**Account Screen**

Der Account-Bildschirm präsentiert dem Nutzer eine Reihe nützlicher Funktionen und Einstellungen. Die Buttons folgen einem einfachen Design-Schema. Ein deutlicher Rahmen umrandet den ansonsten in der Hintergrundfarbe gehaltenen Knopf. Jeder enthält ein entsprechendes Icon, welches die Funktion verdeutlicht, sowie den Titel der Aktion (Ausnahme ist der Slider). Diese Knöpfe unterscheiden sich somit gut von den meisten anderen Buttons und schaffen einen abgekapselten Bereich im Account-Bildschirm, der weniger mit den eigentlichen Rezepten zu tun hat und optional für die Nutzung der App ist. Momentan umfasst er folgende sechs Elemente bzw. Funktionen:

Ein Avatar mit dem Stiftsymbol weist darauf hin, dass dieser angepasst werden kann. Beim Klick darauf öffnet sich die Galerie, aus der man ein Bild auswählen kann. Das ausgewählte Bild wird dann als Kreis ohne Rahmen angezeigt. Dies dient sowohl zur Personalisierung als auch zur schnelleren Identifizierung der Nutzer in der Übersicht, falls dort mehrere Personen hinzugefügt wurden.

Der "Theme ändern"-Button wechselt nacheinander durch das Karussell der verfügbaren Themes, sodass jedes ausgewählt werden kann. Momentan sind drei verschiedene Themes vorhanden. Durch das Karussell, anstelle eines separaten Auswahlfensters, lässt sich schneller durch die Themes navigieren. Die Themes werden direkt aktualisiert, sodass man auf der Account-Seite bereits sehen kann, wie sie aussehen.

Der Slider mit dem ABC-Icon ändert die Schriftgröße in kleinen Schritten. Auch diese Änderung aktualisiert sich in Echtzeit.

Der "Sprache"-Button öffnet ein Popup-Fenster, das alle verfügbaren Sprachen auflistet (momentan Deutsch und Englisch) und sie als separate Buttons darstellt. Wenn man einen davon drückt, wird die App-Sprache entsprechend geändert. Auch dies geschieht in Echtzeit, sodass keine Erfolgsmeldung erforderlich ist, da man die geänderte Sprache direkt im Account-Bildschirm sehen kann.

Der "Gruppe bearbeiten"-Button führt zu einer neuen Seite. Auf dieser Seite sieht man Nutzer, die man zur eigenen Gruppe hinzugefügt hat und deren Rezepte auf dem Home-Bildschirm angezeigt werden. Die beiden darunter liegenden Buttons dienen dem Kopieren und Einfügen der eigenen bzw. einer fremden Nutzer-ID. Letzteres bewirkt das Hinzufügen zur Gruppe. Beide Aktionen werden mit einer Snackbar kommentiert.

Zuletzt ermöglicht der "Abmelden"-Button dem Nutzer, sich aus seinem Konto auszuloggen und zum Anmeldebildschirm zu gelangen.

**Multi Language Support**

Damit Einstellungen wie die ausgewählte Sprache nicht beim Neustart der App verloren gehen, benötigten wir die Möglichkeit, Einstellungen lokal abzuspeichern. Zu diesem Zweck verwenden wir die Shared Preferences aus der gleichnamigen Bibliothek. Diese ermöglichen es uns, plattformunabhängig einfache Werte (int, double, bool, String, List<String>) zu speichern. Das reicht vollkommen aus, um die aktuelle Sprache, Schriftgröße oder das aktive Theme zu speichern. Beim Start der App wird eine Instanz von Shared Preferences erzeugt, auf die von überall aus über die config.dart-Datei zugegriffen werden kann. Durch Getter- und Setter-Methoden (z.B. setInt("Zahl", 2), getInt("Zahl")) können Werte ganz einfach gespeichert und gelesen werden. Für die Schriftgröße verwenden wir einen einfachen double-Wert, während für die Sprache der Sprachcode als String gespeichert wird (z.B. "de" für Deutsch, "en" für Englisch). Da wir keine Objekte direkt speichern können, haben wir für die Themes eine kreative Lösunge gefunden. Statt das gesamte Theme-Objekt zu speichern, speichern wir die aktuelle Position des Theme-Karussells als Integer ab. Beim Initialisieren des Theme-Services wird dieser Wert automatisch ausgelesen und bei einem Wechsel des Themes aktualisiert. Insgesamt bieten die Shared Preferences somit einen sehr intuitiven und schnellen Weg, um einfache Daten zu speichern.

Für die Lokalisierung verwenden wir die Bibliothek flutter\_i18n. Zur Vorbereitung wird der Main-Methode ein „localizationDelegate“ übergeben, der den Pfad zu den Übersetzungsdateien (assets/i18n) definiert und das verwendete Format angibt. Gültige Formate sind JSON, YAML, TOML und XML. Zusätzlich deaktivieren wir die Nutzung von Ländercodes, da wir keine Übersetzungen für Dialekte anbieten.

Im Code selbst muss jeder angezeigte Text durch die Funktion FlutterI18n.translate() ersetzt werden. Diese Funktion benötigt den BuildContext und einen (String-)Key als Parameter. Der zum Key passende Text wird dann, sofern vorhanden, aus der entsprechenden Übersetzungsdatei gelesen. In unserem Fall sind das de.json und en.json für die Lokalisierung auf Deutsch bzw. Englisch. Beide Dateien haben die gleiche Struktur, nur die Werte für die Keys unterscheiden sich.

Wenn ein neuer Text benötigt wird, der noch nicht vorhanden ist, muss dieser unbedingt mit demselben Schlüssel in beiden Dateien hinzugefügt werden. Die Schlüssel sind hierarchisch nach den Screens strukturiert, in denen sie vorkommen. Die Wurzel des Erstellungs-Screens "create" enthält beispielsweise die Sub-Keys "title" und "create". Der kombinierte Schlüssel "create.title" wird dann für die Überschrift des Erstellungs-Screens genutzt. Die Schlüssel können beliebig tief verschachtelt werden, um sie nach Themen zu strukturieren (z.B. create.error.error1, create.error.error2), wodurch sie jedoch schnell sehr lang werden können. Bei PlatePal haben wir eine maximale Verschachtelungstiefe von 3 angestrebt. Zudem haben wir versucht, möglichst kurze Schlüssel zu wählen, weshalb diese auch auf Englisch sind.

Für das eigentliche Wechseln der Sprache haben wir den locale\_service geschrieben. Dieser enthält alle verfügbaren Sprachen und bietet einfache Methoden zum Wechseln und auslesen der aktuellen Sprache an. Außerdem verwendet er das Mixin ChangeNotifier. Dadurch können wir die setState() Methode der Main als Listener hinzufügen um den Wechsel der ausgewählten Sprache zu überwachen und direkt zu aktualisieren.

**Theme Wechsel**

Der Theme-Wechsel funktioniert analog zum Sprachwechsel. Auch der theme\_service verwendet das Mixin ChangeNotifier, fügt setState() in der Main-Methode als Listener hinzu und aktualisiert dadurch das Theme in Echtzeit. Statt einer Map verwendet der theme\_service eine einfache Liste, die alle definierten Themes enthält. Zusätzlich hält er einen Zähler aktuell, der die Position des ausgewählten Themes in der Liste angibt.

**KI-Import**

Für den KI-Import verwenden wir einige spezielle Packages. Zunächst verwenden wir google\_mlkit\_text\_recognition, um den Text aus dem ausgewählten Bild in einen String umzuwandeln. Die Texterkennung ist Teil des Google Machine Learning Kits und steht auf modernen Mobilgeräten zur Verfügung.

Um das erkannte Rezept, das nun als unstrukturierter Text vorliegt, in unser intern verwendetes Recipe-Objekt zu überführen, senden wir es an die Chat-GPT-API von OpenAI. Zusätzlich übermitteln wir ein reduziertes Beispiel-Recipe-JSON und geben an, dass der Fließtext in dieses Format überführt werden soll. Durch die Verwendung eines kleineren JSON-Formats, das nur die benötigten Felder enthält, konnten wir sowohl die Antwortzeit drastisch reduzieren (von teilweise 90 Sekunden auf 20-30 Sekunden) als auch die Fehlerquote verringern. Letzteres war besonders wichtig, da ein unvollständiges oder beschädigtes JSON schwer korrigierbar ist.

Der eigentliche Aufruf der API erfolgt mithilfe des Packages dart\_openai. Dabei geben wir einige Parameter an, wie den API-Key, das zu verwendende Chat-GPT-Modell und unsere Nachricht, und warten auf eine Antwort. Im Erfolgsfall erhalten wir einen gültigen JSON-String als Antwort, aus dem wir die Felder extrahieren und in ein Recipe-Objekt überführen können. Dieses kann dann in der ErstellenView in das Model eingefügt werden, um die GUI zu aktualisieren.

**Packages:**  
google\_mlkit\_text\_recognition: Texterkennung (auf Bildern)

dart\_openai: Aufruf der Chat GPT API

shared\_preferences: (begrenzter) Lokaler Datenspeicher (für z.B. Einstellungen)

flutter\_i18n: Lokalisierung (Multi Language Support)

flutter\_spinkit: Einfache Ladeanimation