading Style Sheets
<b>DRY</b>	Don't Repeat Yourself
<b>EAH</b>	Ernst-Abbe-Hochschule Jena
<b>JSON</b>	JavaScript Object Notation
<b>OSS</b>	Open Source Software
<b>REST</b>	Representational State Transfer
<b>SPA</b>	Single-Page Application
<b>SPOT</b>	Single Point of Truth
<b>TKP</b>	Tiefkühlpizza
<b>UI</b>	User Interface
<b>UX</b>	User Experience
<b>WIP</b>	Work In Progress
<b>WORA</b>	Write once, run anywhere

## Abbildungsverzeichnis

1	Flutter-Logo . . . . .	2
2	Dart-Logo . . . . .	2
3	Vergleich deklaratives & imperatives UI . . . . .	3
4	Beispiel: imperativer Stil . . . . .	4
5	Beispiel: deklarativer Stil . . . . .	4

## Tabellenverzeichnis

1	Übersicht Arten von Apps . . . . .	2
---	------------------------------------	---

# 1 Motivation

Diese Ausarbeitung dokumentiert die Entwicklung einer Cross-Plattform-App um die Daten von meist würzig belegten Fladenbrotten erfassen zu können. Damit gemeint sind vor allem Pizzen sowie deren Varianten. Die ursprüngliche Idee entstand durch einen Beitrag des Subforums *r/dataisbeautiful* der Social-News-Aggregator-Plattform Reddit. Dieses Subforum legt besonderen Wert darauf Datensätze möglichst sinnvoll, ansprechend und zugänglich aufzubereiten. Nicht selten sind diese Datensätze eher skurril und handeln, wie in diesem Fall, auch von Lebensmitteln. Da ich unter anderem häufig und gern Pizzen esse, lag der Entschluss nah, eben diese zu erfassen. Motiviert durch den Einstieg im Wahlmodul *Mobile App Entwicklung I* weiter native Apps zu programmieren sowie aus privatem Interesse, stand die Entscheidung schnell, dieses Mal eine Cross-Plattform-Technologie zu nutzen. Grob zusammengefasst ergeben sich dabei im nächsten Abschnitt folgende Ziele.

## 2 Ziele

- Cross-Plattform-Technologie nutzen
- Ansprechende App im Material-Design
- Single-Page-Application
- Pizzen und deren Daten erfassen/darstellen
- Cloud NoSQL-Datenbank *Firestore* nutzen

### 3 Umsetzung

Im Folgenden wird die Umsetzung insbesondere im Bezug auf die Wahl der Technologie sowie Darstellung der App betrachtet.

#### 3.1 Technologie

Um Apps zu entwickeln gibt es viele Möglichkeiten. Diese lassen sich grob in folgende Arten einteilen

Art	Charakteristik
Hybrid	Web-Apps werden im nativen Kontext in einer WebView eingebunden
Native	Adressieren konkrete Zielplattformen und deren Programmiersprachen. Android (Java, Kotlin, Dart), iOS (Objective-C, Swift)
Web Apps	Über einen Server bereitgestellte plattformunabhängige Anwendungen

Tab. 1: Übersicht Arten von Apps

Ich bin großer Fan von Write once, run anywhere (WORA) und entwickle überlicherweise vor allem Web-Apps. Um etwas neues zu lernen und daran zu wachsen, entschied ich mich, dieses Mal dazu eine Cross-Plattform-Technologie zu nutzen. Die Entscheidung fiel dabei auf das von Google entwickelte Open Source User Interface (UI)-Kit *Flutter*.



Abb. 1: Flutter-Logo



Abb. 2: Dart-Logo

#### 3.2 Besonderheiten von Flutter

##### 3.2.1 Performance

Eine der Besonderheiten von *Flutter* ist es, dass dieses UI-Kit weder eine WebView noch die vom Betriebssystem mitgelieferten Widgets benutzt. Widgets sind in der Welt von *Flutter* alles von Bedienelemente bis hin zu Layout-Helfern. Statt diese mitgelieferten Widgets zu nutzen, setzt *Flutter* auf eine eigene Rendering-Engine welche häufig mit einer 2D-Spiele-Engine verglichen wird. Eines der Entwicklungsziele von *Flutter* war es nämlich, besonders performante Apps entwickeln zu können welche mit einer hohen Hertz-Zahl

(60-120 Hz) laufen. Mit diesem Ansatz ist es möglich, die gesamte UI über den Grafikchip des Systems zu berechnen und die CPU zu entlassen.

### 3.2.2 Dart

*Flutter*-Apps werden in *Dart* entwickelt.

### 3.2.3 Deklaratives UI

Im Gegensatz zu imperativen Frameworks wie dem *Android SDK* oder dem *iOS UIKit* handelt es sich bei *Flutter* um ein so genanntes deklaratives Framework. Dies bedeutet, dass das UI von Flutter immer aktuellen "State" reflektiert. Wird beispielsweise eine Option in den Einstellungen einer App geändert, so ändert sich der "State" der App welches das Neuzeichnen der App auslöst (eine Checkbox wird gefüllt, ein Switch aktiviert). Imperativ würde bedeuten, dass es Methoden wie *widget.setText* gibt um Werte direkt zu ändern. Hier wird der "State" geändert und das UI wird komplett neu gezeichnet.

Technisches Beispiel

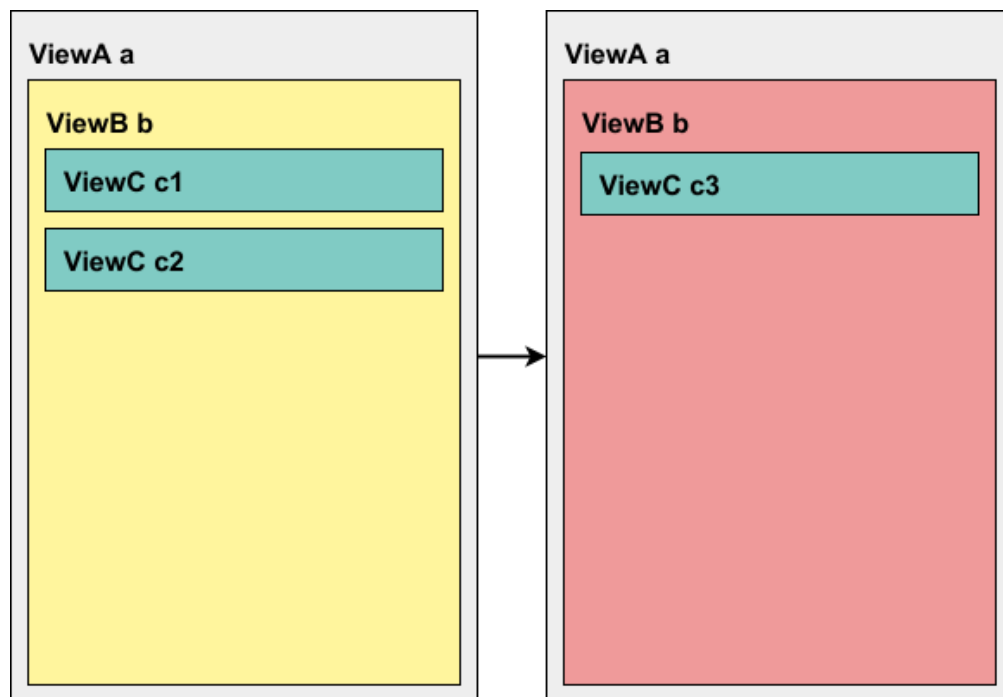


Abb. 3: Vergleich deklaratives & imperatives UI

Im imperativen Stil würde man eine Instanz *b* des so genannten *Owners* der View *ViewB* nutzen und mit Hilfe eines Selektors wie beispielsweise *findViewById* Änderungen



auf dieser anwenden (und somit diese implizit invalidieren). Das kann ungefähr so aussehen:



```
// Imperative style
b.setColor(red)
b.clearChildren()

ViewC c3 = new
ViewC()
b.add(c3)
```

Abb. 4: Beispiel: imperativer Stil

Außerdem müsste dieses Setup im Konstruktor von *ViewB* dupliziert werden, da die UI (der Single Point of Truth (SPOT)) möglicherweise die Instanz *b* überlebt. Im deklarativen Frameworks sind View-Konfigurationen (wie *Flutter's* Widgets) unveränderlich (immutable) und an sich nur simple Blaupausen. Um ein UI zu ändern, löst ein Widget einen Rebuild auf sich selbst aus (in *StatefulWidgets* häufig per *setState()*) und erzeugt damit einen neuen Widget-Subtree.



```
// Declarative
// Style
return ViewB(
  color: red,
  child: ViewC(),
)
```

Abb. 5: Beispiel: deklarativer Stil

### **3.3 Herausforderungen**

#### *3.3.1 Date-Library*

### **3.4 Widgets**

#### *3.4.1 Pizzaltem*

## **4 Fazit**