Roadmap zur Implementierung einer interaktiven 2D/3D Muskel-Heatmap mit Flutter & FlutterFlow

Dein Projektteam

18. Juli 2025

Einleitung

Diese Roadmap beschreibt in 7 Phasen, wie Du aus Deinem bestehenden 2D-Heatmap-CustomPainter und rudimentären 3D-Widget eine attraktive, performant drehbare 3D-Muskel-Heatmap realisierst – idealerweise innerhalb von FlutterFlow mit Custom Code. Jeder Schritt führt Dich systematisch von den Grundlagen bis zur Finalisierung.

1 Phase 0: Voraussetzungen und Setup

0.1 Flutter-Toolchain installieren:
- Flutter SDK (Version 3.7+), Android Studio / VS Code
- Empfohlene Plugins: Dart, Flutter, FlutterFlow CLI

0.2 FlutterFlow-Account anlegen:
- Registrierung unter `https://flutterflow.io`, CLI installieren

0.3 Projektstruktur klären:
- Lokales Flutter-Projekt plus FlutterFlow-Projekt synchronisieren
- Git-Repository aufsetzen für Versionierung

2 Phase 1: Grundlagen FlutterFlow & Custom Widgets

1.1 FlutterFlow-Oberfläche kennenlernen:
- Seiten, Widgets, Actions
- Custom Code Bereich öffnen

1.2 Erstes Custom Widget: „Hello World“ – Test:
- Einbinden eines sehr einfachen StatelessWidget
- Deployment-Test in FlutterFlow Preview

1.3 Projekt lokal klonen / exportieren:

- FlutterFlow → Export → lokales Flutter-Projekt
- Sicherstellen, dass automatische Sync-Funktion aktiviert ist

3 Phase 2: Vektor-Assets für die Heatmap erstellen

2.1 Muskelkonturen als SVG:
- Body-Silhouette und Muskelregionen in einem Vektorprogramm (z.B. Illustrator, Inkscape)
- Export: einzelne Pfade je Region („shoulders.svg“, „quadriceps.svg“ usw.)

2.2 SVG in Flutter importieren:
- `flutter_svg`-Package einbinden
- Test: einzelne SVGs anzeigen lassen

2.3 Color-Map definieren:
- Farbverlauf von Grau über Hellrot zu Dunkelrot
- In Dart: Funktion `Color intensityToColor(double t)` implementieren

4 Phase 3: 2D-Heatmap mit CustomPainter optimieren

3.1 Übersetzen der SVG-Pfaddaten:
- Mü ggf. `PathParser` nutzen, oder manuell aus `MusclePaths`

3.2 Intensitätswerte aus Firestore laden:
- Provider/Bloc → Firestore → MapRegion, Count
- Normierung auf [0,1]

3.3 CustomPainter entwickeln:
- Pfade zeichnen, füllen mit `intensityToColor()`
- Stroke-Linien weiß mit `canvas.drawPath()`

3.4 Performance-Optimierung:
- `RepaintBoundary` rund um den Painter
- ggf. in `isolate` vorrechnen

5 Phase 4: Einbindung in FlutterFlow

4.1 Custom Code Widget anlegen:
- Code für `AdvancedBodyHeatmap` in FlutterFlow → Custom Widgets kopieren

4.2 Bindings definieren:
- Inputs: MapString,double (Region → Intensität)
- Expose als Parameter im FlutterFlow-Widget

4.3 UI in FlutterFlow zusammenstellen:

- Page: Data Query (Firestore) → Custom Widget (Heatmap)

4.4 Preview und Test auf Device:

- Browser Preview → Android / iOS Preview

6 Phase 5: 3D-Heatmap auf Basis eines 3D-Modells

5.1 3D-Modell vorbereiten:

- OBJ/GLTF mit klaren Mesh-Gruppen für Muskelregionen
- Blender: jedem Mesh eine eigene Material-ID geben

5.2 3D-Rendering in Flutter:

- Packages prüfen: `flutter_cube`, `flutter_gl`+`three_dart`, `flutter_scene_kit`

5.3 Dynamische Textur / Vertex-Coloring:

- Jede Muskel-Mesh über Materialfarbe einfärben (intensityToColor)
- Texture-Update: Canvas-Heatmap → ByteData → als Texture anbinden

5.4 Rotation und Interaktion:

- Gesture-Detector: Drehung um Y-Achse (Drag → scene.world.rotation.y)
- Zoom via Pinch → camera.zoom

7 Phase 6: Integration des 3D-Widgets in FlutterFlow

6.1 Custom 3D-Widget in FlutterFlow:

- Gleiche Vorgehensweise wie in Phase 4
- Inputs: MapString,double , initiale Kameraparameter

6.2 UI/UX-Feinschliff:

- Lade-Placeholder, Error-States, Gestenhinweise

6.3 Cross-Plattform-Test:

- iOS/Android/ Web (sofern unterstützt)

8 Phase 7: Finalisierung, Qualitätssicherung & Deployment

7.1 Unit- & Widget-Tests schreiben:

- CustomPainter, 3D-Widget, Farbzuordnung

7.2 Performance-Profiling:

- DevTools: FPS, Render-Time

7.3 Dokumentation:

- Kurzanleitung für weitere Entwickler (README)

7.4 Deployment:

- Android/iOS Release-Builds, AppStore / PlayStore
- FlutterFlow → Production Push

Viel Erfolg bei der Umsetzung!