2022

Vokabeltrainer ScanQ

JUSTUS BENDEL UND MARCEL ROMMEL

Konzeptdaten

App: Vokabeltrainer *ScanQ*
Entwickler: Marcel Rommel (18), Justus Bendel (18)
Thema der App: App-gestütztes Vokabellernen

Kurzfassung

Unsere Konzeptidee beschreibt eine App zur Maximierung der Lerneffektivität von Vokabeln durch interaktive Lernspiele sowie einer Quiz-Funktion. Dabei stehen dem Nutzer verschiedene Features zur Verfügung:

- Schnelles **Einscannen** von Vokabeln
- Automatische Erstellung von **Lern-** & **Quizspielen**
- Übersicht des aktuellen **Lernfortschrittes**[1]
- Per App erstellte **Lernpläne** & **Organisation**[2]
- **Online** in Gruppen Vokabeln trainieren

Durch diese Features möchten wir für unsere Benutzer das Üben neuer Vokabeln zum Vergnügen machen. Hierbei steht die Benutzerfreundlichkeit für uns an oberster Stelle, weswegen eine einfache, intuitive und moderne Bedienoberfläche sowie eine spielerische Wettbewerbsumgebung[3] für mehr Spaß beim Lernen sorgen.

Abschließend ist es für uns von hoher Bedeutung, die App für eine möglichst große Gruppe von Menschen verfügbar zu machen. Aus diesem Grund sind alle unsere App-Inhalte vollkommen werbe- und kostenfrei zugänglich. Zusätzlich stellen wir einen Modus für Sehbehinderte bereit, sodass auch diese einfach per Sprachsteuerung mit unserer Anwendung interagieren können.

[1] Statistiken über die Menge der Vokabeln (gesamt, falsche, unbeantwortete…); zeitlich aufgeschlüsseltes Lernverhalten, etc.
[2] Noch in Arbeit; nur teilweise realisiert
[3] Gemeint ist die Online-Funktion

Hier geht es zur aktuellen App-Version:

App herunterladen

Unser Promotion Video:

Video anschauen

Quellcode der vereinfachten, aktuellen App-Version:

Quellcode einsehen

Inhaltsverzeichnis

Anmerkungen

Aus Gründen der Lesbarkeit verwenden wir stets die maskuline Form, meinen jedoch immer eine geschlechtsneutrale Formulierung (z.B. Benutzer statt Benutzer*In).

Legende:

Kursiv	Fachbegriffe, Eigennamen, Zitate & Verweise
`Code-Style`	Java-*Klassen* & weitere Code-Elemente
`Java Quellcode`	Quellcode-Auszüge
[Hinweis]	Hinweise bezüglich der Lesart
~ Quellname	Kurzreferenz zur Zitatquelle

1 Einleitung

Ziel unserer App ist es, jedem, der Vokabeln lernen möchte, mit ScanQ ein Instrument zu geben, um dies erfolgreich und effizient zu tun. Simultan soll eine Abkehr vom monotonen und freudlosen Lernen hin zu einem von Abwechslung und Interaktivität geprägten Lernerlebnis stattfinden.
Die Notwendigkeit hierfür wird durch Forschungsergebnisse begründet, die belegen, dass stumpfsinniges Auswendiglernen für dauerhaften Lernerfolg kontraproduktiv ist. Denn hierbei gelangen die Inhalte nicht in das Langzeitgedächtnis.
Unser Vorhaben ist es daher, unsere App so zu entwickeln, dass vor allem Schülern das Erlernen eines fremdsprachlichen Wortschatzes erleichtert wird und dieser langfristig verinnerlicht wird, indem durch spielerischen Anreiz und kleine Competition die Motivation gesteigert wird.

Zusätzlich versuchen wir mit *ScanQ* eine möglichst große Zielgruppe, die Schüler aller Jahrgangsstufen, Menschen mit körperlicher Beeinträchtigung sowie die verschiedensten Lernpräferenzen umfasst, abzudecken. Davon abgesehen stehen alle Inhalte kosten- und werbefrei in uneingeschränktem Umfang zur Verfügung, sodass sich unsere App ideal für den Einsatz im Schulalltag eignet. Aus diesem Grund ist zunächst eine Einführung an unserer Schule geplant.

1.1 Unsere Motivation

Im Laufe unserer bereits zwölf Jahre andauernden Schulzeit haben wir selbst schon viele Vokabeln in verschiedenen Fremdsprachen lernen müssen und konnten dabei auf verschiedenste Lernkonzepte zurückgreifen; sowohl die klassischen Vokabelkarten, als auch bereits existierende Lernsoftware haben uns dabei unterstützt. Jedoch stellten wir im Laufe der Zeit zahlreiche Schwachstellen und Defizite fest:

So fanden wir ausschließlich[4] Apps, die…

- … auf **vorgeschriebene** Vokabellisten zurückgreifen.
- … **kostenpflichtig** oder **abonnementgebunden** zu beziehen sind.
- … nichts weiter als ein Karteikartensystem unterstützen.
- … umständliches **Eintippen** voraussetzen.
- … strikt an einen **Verlag** gebunden sind.
- … ein **veraltetes** und/oder **benutzerunfreundliches** *UI*[5] aufweisen.

All diese Punkte stellten für uns eine zusätzliche Motivation dar, eine App mit der Zielsetzung zu entwickeln, entsprechende Defizite zu umgehen. Ergo schließt unsere App eine zurzeit bestehende Marktlücke und treibt das *e-Learning*[6] in diesem Bereich maßgeblich voran.

1.2 wissenschaftlicher Hintergrund

> *„The advantages of computer-only learning are that students can get correct answers **sooner** and work on many more exercises in **shorter** periods of time.“ [1]*
>
> ~ Tokyo Keizai University

Diese Vorteile werden durch unsere App *„ScanQ“* aufgegriffen, indem durch direktes Einscannen der Vokabeln Zeit eingespart wird sowie durch unmittelbare Fehlerkorrektur wesentlich schneller Resultate erzielt werden.

[4] Alle Apps erfüllen mindestens einen der genannten Kritikpunkte
[5] User Interface ≙ Benutzeroberfläche
[6] Für genauere Erläuterungen siehe 1.3

„Spielmotivation [hat] einen positiven Einfluss auf den Lernerfolg. […]
Umso größer die Spielmotivation […], desto **mehr Aufmerksamkeit**
widmen [Schüler] den Inhalten […]“ *[2]*

~ Wissensprozesse und digitale Medien, Bd. 20

Vier verschiedene Quiz-Modi garantieren eine spielerisch geprägte Lernumgebung für den App-Nutzer, wodurch die oben genannten Vorteile *(vgl. [2])* aufgegriffen werden; einerseits bietet unser Vokabeltrainer ein Multiple-Choice- und zwei Direkteingabe-Quiz, andererseits auch ein Quiz mit Spracheingabe, welches zusätzlich die korrekte Aussprache der Vokabel für die richtige Beantwortung voraussetzt und die Abfragefunktion für blinde Menschen zugänglich gestaltet. Daher haben wir uns bei der Entwicklung der Spiele darauf konzentriert, durch ein ansprechendes *UI*[6], ein simples Spielekonzept sowie ein Belohnungssystem[7] eine möglichst hohe Spielemotivation zu gewährleisten.

Zusätzlich bietet unsere App einen Onlinemodus, dessen Intention in der Bereitstellung einer spielerischen Wettbewerbsumgebung liegt. Dieser Modus greift Erkenntnisse zur positiven Wirkung des Wettbewerbs beim Lernen auf *(vgl. Abb. 1)*, jedoch in Form einer sogenannten Koopetition. Dieser Begriff stammt aus der Wirtschafts-theorie und bezeichnet die Symbiose aus Kooperation und Konkurrenz.

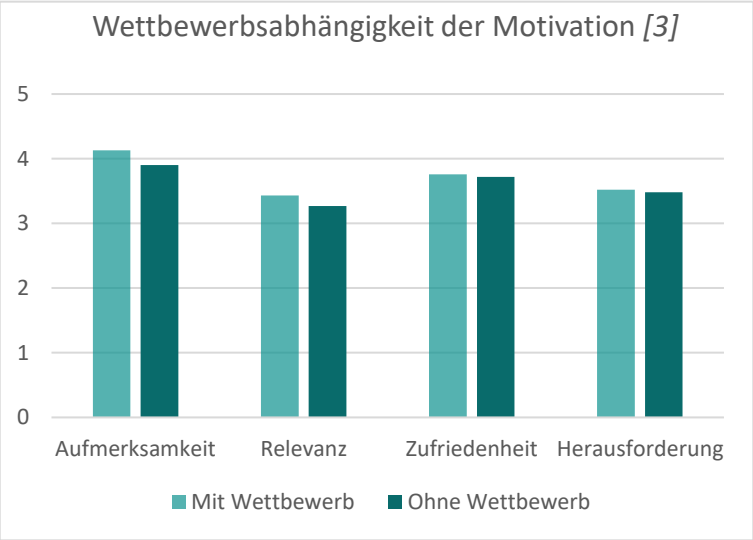

Abb. 1

Diese beiden Elemente zeigen sich in unserer Anwendung folgendermaßen:

Kooperation in Form gemeinschaftlichen Übens für Klausuren und Tests. Die von sozialen Gruppen ausgehende Atmosphäre wirkt sich diesbezüglich höchst förderlich aus.

Konkurrenz manifestiert sich hingegen im Vergleich mit Freunden durch unterschiedliche Ranglisten sowie im Wettstreit während der Online-Quiz.

2 Technische Hintergründe

2.1 Vorbereitung

Zur Realisierung unserer App standen uns mehrere Möglichkeiten der Programmierung zur Verfügung. Entweder als native Android- (Java-Applikation) oder hybride Web-Anwendung mit folgenden Vor- und Nachteilen:

Java-Applikation (nativ)	Web-Anwendung (hybrid)
mehr Möglichkeiten für Android	Mehr Erfahrung
bessere Geräteinteraktion	keine Bindung an Android
auch offline vollständig nutzbar	Wiederverwendbarkeit alter Projekte
direkte Anpassung an Systemupdates	
Android Studio (Autogenerierung, etc.)	

[7] Erfolge werden mit digitalen Trophäen und Auszeichnungen belohnt

Da mangelnde Erfahrung und die Notwendigkeit, alles von Grund auf neu zu programmieren, für uns weniger eine Hürde als eine Chance respektive Herausforderung darstellten, entschieden wir uns für Java. Einzig und allein die strikte Bindung an Android verkörperte einen ernst zu nehmenden Nachteil, welcher jedoch durch die horrenden Kosten[8] des Apple App Stores relativiert wurde. Zudem ist der Marktanteil von Android-Smartphones in Deutschland mit 80,9 % deutlich größer als bei Apples IOS (18,9 %) *[4]*, weshalb wir uns für Android entschieden, um eine breitere Zielgruppe ansprechen zu können.

2.2 Activities

In der Regel bestehen Android-Apps aus mehreren Unterseiten, genannt *Activities*. Sie setzen sich aus *XML*-Layouts, für die optische Gestaltung der Bedienoberfläche, und Java-*Klassen*, für Funktionalität und Benutzer-Interaktion der App, zusammen.
Hier wird insbesondere unsere Vorgehensweise aus informatischer Sicht beschrieben. Abläufe aus Nutzersicht werden in den Ergebnissen thematisiert.

2.2.1 Haupt-Activity: Einscannen

Diese *Activity* ist das Kernstück unserer App; sie behandelt den Prozess der Integration neuer Vokabeln, indem diese aus Bildern erkannt und in einer Datenbank abgespeichert werden. Hierfür wird eine *OCR*[9]-Bibliothek benötigt, von denen mehrere vergleichbare zur Verfügung stehen. Für uns standen folgende zur Auswahl *[5]*:

	Abbyyocr*	Gocr**	Tesseract**
Version	8.0	0.48	SVN r402
Input Format	PNG	PNM	TIF/BMP
Courier	100 %	67 %	81 %
Times New Roman	100 %	76 %	92 %

* kostenpflichtige Software ** kotenlose Open-Source Software

Wir haben uns nach reiflicher Überlegung für die Open-Source-Software *„Tesseract“* entschieden, da sie über eine sehr präzise Texterkennung verfügt. Das deutlich genauere *„Abbyyocr“* kam zunächst[10] aus finanzieller Sicht nicht für uns in Frage.
Abgesehen davon deckt *Tesseract* bereits 116 Sprachen ab, was eine spätere, uneingeschränkte Weiterentwicklung der App ermöglicht. Im Gegensatz zur ebenfalls kostenfreien Alternative *„Gocr“* bietet *Tesseract* zudem Möglichkeiten der Textanalyse[11], die wir auch in der weiteren Entwicklung verwendeten.
Anders als *„Gocr“* unterstützt *Tesseract* sowohl mehrspaltige Layouts als auch serifenhaltige Schriftarten. Schließlich ist *Tesseract* nicht an ein *UI* gebunden, was die Implementierung in unsere App erleichtert.

Erster Entwurf:

Hierbei liegt eine simple *Activity*, die aus einem `EditText` (Texteingabefeld), zwei `Buttons` und einem `ImageView` (Bildbereich) besteht, vor.
Der Nutzer wählt über einen via `Intent` übergebenen `Integer`-Wert zwischen Galerie und Kamera. Das selektierte Bild wird für das Zuschneiden durch den Benutzer in einer separaten *Activity* geladen. Nun wird das zugeschnittene Bild im `ImageView` angezeigt und als `Bitmap` in einem `Frame` an *Tesseract* übergeben. *Tesseract* erkennt den Text und dieser wird im `EditText` angezeigt, um ggf. durch den Nutzer korrigiert zu werden *(vgl. Abb. 2 Phase 1)*.
Nachfolgend extrahieren wir mit Hilfe dieses Befehls `Arrays.asList(s.toString().split("\\n"));` die Vokabeln aus dem Text und speichern sie als `ArrayList`-Elemente. Schließlich sollten die

[8] 99 USD pro Jahr und App
[9] Optical Character Recognition ≙ Texterkennung
[10] Umstieg zu einem späteren Zeitpunkt angedacht
[11] Erkennung von Schriftarten und deren Derivationen sowie konkreten Positionen *(Bounding Box)* von Zeichen, Wörtern, Zeilen und Absätzen

Vokabeln in eine *SQLite*-Datenbank abgespeichert werden, jedoch führte dies zu Komplikationen, weswegen wir unsere gesamte Datenbank auf *RoomDB* (vgl. 2.2.2) umstellten.

Zweiter Entwurf:

Dieser Entwurf brachte weitestgehend Layout-Änderungen mit sich, die jedoch wichtige Auswirkungen auf Datenbank und Texterkennung bedingten. Wir führten ein zweites `EditText`-Feld ein *(vgl. Abb. 2 Phase 2)* und ließen den bereits zuvor beschriebenen Prozess getrennt für englische Vokabeln und deren Übersetzung ausführen. Dadurch konnte die Datenbankkommunikation erheblich vereinfacht und die Genauigkeit *Tesseracts* wesentlich gesteigert werden.

Dritter Entwurf:

Laut der Dokumentation *Tesseracts [7]* empfiehlt es sich zur Verbesserung der Erkennungsqualität eine vorangestellte Bildbearbeitung anzuwenden. Dafür zogen wir verschiedene Variationen in Betracht, entschieden uns aber schlussendlich für *OpenCV*, da es uns die umfangreichsten Bearbeitungsmöglichkeiten bot.

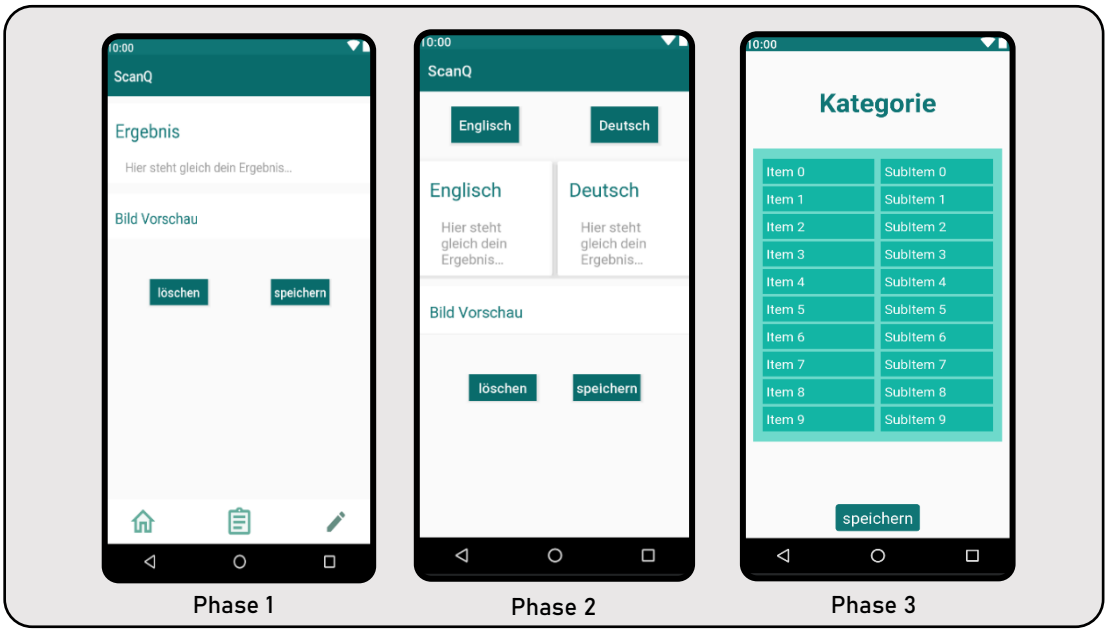

Abb. 2: Entwicklung der Einscann-Activity

Daher begannen wir mit unterschiedlichen Bildbearbeitungsoptionen zu experimentieren und kamen dabei zu folgenden Optionen der Resultatverbesserung:

- Konvertierung in **Graustufen**
- Gezielte **Kontrastanpassung**
- Entfernung von **Bildrauschen**
- **Begradigung** des Bildes (*Deskewing*)

Die Umsetzung dieser Bildverbesserungen wurde übergangsweise mittels *Chaquopy*[12] in Python statt Java realisiert. Jedoch verursachte dies eine enorme Erhöhung der Speichergröße unserer App sowie eine Verringerung der Performance.

Vierter Entwurf:

Zur Vereinfachung der Usability fügten wir in diesem Entwurf eine Anleitung hinzu, welche den Benutzer schrittweise durch das Einscannen leitet. Zusätzlich erfolgte eine weitere Änderung im Bereich der Präsentation der Vokabeln, sodass die ehemaligen beiden `EditText`-Felder in einem `TabelLayout`, bestehend aus zwei Spalten mit je einem `EditText` pro Zelle, zusammengefasst wurden *(vgl. Abb. 2 Phase 3)*. Folglich blieb die Bearbeitbarkeit erhalten und das Layout gewann an

[12] Hiermit können Python-Skripte unter Android verwendet werden

Übersichtlichkeit und moderner Optik. Neben der Bildbearbeitung lässt sich das Ergebnis der Erkennung durch *Tesseract* auch auf andere Weise optimieren, was diese Entwicklungsstufe prägt.

Zum einen teilt die App das ausgewählte Bild in mehrere Teilbilder, wenn viele Vokabeln darin enthalten sind, sodass *Tesseract* immer nur mit kleineren Datenmengen arbeiten muss. Dies musste zuvor manuell erledigt werden, jedoch wird mittlerweile die Anzahl der Vokabeln automatisiert erkannt und mittels *Bounding Box*[11] die Position der Schnittkante berechnet. Am Ende werden die Vokabeln, aus den Einzelbildern, in einer `ArrayList` zusammengefügt.

Zum anderen gelang es uns ein langanhaltendes Problem zu lösen, welches in der Entfernung der Lautschriften bestand. Auch dies ließ sich unter Zuhilfenahme der *Bounding Boxen* beheben. Dafür verwenden wir einen `OnTouchListener`, welcher via `MotionEvent` die Position der Berührung übergibt.
Anschließend vergleicht `boundingBox.contains(motionEvent.getX(), motionEvent.getY());` die Position mit der `Bounding Box` und blendet die Lautschrift, oder andere unerwünschte Ausdrücke, bei Übereinstimmung aus *(vgl. Abb. 3 Schritt 3)*.
Schließlich erfolgt die Löschung nicht notwendiger Sonderzeichen und Zahlen durch *RegEx*[13].

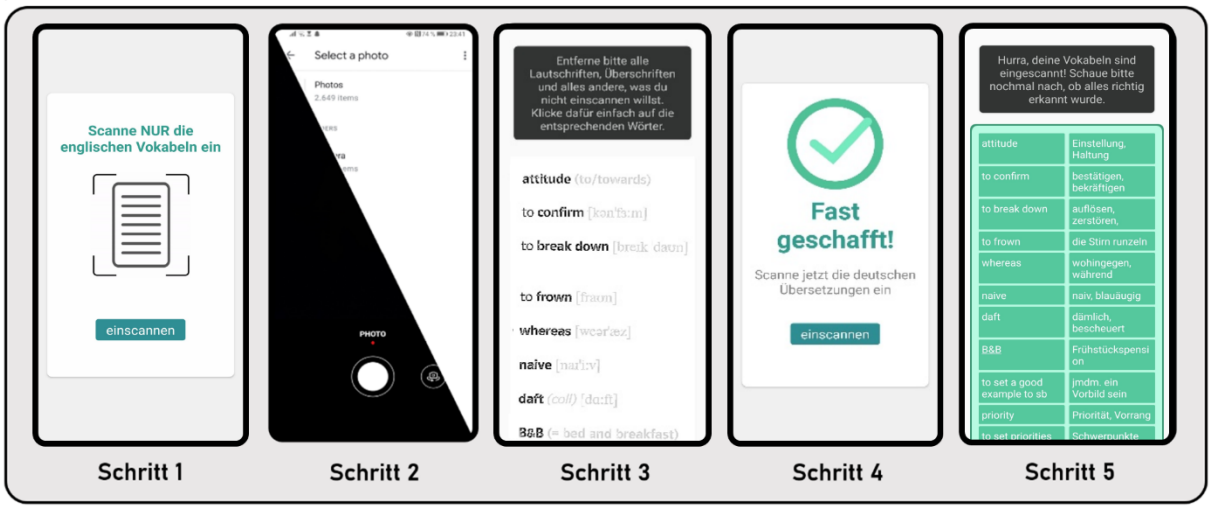

Abb. 3: Ablauf des Einscann-Vorgangs

aktueller Entwurf:

Das letzte verbleibende Problem bestand darin, dass Vokabel und Übersetzung fehlerhaft zugeordnet wurden, was sich auf mangelhafte Texterkennung und unzureichende Bildqualität zurückführen ließ. Infolgedessen erweiterten wir die Bildbearbeitung um eine Funktion zur Entzerrung im dreidimensionalen Raum *(Dewarping)* mittels `Imgproc.warpPerspective(image);`. Dabei ergibt sich ein weiterer Vorteil, der in dem automatisierten Zuschnitt des Bildes auf eine Standardgröße, besteht. Des Weiteren beinhaltet dieser Entwurf Verfeinerungen der *Tesseract*-Konfigurationen, welche die Erkennungsqualität zum Beispiel durch die genaue Festlegung der zu erkennenden Zeichen (Black- / White-Listing) erhöht.
Das bisherige Layout barg das Risiko einer Verwechslung der einzelnen Etappen des Einscannens, da sich deren jeweilige Designs zu stark ähnelten. Die Behebung erfolgte in diesem Entwurf durch kleine Layout-Anpassungen *(vgl. Abb. 3 Schritt 4)*.

2.2.2 Quiz-*Activities* und Datenbank

Neben dem Einscannen ist dies das nächstwichtige Element unserer App, welches untrennbar mit der Datenbank verknüpft ist, weswegen sie erst in diesem Abschnitt thematisiert wird.

[13] Regular Expression ≙ Muster zur Erkennung bestimmter Zeichenketten

Zunächst gibt es eine Übersicht aller Quiz in einem `ScrollView`, auf welche eine *Activity* mit den genauen Spieleinstellungen folgt. Die dort erstellten Werte werden an das gewählte Quiz übergeben und dieses wird vorbereitet. Ein solches folgt stets einem ähnlichen Aufbau, bestehend aus einem `CardView` mit der Frage im oberen Drittel, einem Antwortbereich und einer Navigationsleiste am unteren Rand. Bevor wir im Detail auf die Kommunikation zwischen den Quiz und der Datenbank **Fehler! Verweisquelle konnte nicht gefunden werden.Fehler! Verweisquelle konnte nicht gefunden**

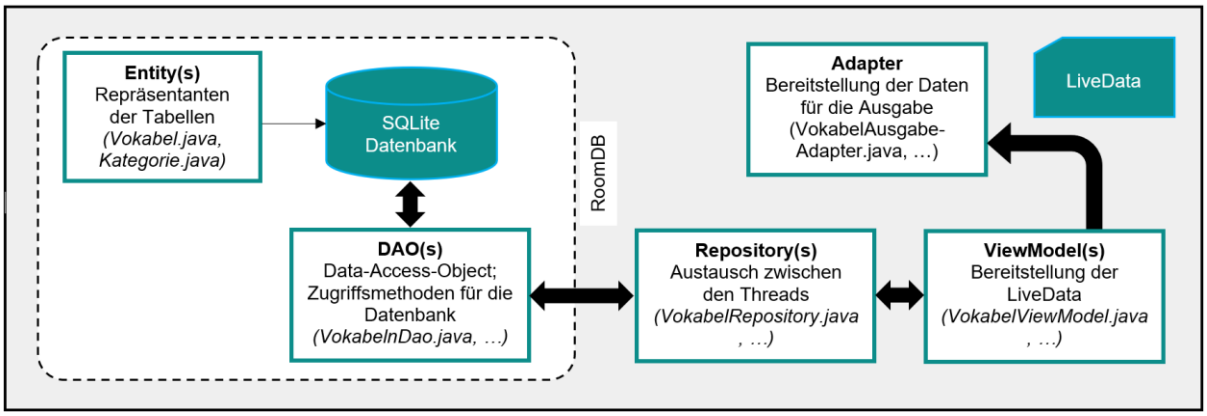

Abb. 4: Unsere lokale Datenbankstruktur

werden.eingehen, folgt erst einmal eine Erläuterung zur Funktionsweise der *RoomDB*[14] in unserer App.

Unsere *RoomDB* besteht aus einer generierten *SQLite*-Datenbank, einer `Datenbank`-*Klasse*, mehreren `Entity`-*Klassen*, `DAOs`, `Repositories`, `Adapter`-*Klassen* und `ViewModels`. Die Beziehungen dieser Komponenten können der Abbildung 4 entnommen werden.
Die *SQLite*-Datenbank wird von *RoomDB* verwaltet und stellt die reale Datenbank dar. Der Aufbau der Datenbank-Tabellen wird durch die `Entity`-*Klassen* definiert, wohingegen die `DAOs` Funktionen (Java-*Methoden)* mit *SQL*-Befehlen verknüpfen. Die weiteren Bestandteile gehören zwar nicht mehr direkt zur Datenbank, sind jedoch für deren Nutzung essentiell. So bilden die `Repositories` den Übergang vom *Main-Thread*[15] zu den *Threads*, die von der `Datenbank`-*Klasse* bereitgestellt werden. Die Funktion unserer `ViewModels` liegt in der Bereitstellung von `LiveData`, die über das `Repository` abgefragt werden, und die Bearbeitung von Datenbankinhalten in Echtzeit zulassen. Schließlich werden die Daten für den Nutzer über `Adapter`-*Klassen* zur Ausgabe vorbereitet, was den letzten Schritt in der Datenbankarbeit darstellt.

Bei der Verknüpfung des Quiz‘ mit der Datenbank traten zuerst folgende Fehler auf:

Android erlaubt keine rechenaufwendigen Operationen im *Main-Thread*, die zur Verlangsamung der App führen, weshalb Datenbankabfragen in einem separaten *Thread* ausgeführt werden müssen. Dadurch entstand der Bedarf für eine solch komplexe Datenbankstruktur.

```
do{…} while(!falsche.contains(vokabel));
```
führte zu übermäßigem Rechenaufwand und einem App-Absturz, weswegen wir einen neuen Ansatz zur Generierung der falschen Antworten mit
```
Collections.shuffle();
```
wählten.

Die aktuelle fehlerfreie Version setzt zunächst einen `Observer`[16] auf die `LiveData` im `ViewModel` und bewirkt das Anzeigen der Vokabeln im Quiz, sobald diese abgefragt wurden.
Damit ist die Datenbank-Kommunikation vorerst abgeschlossen und es wird eine `ArrayList` erstellt. Nun werden die falschen Antwortmöglichkeiten über besagtes *Shuffeln* dieser Liste generiert. Erreicht man das Ende der Vokabel-Liste, werden die Ergebnisse angezeigt, welche aus der Punktzahl ermittelt werden. Des Weiteren werden die individuellen Lernstatistiken aus den Resultaten der Quiz berechnet.
Die verschiedenen Quiz-Modi, welche in Abb. 5 gezeigt sind, funktionieren im Wesentlichen alle nach dem genannten Schema. Die Unterschiede hingegen liegen vor allem in der Spielweise, die in den Ergebnissen thematisiert wird.

[14] Bibliothek zur Optimierung und zum Umgang mit einer lokalen *SQLite*-Datenbank
[15] Ausführungsstrang; Es können mehrere Prozesse *(Threads)* asynchron abgearbeitet werden
[16] Ein Observer („Beobachter“) definiert die Reaktion auf eine Veränderung der `LiveData`

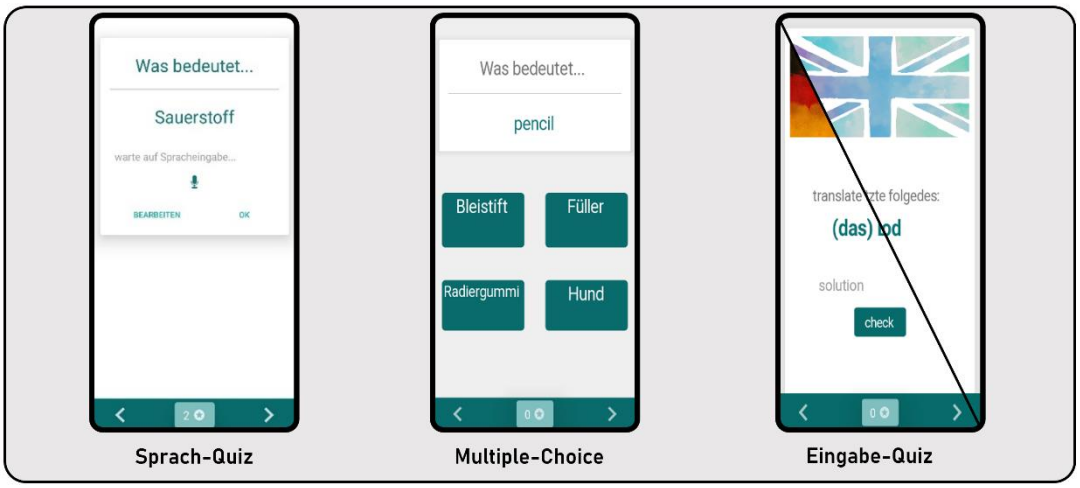

Abb. 5: Unsere verschiedenen Quiz-Modi

2.2.3 Statistik-*Activity* und Datenbankauswertung

In dieser *Activity* kann der Benutzer eine Übersicht seines aktuellen Lernfortschrittes einsehen, welcher sich aus verschiedenen Datenbankwerten zusammensetzt. Hierbei stehen dem Nutzer ein Kreisdiagramm zum Verhältnis aus gut, schlecht und ungelernten Vokabeln, ein Balkendiagramm mit den dazugehörigen absoluten Werten[17] sowie eine Aktivitätsübersicht der letzten 7 Tage zur Verfügung. Zudem ist eine vereinfachte Version dieser Statistiken in der Home-*Activity* einsehbar, worüber diese *Activity* gleichzeitig auch aufgerufen wird (vgl. Abb. 6).

Zunächst wird im *XML-Layout* der Diagramm-Typ-Container platziert und mit der entsprechenden Java-*Klasse* verknüpft, woraufhin Statistik-Daten aus der Datenbank geladen werden.

Abb. 6: Alle Statistiken im Überblick

Danach kommt es zur Abfrage der Daten über anwendungsorientierte SQL-Befehle. Beispielsweise fragt der Ausdruck `SELECT AVG(NumberOfTrainedVocabs) FROM Weekdays;` die durchschnittliche Anzahl der täglich gelernten Vokabeln ab und gibt das Ergebnis als `Double`[18] an ein `ViewModel` weiter. Für komplexere Auswertungen (z.B. für das Kreisdiagram) kombinieren wir die Rückgabewerte der SQL-Abfrage mit Berechnungen in Java. Damit erzeugt man eine `ArrayList` mit den Diagramm-Daten und übergibt sie an die (*Instanz* einer) `DataSet`-*Klasse*. Schließlich werden die zugehörigen Grafiken mit Hilfe der Bibliothek *MPAndroidChart* erstellt.
Problematisch gestaltete sich jedoch zum einen die Verwaltung der Wertepaare im 7-Tage Verlauf, da mit jedem App-Start alte Werte überschrieben, zurückgesetzt und neu sortiert werden müssen. Zur

[17] Bestehend aus: Menge der gut, schlecht & ungelernten Vokabeln
[18] 8 Byte Fließkommazahlen-Datentyp in Java

Lösung dessen legten wir eine zusätzliche Hilfstabelle „Weekdays“ an, in die dieser Prozess ausgelagert werden konnte. Zum anderen bedurfte es schließlich noch einer optischen Feinabstimmung, um störende Prozentwerte im Kreisdiagramm auszublenden.

2.2.4 Online-Mehrspieler-Quiz

In diesem Modus kann der Nutzer online gegen seine Freunde antreten, was, wie bereits in 1.3 erläutert, zu einer zusätzlichen Lernmotivation führt. Hier können bis zu 10 Spieler dieselben Vokabeln zusammen üben und ihr gewonnenes Wissen unter Beweis stellen.
Zur Umsetzung wird in diesem Fall eine online Datenbank benötigt, wobei wir zunächst auf Googles *Firebase Cloud Firestore* zurückgriffen.
Erstellt ein Benutzer (Administrator) ein Spiel und legt die Spieleinstellungen fest, werden diese anschließend zusammen mit den benötigten Vokabeln an *Firebase* übergeben. Andere Spieler können mit einem zuvor vom Administrator festgelegten Code einer Lobby beitreten und erhalten die benötigten Vokabeln von der Datenbank. Daraufhin beginnt das Spiel, welches genauso abläuft, wie die offline Variante. Erst nach Beendigung der Runde wird die Datenbank wieder aktiv, sodass geprüft werden kann, ob alle Teilnehmer das Quiz beendet haben und die entsprechenden Punktestände übermittelt werden können. Ist `if (notFinished) {…}` *true* werden die Resultate präsentiert und es kann nach Belieben erneut gespielt werden.
Jedoch bot dieser Ansatz einerseits die Problematik, dass die Kommunikation mit der Datenbank erst am Ende eines Spieles erfolgen konnte. Dadurch konnte beispielsweise nicht registriert werden, ob ein Spieler die Runde vorzeitig verlassen musste, wodurch die Mitspieler in eine endlose Warteschleife gerieten und diese das Quiz nicht abschließen konnten. Andererseits stellt *Cloud Firestore* lediglich einfache Speicherfunktionen bereit, sodass keine komplexen *EventListener*, die Änderungen in der Datenbank registrieren, zur Verfügung stehen. Deshalb musste das gesamte online Datenbanksystem von *Cloud Firestore* auf die *Firebase Realtime Database*[19] *(vgl. Abb. 7)* umgestellt werden.

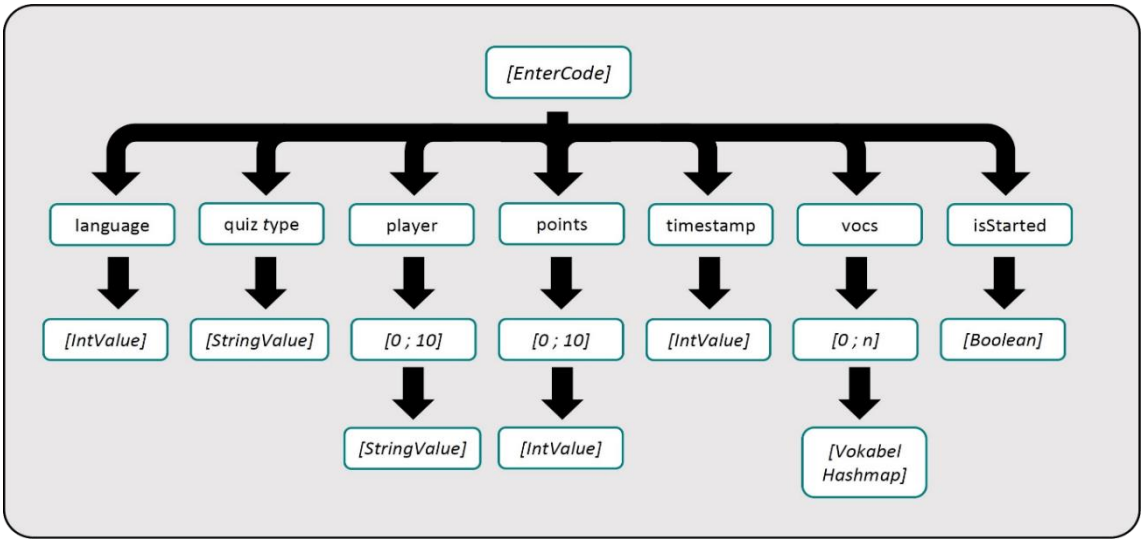

Abb. 7: Übersicht der Child-Elemente (Realtime Database)

Das erste Problem ließ sich, dank des erhöhten Abfragekontingents der Echtzeitdatenbank, mit sekündlichen Datenbankzugriffen lösen. Hierfür verwenden wir folgende `TimerTask`:
`new Timer().scheduleAtFixedRate(new TimerTask() {…}, 1000);`.

Davon abgesehen konnten mithilfe von *EventListenern*[20], wie z.B.:

```
addValueEventListener(new ValueEventListener() {
        @Override
        public void onDataChange(@NonNull DataSnapshot snapshot) {
                snapshot.getValue();
}});
```

,

[19] Echtzeitdatenbank, die die Live-Abfrage von Datensätzen ermöglicht
[20] Der *Listener* wird nur auf bestimmte Child-Elemente (z.B. isStarted) der Datenbank angewendet. So werden ausschließlich relevante Änderungen erfasst.

Methoden in Reaktion auf Datenbankänderungen *(Trigger)* ausgeführt werden. Das bedeutet beispielsweise, dass der Lobby-Administrator das Spiel startet und alle anderen Mitspieler in Echtzeit auf diese Änderung reagieren, wodurch das Quiz beginnt. Besagte Änderungen sind im *Objekt* der `DataSnapshot`-*Klasse* enthalten und können dadurch verarbeitet werden.
Zusätzlich brachte die Echtzeitdatenbank den Vorteil, bestimmte Datenbankoperationen durch einen externen Server ausführen zu lassen, sodass diese nicht mehr dezentral von einem Nutzergerät aufgerufen werden. Um *Firebase Cloud Functions* nutzen zu können, stiegen wir vom kostenlosen *Firebase Spark* auf das kostenpflichtige *Blaze* Paket um. Diese *Cloud Functions* ermöglichen wiederum den Zugriff auf den zuvor genannten Server. Hierfür verwenden wir *Node.js*[21], welches sich als Paket aus einer *index.js* und diversen Konfigurationsdateien zusammensetzt. Dabei befinden sich die eigentlichen *Cloud Functions* in der *index.js.* Ein Beispiel für eine solche *index.js* stellt unsere Datenbankbereinigung dar, welche ungenutzte Lobbies folgendermaßen automatisch aus der Datenbank löscht:

```
FUNCTIONS.DATABASE.REF('{ENTERCODE}/').ONWRITE(ASYNC (CHANGE) => {
            ⋮
  SNAPSHOT.FOREACH(CHILD => {
    UPDATES[CHILD.KEY] = NULL;
  });
  RETURN REF.UPDATE(UPDATES);
});
```

.

Für *Cloud Functions* gibt es vier verschiedene *Trigger*[22], von denen `onWrite` am häufigsten aufgerufen wird, weil er die restlichen drei umfasst. Daher ist dieser für die Datenbankreinigung am besten geeignet. Hierfür wird jener *Trigger* auf das *root*-Verzeichnis (`{entercode}`) gelegt, sodass jeder veraltete Pfad gelöscht wird, indem alle *Child*-Elemente *(vgl. Abb. 7)* auf `null` gesetzt werden.
Zur Ausführung der *Cloud Functions* muss die *index.js* nun abschließend auf den *Firebase*-Server hochgeladen werden. Dazu installieren wir *Firebase* mittels *npm*[23] in der *Shell* und senden über `firebase deploy --only functions` *Node.js* Dateien an den *Cloud Functions Server*.

2.2.5 Barrierefreiheitsfunktion für Sehbehinderte

In Folge einer Warnmeldung, dass bestimmte Funktionen für sehbehinderte Menschen unzugänglich sind, entschieden wir uns für die Entwicklung eines Modus‘ für Sehbehinderte, welcher die App auf verbale Interaktion umstellt. Wird dieser Modus in den Einstellungen aktiviert, leiten Sprachkonserven den Nutzer durch alle Funktionen und geben ihm konkrete Anweisungen zur Bedienung. Dafür nutzt dieser einen Doppeltipp, um einen Sprachbefehl einzuleiten.
Im Blindenmodus erzeugt jede *Activity* eine Instanz der `VoiceControl`-Klasse, welche sich im Anschluss um die Verwaltung der Spracheingabe mit der `VoiceInput`-Klasse kümmert und diverse vordefinierte Reaktionsmechanismen (Vokabelanzahlabfrage, Befehlsauflistung, etc.) und Sprachausgabe-Methoden *(TTS*[24]*)* bereitstellt.

Im ersten Entwurf konzentrierten wir uns auf die Aktivierung der Spracheingabe über ein Aktivierungswort, wie es von diversen Sprachassistenten bereits bekannt ist. Dafür verwendeten wir die `SpeechRecognizer`-Bibliothek von Google, welche die KI-basierte Spracherkennung übernimmt. Daraus entstand das Problem, dass durch wiederholtes Abspielen eines Signaltons eine negative Benutzererfahrung entstand. Mit diesem Signalton kennzeichnet Android die Aktivierung des Mikrophons und schützt so vor Missbrauch. Daraufhin entschieden wir uns für den Umstieg auf eine Aktivierung via Touch-Geste. Am geeignetsten erschien uns hierfür der Doppeltipp.
Bei der Testung der resultierenden App-Version fiel auf, dass es aus nicht vollständig geklärten Umständen auf diversen Geräten zu Fehlern beim Aufruf des Spracherkennungsdienstes kam. Auch der Umstieg auf eine neue Bibliothek sowie Anpassungen ihres Open-Source-Quellcodes konnten das Problem nicht lösen. Aus diesem Grund entscheiden wir uns dafür den gesamten Quellcode erneut umzuschreiben und auf den Android-`RecognizerIntent` zurückzugreifen, der über den Befehl `new Intent(RecognizerIntent.ACTION_RECOGNIZE_SPEECH);` instanziiert wird und schlussendlich

[21] Laufzeitumgebung für *JavaScript*
[22] `onCreate`, `onUpdate`, `onDelete` & `onWrite`
[23] *JavaScript* Paketmanager, der unter anderem die Installation von *Firebase* ermöglicht
[24] Text To Speech ≙ (engl.) Text zu Sprache

einen Systemdialog zur Spracheingabe öffnet.
Nachdem der Modus für Sehbehinderte nun grundlegend funktionierte, begannen wir mit dessen Implementierung in allen *Activities* und erkannten schließlich den Bedarf komplexerer App-Nutzer-Dialoge. Zu diesem Zweck entwickelten wir schließlich das Konzept der von uns als Prozeduren bezeichneten Frage-Antwort-Mechanismen. Hierfür erzeugt jede *Activity* ein `Procedure`-Objekt:

```
new Procedure() {
    @Override
    public ArrayList<String> getQuestions() {…}
    @Override
    public void onAnswer(int index, @NotNull String question, @NotNull String answer) {…}
    @Override
    public boolean onAnswersAvailable(@NotNull ArrayList<String> answers) {…}
};
```

Anschließend wird die Prozedur mit dem Kommando „starte *[Name der Prozedur]*“ gestartet, woraufhin die entsprechenden Fragen über `getQuestions` abgerufen werden. Die gegebenen Antworten verarbeitet hingegen die `onAnswer`- bzw. `onAnswersAvailable`-*Methode*. Durch diese App-Nutzer-Dialoge ist nun eine variationsreiche App-Bedienung mit beliebig vielen Prozeduren pro *Activity* möglich.

3 Veröffentlichung

3.1 Optimierung der App-Performance

Nachdem das Zusammensetzen der App vollständig abgeschlossen war, bestand die Notwendigkeit diese leistungsfähiger und weniger speicherintensiv zu gestalten. Dies erreichten wir einerseits durch die Bereinigung unseres Quellcodes *(Code-Cleanup)*, indem wir Kommentare, ungenutzte *Variablen*, *Methoden* und Dateien, überflüssigerweise importierte *Klassen* sowie nicht benötigte *XML-Container* (Layout-Elemente) entfernten.

Andererseits nutzten wir verschiedene *Android Studio* Dienste (*Android Size Analyzer*, *Lint* und *APK Analyzer*), um verstecktes Verbesserungspotenzial aufzudecken. Hierdurch kamen wir beispielsweise auf die Idee, die meisten unsere Bild-Dateien in Web-Pictures zu konvertieren.
Die zuvor beschriebenen Maßnahmen richteten sich ausschließlich an Java- und *XML*-Dateien, jedoch verfügt jede Android-App zudem über ein *Build-Tool (Gradle)*, welches die App aus den Quell-Dateien erstellt *(Build)*, diese ausführt *(Run)* und Tests *(Debugging)* ermöglicht. Daher veränderten wir auch einige *Gradle*-Eigenschaften und entfernten überflüssig gewordene *Dependencies*[25].
Abgesehen davon haben wir einige Codeänderungen zur Verbesserung der Performance vorgenommen. Dafür kennzeichneten wir z.B. alle *Methoden* und *Variablen*, bei denen es möglich war, als unveränderlich während der Laufzeit (`final`), oder als unabhängig von der Instanz (`static`).

Zum Schluss veröffentlichten wir unsere App als *Android App Bundle („.aab“), welches* eine Alternative zum gängigen Format *APK*[26] darstellt. Dadurch kann laut offiziellem *Android Developers Blogs [8]* die Dateigröße um durchschnittlich 15 % verringert werden. Dies wird dadurch erzielt, dass Google Play automatisch zwei unterschiedliche *APKs* aus einer *„.aab“* Datei erstellt, wobei eine als *Basis APK* (für alle Geräte) und eine als *Konfigurations-APK* (gerätespezifisch) fungiert.

Letztendlich gelang es uns, als Resultat dieser Optimierungen, die Größe der Installationsdatei von ca. 325 MB auf ca. 80 MB zu senken.

3.4 Durchführung und Auswertung einer Studie

Zur Evaluation unserer aktuellen App-Version starteten wir einen sogenannten „Offenen Tests“ im Google Play Store. Anschließend erstellten wir eine Online-Umfrage mit 20 Fragen über die gesammelten Erfahrungen der Tester.

Für die Auswertung der Rohdaten gliederten wir diese zunächst nach Jahrgangsstufen, Hersteller, Gerätetyp und Android-Version, sodass wir korrelative Tendenzen innerhalb der zuvor genannten

[25] Zur Einbindung externer *Bibliotheken* und *Modulen*
[26] *Android Package* ≙ Dateiformat für Installationsdateien unter Android

Gruppen ausfindig machen konnten. Daraufhin leiteten wir aus den verschiedenen Datengruppen problembehaftete sowie bereits zufriedenstellende Bereich der App ab. Dadurch gelang es uns eine Planungsgrundlage für die zukünftige Entwicklungsarbeit zu schaffen.
Konkret ließen sich aus der Umfrage folgende Haupterkenntnisse ziehen:

- **Kompatibilitätsprobleme** zwischen bestimmten Eingabefeldern und Darkmode
- **App-Design** hat eine mehrheitlich zufriedenstellende Wirkung
- Angemessener Grad der **Benutzerfreundlichkeit** (vgl. Abb. 9)
- Schwierigkeiten mit der **Einscann-Funktion**
- Bedarf an **App-Tutorials** bestätigt
- App-Idee wird positiv bewertet
  ⇒ Hohes **Marktpotential** bestätigt
- Erhebliche Nachfrage durch **IOS**-Nutzer

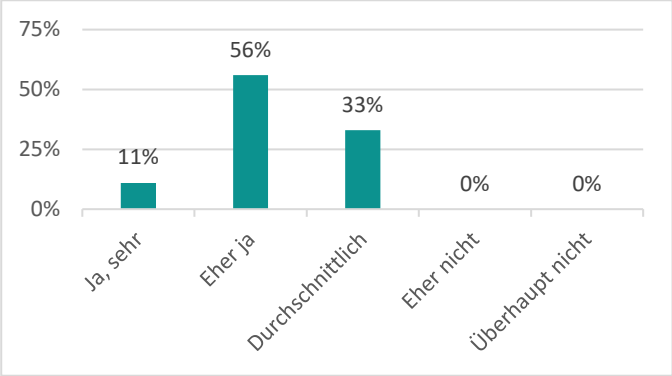

Abb. 9: Benutzerfreundlichkeit

3.5 Aufarbeitung der Studienergebnisse

Zunächst befassten wir uns mit mehreren kleineren Problemen, wie bspw. der Kompatibilität von Darkmode und Eingabefeldern. Ungeachtet der äußerst positiven Rückmeldungen bezüglich des Layouts sowie der Benutzerfreundlichkeit, entschieden wir uns für eine gänzliche Überarbeitung des Navigationskontenpunktes der App, das Home-Menü. Zur Steigerung der Übersichtlichkeit lagerten wir die gesamte Navigation in einen `FloatingAppButton`[27] in der unteren, rechten Ecke aus, wodurch die Mehrheit der Funktionen lediglich bei Bedarf eingeblendet wird. Des Weiteren führten wir einen Schnellzugriff für die Kernfunktionalitäten, also die Offline- und Online-Quiz sowie die jeweiligen Vokabelkategorien ein. Zur nachhaltigen Verbesserung des Wortschatzes inkludieren wir ebenfalls eine Tabelle mit zufälligen Vokabeln zur täglichen Wiederholung am unteren Ende des Home-Menüs. Eine Kurzübersicht ausgewählter Statistiken (vgl. 2.2.3) stellt für den Nutzer einen weiteren direkten Mehrwert bereit, welcher sofort nach App-Start zur Verfügung steht.

Aufgrund aktueller Markttendenzen zur Logo-Simplifizierung entschieden auch wir uns für die Anpassung unseres App-Logos (vgl. Abb. 10):

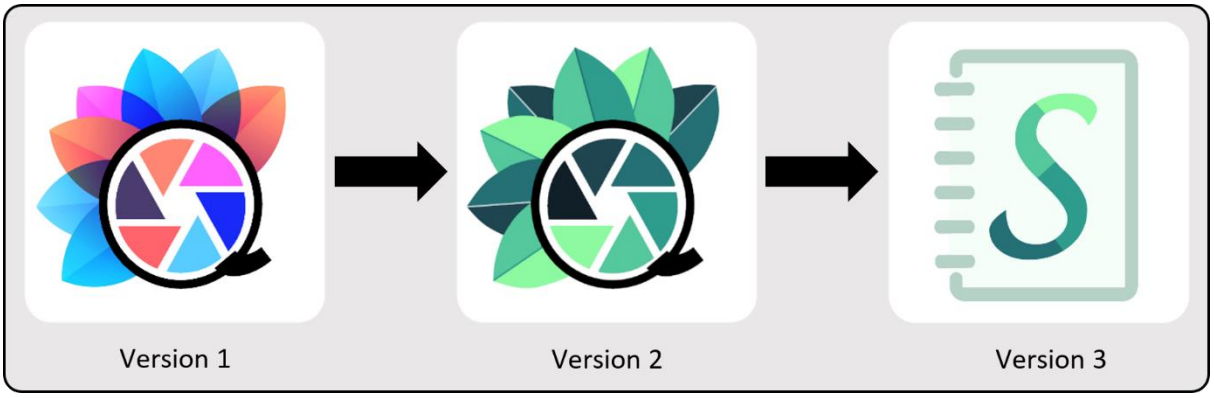

Abb. 10: Logovergleich

Die aufgekommenen Probleme mit der Einscann-Funktion stellten einen wesentlich höheren Arbeitsaufwand dar als die restlichen Schwierigkeiten. Aus diesem Grund erklärten wir diesen Sachverhalt zu unserem Entwicklungsschwerpunkt, weswegen insbesondere hinsichtlich weiterer Konfigurationsmöglichkeiten von *Tesseract* als auch komplexerer Bildoptimierung gearbeitet wird.
Als weniger relevant befinden wir die Erstellung weiterer detaillierter Tutorials, da diese vergleichsweise wenig Entwicklungsaufwand erfordern, weshalb wir diese erst in einem letzten Arbeitsschritt kurz vor der Veröffentlichung ScanQs implementieren wollen.
Bezüglich der hohen Nachfrage nach einer IOS-Version der App, steht für uns nun fest, dass eine spätere Veröffentlichung im Apple App Store zweifelsohne Notwendigkeit besitzt.

[27] Button mit fixierter Position zur Einblendung eines animierten Pop-Up-Menüs

4 Was haben wir erreicht?

4.2 Auswirkungen auf den Schulalltag

Die klassische Vorbereitung eines Schülers auf einen Vokabel-Test verläuft wie folgt:
Der Schüler kommt nach Hause und schreibt die zu lernenden Vokabeln aus seinem Schulbuch in ein Vokabelheft ab. Nun ist er an dieses gebunden und kann ohne es nicht überall lernen.

Dahingegen ermöglicht *ScanQ*, als ständiger Begleiter, jederzeit, ortsungebunden, ob zuhause, im Bus, auf dem Weg zur Schule oder in Freistunden zu lernen. Vokabeltraining kann somit komfortabel in jede Alltagssituation integriert werden.
Zudem bietet *ScanQ* die gezielte Erkennung von Lerndefiziten und deren Priorisierung in den Quiz. Ferner deckt die Vielzahl der Quiz unterschiedliche Abfragetypen ab, sodass dieser in der Lage ist, Vokabel-Fragen in wechselnden Kontexten zu beantworten.
Abgesehen davon werden bei gleichbleibender Reihenfolge der Vokabeln, weniger die Vokabeln selbst, als deren Muster und Reihenfolge verinnerlicht. Darum überprüft die App Vokabeln stets in zufälliger Abfolge. Überdies schafft unsere App eine digitale Alternative für das gemeinschaftliche Lernen in Pandemiezeiten. Dies ist unser *Online-Multiplayer-Quiz (vgl. 2.4)*.

4.3 Vergleich mit anderen Apps (zu 1.2)

Im Vergleich mit anderen, ähnlichen Apps verfügt unser fertiger Software-Entwurf über zahlreiche Vorteile. Die untenstehende Tabelle veranschaulicht diese in Bezug auf beliebte Play Store Apps mit über 100.000 Downloads.

Hierbei erhalten die jeweiligen Sieger einer Kategorie einen Vergleichspunkt, welcher in die Gesamtwertung einfließt.

[Aus Gründen der Vereinfachung wird angenommen, dass alle Kategorien als gleichwertig zu werten sind, wodurch die Vergleichbarkeit in der Gesamtwertung erhalten bleibt.]

	ScanQ	Green Line	Cabuu	Pons	Vokabel-trainer[28]
Kosten	**kostenlos**	0,59 - 5,99€ pro Einheit	47,88€ / Jahr	0,59 - 16,99€ pro Vokabel-erweiterung	**kostenlos**
Freie Vokabel-eingabe	**Ja**	Nein	teilweise[29]	Nein	**Ja**
Online-Multiplayer	**Ja**	Nein	Nein	Nein	Nein
Layout	**appeigenes Layout**	Android-Standard-Layout	**appeigenes Layout**	Android-Standard-Layout	Android-Standard-Layout
Einscannen	**Ja**	Nein	**Ja**	Nein	Nein
Werbung	**Nein**	**Nein**	**Nein**	**Nein**	**Auf Wunsch**
Quiz-Spiele	3 Spiel-Modi	**5 Spiel-Modi**	3 Spiel-Modi	**5 Spiel-Modi**	3 Spiel-Modi
Statistiken	**Ja**	Nein	**Ja**	Nein	**Ja**
Blindenmodus	**Ja**	Nein	Nein	Nein	Nein
Lernpläne	In Arbeit	Nein	**Ja**	Nein	Nein
Gesamt	**8**	2	5	2	4

Betrachtet man den tabellarischen Vergleich, so ist zu erkennen, dass *ScanQ* mit 8 von 10 Punkten in den meisten Kategorien die größten Vorzüge bietet, wohingegen *Cabuu*, mit drei Punkten weniger (5 / 10), den zweiten Platz belegt. Dies kann auf unsere Entwicklungsstrategie zurückgeführt werden, die darauf basiert, Defizite frühestmöglich zu erkennen, diese zu beheben und zugleich aus Fehlern

[28] Uneindeutiger App-Name; Entwickler: Ruben Gees
[29] Nur für im Langenscheidt-Wörterbuch enthaltene Vokabeln

anderer Anwendungen zu lernen. Insbesondere der *Online-Multiplayer-Modus* verkörpert ein Alleinstellungsmerkmal unserer App und unterstreicht ihren innovativen Charakter.
[Eine auf Konsequenzen bezogene Auswertung der Tabelle erfolgt in der Ergebnisdiskussion (4.4).]

4.4 *ScanQ* als Marktinnovation

Als Marktinnovation bezeichnet man gemeinhin ein Angebot, in unserem Fall eine App, die in ihrem jeweiligen Markt (*Play Store*) in bestimmten Aspekten eine vollkommene Neuerung darstellt.

Dafür verglichen wir *ScanQ* mit den verfügbaren Angeboten des App-Mark *(vgl. 4.3)*. Daraus ergab sich das Resultat, dass unsere Anwendung, unter allen Vergleichspartnern, die meisten Funktionen abdeckt. Der so entstehende Wettbewerbsvorteil gründet auf folgenden Innovationen:

- **Freies Einscannen** ohne vorgeschriebene Vokabellisten
- Einzigartige, nie dagewesene Features, wie …
  - … einem **Blindenmodus**
  - … einem **Online-Mehrspieler-Modus**

5 Was ist noch geplant?

Auf langfristige Sicht ist die Ergänzung der App *(vgl. Abb. 11)* um einen viel aufwendigeren *Online-Multiplayer-Modus* geplant, welcher zudem eine Login- und Registrierungsfunktion erfordert. Diese wurde zwar bereits in früheren Stadien der Ausarbeitung erstellt, jedoch vorerst verworfen, da sie in den ersten Versionen nicht benötigt wurde. Erst mit Erweiterung der Online-Nutzung um Spieler-Profile, Ranglisten und Gruppen wird diese Funktion Notwendigkeit erhalten.

Ebenso soll *ScanQ* in Zukunft über einen Modus zur Erstellung individueller Lernpläne verfügen, welchem wir uns in unserer Entwicklung, nach der erfolgreichen Fertigstellung von *Version 2*, widmen. Diese erstellen spezielle „Lern-Quiz“ zur Vorbereitung auf Tests und Überprüfungen, die definitiv separat von den bisherigen Quiz-*Activities (vgl. 2.2.2)* betrachtet werden müssen.

Zur Erstellung eines solchen Lernplans gibt der Schüler seinen Prüfungstermin und die zu lernenden Vokabeln an. Daraus berechnet die Anwendung die tägliche Menge an zu lernenden Wörtern unter Einbeziehung der individuellen Präferenzen des Nutzers, indem dieser beispielsweise einstellt, ob Vokabeln mehrfach oder einfach zur Vorbereitung abgefragt werden sollen. Versäumt es ein Schüler an einem Tag zu lernen, passt sich die Anzahl der in den nächsten Tagen zu lernenden Wörter automatisch an.

Nun ist der Lernplan erstellt, im System-Kalender ein Eintrag, bezüglich des Prüfungstermins, hinzugefügt und der Benutzer erhält, je nach persönlicher Einstellung, eine oder mehrere Erinnerungen zu lernen. Diese Funktion bietet gerade unerfahrenen, jungen Schülern, denen oftmals das Einschätzungsvermögen für Dauer und Lernaufwand fehlt, eine Möglichkeit Struktur und Ordnung in ihre Vorbereitungen zu bringen.

Weiterhin ist auch die Ergänzung der von vielen Testern gewünschten weiteren Tutorials fest eingeplant, um die Bedienung noch verständlicher zu gestalten. Diese sollen sich stilistisch an den bereits bestehenden Tutorials orientieren, wobei hierfür das App-Intro angeführt werden kann.

Als letzte Ergänzung sehen wir vor, dass bei jedem App-Start Tipps und Lernstrategien eingeblendet werden, was auf Wunsch deaktiviert werden kann. Zur Realisierung gedenken wir erneut die lokale Datenbank von *RoomDB* zu verwenden, damit diese Hinweise jedoch stetig erweitert werden können, planen wir bei jedem App-Start einen Abgleich mit einer *Firebase* Datenbank durchzuführen. In diesem Fall würden wir den *Cloud Firestore* von Firebase verwenden, da keine Notwendigkeit für eine Echtzeitdatenbank besteht. In dieser sollen dann Versionsnummern für den Datensatz hinterlegt sein, sodass stets verglichen werden kann, ob eine Aktualisierung vorliegt. Ist dies der Fall, werden alle lokal gespeicherten Datensätze mit den neueren überschrieben.

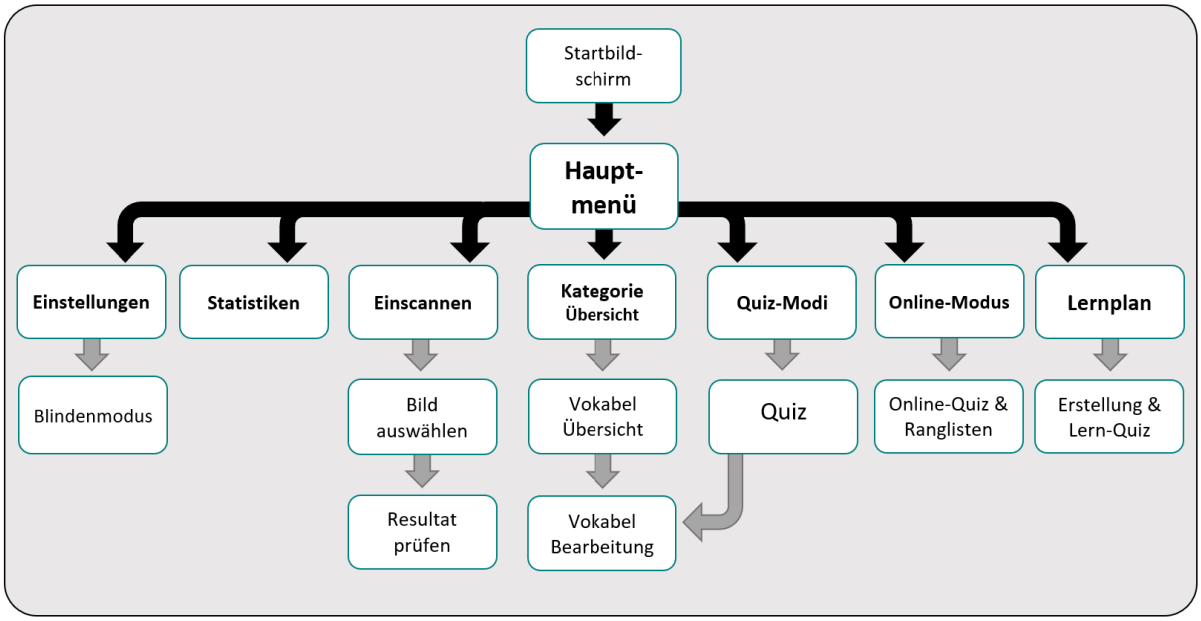

Abb. 11: zukünftige App-Struktur

6 Zusammenfassung

Unser Ziel war es, das Vokabellernen effektiver, moderner und amüsanter zu gestalten und das schlichte Üben in ein effizienteres Vergnügen umzuwandeln.
Daher entschieden wir uns für die Entwicklung einer Android-App, die das Lernen erleichtert, indem Vokabeln schnell eingescannt *(vgl. 2.2.1)* und automatisch in Lernspielen *(vgl. 2.2.2)* bereitgestellt werden.
Ebenfalls von großer Bedeutung war das Erreichen einer größtmöglichen Zielgruppe, weswegen wir uns in der Entwicklung für eine kosten- und werbefreie Anwendung, mit neutralem Layout und barrierefreier Benutzerführung *(vgl. 2.2.5)*, entschieden.

Die Umsetzung dieser Ziele gelang uns zudem lediglich durch die Tatsache, dass wir in unserer Erarbeitung jedes Problem so lange zu beheben versuchten, bis eine Lösung oder ein Weg, es zu umgehen, gefunden war. Diese Arbeitsweise erforderte zwar viel Zeit und Aufwand, weswegen der Fortschritt auch teilweise stagnierte, jedoch überzeugt uns der Mehrwert geringerer Fehleranfälligkeit und angenehmerer Usability.

Aufgrund unserer ausgesprochen kurzen Entwicklungszeit von fast zwölf Monaten sind noch nicht alle Funktionen in einer Qualität entwickelt, in der wir sie uns wünschen. Besonders die Ergebnisse der Einscann-Funktion sind zwar unter normalen Bedingungen zufriedenstellend, jedoch erachten wir es für sinnvoll noch diverse Anpassungen, wie in 3.5 beschrieben, vorzunehmen. Zudem erhoffen wir uns durch einen Sponsor, unseren OCR-Dienst auf die Cloud-basierte Lösung von „Abbyyocr“ *(vgl. 2.2.1)* umstellen zu können.

Darüber hinaus befinden sich zahlreiche Funktionen noch in der Planungsphase, die, nach der vollendeten Optimierung zuvor genannter Features, umgesetzt werden. Hier sind beispielsweise die Implementierung weiterer Fremdsprachen und der individuelle Lernplan *(vgl. 5)* vorgesehen.

Abschließend planen wir erneut auch unsere aktuelle Version von *ScanQ* (*Version 2*) in einer praktischen Testphase mit Nutzern aus den verschiedenen Jahrgängen unserer Schule zu testen. Von der Auswertung der Resultate erhoffen wir uns weitere Anregungen für zusätzliche Funktionen und die Möglichkeit letzte Fehler in den bestehenden Funktionen zu beheben. Ebenfalls gedenken wir dadurch, nachhaltig eine weitere Verbreitung ScanQs zu fördern.

Quellen- und Literaturverzeichnis

[1] Hiroko Miyake und Rie Owaku (Tokyo Keizai University, Japan), Implementing the Activity of Copying Text by Hand (CTH) in “English E-Learning”: A Case Study at Tokyo Keizai University, https://infonomics-society.org/wp-content/uploads/ijels/published-papers/volume-2-2012/Implementing-the-Activity-of-Copying-Text-by-Hand-CTH-in-English-e-learning-A-Case Study-at-Tokyo-Keizai-University.pdf, besucht am 26.12.2020

[2] Dr. Anna Hawlitschek; „Spielend Lernen - Wissensprozesse und digitale Medien, Bd. 20“; Logos Verlag, Berlin; 2013; S. 190

[3] Springer Nature, Wettbewerbsmotivation, https://media.springernature.com/lw685/springer-static/image/art%3A10.1365%2Fs40702-018-00481-7/MediaObjects/40702_2018_481_Fig5_HTML.png, besucht am 27.12.2020

[4] Statista GmbH, Vergleich der Marktanteile von Android und iOS am Absatz von Smartphones in Deutschland von Januar 2012 bis September 2020, https://de.statista.com/statistik/daten/studie/256790/umfrage/marktanteile-von-android-und-ios-am-smxartphone-absatz-in-deutschland, besucht am 28.12.2020

[5] Sergej Neumann (Universität Koblenz-Landau, Bachelor-Arbeit), Entwicklung einer Android-App zur Erkennung und Übersetzung von Worten in Kamerabildern, https://kola.opus.hbz-nrw.de/opus45-kola/frontdoor/deliver/index/docId/699/file/bach222.pdf, besucht am 27.12.2020

[6] Kelsey Taylor, List of Top 5 Open Source OCR Tools, https://www.hitechnectar.com/blogs/open-source-ocr-tools/, besucht am 28.12.2020

[7] GitHub Pages, Improving the quality of the output, https://tesseract-ocr.github.io/tessdoc/ImproveQuality.html, besucht am 02.09.2020

[8] Hoi Lam (Developer Relations Engineer), New Android App Bundle and target API level requirements in 2021, https://android-developers.googleblog.com/2020/11/new-android-app-bundle-and-target-api.html

Bildquellen:

Alle Grafiken wurden eigenständig erstellt. Sofern externe Daten verwendet wurden, ist dies an entsprechender Stelle vermerkt.

Alle Bilder wurden eigenständig als Screenshots der *com.rommelbendel.scanq* Applikation erstellt.