Universität Ulm | 89075 Ulm | Germany

Fakultät für Ingenieurwissenschaften und Informatik
Institut für Datenbanken und Informationssystemen

Andreas Kilian
andreas.kilian@uni-ulm.de

psyq – Entwicklung eines App-Systems zur spielerischen Wissensvermittlung und deren Verknüpfung mit psychologischen Fragebögen am Beispiel des Android-Frameworks

Masterarbeit – Dokumentation

2015

Vorgelegt von: Andreas Kilian

Gutachter: Prof. Dr. Manfred Reichert
Jun.-Prof. Dr. Claudia Schrader

Betreuer: Dipl.-Inf. Rüdiger Pryss

Fassung vom 23. Januar 2015

Inhaltsverzeichnis

1 Einleitung 6

1.1 Motivation 6

1.2 Idee und Zielsetzung 7

2 Related Work 9

2.1 Anki 9

2.2 Repetico 11

3 Anforderungsanalyse 13

3.1 Technische Umgebung 13

3.2 Anforderungen 16

3.3 Anwendungsfälle 21

4 Entwurf und Konzept 22

4.1 Allgemeines Konzept 22

4.2 Begrifflichkeiten 23

4.2.1 Akteure 23

4.2.2 Technische Teilnehmer 24

4.2.3 Themenbereiche 24

4.2.4 Grund-Elemente der Quizze und Fragebögen 25

4.2.5 Played-Elemente (gespielte Quizze und ausgefüllte Fragebögen) 26

5 Architektur 28

5.1 Basis-Architektur 28

5.2 Struktur der Daten 29

5.2.1 Serverseite und App 31

5.2.2 Rahmendaten in der App 33

5.3 Informationsfluss-Struktur 35

5.3.1 Client-Server-Kommunikation ... 35
5.3.2 Kommunikation innerhalb der App ... 35
5.4 App: Paket- und Klassenstruktur ... 36
5.5 App-Dialogstruktur ... 38
5.6 Bibliotheken (Server) ... 40
5.6.1 RedBeanPHP ... 40
5.6.2 Bootstrap ... 40
5.7 Bibliotheken (App) ... 40
5.7.1 Support-Library ... 40
5.7.2 Realm ... 41
5.7.3 Gson ... 42
5.7.4 EventBus ... 43
5.7.5 Android Priority Job Queue ... 44
5.7.6 Weitere Bibliotheken ... 44
6 Implementierung ... 46
6.1 App ... 46
6.1.1 Material Design und Android Design Guidelines ... 46
6.1.2 Fragments ... 52
6.1.3 Rahmendaten ... 55
6.1.4 Jobs ... 58
6.1.5 Events ... 60
6.1.6 Herausforderungen ... 62
6.2 API ... 63
6.2.1 JSON-Format ... 64
6.2.2 Rückgabedaten ... 66
6.3 Backend ... 66
6.3.1 Daten-Verwaltung ... 67
7 Walkthroughs ... 69
7.1 Szenario 1: Anlegen eines Decks im Backend ... 69

7.2 Szenario 2: Spielen eines Decks mit der App ... 75

8 Anforderungsabgleich ... 80

9 Ausblick und Fazit ... 83

9.1 Mögliche Erweiterungen ... 83

9.1.1 Weitere Game-Modes ... 83

9.1.2 Spielen gegen andere Spieler und Punktevergleich ... 84

9.1.3 Fragen vorschlagen ... 85

9.1.4 Nutzung vorhandener Logins ... 85

9.1.5 Antwortzeit ... 86

9.1.6 Userspezifische Decks ... 86

9.1.7 Gruppierung oder Tagging von Decks ... 86

9.1.8 Suche nach Decks ... 87

9.1.9 Teams ... 88

9.1.10 Unterstützung weiterer Plattformen ... 88

9.2 Fazit ... 88

Abbildungsverzeichnis ... 90

Listingverzeichnis ... 92

Tabellenverzeichnis ... 93

Literaturverzeichnis ... 94

Glossar ... 100

CD-Inhalt ... 101

Erklärung ... 103

1 Einleitung

1.1 Motivation

Die Weiterentwicklung und die zunehmende Verbreitung von Smartphones sowie zunehmend auch von Tablets in den letzten Jahren haben eine große Vielfalt von neuen Nutzungsmöglichkeiten für Anbieter und Anwender hervorgebracht. Hauptgrund hierfür sind die Apps, die in sämtlichen mobilen Infrastrukturen möglich sind und die einen gewaltigen Mehrwert gegenüber klassischen Mobiltelefonen und sog. *Featurephones* bieten. Über die mobilen Applikationen lassen sich Funktionsumfang und mögliche Facetten eines Geräts binnen Sekunden um ein Vielfaches erweitern – oft ohne dass die Idee oder die Möglichkeit dazu zum Zeitpunkt des Kaufs überhaupt schon bestand.

Durch eine eigene App ist der jeweilige Anbieter dauerhaft auf dem Gerät des Nutzers präsent. Daher sind v. a. auch Firmen und Diensteanbieter daran interessiert, sich in der stetig wachsenden Schar von Apps möglichst gut zu präsentieren und zu positionieren.

Nicht nur die Apps selbst, auch die Eigenschaften der Anwender und insbesondere die von ihnen durch Nutzung der Apps generierten Daten sind wertvoll für die Anbieter. Zahlreiche Frameworks zur Feststellung, Messung und Verarbeitung unterschiedlichster Größen im Kontext der App-Nutzung bestätigen dies [1].

Ebenfalls interessant für die Erstellung von Apps sind persönliche Faktoren, wie z. B. die Stimmung oder die Zufriedenheit des Anwenders. Diese nur schwer messbaren Größen interessieren insbesondere dann, wenn sie sich auf die Art der Nutzung auswirken oder gar durch diese beeinflusst werden.
So liegt es nahe, solche Faktoren durch den Anwender selbst einordnen zu lassen. Zwar sind diese Werte immer subjektiv, doch lassen sie sich mit objektiv messbaren Größen verknüpfen sowie anhand einer psychologisch durchdachten Kombination von Fragen und einer Auswahl bestimmter Antwortmöglichkeiten oder -bereiche angemessen auswerten.

1.2 Idee und Zielsetzung

Schon lange vor dem digitalen Zeitalter war die spielerische Wissensvermittlung mithilfe unterschiedlichster Frage-Antwort-Ansätze sowohl eine beliebte Freizeitbeschäftigung als auch eine interessante Lernform.
In Zeiten von immer komplexer werdenden PC-Anwendungen wurde konsequenterweise auch dieser Ansatz adaptiert und führte zu einigen spannenden Quiz-Spielen für den heimischen Computer. So verwundert es wenig, dass auch die Entwicklung im mobilen Bereich sehr schnell entsprechende "Quiz-Apps" nach sich zog.

Für verschiedene mobile Endsysteme stehen unzählige Anwendungen mit vordefinierten oder redaktionell gepflegten Fragen bereit, die für das entsprechende Themengebiet oft durchaus geeignet sind. Neben diesen Anwendungen entstand aber auch schnell der Bedarf an *"Rahmen-Systemen"*. In diesen sind der strukturierte App-Ablauf, das Bewertungssystem und das Einpflegen der Fragen und Inhalte nicht mehr oder nur noch sehr lose mit dem eigentlichen Inhalt verbunden.

Durch diese lose Kopplung ist es möglich, verschiedene Themengebiete und unterschiedliche Anwendergruppen abzudecken, ohne die App oder die Server-Komponente für jeden Einsatzzweck neu entwickeln zu müssen.
Mithilfe eines nutzerfreundlichen Backends können Nutzer, Lehrende, fachliche Experten oder Institutionen (im Folgenden alle als *Autoren* bezeichnet) eigene Inhalte durch das ausgelagerte Anlegen von Fragen und Fragenkatalogen sehr viel einfacher und ohne technisches Fachwissen einbringen.

Diese Herangehensweise verfolgt auch *psyq* – gepaart mit der eingangs erwähnten Möglichkeit, ein subjektives Nutzerfeedback bzgl. der psychologisch interessanten Faktoren zu erheben. Dieser Themenbereich gilt als anspruchsvoll, da die Realisierung von psychologischen Fragebogen-Systemen auf mobilen Endgeräten zahlreiche verschiedene Herausforderungen mit sich bringen kann [2], [3], [4], [5], [6], [7].

Durch den angestrebten Ansatz sollen zum einen Inhalte mühelos generiert, zum anderen soll gleichzeitig Wissen spielerisch vermittelt werden. Des Weiteren sollen die bei der Nutzung entstandenen Ergebnisse gemessen sowie das abgefragte Anwender-Feedback gesammelt werden und damit eine aufschlussreiche Datengrundlage für die psychologische Auswertung liefern.

2 Related Work

2.1 Anki

Anki [8] ist eine Rahmen-System-Infrastruktur zur Eingabe und Abfragen von sog. *Flashcards*. Das System wird bereits seit geraumer Zeit stetig weiterentwickelt und kann daher auf eine große Community und mehrere Entwickler aufbauen.

Anki ist sowohl als Android-App als auch für diverse weitere technische Plattformen erhältlich.

Die Hauptanwendung befasst sich nahezu ausschließlich mit *Flashcards*. Dabei handelt es sich um die digitalen Repräsentation von klassischen Karteikarten. Diese können Begriffe sowie Bilder und Tonaufnahmen enthalten. Zudem erlaubt *Anki* verschiedene Designs, die mithilfe von CSS (Cascading Stylesheets) vergeben werden können.
Weiterhin ist inzwischen die Unterstützung für die Auszeichnungsspache *LaTeX* integriert. Der damit erreichte Mehrwert ist enorm – insbesondere im Hinblick auf naturwissenschaftliche Themen. LaTeX eignet sich optimal, um z. B. komplexe mathematische Formeln oder die Notation von Molekülen im Bereich der Chemie darzustellen.

Seit kurzem ist zudem eine Handschrifterkennung in die Android-App von *Anki* integriert, die die Eingabe von Antworten durch den Anwender erleichtern soll. Die von Nutzern erstellten Decks lassen sich zentral veröffentlichen. Auf diese Weise können von anderen Anwendern eingepflegte Inhalte verwendet und gelernt werden.

Beim Spielprinzip von *Anki* werden dem Spieler nacheinander *Flashcards* angezeigt, die eine Fragestellung, eine Formel, eine Übersetzung o. Ä. enthalten. Der Spieler erhält dabei allerdings keine Antworten zur Auswahl und keine Tipps durch die App. Meint er die richtige Antwort zu kennen, oder gibt er auf, beendet er diese Aufgabe und bekommt anschließend die korrekte Antwort präsentiert. Anhand dieser muss er für sich entscheiden, ob seine eigene Lösung korrekt war und wie sicher er sich dabei war. Er kann die Karte nun in eine von drei Gruppen einordnen, welche drei Schwierigkeitskategorien repräsentieren und damit festlegen, in welchem zeitlichen Abstand die betreffenden Karten wiederholt werden.
Anki bietet allerdings keine Möglichkeit, um zusätzlich falsche Antwortmöglichkeiten zu erstellen. Dadurch entfällt die *ChooseOne*-Spielvariante. *psyq* setzt diese Spielvariante um und bietet dem Spieler die korrekte sowie eine Anzahl falscher Antworten zur Auswahl an.

Die Resultate gespielter Karten sind lediglich für den jeweiligen Spieler ersichtlich. Eine „Autoren"-Funktion, durch die der Verfasser einen Einblick in die mit seinen Decks erzielten Resultate erhielte, ist nicht vorhanden.

Zudem findet die Synchronisierung über die *Anki*-eigenen Server statt, was gerade bei sensiblen Daten ein datenschutzrechtliches Manko darstellt.

Die Realisierung zusätzlicher Fragebögen jeglicher Art ist nicht vorgesehen. Somit ist eine Verknüpfung der Spielergebnisse mit Antworten auf psychologisch relevante Fragestellungen nicht – oder nur mit zusätzlichem Aufwand, erhöhter Durchführungsdauer sowie einer zusätzlichen App – möglich.

2.2 Repetico

Repetico [9] ist ein *Social Learning Network*, um unterschiedliche Arten von Inhalten mithilfe von Online-Karteikarten zu lernen. Die zugehörige App ist daher auf den Bereich des digitalen Lernens spezialisiert und ist sowohl für die Android- als auch für die iOS-Plattform verfügbar. Auch ein Wechsel zwischen verschiedenen Geräten ist möglich, da *Repetico* die Daten in Echtzeit synchronisiert.

Ein hervorzuhebendes Merkmal von *Repetico* ist der Lernplan mit der eigenen Lernvergangenheit, einer aktuellen To-Do-Liste sowie einem Plan für die folgenden Tage. Die Reihenfolge und Anzahl der Wiederholungen wird automatisiert festgelegt. Dabei sollen Aspekte der Lern-Lehr-Forschung und der Neurowissenschaft berücksichtigt werden. Es existieren verschiedene Lernmodi, aus denen der Nutzer wählen kann [10]:

- "Langzeitgedächtnis": *Repetico* entscheidet für jede Karteikarte selbst, wann die nächste Wiederholung angebracht ist.
- "Kurzzeitgedächtnis": In diesem Modus werden alle Karten gelernt.
- "Favoriten": Nur Karten, die der Anwender favorisiert hat, werden abgefragt.

Um zu lernen muss sich der Nutzer zunächst einloggen und kann anschließend selbstständig verschiedene Karteikartensätze anlegen, die in Kategorien unterteilt werden können. Die App bietet aber auch die Möglichkeit, Kartensätze von angesehenen Fachverlagen zu kaufen, wodurch ein hoher Qualitätsstandard sichergestellt wird. Die Import- und Exportfunktion ermöglicht die Verwendung von Inhalten aus anderen Formaten. Eine Karteikarte kann entweder eine Frage-Antwort-Karte sein oder eine *Multiple-Choice*-Frage abprüfen. Aufgrund der nicht zu jeder Zeit vorhandenen mobilen Internetverbindung, besteht die Möglichkeit, die Kartensätze herunterzuladen.

Der *Social Learning* -Gedanke von *Repetico* zeichnet sich in der Community-Funktionalität ab. Hier können Karteikarten gemeinschaftlich erstellt werden und es kann über den Lernstoff diskutiert werden. Zudem lassen sich die Lernaktivitäten und -erfolge der Freunde verfolgen. Für Nutzer, die sich gerne gegenseitig herausfordern, gibt es die Möglichkeit, Erfolge und Fortschritte via *Twitter* und *Facebook* zu teilen.
Einzelpersonen bietet *Repetico* Möglichkeiten, sich selbst zu kontrollieren. Ist der Spieler mit dem Lernstoff noch nicht ganz vertraut, kann er sich die

Karteikarten zunächst sowohl mit der Frage als auch mit der Antwort einblenden lassen. Weiterhin muss das Antwort-Feld beim Lernen nicht zwingend ausgefüllt werden. So ist kein Fehlversuch notwendig, um die korrekte Antwort zu erfahren. Karteikarten können während der Abfrage zu den Favoriten hinzugefügt werden.
Der eigene Lernerfolg kann durch ausführliche Lernstatistiken überprüft werden.

Die Einbindung von ergänzenden Fragebögen und deren Zuordnung zu Karteikarten oder Kartenstapeln ist in *Repetico* derzeit nicht möglich.

3 Anforderungsanalyse

3.1 Technische Umgebung

Die App-Komponente von *psyq* wird als Applikation für das *Android*-Framework entwickelt. Die Umsetzung von mobilen Apps birgt diverse Stolpersteine, welche nicht immer sofort ersichtlich sind. Daher bedarf es einer angemessenen Planung, um die voraussichtliche Komplexität abzuschätzen und potentielle Hürden idealerweise vermeiden, zumindest aber vorhersehen und berücksichtigen, zu können [11], [12], [13], [14], [15], [16], [17].

Obwohl die neuesten Android-Referenzgeräte der *Nexus*-Produktreihe zum Zeitpunkt dieser Dokumentation bereits Android 5.0 ("Lollipop", API-Version 21) mitbringen, ist für *psyq* Android 2.3.3 ("Gingerbread", API-Version 10) als Mindestanforderung vorgesehen. Die so erreichte Abwärtskompatibilität sorgt zwar für erhöhten Aufwand bei Planung und Implementierung, garantiert aber eine Kompatibilität mit 99,6 % aller Android-Geräte (Stand vom 5. Januar 2015, [18]; siehe Abb. 3.a). Zudem bietet Google mit seiner *Support-Library* abwärtskompatible Versionen von einigen wichtigen und modernen Android-Framework-APIs für ältere API-Levels an. *psyq* macht von dieser hilfreichen Möglichkeit Gebrauch.

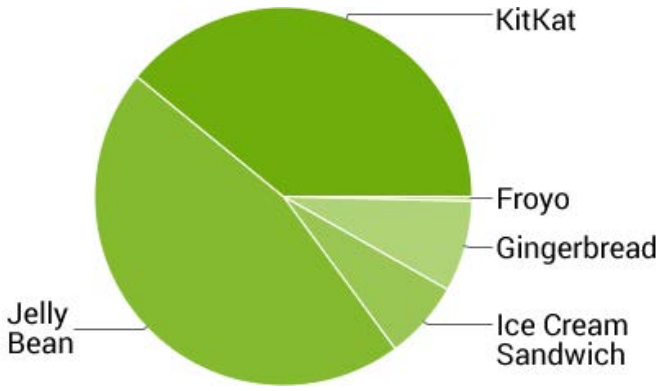

Abb. 3.a: Fragmentierung der Android-Versionen (Stand: 5. Januar 2015) [18]

Zur Entwicklung der App dient die *Android Studio*-IDE in Version *0.8.9 (Beta)*, welche auf *IntelliJ IDEA* von *JetBrains* basiert, und bereits sämtliche Android-spezifischen Einstellungen und Eigenschaften beinhaltet. Nach jahrelanger Dominanz von *Eclipse* im Zusammenspiel mit dem Android-SDK sowie dem Android-ADT gilt *Android Studio* als neuer Standard für die Android-Entwicklung. Auch Google selbst empfiehlt die Nutzung dieser IDE als "offizielle Entwicklungsumgebung" [19] und aktualisiert die eigenen Anleitungen und Trainings für Entwickler nach und nach entsprechend. Im Dezember 2014 wurde nach zwei Jahren der Betaphase Version 1.0 der IDE veröffentlicht.
Ergänzend wird das in *Android Studio* integrierte Build-Management-Automatisierungs-Tool *gradle* verwendet, welches den Build-Prozess erheblich vereinfacht.
Auf Serverseite kommen PHP und MySQL zum Einsatz.

Weiterführende Erklärungen zu Architektur und Implementierung der App sowie Details zum Aufbau und zur Umsetzung der Server-Komponente können aus Kap. 5 und Kap. 6 entnommen werden.

Zur Kontrolle der Entwicklungsschritte ist ein mobiles Gerät in vielen Punkten deutlich komfortabler als die Nutzung des Emulators. Zu diesem Zweck wird hauptsächlich ein Smartphone des Typs *Motorola RAZR HD* mit Android-4.1.2 *Stock-ROM* verwendet.
Für Tests der App unter realitätsnahen Bedingungen kommen mehrere verschiedene mobile Endgeräte zum Einsatz. Es wird darauf geachtet, sowohl die älteste unterstützte als auch die momentan neueste Android-Version als Betriebssystem einzubeziehen, um möglichst unterschiedliche Voraussetzungen abbilden und berücksichtigen zu können. Des Weiteren

werden auch bewusst Geräte mit verschiedenen Auflösungen (480 × 800 Pixel bis 2048 × 1536 Pixel) und Displaygrößen (3,5 " bis 8,9 ") für die Tests gewählt.

psyq wird auf den folgenden mobilen Geräten getestet:

- *Motorola RAZR HD* mit Android 4.1.2 Stock-ROM
- *HTC One (M8)* mit Android 4.4.4 Stock-ROM
- *Motorola Moto G (2013)* mit Android 4.4.4 Stock-ROM
- *ASUS MeMO Pad HD 7* mit Android 4.2.2 Stock-ROM
- *HTC Nexus 9* mit Android 5.0.1
- *Orange San Francisco (ZTE Blade)* mit *Custom-ROM* "*CyanogenMod-7.1.0-Blade*" (basierend auf Android 2.3.7)

Zur Analyse sämtlicher Logs kommt die in *Android Studio* integrierte *LogCat*-Ansicht zum Einsatz. Sie bietet dem Entwickler einen Überblick über alle Logs sämtlicher auf dem Gerät laufenden Prozesse, kann diese aber auch gefiltert darstellen. Auf diese Weise können die Inhalte der Logs während der Laufzeit überprüft werden, z. B. die beim Update aufgerufene URL, Statusmeldungen des Systems oder etwaige Fehlermeldungen. Abb. 3.b zeigt beispielhaft einen Auszug dieser Ansicht.

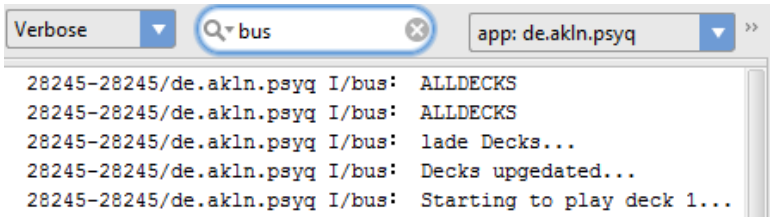

Abb. 3.b: Ausschnitt der *LogCat*-Ansicht, gefiltert nach dem *Tag* "bus"

Auch die Server-Komponente erhält während der Tests Daten wie z. B. die Ergebnisse fertig gespielter *Decks* und pflegt diese in der Datenbank ein. Viele dieser Änderungen sind im Statistikbereich des Backends sichtbar. Im kompletten Umfang sind die veränderten oder neu hinzugekommenen Datensätze in der Datenbank direkt erkennbar. Entsprechend werden die Veränderungen innerhalb der Server-Datenbank während der Testläufe mittels *phpMyAdmin* kontrolliert.

3.2 Anforderungen

Dieses Kapitel behandelt die erarbeiteten funktionalen und nichtfunktionalen Anforderungen, die an *psyq* gestellt werden. Diese sind in Vorgesprächen entstanden und betreffen sowohl die Funktionalität als auch die Kompatibilität und die Benutzerfreundlichkeit des Gesamtsystems.
Kap. 8 betrachtet die Umsetzung der hier gesteckten Anforderungen sowie mögliche Herausforderungen bei deren Realisierung.

psyq muss Autoren den Rahmen bieten, *Decks* und *Tasks* anzulegen. Die Wortlaute von Fragetexten und Antwortmöglichkeiten sollen vollständig frei formulierbar sein. Ist ein Deck vollständig angelegt, kann es veröffentlicht und damit spielbar gemacht werden. Nach diesem Schritt ist das Hinzufügen von weiteren Tasks zu diesem Deck nicht mehr möglich. Dies stellt sicher, dass ausschließlich solche Ergebnisse verglichen werden, welche auf identischer Grundlage zustande kamen. Diese Voraussetzung hilft somit, die Korrektheit der Auswertungen zu gewährleisten.
Weiterhin sollen *Questionnaires* und *Psychs* erstellt und auf identische Weise verwaltet werden können.
Die *Statistik*-Ansicht soll Autoren die Messwerte und Ergebnisse der Spieler grafisch ansprechend aufbereitet präsentieren.
Das Backend soll plattformunabhängig in jedem gewöhnlichen Browser funktionieren und keinerlei zusätzliche PlugIns o. Ä. voraussetzen. Zudem soll die Pflege der Daten und die Kontrolle der Resultate im Backend auf verschiedenen Bildschirmgrößen möglich sein – auch für jene Autoren, die weder die genaue technische Umsetzung von *psyq* noch die generelle Funktionsweise einer Datenbank kennen. Somit genügt für jeden potentiellen Autor eine kurze Einführung, und die Position des "Technikers" zur Pflege der Datenbank kann entfallen.

Die Daten auf Serverseite müssen persistent gespeichert werden, um Inkonsistenzen zu vermeiden.

Als Schnittstelle zwischen App und Server soll eine API einen standardisierten und schnellen Datenaustausch garantieren. Auf diese Weise sind weitere Realisierungen auf anderen Betriebssystemen ohne komplexe API-Anpassungen denkbar. Zudem soll der Datentransfer mit wenig *Overhead* erfolgen.

Für die Android-App wird eine möglichst hohe Kompatibilität angestrebt. So sollen alte Versionen des mobilen Betriebssystems ebenso unterstützt werden wie verschiedene Bildschirmgrößen und -auflösungen. Zudem soll *psyq* die Android-Empfehlungen hinsichtlich Struktur, Layout und Design angemessen umsetzen, um bereits erlernte Interaktionskonzepte der Anwender zu nutzen und den App-übergreifenden Gesamteindruck zu unterstützen. Die App soll somit ohne hohen Lernaufwand verwendet werden können.
Updates von Quiz- und Fragebogen-Daten sollen via Internet mühelos möglich und manuell startbar sein. Um die eigenen Daten absolvierter Decks zum Server zu senden, bedarf es keiner expliziten Aktion seitens des Spielers.
Decks sollen intuitiv spielbar sein, der Spieler soll über seinen Spielfortschritt innerhalb des Decks informiert werden und Zwischenstände sowie Resultate sollen spielerspezifisch gesichert werden.
Die Antwortmöglichkeiten sollen bei jedem Start eines Decks erneut zufällig gemischt werden, um die Erinnerung an die korrekte Lösung – statt nur an deren Position innerhalb der Liste – zu fördern.
Decks sollen als Favoritendeck markierbar und als solche erkennbar sein – ebenso wie laufende Spiele als solche kenntlich gemacht werden. Diese Zuordnungen sollen nach einem Neustart der App oder des Geräts ebenso erhalten bleiben wie die gespeicherten Statistiken der bereits gespielten Decks.
Mithilfe einer konsequenten Userverwaltung können sich verschiedene Spieler am selben Gerät anmelden, eigene Ergebnisse erspielen, individuelle Favoritendecks anlegen und die persönlichen Statistiken betrachten. Somit steht der gemeinsamen Nutzung eines einzelnen Geräts, z. B. im Rahmen einer beaufsichtigten Evaluation, nichts im Wege.
Sowohl die vom Server bereitgestellten Quiz- und Fragebogen-Daten als auch die individuellen Ergebnisse aller am Gerät spielenden Benutzer sollen lokal auf dem Gerät vorgehalten werden.

Die allgemeinen Anforderungen umfassen neben guter Performance und einer möglichst hohen Kompatibilität auch eine intuitive Benutzeroberfläche. Der Einsatz bekannter Interaktionsparadigmen sowie aussagekräftige Bezeichnungen und hilfreiche Hinweise leisten ihren Beitrag hierzu.
Weiterhin ist die Planung sämtlicher Aspekte im Hinblick auf ihre Zukunftssicherheit ausgelegt. Bei der Strukturierung der Daten wurde z. B. bereits darauf geachtet, dass zukünftige weitere *Game-Modes* wie *Multiple-Choice* möglichst nahtlos in die bestehende Architektur eingefügt werden können. Die dafür nötige generelle Flexibilität des Projekts bietet nicht nur bessere Erweiterungsmöglichkeiten für etwaige zukünftige Anpassungen,

sondern sorgt durch variable Attribute der vorhandenen Eigenschaften bereits aktuell für einen erheblichen Mehrwert.
Tabelle 3.a und Tabelle 3.b fassen die funktionalen und nichtfunktionalen Anforderungen an *psyq* zusammen.

Nr.	Anforderung	Beschreibung	Zuord-nung	Art	
				funktional	nicht-fktl.
1	Anlegen von Spielern, Decks, Tasks, Question-naires und Psychs	Autoren können die Daten im Backend erstellen und einander zuordnen.	Backend	✓	
2	Bilder-Upload für Decks, Tasks und Spieler	Spieler erkennen Decks leichter wieder. Ein Avatar wirkt persönlich ansprechend.	Backend	✓	
3	Freier Text für Fragen, Antworten, Psych-Texten und Min.-/Max.-Labels	Die Formulierungen der Quizdaten, der Psych-Texte und der SeekBar-Beschriftungen können vom Autor frei verfasst werden.	Backend & App	✓	
4	Globale Statistikanzeige	Im Backend werden die Daten aller Spieler in Diagrammen visualisiert.	Backend	✓	
5	Plattformunabhängiges Backend	Der Autor kann die Daten mit jedem Browser pflegen und einsehen.	Backend	✓	
6	Einfache Handhabung des Backends	Auch technisch weniger versierte Autoren können das Backend intuitiv bedienen.	Backend		✓
7	Vielfältige Einstellmög-lichkeiten für Autoren	Decks und Questionnaires besitzen diverse modifizierbare Parameter.	Backend & App		✓
8	Flexible Questionnaire-Einbindung	Questionnaires können lose an beliebige Tasks oder Decks gekoppelt werden.	B'end, DB, API & App	✓	
9	Unterstützung bei der Erstellung	Eindeutige Bezeichnungen, Hinweise und Beispiele helfen bei der Dateneingabe.	Backend		✓
10	Feingranulare Datenbankstruktur	Der Aufbau der Datenbank eignet sich gut für etwaige Erweiterungen.	DB & Backend		✓
11	Datenabruf über API mit Wiederverwendbarkeit	Die Schnittstelle liefert Daten in einem einheitlichen Format und ermöglicht so weitere Clients.	API	✓	
12	Update vom Server	Die App erhält per Update die aktuellen Daten.	API & App	✓	
13	Login / Logout	Die Ergebnisse lassen sich eindeutig den Spielern zuordnen.	App & API	✓	
14	Starten / Fortsetzen von Decks	Nach einer Unterbrechung beginnt das Spiel an der zuletzt aktiven Position.	App	✓	
15	Fortschrittsanzeige	Spieler erhalten eine visuelle Rückmeldung über ihren Fortschritt im gespielten Deck.	App		✓
16	Antwortmöglichkeiten mischen	Die Antwortmöglichkeiten eines Tasks werden bei jedem Aufruf erneut zufällig angeordnet, um den Lerneffekt zu steigern	App	✓	

Tabelle 3.a: Anforderungen - Teil 1

Nr.	Anforderung	Beschreibung	Zuord-nung	Art	
				funktional	nicht-fktl.
17	Rückmeldung zu korrekten und falschen Antworten	Die Spieler erfahren, ob die abgegebene Antwort korrekt war. Nach einem Fehlversuch wird die korrekte Lösung hervorgehoben.	App	✓	
18	Decks favorisieren	Zur Erleichterung der Auswahl kann jedes Deck als "FavoriteDeck" markiert werden.	App	✓	
19	Anzeige begonnener Spiele	Bereits laufende Spiele sind als solche zu erkennen.	App		✓
20	Spiel-Ende erzwingen	Begonnene Spiele müssen zu Ende gespielt werden, bevor sie erneut gestartet werden können. Dies stellt einheitliche Vergleichsdaten sicher.	App	✓	
21	Senden der Ergebnisse an den Server	Resultate der Quizze und der Fragebögen werden automatisch an den Server übermittelt.	App & API	✓	
22	Eigene Statistik in App	Spieler erhalten Innerhalb der App Einblick in die eigene Statistik.	App	✓	
23	Kompatibilität	Die App ist kompatibel mit unterschiedlichsten Android-Versionen und Gerätevoraussetzungen.	App		✓
24	Intuitive Bedienung der App und Einhaltung von Design-Richtlinien	Nutzer der App können bekannte Interaktionsmuster anwenden und eine einheitliche Designsprache wahrnehmen. Das fördert die leichte Erlernbarkeit der App-Bedienung.	App		✓
25	Speichern von Nutzer- und spielspezifischen Daten	Login-Informationen, Spielfortschritte und ergebnisse, Favorite-Decks und Statistiken sind auch nach dem Neustart der App oder des Geräts vorhanden und ersichtlich.	App	✓	
26	Nach Spielern getrennte Datenhaltung	Pro Gerät können mehrere Anwender die App unabhängig voneinander nutzen.	App	✓	
27	Flexible Architektur der App	Strukturierung und Kommunikation der App sind logisch und modular gehalten. Das erleichtert die Wartung, Anpassbarkeit und Erweiterbarkeit.	App		✓
28	Konsistenz bei Begriffen und Verhalten	App und Backend reagieren in sich stim-mig und konsistent. Begriffe - insbeson-dere projektspezifische - werden konsequent einheitlich verwendet.	App & Backend		✓

Tabelle 3.b: Anforderungen - Teil 2

3.3 Anwendungsfälle

Basierend auf den in Kap. 3.2 analysierten Anforderungen lassen sich die wichtigsten Anwendungsfälle wie in Abb. 3.c dargestellt veranschaulichen. Hauptakteure sind neben dem Spieler und dem Autor die beiden beteiligten technischen Komponenten von *psyq*, *App* und *Server*.

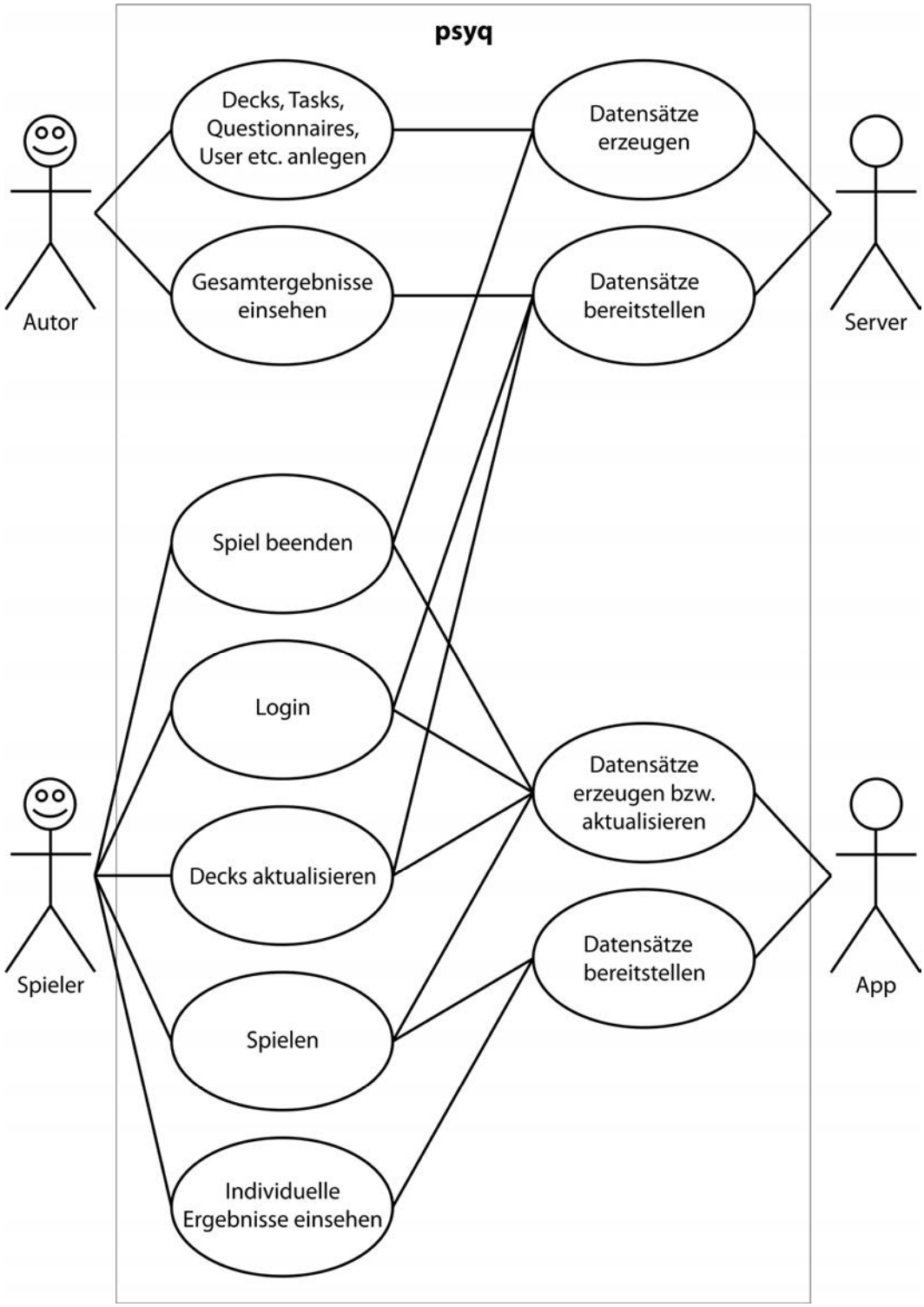

Abb. 3.c *psyq*-Anwendungsfalldiagram

4 Entwurf und Konzept

4.1 Allgemeines Konzept

Ziel ist die Erstellung einer mobil einsetzbaren Quiz-App auf der Android-Plattform zur spielerischen Wissensvermittlung. Der zugehörige Themenbereich *Serious Games* stellt sich in vielerlei Hinsicht als komplex und facettenreich dar. Stetige Neuentwicklungen haben einen anhaltenden Wandel der möglichen Ansätze zur Folge [20], [21], [22], [23], [24].
Neben der Realisierung der App wird zudem ein komfortables Backend angestrebt, welches zwei Hauptfunktionen bietet: Zum einen das Anlegen und Verwalten von Fragen, Antworten und Spielern sowie zum anderen die übersichtliche Darstellung dieser Daten und der Ergebnisse.
Darüber hinaus sollen psychologische Fragebögen erstellt werden können, die mit den Quiz-Fragen bzw. Decks verknüpfbar sind. Auf diese Weise können entsprechende psychologische Fragen explizit eingeschoben werden, z. B. nach besonders schwierigen oder irritierenden Quiz-Fragen, um die Glaubwürdigkeit und die Qualität der Antworten besser einschätzen zu können. Auch die psychologischen Fragebögen sowie deren Ergebnisse sollen im Backend verwaltet werden können.

4.2 Begrifflichkeiten

Während das Glossar (S. 100) eine Erklärung zu wichtigen generellen Fachbegriffen bereithält, verwendet die vorliegende Dokumentation einige Terme, die exklusiv für *psyq* eingeführt wurden oder in dessen Kontext eine spezielle Bedeutung besitzen. Dieses Kapitel erläutert eben jene Begriffe, da es sich dabei um essentielle Akteure und Komponenten im Aufbau von *psyq* handelt.

Die Zusammenhänge der nachfolgend erklärten Komponenten werden in Kap. 5.2 ausführlich behandelt.

4.2.1 Akteure

Akteure sind Personen, die mit *psyq* interagieren. Je nachdem, mit welcher Systemkomponente sie agieren, werden sie wie folgt unterschieden.

4.2.1.1 Autor

Ein *Autor* erstellt Quiz-Fragen sowie psychologische Fragebögen und ordnet die Aufgaben den *Decks* zu. Er besitzt außerdem den Überblick über die existierenden *Spieler*, über die Ergebnisse bzw. Statistiken der gespielten *Decks* und über die beantworteten *Questionnaires*. Für alle diese Aktionen nutzt er das Backend.

4.2.1.2 Spieler

Ein *Spieler* nutzt die App als Anwender auf seinem mobilen Android-Endgerät, spielt also typischerweise ein oder mehrere *Decks* und beantwortet die ggf. darin integrierten *Questionnaires*. Um spielen zu können, benötigt der Spieler einen *Account*.
Die Angaben, die im *Account* des Spielers gespeichert werden reichen von seiner E-Mail-Adresse, die gleichzeitig als Login-Name sowie zur Kontaktaufnahme dienen kann, über Name und Geschlecht bis hin zum Geburtsjahr und einem Kommentarfeld. Zudem ist für den Login ein Passwort nötig, dessen *Hash* ebenfalls im jeweiligen Account gespeichert wird.

4.2.2 Technische Teilnehmer

psyq setzt den bewährten Server-Client-Aufbau um. Die daran beteiligten Hauptkomponenten, zusammenfassend als „technische Teilnehmer" bezeichnet, sind die folgenden.

4.2.2.1 Server-Komponente

Die Serverseite von *psyq* dient der Eingabe und Pflege neuer Daten sowie der Präsentation der Ergebnisse. Zur Datenhaltung dient eine zentrale Datenbank. Die Eingaben des *Autors* erfolgen über das Backend, einer einfach zu bedienenden Ansicht mit Eingabemasken etc. zur bequemen Datenpflege und -abfrage.

4.2.2.2 API (Application Programming Interface)

Die Programmierschnittstelle stellt die Kommunikation zwischen *Server-Komponente* und *Client-Komponente (App)* sicher und liefert Daten in beide Richtungen. Bei der Realisierung der *API* muss auf die entsprechend korrekten Reaktionen auf Anfragen sowie auf möglichst geringen Datenverkehr geachtet werden.

4.2.2.3 Client-Komponente (App)

Die *App* (oder auch *App-Komponente*) von *psyq* ist der fundamentale Bestandteil des Systems. Sie ermöglicht das Ziel von *psyq*, nämlich die Beantwortung der Fragen durch den *Spieler*. Zudem ist die *App* für dessen Zufriedenheit mit dem System zuständig, da sie die Kontaktstelle zwischen Mensch und Software bildet.

4.2.3 Themenbereiche

psyq umfasst zwei große Themenbereiche, die bewusst nur lose gekoppelt wurden, um eine möglichst hohe Variabilität zu gewährleisten.

4.2.3.1 Quiz

Der Themenbereich *Quiz* beinhaltet sämtliche Quizfragen und damit die *Decks*, *Tasks*, *Questions* und *Answers*. Außerdem gehören ihm die dazugehörigen Resultate, also *PlayedDecks*, *PlayedTasks* und *GivenAnswers* an.

4.2.3.2 Fragebögen

Der Themenbereich *Fragebögen* umfasst sämtliche *Questionnaires* und *Psychs* sowie die ausgefüllten *PlayedQuestionnaires* und *PlayedPsychs*.

4.2.4 Grund-Elemente der Quizze und Fragebögen

Quizze und Fragebögen können von *Autoren* im Backend angelegt und befüllt werden. Anschließend werden diese Elemente nicht weiter verändert, sie werden von der API im Zuge eines Updates an die App weitergegeben und bilden die Grundlage für Spielresultate.

4.2.4.1 Deck

Ein *Deck* repräsentiert einen definierten Katalog verschiedener Aufgaben (*Tasks*). Idealerweise werden zusammengehörige *Tasks* zu einem *Deck* gebündelt. Nach welchen Merkmalen die Bündelung erfolgt bleibt vollständig dem *Autor* überlassen. Titel und Beschreibung eines *Decks* sowie das optionale Icon dienen der schnellen Erfassung seines Inhalts durch den Nutzer.
Jedes Deck kann sich in einem von mehreren Stati befinden. Der Status kennzeichnet den Zustand im Lebenszyklus des betreffenden *Decks*.
Weiterhin kann ein *Questionnaire* mit dem Deck mittels `start_questionnaire` verknüpft sein.

4.2.4.2 Task

Ein *Task* beschreibt eine zu lösende Aufgabe. Er ist an ein *Deck* gekoppelt, bündelt sowohl eine Frage (*Question*) als deren auch Antwortmöglichkeiten (*Answers*) und verfügt über die Möglichkeit eines nachgeschalteten *Questionnaires*.

4.2.4.3 Question

Die textuelle Ausformulierung der Frage sowie ein optionales Bild zur Verdeutlichung werden in einer *Question* zusammengefasst. Sie ist einem *Task* untergeordnet.

4.2.4.4 Answer

Jede *Answer* gehört ebenfalls zu jeweils genau einem *Task*. Sie beinhaltet den Antworttext und ein elementares *Flag*, das angibt, ob sie im gegebenen Kontext, also für den verknüpften *Task*, korrekt ist.

4.2.4.5 Questionnaire

Ein *Questionnaire* bündelt beliebig viele *Psychs.* Er stellt den Grundbaustein für die eingestreuten psychologischen Fragebögen dar. Der *Questionnaire* selbst hat lediglich einen Titel, ein Bild und einen Status vorzuweisen – die konkreten Inhalte sind in den beinhalteten *Psychs* enthalten.

4.2.4.6 Psych

Ein *Psych* ist eine psychologische Abfrage innerhalb eines *Questionnaires*. Sie beinhaltet einen Text zur Erklärung bzw. Fragestellung, Minimal- und Maximalwerte sowie deren Beschriftungen und ist von einem bestimmten *Psychtype*.
Der *Psychtype* gibt an, um welche Art der Abfrage es sich beim jeweiligen *Psych* handelt. Aktuell werden folgende *Psychtypes* unterstützt: *Boolean*, *range*, *free text*.

4.2.5 Played-Elemente (gespielte Quizze und ausgefüllte Fragebögen)

Die in Kap. 4.2.4 vorgestellten Grund-Elemente, welche vom Backend über die Datenbank via API in der App landen, dienen der Anzeige von Aufgaben und von psychologischen Fragebögen. Das Interesse der *Autoren* gilt den daraus produzierten Ergebnissen.
Diese sog. Played-Elemente werden von der App über die API an den Server geschickt.

4.2.5.1 PlayedDeck

Ein *PlayedDeck* steht für ein gespieltes *Deck*. Es ist mit diesem verknüpft und beinhaltet zusätzliche Angaben wie den spielenden *Spieler* und den Timestamp der Fertigstellung.

4.2.5.2 PlayedTask

Ein *PlayedTask* bezeichnet einen abgeschlossenen *Task*. Auch er beinhaltet einen Timestamp sowie die Verknüpfung zum Ursprungs-*Task* und zum übergeordneten *PlayedDeck*.

4.2.5.3 GivenAnswer

Eine *GivenAnswer* repräsentiert eine abgegebene Antwort. Sie verweist auf die zugehörige *Answer* ebenso wie auf den *PlayedTask*, dem sie zugeordnet ist.

4.2.5.4 PlayedQuestionnaire

Ein *PlayedQuestionnaire* ist ein ausgefüllter und abgeschickter *Questionnaire*. Er enthält Angaben zum *Spieler* sowie zu einem Timestamp und beinhaltet *PlayedPsychs*.

4.2.5.5 PlayedPsych

Ein *PlayedPych* stellt ein ausgefülltes *Psych* dar; es besitzt eine Verknüpfung zum zugehörigen *PlayedQuestionnaire*, zum entsprechenden *Psych* und beinhaltet zudem die gegebene Antwort als Wert (z. B. numerischer Wert oder Text).

5 Architektur

Die Applikation auf dem mobilen Endgerät einerseits und die Datenbank sowie das Backend auf der Serverseite andererseits bilden die beiden Hauptbestandteile von *psyq*.
Eine standardisierte Kommunikation zur fehlerfreien Kooperation dieser beiden maßgeblichen Akteure stellt ebenso ein wesentliches Ziel der Konzeption von *psyq* dar wie eine konsistente Datenhaltung auf beiden Seiten.

Das folgende Kapitel diskutiert die dafür notwendige Planung und die erarbeitete Modulstrukturierung. Nachdem zunächst die grundlegende Architektur hinter *psyq* erläutert wird, gehen Kap. 5.2 und folgende detaillierter auf essentielle Unterthemen ein. Auftretende Hindernisse werden analysiert und die zur Behebung nötigen Lösungswege werden aufgezeigt und diskutiert.

5.1 Basis-Architektur

Die Basis-Architektur von *psyq* (Abb. 3.a) zeigt zunächst eine Trennung in zwei wesentliche Bestandteile: Die mobile App-Komponente in der Rolle des Client einerseits und die serverseitige Implementierung, welche das Backend sowie die Clients bedient, andererseits. Die dafür nötige Kommunikation nutzt eigens entwickelte Schnittstellen (Details zur verwendeten API siehe Kap. 6.2), um die

jeweiligen Daten in beide Richtungen bereitzustellen. Die Übermittlung erfolgt im standardisierten JSON-Format. Kap. 5.3.1 geht genauer auf den Datenfluss ein.

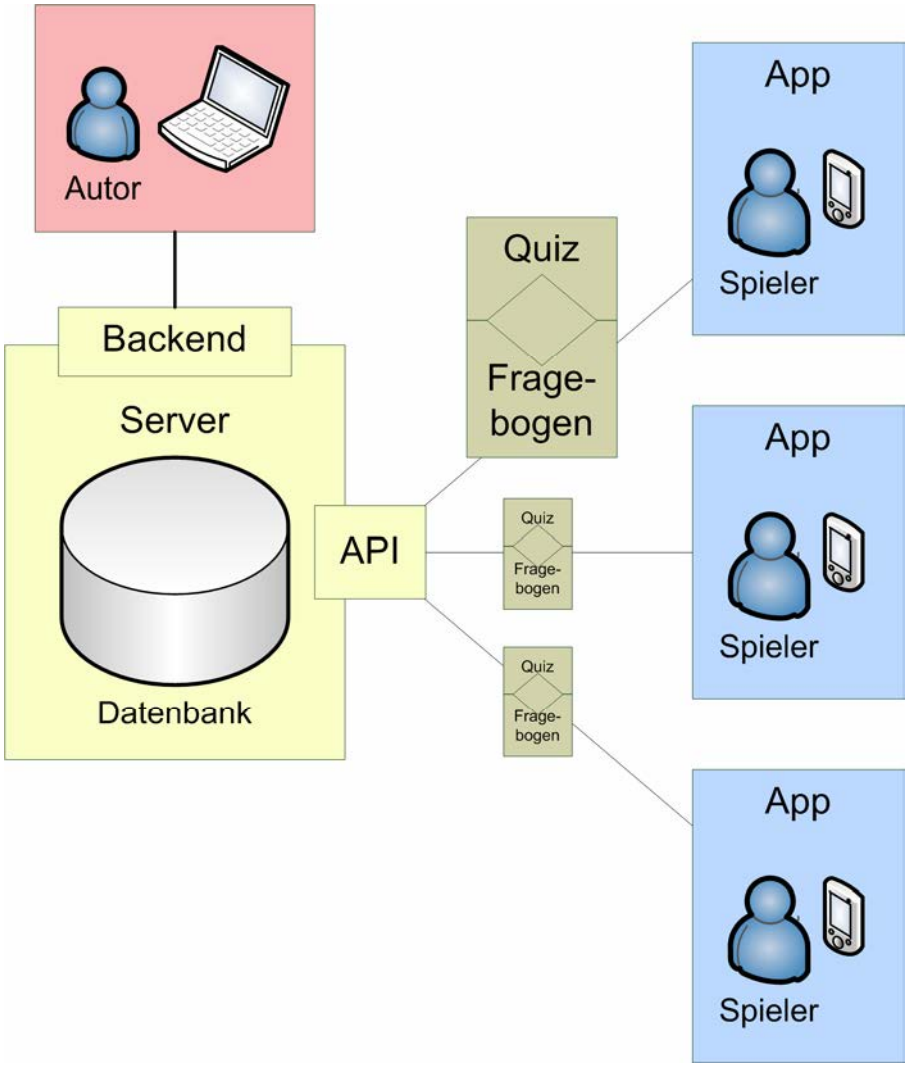

Abb. 5.a: Vereinfachte Basis-Architektur von psyq

5.2 Struktur der Daten

Eine zentrale Rolle in der Planung und Implementierung von *psyq* spielt die Datenhaltung und deren Strukturierung. Eine strategisch sinnvolle Planung der Datenbankstruktur vereinfacht die Erweiterbarkeit um ein Vielfaches und sorgt dadurch auch für optimale Zukunftssicherheit. Daher erforderte dieser Teil der Arbeit einen hohen zeitlichen Aufwand.

Eine geeignete Hierarchie und Modularisierung bewirken, dass Erweiterungen, z. B. die Erstellung neuer *Game-Modes* (siehe Kap. 9.1.1), mit möglichst geringem Aufwand hinsichtlich Änderungen der Datenbankstruktur hinzugefügt werden können. Auch die Möglichkeiten der Statistikgenerierung werden durch eine feingranulare Aufspaltung zusätzlich verbessert.

Zudem wurden die beiden Bereiche *Quiz* und *Fragebögen* bewusst nur lose gekoppelt. Auf diese Weise lassen sich Questionnaires z. B. mühelos hinter mehrere verschiedene Tasks in verschiedenen Decks schalten.
Auf diese Weise können dieselben Questionnaires aus unterschiedlichen Decks heraus beantwortet werden. Die so entstehenden Resultate beziehen sich auf die identische Fragestellung, sind aber unter verschiedenen Voraussetzungen entstanden und daher besonders interessant.

Die in Kap. 4.2 bereits besprochenen Komponenten stehen in Beziehung zueinander, hängen zum Teil voneinander ab und besitzen verschiedene Eigenschaften, Themengebiete und beteiligte Akteure. Tabellarisch können sie folgendermaßen gegliedert werden (Abb. 5.b).

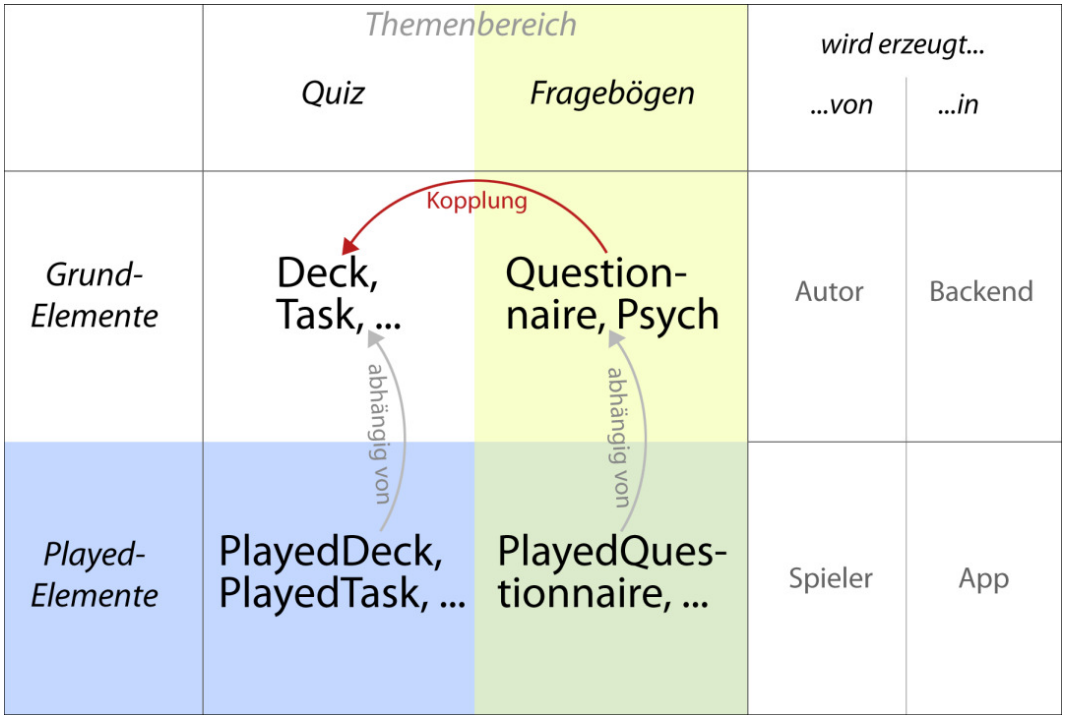

Abb. 5.b Datenstruktur innerhalb von *psyq*

5.2.1 Serverseite und App

Der Aufbau der MySQL-Datenbank weist eine strenge Hierarchie bzgl. der einzelnen Komponenten auf.
Abb. 5.c zeigt die Strukturierung der MySQL-Datenbank sowie die Beziehung zwischen den enthaltenen Tabellen.

Die Struktur der Daten auf Seite der mobilen Applikation gleicht der Datenbankstruktur des Servers. Die verwendete Datenhaltungs-Bibliothek *Realm* (siehe Kap. 5.7.2) kann die dargestellten Relationen der einzelnen Komponenten zueinander – auch ohne den Einsatz von *SQLite* – mithilfe ihrer Objekt-Beziehungen zueinander herstellen.

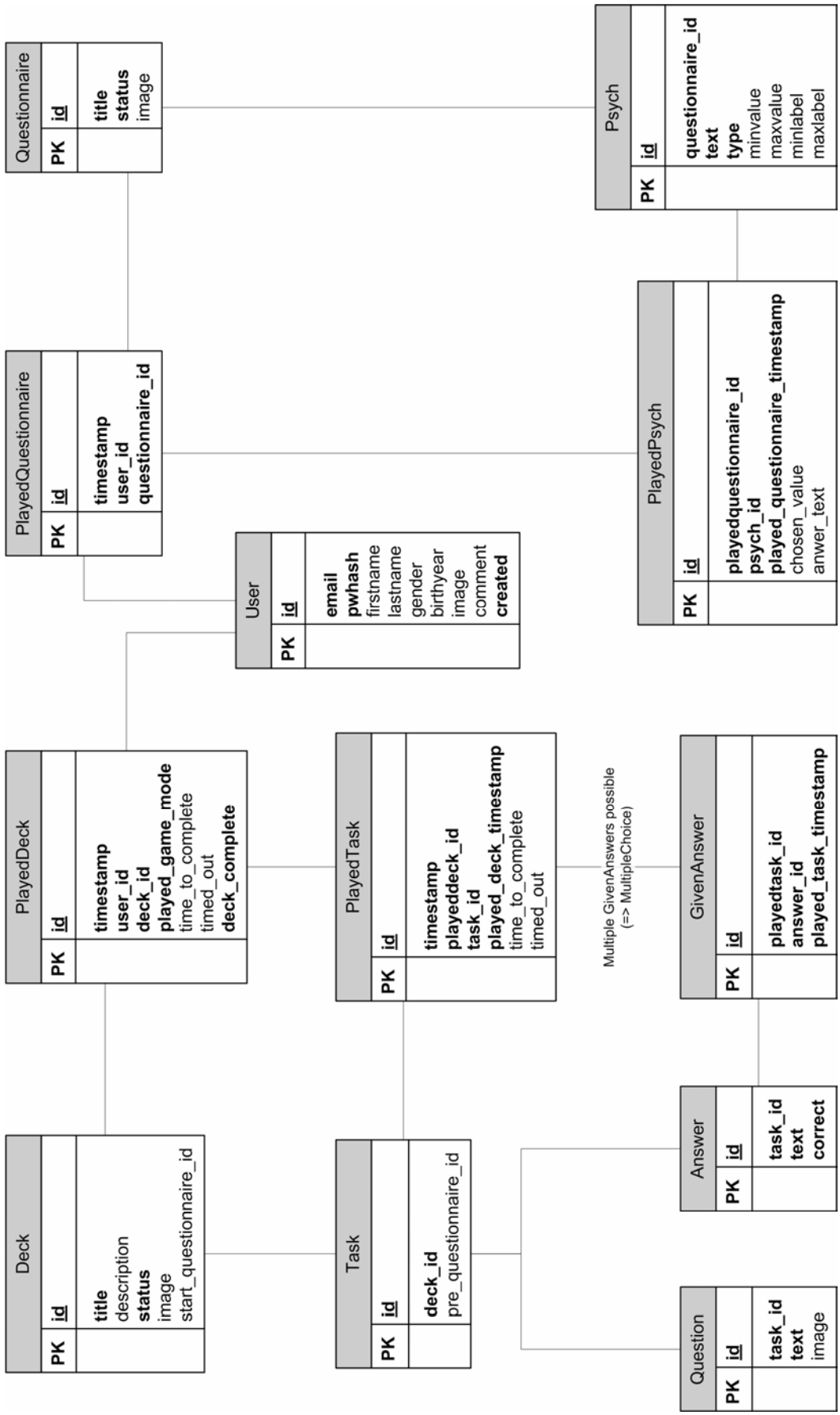

Abb. 5.c Datenbankstruktur von *psyq*

Die in Abb. 5.c ersichtliche mehrschichtige Struktur der Daten bietet zusammen mit der losen Kopplung der Themenbereiche die zuvor erwähnten Vorteile, hat aber in einigen Gebieten auch Nachteile. So führt die Unterteilung oft zu aufwendigeren Datenbankabfragen und Objektoperationen etc., um alle benötigten Daten zusammenzuführen.

5.2.2 Rahmendaten in der App

Die App erzeugt „Rahmendaten“, die dem Spielablauf oder der angenehmeren Nutzung dienen. Diese sind ausschließlich lokal verfügbar und müssen nicht mit dem Server synchronisiert werden. Beispiele hierfür sind die Verwaltung von favorisierten Decks und das Speichern der Login-Daten über den Geräteneustart hinweg. Auch der Fortschritt innerhalb eines Decks zählt zu dieser Art von Rahmendaten und wird benötigt, um ein begonnenes Spiel an der korrekten Position fortsetzen zu können.
Mit Ausnahme der Login-Informationen, welche einmalig global gesichert werden, sind diese Daten an einen Spieler bzw. an dessen Account gebunden. Somit können verschiedene Spieler am selben Endgerät völlig unabhängig agieren und unterschiedliche Spielstände und Favoriten generieren.

Die Rahmendaten werden als einfache *Key-Value-Paare* gespeichert. Kap. 6.1.3 vertieft die Realisierung dieses Schritts.

Abb. 5.d zeigt die Speicherung der spielerbezogenen Rahmendaten sowie des einmalig gespeicherten Logins. Abb. 5.e verdeutlicht, wie die Daten eines weiteren Spielers am selben Gerät individuell und separat gesichert werden.

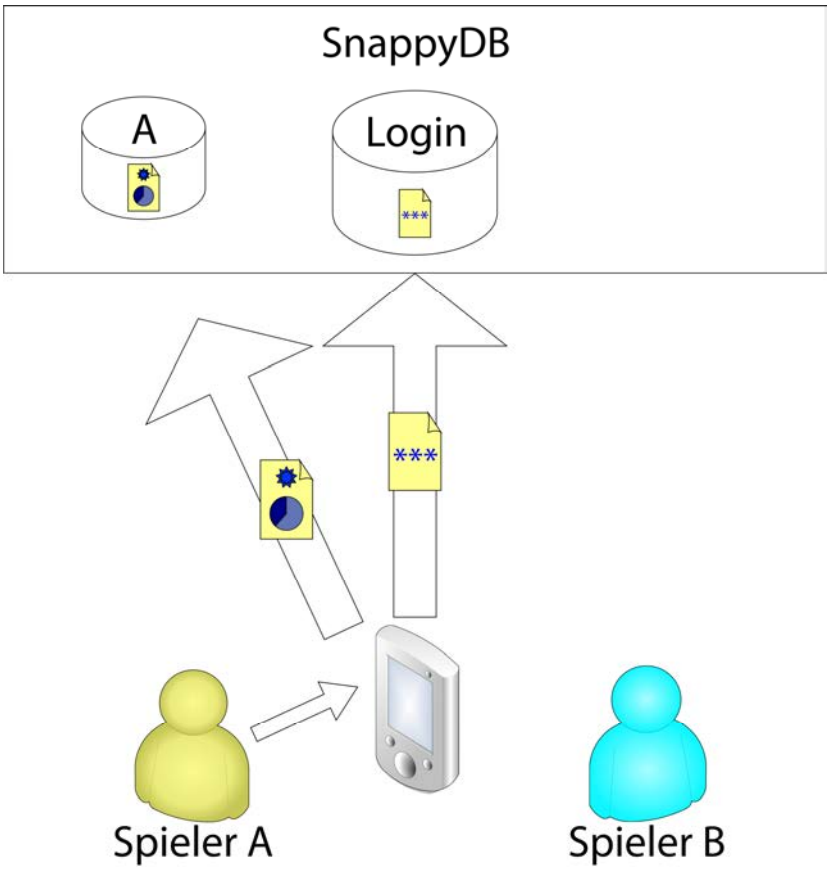

Abb. 5.d: Speicherung von Rahmendaten - Schritt 1

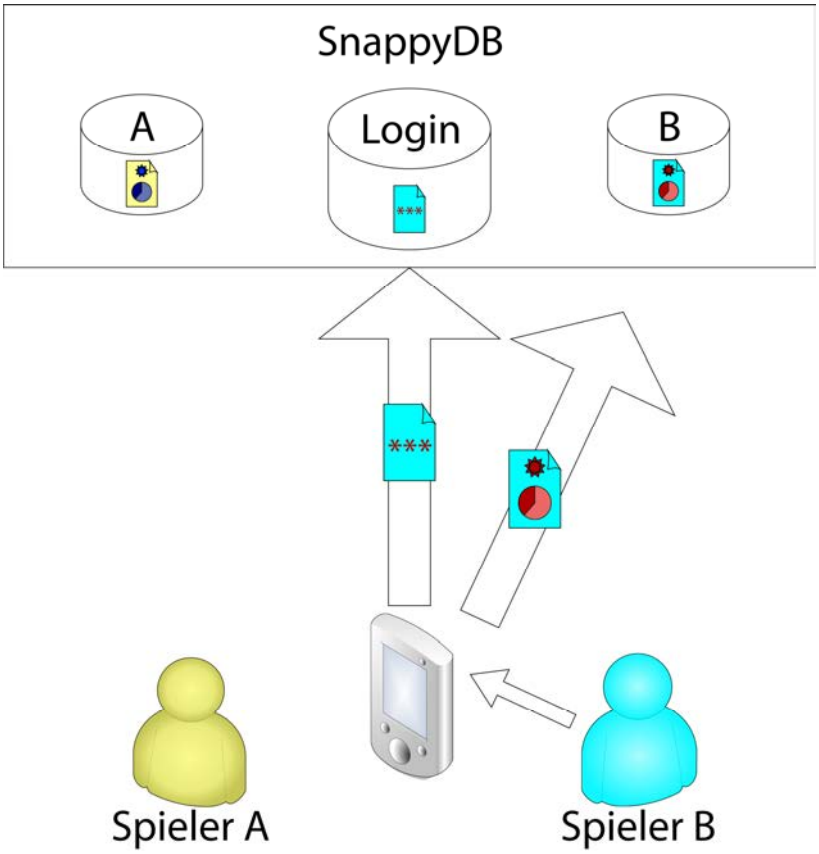

Abb. 5.e: Speicherung von Rahmendaten - Schritt 2

5.3 Informationsfluss-Struktur

Ein zügiger und fehlerfreier Informationsfluss ist wesentlich für die korrekte Funktionalität eines Systems und erhöht zudem maßgeblich den Bedienkomfort, z. B. durch geringere Ladezeiten.
Dieses Kapitel zeigt die verschiedenen Informationsflüsse in *psyq* strukturell auf, während die exakte Implementierung der einzelnen Kommunikationswege in Kap. 6. erläutert wird.

5.3.1 Client-Server-Kommunikation

Die Kommunikation zwischen der Server-Komponente und der App stellt die Basis für ein laufendes Gesamtsystem dar. Nur so können eingepflegte Daten in der App angezeigt und erspielte Resultate im Backend betrachtet werden.
Die API erledigt den gesamten nötigen Informationsaustausch adäquat und effizient.
Abb. 5.f verdeutlicht, dass die API einen geeigneten Kommunikationskanal in beide Richtungen bietet und die Daten im herkömmlichen JSON-Format liefert.

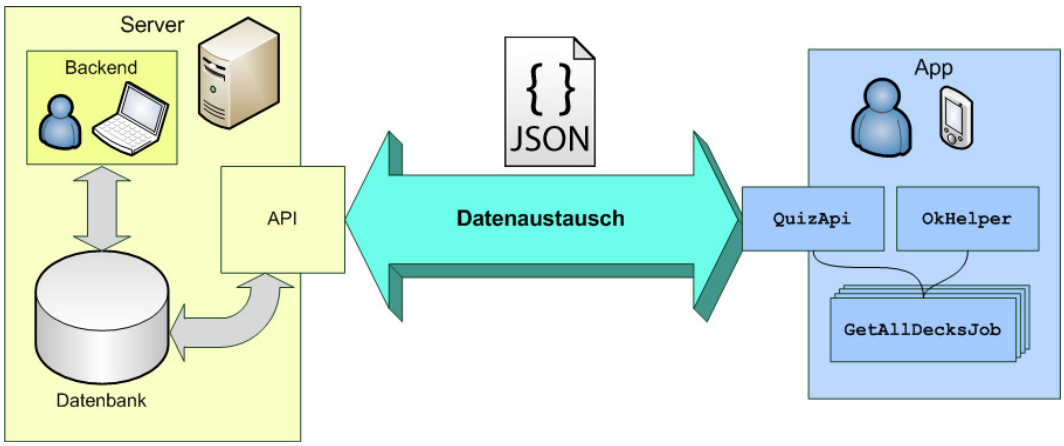

Abb. 5.f Informationsfluss: Client-Server-Kommunikation

5.3.2 Kommunikation innerhalb der App

Von grundlegender Bedeutung hinsichtlich Ablauf, Performance und Aufbau einer App sind der Informationsfluss und die so zustande kommende Kommunikation zwischen den einzelnen Komponenten.
In *psyq* ist die dafür nötige Architektur möglichst strukturiert gehalten und durch feste Regeln sowie zentrale Stellen gesichert.

psyq verwendet eine zentrale *Job*-Verwaltung, die komplexe Aufgaben kapselt und verwaltet. Eventuelle Fehler werden von ihr angemessen behandelt und entsprechende Ergebnisse an die aufrufenden Komponenten zurückgegeben.
Diese Rückmeldungen werden – ebenso wie diverse andere Nachrichtenflüsse innerhalb der App – über *Events* auf dem *EventBus* realisiert (siehe Kap. 6.1.5). Dieser zentrale Kommunikationskanal verbindet sämtliche beteiligten Komponenten des Systems und sorgt für eine reibungslose und nicht redundante Kommunikation.

5.4 App: Paket- und Klassenstruktur

Eine durchdachte Strukturierung der einzelnen Klassen und Packages ist für einen logischen Aufbau der App zwingend erforderlich.
Die so erreichte Modularisierung ermöglicht zum einen eine vereinfachte Wartbarkeit sowie verbesserte Erweiterungsmöglichkeiten. Zum anderen können dadurch Code-Fragmente oder Methoden mit geringem Aufwand wiederverwendet werden.

Abb. 5.g zeigt die Architektur der App-Komponente von *psyq* und listet die enthaltenen Bestandteile anschaulich auf.

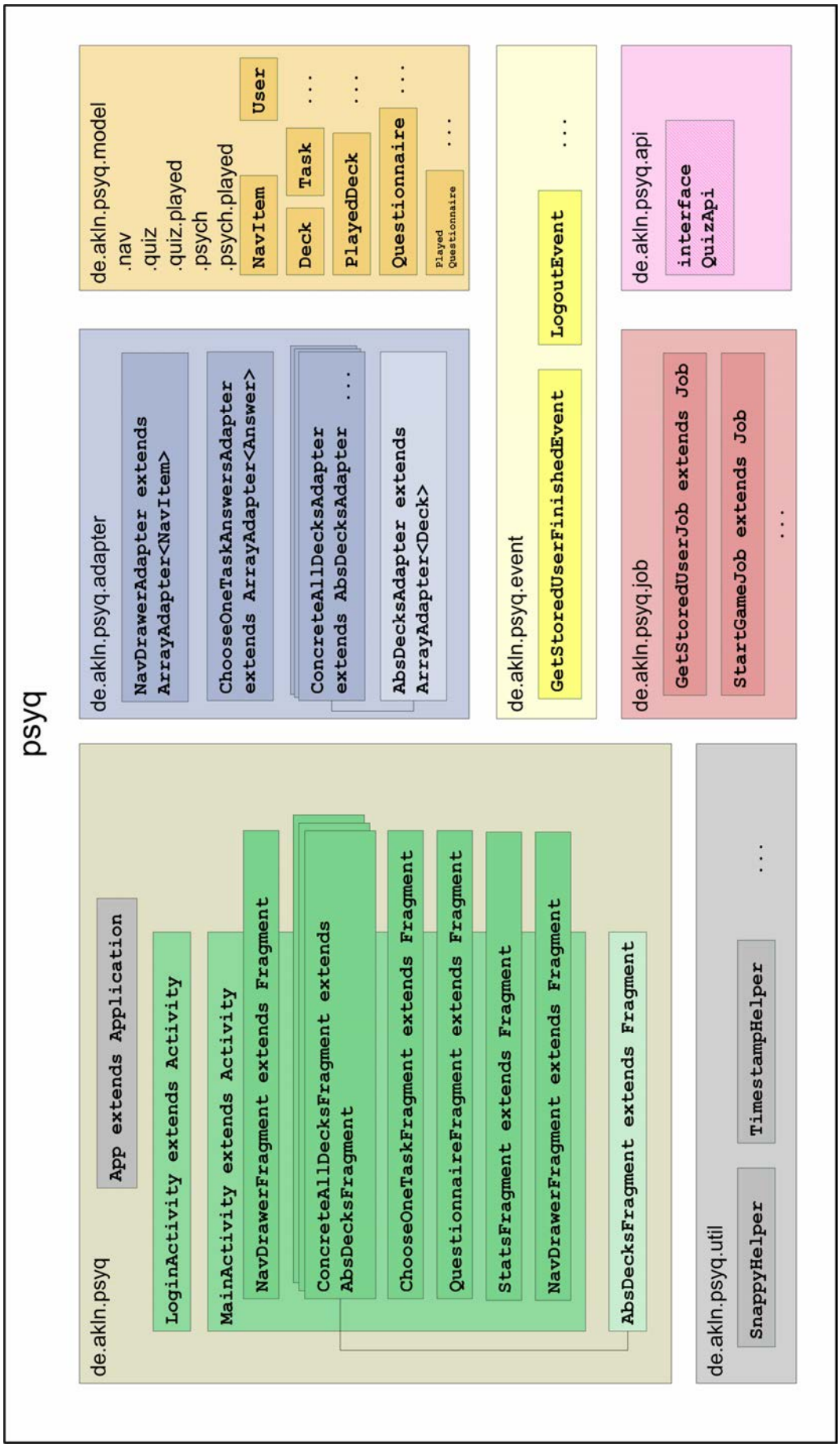

Abb. 5.g Klassen- und Paketstruktur von *psyq*

5.5 App-Dialogstruktur

Um den Weg des Anwenders durch die App optimal zu unterstützen, muss auf eine geeignete Reihenfolge der für ihn relevanten Ansichten geachtet werden. Ein intuitiver Programmablauf sowie erwartbare Reaktionen und visuelle Rückmeldungen des Systems stellen hierbei ein wichtiges Ziel dar.
Dieser Ablauf bedarf deshalb einer durchdachten Planung – insbesondere hinsichtlich der *Zurück-Navigation* und der *Up-Navigation* (falls eingesetzt). Zudem werden bekannte und empfohlene Navigationsmuster angewendet, um den Lernaufwand des Anwenders gering zu halten.

Abb. 5.h veranschaulicht die möglichen Abläufe dieser Ansichten sowie die zugehörigen Interaktionen des Anwenders.

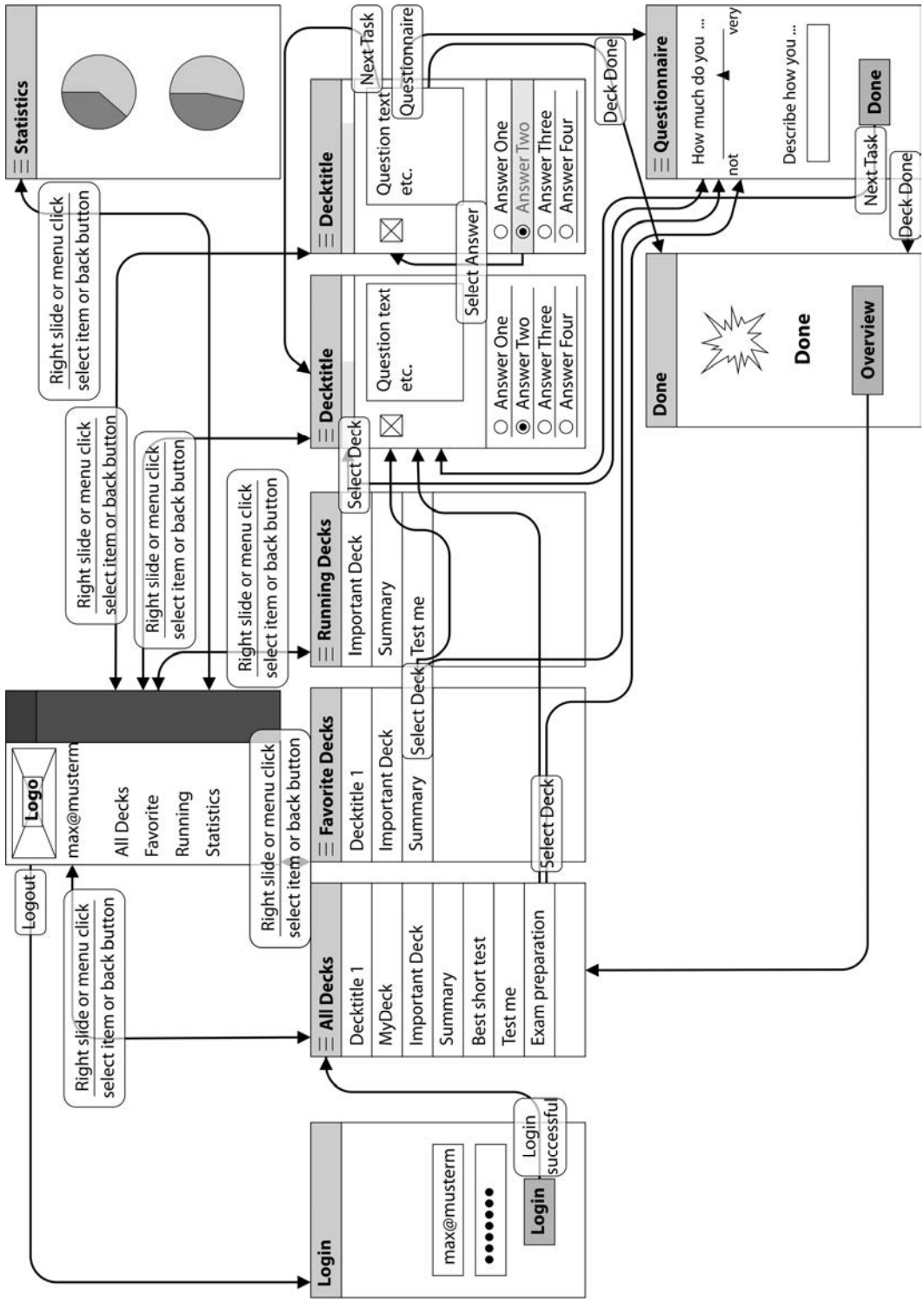

Abb. 5.h Dialogstruktur der App-Komponente

5.6 Bibliotheken (Server)

5.6.1 RedBeanPHP

Serverseitig verwendet *psyq* die PHP-Bibliothek *RedBeanPHP* (Version 4.0) [25], welche dem *Object Relationship Mapping* (ORM) dient. Sie vereinfacht die Erzeugung von Datenbankeinträgen aus PHP-Objekten. Umgekehrt unterstützt sie den Entwickler ebenfalls bei der Abfrage und Anzeige von bereits bestehenden Daten. Eine besonders wertvolle Eigenschaft stellen hier die automatisch erkannten und verarbeiteten Relationen zwischen den Daten dar.
Kap. 6.3.1 geht näher auf die dazugehörigen Implementierungsaspekte ein.

5.6.2 Bootstrap

Zur flexiblen Ansicht des Backends auf verschiedenen Bildschirmgrößen und -auflösungen wurde das Bootstrap-Framework [26] eingesetzt. Das so erreichte *Responsive Design* macht das Backend unter sämtlichen Voraussetzungen bedienbar und wirkt visuell ansprechend.

5.7 Bibliotheken (App)

Die App-Komponente von *psyq* nutzt einige bewährte Programmbibliotheken, welche die benötigten Funktionalitäten anbieten. Sie ermöglichen es, die architektonische Struktur und den resultierenden Programmablauf um ein Vielfaches modularisierbarer zu gestalten.
Dieses Kapitel beleuchtet die wichtigsten Bibliotheken, ihre Einsatzzwecke und die jeweiligen Vorteile näher.

5.7.1 Support-Library

Viele neuartige Design-, Bedien- und Programmierkonzepte, welche Google in nachfolgenden Android-Versionen eingeführt hat, sind auf älteren Android-Versionen nicht vorhanden und daher nicht nutzbar. Die von Google bereitgestellte Support-Library [27] garantiert jedoch eine solche Abwärtskompatibilität zu älteren Android-Versionen, indem sie die

betreffenden Klassen durch eine alternative Implementierung (Nutzung der alten APIs) oder Nachbildungen bereitstellt.
So können verschiedene Neuentwicklungen auch in Kombination mit niedrigeren API-Versionen genutzt werden. Dadurch war es für *psyq* nicht erforderlich, sich für eine der folgenden, jeweils nachteiligeren Varianten zu entscheiden:.

- Abwärtskompatibilität, aber veraltetes Design und keine Nutzung neuartiger Konzepte wie z. B. *Fragments*
- Verwendung von Neuerungen, aber keine Lauffähigkeit auf alten Android-Versionen und -Geräten

Die Verwendung der Support-Library ist daher dringend zu empfehlen.
psyq verwendet die folgenden offiziellen Support-Libraries: `com.android.support:support-v4:21.0.0`, `com.android.support:appcompat-v7:21.0.0`.

5.7.2 Realm

Realm ist eine junge Programmbibliothek für iOS und Android, die sich der Datenhaltung verschrieben hat. Ihr Anspruch, SQLite und ORMs ersetzen zu wollen, die Realisierung über NoSQL sowie die Umsetzung der Kernfunktionalität in C++ unterscheidet Realm von anderen Datenhaltungs-Bibliotheken wie z. B. *OrmLite*, oder *greenDAO*. Diese Alternativen, welche im Kern hauptsächlich auf das Android-eigene Datenbank-Management-System SQLite setzen, können in Lese- und Schreibgeschwindigkeit nicht mit *Realm* mithalten [28]. Abb. 5.i [29] zeigt dies beispielhaft für Schreiboperationen.

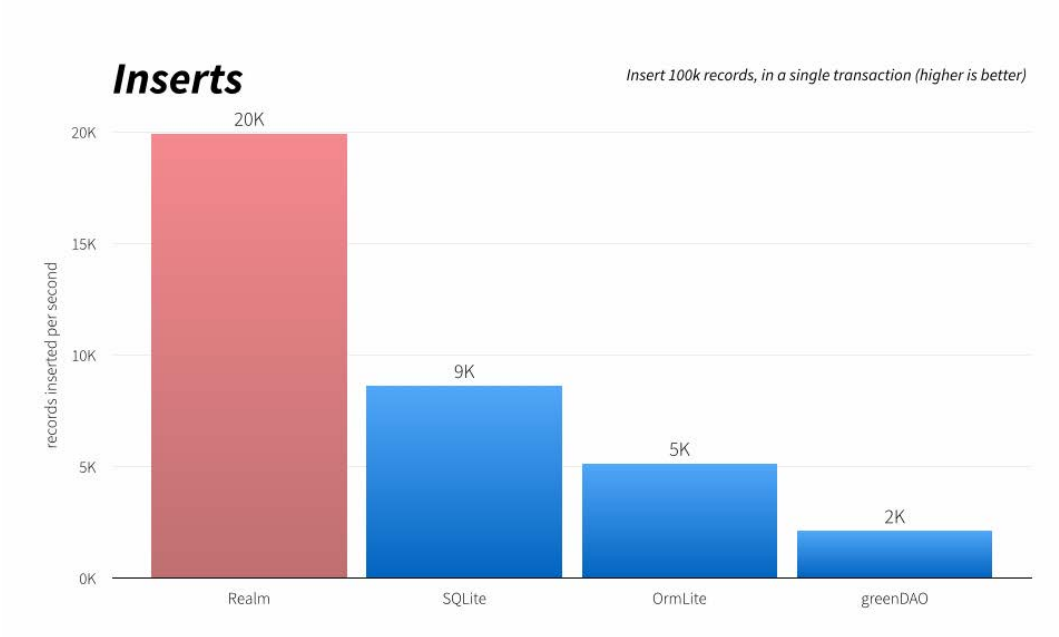

Abb. 5.i Performance-Vergleich (Inserts pro Sekunde): Realm, SQLite, OrmLite, greenDAO [29]

Der von *Realm* verfolgte Ansatz kommt ohne *Data Access Objects* (DAOs) aus und verlangt daher, immer direkt mit den originalen Datensätzen zu arbeiten. Hierdurch und wegen des jungen Alters von *Realm* sind allerdings auch einige wünschenswerte Funktionalitäten noch nicht vorhanden und bedürfen manueller Workarounds. Die detaillierten Herausforderungen werden in Kap. 0 genauer beschrieben.
Realm wird in Version 0.75.1 verwendet.

5.7.3 Gson

Gson von Google [30] dient der Serialisierung und Deserialisierung von Java-Objekten in bzw. von JSON-Daten. Die hohe Verbreitung, die stetige Weiterentwicklung und die umfangreichen Konfigurationsmöglichkeiten garantieren eine gute Integration auch mit anderen Bibliotheken. Daher wurde *Gson* gegenüber *FasterXML Jackson* der Vorzug gewährt und Version 2.2.4 in *psyq* verwendet.
Gson arbeitet gut mit unterschiedlichen Bibliotheken zusammen und wird von einigen auch aktiv unterstützt oder gar für deren Funktionalität vorausgesetzt. Jedoch stellte sich während der Implementierung ein Fehlverhalten von *psyq* heraus, welches in der Verwendung von *Gson* in Verbindung mit einem *Realm-Workaround* begründet lag. Eine detaillierte Beschreibung der Fehlerursache sowie dessen Behebung findet sich in Kap. 0.

Gson bildet *Plain Old Java Objects* (POJOs) mithilfe der Klassen-Angabe aus JSON-Dokumenten (siehe Abb. 5.j).

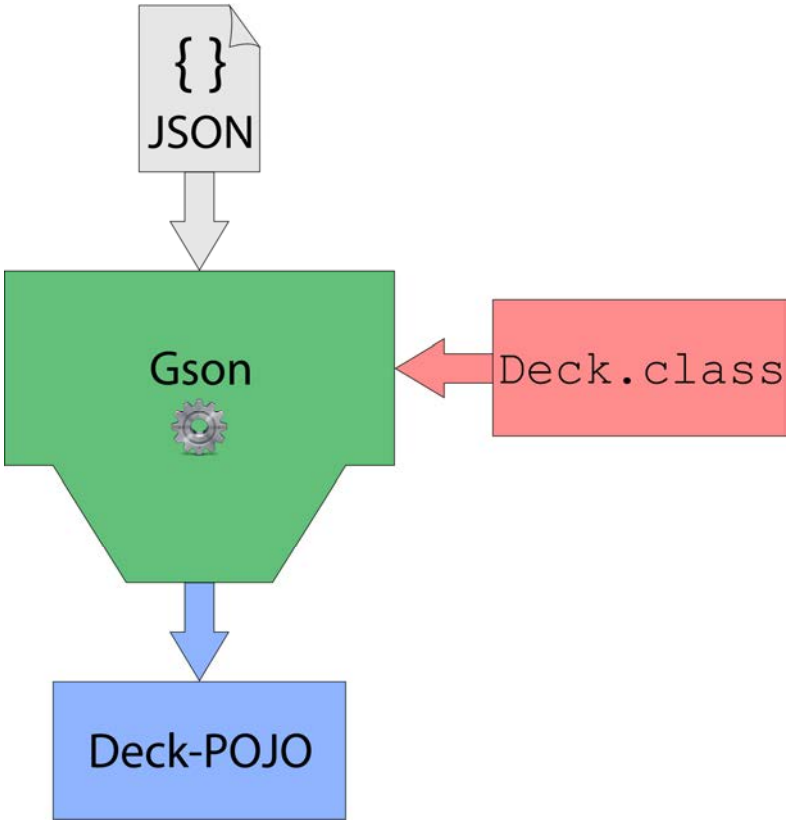

Abb. 5.j: Erzeugung eines POJOs durch Gson

5.7.4 EventBus

Für die Kommunikation zwischen den einzelnen Komponenten der App kommt ein *EventBus* zum Einsatz. Der *greenrobot EventBus* (Version 2.2.1) [31] standardisiert die Kommunikation z. B. zwischen *Activities* und *Fragments*, oder aber auch zwischen View und Hintergrundaktivitäten wie Updates oder dem lokalen Abfragen von Daten (siehe Abb. 5.k [32]). Dadurch können zahlreiche ansonsten nötige manuelle *Listener*-Konstruktionen entfallen.

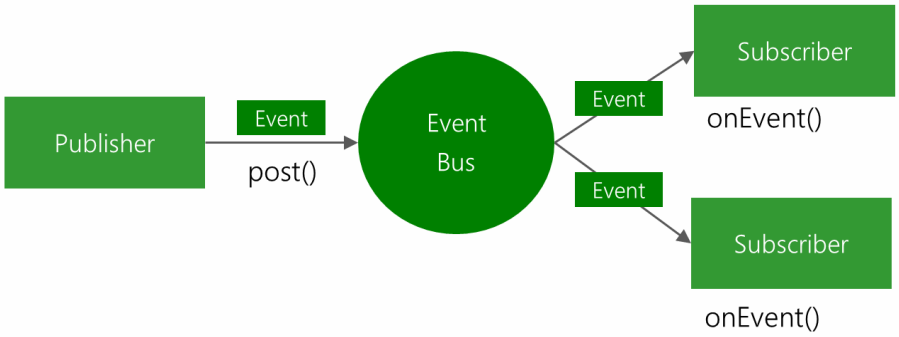

Abb. 5.k Ablauf der Kommunikation über den EventBus [32]

Zudem bieten die sog. *StickyEvents* den Mehrwert, Ereignisse "anzuheften", um auch nach ihrem Eintreten noch auf sie Bezug nehmen zu können. Als Beispiel sei hier das *StickyEvent* `GetStoredUserFinishedEvent` genannt, durch das unterschiedliche Abnehmer jederzeit das `User`-Objekt des aktuell eingeloggten Spielers erhalten können.
Diese Funktionalität war neben der deutlich besseren Performance [33] das Entscheidungskriterium für *greenrobot EventBus* und gegen die ebenfalls beliebte *EventBus*-Programmbibliothek *Otto* von *Square, Inc*.

5.7.5 Android Priority Job Queue

Das Abarbeiten von komplexen (Hintergrund-)Aufgaben ist seit jeher ein umstrittenes Thema in der Android-Programmierung. Um *Activities* und *Fragments* schlank zu halten, wurden aufwendigere Aufgaben in umständliche `AsyncTasks` ausgelagert, die allerdings ihrerseits Probleme verursachen können. Da der von Google eingeführte `Loader`-Ansatz zwar deutlich besser, doch ebenfalls nicht optimal für Netzwerkanfragen geeignet ist, besteht auf diesem Gebiet nach wie vor eine große Nachfrage nach einer komfortablen Lösung.
psyq verarbeitet daher sämtliche umfangreicheren Aufgaben mittels *Android Priority Job Queue* (Version 1.1.2) [34], einer Bibliothek, welche die zuvor beschriebene Problematik beheben möchte.

5.7.6 Weitere Bibliotheken

- *Butter Knife* 6.0.0 und *NineOldAndroids* 2.4.0:
  Butter Knife [35] von *Jake Wharton* optimiert das identifizieren von Layout-Elementen in den zugehörigen Klassen durch *View-Injection*. Die Bibliothek *NineOldAndroids* [36] – ebenfalls von *Jake Wharton* – stellt ansprechende Animationen auch für niedrige API-Levels bereit.

- *SnappyDB* 0.5.0 und *kryo* 2.24.0:
  Die in Kap. 6.1.3 erläuterte Speicherung von Rahmendaten wird durch den Einsatz der *Key-Value*-Speicherung in *SnappyDB* [37] (benötigt *kryo* [38]) realisiert.

- *Retrofit* 1.7.1 und *OkHttp* 2.0.0 (mit *OkHttp-UrlConnection* 2.0.0)
  Diese von *Square, Inc.* entwickelten Bibliotheken dienen der API-Abfrage. Mittels *Retrofit* [39] wird die API von *psyq* als Interface verwendbar gemacht. *OkHttp* [40] gilt als extrem leistungsstarker HTTP-Client, der auch *SPDY* unterstützt und in Android 4.4 sogar die originale `HttpUrlConnection` von Android antreibt [41].

- *UniversalImageLoader* 1.9.3:
  Bilder, die die App vom Server bezieht, wie z. B. Deck-Images oder Spieler-Avatare, werden mithilfe dieser Bibliothek [42] effizient geladen und können in einem Cache vorgehalten werden.

- *TextDrawable* 1.0.0:
  TextDrawable [43] bietet die Möglichkeit eines spezialisierten `TextViews`. Die Bibliothek eignet sich daher hervorragend zur Darstellung der nummerierten Listen-Indizes in der Antwort-Liste.

- *EazeGraph* 1.1.14-SNAPSHOT:
  Die grafische Anzeige der Statistik in der App wird durch *EazeGraph* [44] realisiert. Die Bibliothek ist sehr variabel und erzeugt ansprechende Diagramme.

6 Implementierung

Im Folgenden wird detailliert auf wichtige Aspekte bei der technischen Umsetzung von *psyq* sowie auf deren Besonderheiten und Herausforderungen eingegangen. Kap. 6.1 beschäftigt sich zunächst mit der Implementierung der mobilen App. Die nachfolgenden Kapitel betrachten die API und das Backend.

6.1 App

Über die App kann der Nutzer mit dem System interagieren und somit auch auswertbare Daten liefern. Es gilt, die in Kap. 3.2 diskutierten Anforderungen ebenso umzusetzen wie einen geringen Datenverkehr und eine schnelle Reaktionszeit zu erzielen – insbesondere im mobilen Kontext sind diese Kriterien von hohem Stellenwert.

6.1.1 Material Design und Android Design Guidelines

psyq soll bezüglich Erscheinungsbild und Verhalten aktuell wirken und bzgl. des Bedienkonzepts konsistent mit anderen Apps sein, um sich so nahtlos in die allgemeine App-Landschaft einzufügen. Durch die Übernahme verschiedener Design- und Bedienkonzepte, welche im

Android Design Guide [45] definiert und erklärt werden, entsteht eine einheitliche App-übergreifende Nutzererfahrung.
In *psyq* werden unterschiedliche Design-Vorgaben umgesetzt, um das aktuelle *Material Design* aufzugreifen bzw. abzubilden. Durch den Einsatz der von Google bereitgestellten *support-library-21-v7* ist es möglich, die wichtigsten der nötigen View-Elemente ebenso unter älteren Android-Versionen zu nutzen oder zumindest zu simulieren. Auf diese Weise kommen auch ältere Geräte in den Genuss neuartiger Konzepte – eine womöglich teure Neuanschaffung kann somit entfallen oder zumindest verschoben werden.

6.1.1.1 ToolBar

Sofort auffallend in der neuen Designsprache ist die überarbeitete *ActionBar* am oberen Rand des Bildschirms. Diese nennt sich im Rahmen des *Material Design* nun *ToolBar* und ist erstmals ein separates View-Element. Auf diese Weise kann es in den XML-Layouts deutlich einfacher angepasst werden als die relativ starre *ActionBar*. Für die Nutzung der *ToolBar* ist es wesentlich, die ursprüngliche *ActionBar* im *Theme*, wie in Listing 6.a: Deaktivierung der ActionBar gezeigt, zu deaktivieren.

```
1 | <item name="windowActionBar">false</item>
```

Listing 6.a: Deaktivierung der ActionBar

Unter Zuhilfenahme der Support-Library kommt die *ToolBar* auch auf älteren Versionen zum Einsatz. Listing 6.a zeigt die dafür nötigen Schritte innerhalb der betreffenden *Activity*.

```
1  | @Override
2  | protected void onCreate(Bundle savedInstanceState) {
3  |       // (...)
4  |       // get toolbar from layout, set as
5  | SupportActionBar
6  |       ToolBar toolbar = (ToolBar)
7  | findViewById(R.id.toolbar);
8  |       setSupportActionBar(toolbar);
9  |       // (...)
10 | }
```

Listing 6.b: ToolBar als SupportActionBar

Im Zuge dieser Aktualisierung wurde auch das zuvor meist obligatorische, da automatisch eingefügte App-Icon aus der *ToolBar* entfernt. Zwar ist es nach wie vor möglich, ein Icon einzublenden, die Design-Richtlinien von Google nehmen davon aber Abstand [46].

6.1.1.2 Baseline Grid und Keylines

Abstände sind in den Design-Vorgaben ebenso definiert wie Icon-Größen, Grauwerte und Schriftstile. In *psyq* wurde auf diese Richtwerte geachtet und u. a. das Modell des *8 dp-Baseline-Grid* konsequent umgesetzt. Hierfür werden sämtliche *User-Interface*-Elemente an einem 8 dp-Raster ausgerichtet, was einer ansehnlich geordneten Benutzeroberfläche zugutekommt.
Zudem empfiehlt Google für diverse Einsatzzwecke konkrete Abstände zwischen den UI-Elementen oder zum Bildschirmrand.
Abb. 6.a[47] veranschaulicht einige dieser Empfehlungen.

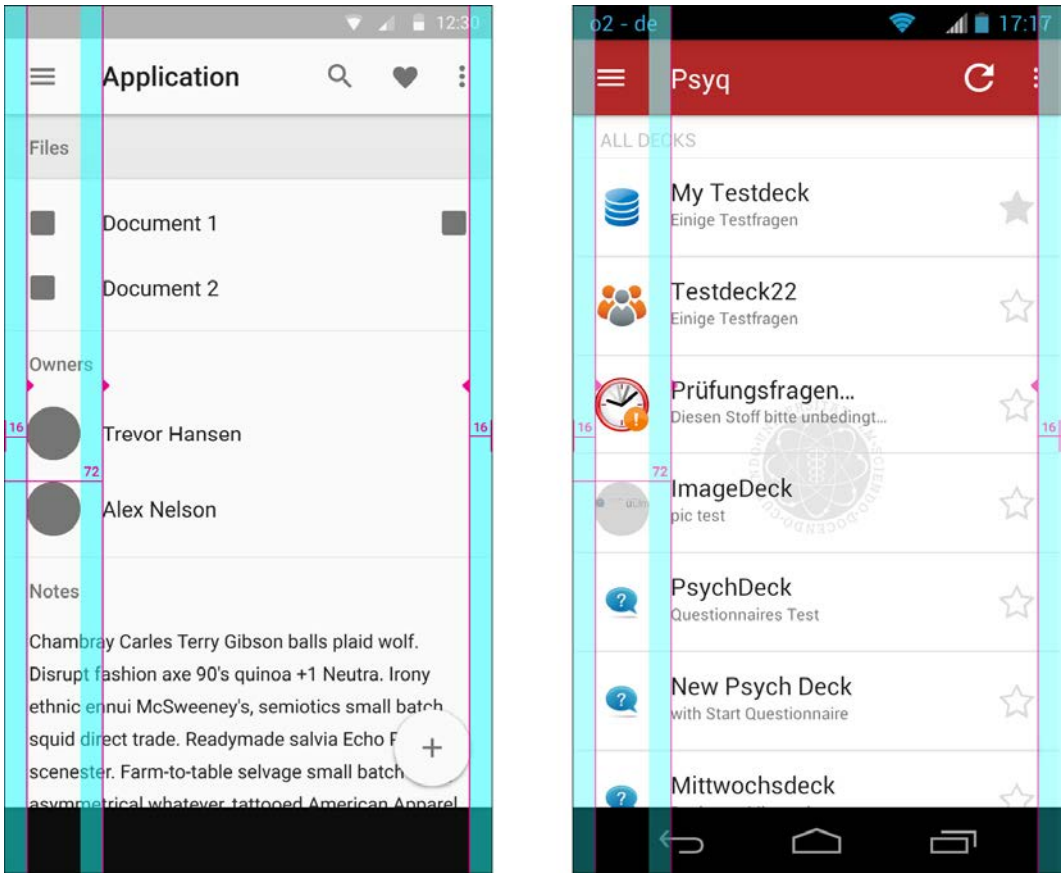

Abb. 6.a: Beispiel zu Google-Vorgaben bzgl. einheitlicher Abstände (Angaben in dp) [47] und deren Umsetzung in *psyq*

Solche Grid- und Keyline-Vorgaben beziehen sich ebenfalls auf die in Kap. 6.1.1.1 erwähnte *ToolBar*. Ihr Aussehen ist hinsichtlich der App-übergreifenden optischen Konsistenz von besonderer Bedeutung.
So soll u. a. darauf geachtet werden, dass der angezeigte Titel-Text der *ToolBar* - oftmals ist dies schlichtweg der Name der App – exakt 72 dp vom linken Rand des Smartphone-Bildschirms entfernt ist. Auf diese Weise möchte man eine visuelle Konsistenz zu Listenelementen im Hauptteil der App schaffen, deren Titel und Beschreibung denselben Richtlinien entsprechen.

Programmiertechnisch gibt es allerdings keine simple Unterstützung dieser Vorgabe für die *ToolBar*, mit deren Standardeinstellung der Titel deutlich zu nah am linken Bildschirmrand platziert wird. Die erwartete Methode `toolbar.setTitleStartPadding(72)` oder ihr Pendant als XML-Attribut zur Angabe im entsprechenden Layout-Dokument ist schlichtweg nicht vorhanden.

psyq nutzt daher einen Trick, um den geforderten Abstand dennoch einhalten zu können: Mittels der in Listing 6.c dargestellten Angabe in der `MainActivity` der Applikation wird definiert, dass die *ToolBar* ein transparentes Icon namens `actionbar_spacer` einbinden soll. Diese Möglichkeit wurde bereits in Kap. 6.1.1.1 erwähnt.

```
1  @Override
2  protected void onCreate(Bundle savedInstanceState) {
3        // (...)
4        // set transparent drawable as icon to push
5        // toolbar title margin to desired value
6        getSupportActionBar()
7              .setIcon(R.drawable.actionbar_spacer);
8        // (...)
9  }
```

Listing 6.c: Anpassung des Titelabstands in SupportActionBar

Statt einer Grafik ist das angegebene Icon allerdings ein per XML definiertes `Drawable`. Diese Art der Grafikerstellung kann in Android dazu genutzt werden, geometrische Formen, Farbverläufe etc. textuell zu erzeugen – ähnlich der Erstellung von SVG-Grafiken. In diesem Fall wird lediglich ein transparenter Quader erzeugt, um den Titel der *ToolBar* zu verschieben. In diesem Fall sind weitere 12 dp nötig, um den vorgegebenen Wert zu erreichen.

Der Inhalt des dafür erzeugten Dokuments `drawable/actionbar_spacer.xml` ist in Listing 6.d ersichtlich:

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <shape
3        xmlns:android="http://schemas.android.com/apk/re
4  s/android"
5        android:shape="rectangle">
6        <solid
7              android:color="#00FFFFFF"/>
8        <stroke android:width="0dp"
9              android:color="#00aaaaaa" />
10       <size
11             android:width="12dp"
12             android:height="2dp"/>
13 </shape>
```

Listing 6.d: Transparente Grafik um den geforderten Abstand herzustellen

6.1.1.3 NavigationDrawer

Das Konzept des *NavigationDrawer* als von der Seite erreichbares Menü (siehe Abb. 3.a) besteht bereits seit einiger Zeit. Mussten zunächst noch externe Bibliotheken bemüht werden, bietet Google inzwischen eine eigene Implementierung des *DrawerLayouts*, welches praktischerweise ebenfalls über die Support-Library abwärtskompatibel gemacht wurde.

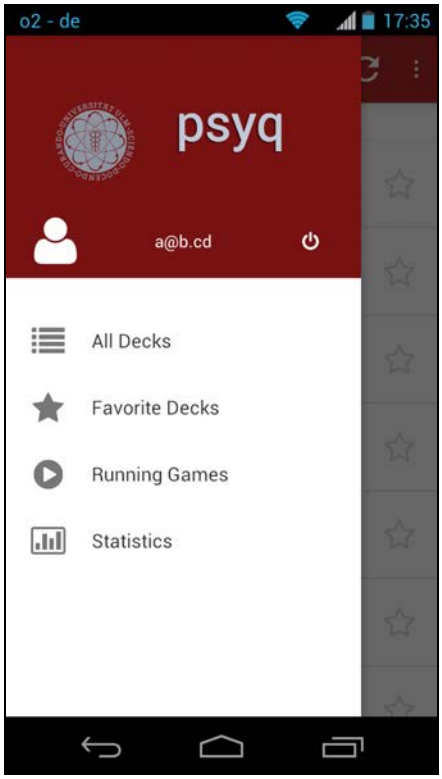

Abb. 6.b: App-Menü via NavigationDrawer

Wegen der enormen Vorteile des *NavigationDrawer*-Ansatzes hat sich diese Art der App-Strukturierung zunehmend verbreitet und wird von Google auch für viele Szenarien als ideales Navigationsmuster empfohlen [48].
Im Gegensatz zur zuvor sehr beliebten Tab-Navigation können so mehr Menüpunkte gleichzeitig angezeigt werden, die ihrerseits auch wesentlich komplexer sein dürfen. Daher ist es nicht unüblich, dass Einträge im *NavigationDrawer* eigene Untermenüpunkte, Counter oder *Badges* (z. B. Zähler für ungelesene Nachrichten) beinhalten. Bei der Tab-Navigation sind solche erweiterten Navigationselemente selbst mit *Custom-Tabs* [49] nur eingeschränkt möglich. Zudem belegt der *NavigationDrawer* keinen wertvollen Display-Raum, der durch die Tab-Leiste zumeist deutlich eingeschränkt wurde.
Einziger Nachteil gegenüber dem Tabs-Konzept ist der zusätzlich nötige *Touch-Tap* auf das Menü-Icon bzw. die *Slide*-Geste vom linken Bildschirmrand, um den *NavigationDrawer* anzuzeigen.

Des Weiteren ist bei der App-Planung darauf zu achten, wie die hierarchische und historische Struktur innerhalb der App behandelt und die *Up*-Navigation realisiert wird, um den *NavigationDrawer* zu jedem Zeitpunkt schnell erreichbar zu halten. Die Dialogstruktur von *psyq* wurde dementsprechend geplant und ist in Kap. 5.5 ausführlich dargelegt.

Allgemeine Verwirrung gab es bei Android-Entwicklern in Bezug auf die Anordnung des *NavigationDrawers* innerhalb einer App. Zwar fordert Google in seiner neuen Spezifikation (im Gegensatz zur alten Richtlinie [48]), dass der *NavigationDrawer* beim Herausziehen alle anderen Komponenten der zugehörigen App überdecken soll [50]. Doch wurde diese Forderung in einigen eigenen Apps des Unternehmens selbst nicht umgesetzt [51] und die *ActionBar* bzw. *ToolBar* nicht wie vorgegeben überlagert. Abb. 6.c [52] von veranschaulicht die unterschiedlichen Umsetzungen innerhalb der verschiedenen Google-Apps.

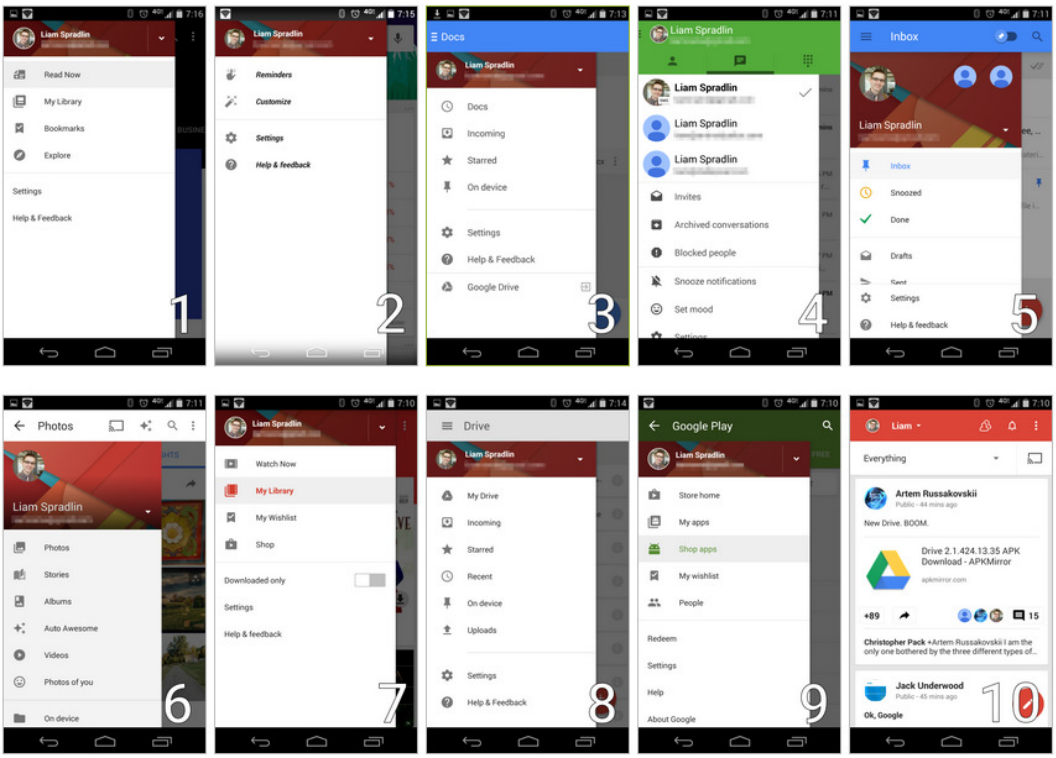

Abb. 6.c Unterschiedliche Umsetzung des NavigationDrawers in den Google-Apps [52]

Dies hatte Diskussionen zur Folge, da die "Google-Apps" von vielen Nutzern und auch Entwicklern als Vorlage für weitere Apps angesehen werden und mit gutem Beispiel vorangehen sollten. Auch das neue animierte "Hamburger-Icon", welches sich beim Herausziehen des *NavigationDrawers* in einer ansprechenden Animation zum "Zurück-Pfeil" transformiert, trug zur Irritation bei, wird es doch bei Beachtung der Richtlinie komplett überlagert.

Mit Version 5.1.11 hat Google nun aber auch seine *PlayStore*-App an die selbst gestellten Vorgaben angepasst [53]. Die Animation des "Hamburger-Icons" wird infolgedessen kurz nach ihrem Start überlagert.

psyq folgt dieser Vorgabe ebenfalls, um ein konsistentes Nutzererlebnis zu ermöglichen.

6.1.2 Fragments

Die Kern-View-Komponente von Android stellen seit jeher die *Activities* dar. Eine Activity repräsentiert eine bestimmte Bildschirm-Ansicht und beinhaltet untergeordnete Views, wie z. B. Buttons, Textfelder oder auch komplexere *Compound*-Views, wie z. B. Listen oder scrollbare Container.

Mit dem Android-API-Level 11 führte Google *Fragments* ein und stellt sie unter Zuhilfenahme der Support-Library auch für niedrigere API-Levels zur Verfügung.
Fragments werden i. d. R. als Zwischenschicht zwischen *Activities* und deren untergeordneten Komponenten genutzt. *Fragments* besitzen einen eigenen Lebenszyklus und ermöglichen es ebenfalls, *CompoundView*s und *Views* zu kapseln. Seit API-Version 17 können *Fragments* sogar selbst *Fragments* enthalten. Mehrere Fragments können unter einer *Activity* gebündelt werden.
So muss die *Acitvity*, die einer Bildschirmansicht entspricht, nichts mehr über den konkret dargestellten Inhalt wissen, sondern nur ihre *Fragments* kennen. Mit diesem Konzept lassen sich praktische Gruppierungen auf UI-Ebene erzeugen. Eine App kann z. B. auf dem Smartphone *ActivityA* anzeigen, welche *FragmentA* beinhaltet und bei einem Klick zu *ActivityB* wechseln, die *FragmentB* anzeigt. Über die *Size Qualifier* kann erreicht werden, dass dieselbe App auf Tablets automatisch ein anderes Layout lädt [54]. Im genannten Beispiel wird nun *ActivityA* dargestellt, die *FragmentA* und *FragmentB* nebeneinander anzeigt (siehe Abb. 6.d [55]). Die Implementierung der Fragments selbst bleibt davon weitgehend unberührt und eignet sich so optimal zur Wiederverwendung.

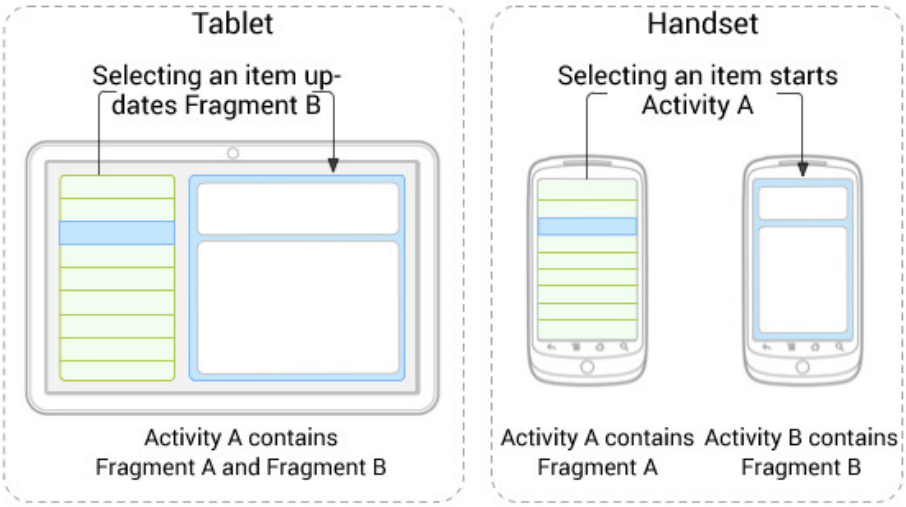

Abb. 6.d Vorteile durch die Verwendung von Fragments [55]

Daher verwendet *psyq* ebenfalls das vorteilhafte Fragment-basierte Konzept und geht auch auf programmiertechnischer Ebene den Weg der Abstraktion, Modularisierung und Wiederverwendbarkeit, wie nachfolgend aufgezeigt wird.

Nachdem die Login-*Activity* aufgrund eines erfolgreichen Login-Vorgangs verlassen oder wegen bereits gespeicherter Login-Informationen übersprungen wurde, werden sämtliche Anzeigen in `MainActivity`, der einzig vorhandenen *Activity*, realisiert. Alle unterschiedlichen Ansichten mit verschiedenen Inhalten und Layouts sind ausschließlich durch *Fragments* realisiert, wie nachfolgend erläutert wird.
Die `MainActivity` "kennt" auf Layout-Ebene lediglich die `ActionBar`, das Menü des *NavigationDrawers* sowie einen "leeren" Container (`@+id/frame_layout`), der das jeweils benötigte, dynamisch erzeugte *Fragment* aufnimmt.
Listing 6.e zeigt das zugehörige Layout `activity_main.xml`.

```
1  <android.support.v4.widget.DrawerLayout
2        android:id="@+id/drawer_layout"
3        ...>
4
5  <LinearLayout
6        android:layout_width="match_parent"
7        android:layout_height="match_parent"
8        android:orientation="vertical" >
9
10       <android.support.v7.widget.Toolbar
11             android:id="@+id/toolbar"
12             android:layout_width="match_parent"
13             android:layout_height="56dp"
14             ... />
15
16       <!-- The main content view -->
17       <FrameLayout
18             android:id="@+id/frame_layout"
19             android:layout_height="match_parent"
20             android:layout_width="match_parent"
21             ...>
22       </FrameLayout>
23
24 </LinearLayout>
25
26 <fragment
27       android:id="@+id/navigation_drawer"
28       android:name="de.akln.psyq.NavDrawerFragment"
29       android:layout_height="match_parent"
30       android:layout_width="match_parent"
31       ... />
32
33 </android.support.v4.widget.DrawerLayout>
```

Listing 6.e: Layout activity_main.xml der MainActivity

Auf Code-Ebene übernimmt die `MainActivity` dagegen weitere Aufgaben, v. a. solche, die von einer zentralen Behandlung profitieren. Listing 6.f zeigt beispielhaft die Handhabung des `GetStoredUserFailedEvents`.

```
1  public void onEventMainThread(
2        GetStoredUserFailedEvent mEvent) {
3        // show reason to the user
4        Toast.makeText(this, mEvent.getReason(),
5              Toast.LENGTH_SHORT).show();
6        // redirect to login screen
7        forwardToLoginActivity();
8  }
```

Listing 6.f: Behandlung des GetStoredUserFailedEvents in der MainActivity

Zudem wird durch die `MainActivity` das jeweils benötigte Fragment, welches den hauptsächlichen Inhalt der Ansicht darstellt, erzeugt und in `@+id/frame_layout` "eingesetzt".

Einige dieser Fragments sind komplexer gehalten und aufwendiger zu implementieren. Zu deren besserer Wiederverwendbarkeit, und um Code-Redundanz auszuschließen, wurde in *psyq* eine Zwischenschicht geschaffen.
So existiert z. B. eine abstrakte Klasse `AbsDecksFragment`, die die Anzeige von Decks in Form einer Liste vorbereitet. Diese Klasse vereint die dafür notwendigen allgemeinen Methoden wie z. B. das Erfassen und Behandeln eines *Touch-Taps* auf einen Listeneintrag. Von ihr erben die konkreten Implementierungen `ConcreteAllDecksFragment`, `ConcreteFavDecksFragment` und `ConcreteRunningDecksFragment`, die allesamt Decks in einer Listenansicht anzeigen, sich allerdings in den Details unterscheiden: Während die Favoriten-Ansicht nur favorisierte Decks listet, stellt das Running-Fragment nur solche Decks dar, mit denen bereits ein Spiel begonnen wurde.
Auch die jeweiligen *Quick-Actions* in den Listeneinträgen unterscheiden sich: In der Gesamtansicht ist es möglich, Decks zu den Favoriten hinzuzufügen. In der Favoriten-Ansicht ist diese Aktion obsolet. Hier wurde nur die Möglichkeit realisiert, Decks von der Favoritenliste zu entfernen.
Um diese Unterscheidung zu ermöglichen und andere abweichende Verhalten zu implementieren, wurden individuelle Adapter entwickelt, die von `adapter/AbsDecksAdapter`, ebenfalls eine abstrakte Basis-Klasse, erben.

6.1.3 Rahmendaten

Die Nutzung und der Spielablauf von *psyq* führen dazu, dass die App „Rahmendaten“ erzeugt. Diese speichern z. B. Favoriten-Decks oder die Fortschritte innerhalb der Decks und sichern zudem die Login-Daten bis zum nächsten Logout. Sie liegen lokal vor, eine Synchronisation mit dem Server ist nicht nötig.
Durch die Rahmendaten wird gewährleistet, dass der Spieler angefangene Spiele auch nach wochenlanger Pause und diversen Neustarts des Geräts an der entsprechenden Stelle fortsetzen kann und seine zuvor favorisierten Decks als solche eingeordnet vorfindet.

Die Speicherung der Zugangsdaten erfolgt einmalig in der App, da pro App-Installation nur ein Spieler gleichzeitig eingeloggt sein kann. Die weiteren erwähnten Eigenschaften verweisen auf den Account des zum Erstell-Zeitpunkt eingeloggten Spielers, um so eine korrekte Zuordnung herzustellen.

Da diese vergleichsweise simplen Daten keine komplexe Hierarchie beinhalten und auch nur lokal vorgehalten werden müssen, benötigt ihre Speicherung kein aufwendiges Datenbanksystem.
Im Standardumfang des Android-Frameworks steht das Konzept der sog. *SharedPreferences* zur Verfügung, die sich für solche Einsatzzwecke eignen. Sie speichern einfache *Key-Value-Paare* und bieten sich somit für einen solchen Einsatzzweck an.
Einen ähnlichen Ansatz – allerdings mit umfangreicherem Funktionsumfang – stellt die Bibliothek *SnappyDB* von *Nabil Hachicha* zur Verfügung. *SnappyDB* basiert auf *leveldb*, ist aus Gründen der besseren Leistung in nativem Code geschrieben (Android *NDK*), kann *Keys* durchsuchen und neben primitiven Datentypen und Arrays auch komplexe Objekte als *Value* speichern. Das Java-*Interface* `Serializable` wird intern genutzt, muss aber nicht von jeder betroffenen Klasse manuell implementiert werden [37]. Obwohl es naheliegt, kommt das verwandte Android-*Interface* `Parcelable` hier nicht zum Einsatz. *SnappyDB* nutzt zum einen den *Snappy*-Kompressionsalgorithmus, um redundante Inhalte platzsparend zu sichern, sowie das außerordentlich schnelle Objekt-Graph-Serialisierungs-Framework *Kryo* zur effizienten *Serialisierung* der Daten.

Die ermöglichte Speicherung von Objekten sowie die sehr gute Performance begründen den Einsatz von *SnappyDB* in *psyq* für die anfallenden Rahmendaten.

Das nachfolgende Listing 6.g zeigt anschaulich die Sicherung der Login-Informationen in *SnappyDB*. Entnommen wurde dieser Ausschnitt aus `job/AuthUserJob.java`.

```
1  // (...)
2
3  // get user object
4  User u = gson.fromJson(result, User.class);
5
6  // (...)
7
8  // store this users id in snappy
9  try {
10       DB snappydb = DBFactory.open(App.getContext());
11       snappydb.putBoolean("userLoggedIn", true);
12       snappydb.put("user", u);
13       snappydb.close();
14
15       // post success event
16       EventBus.getDefault().post(
17             new AuthUserFinishedEvent());
18
19 } catch (SnappydbException e) {
20       // post fail event
21       EventBus.getDefault().post(
22             new AuthUserFailedEvent("Could not store
23 user id"));
24 }
25
26 // (...)
```

Listing 6.g: Speicherung des Logins mittels SnappyDB

Neben dem `User`-Objekt wird auch noch das Boolean-*Flag* `userLoggedIn` gesichert, da dessen Abfrage als primitiver Datentyp deutlich schneller als die des `User`-Objekts vonstatten geht und für viele Entscheidungen ausreicht.

Die Abfrage eines gespeicherten Wertes stellt sich ebenfalls vergleichsweise komfortabel dar. Listing 6.h (entnommen aus `job/GetStoredUserJob.java`) veranschaulicht die Abfrage des gespeicherten Users beispielhaft.

```
1  try {
2        DB snappydb = DBFactory.open(App.getContext());
3
4        // get user from snappy
5        User user = snappydb.getObject("user",
6              User.class);
7        // (...)
8        // post user object to eventbus
9        EventBus.getDefault().postSticky(
10             new GetStoredUserFinishedEvent(user));
11       // (...)
12       }
13       snappydb.close();
14 }
15 catch (SnappydbException e) {
16       e.printStackTrace();
17       EventBus.getDefault().post(
18             new GetStoredUserFailedEvent("no user
19 stored."));
20 }
```

Listing 6.h: Abfragen des User-Objekts

Für die Abfrage von spielerspezifischen Rahmendaten bedarf es einer etwas ausführlicheren Anfrage, da die Anfrage hier die Trennung nach Spielern berücksichtigen muss.

6.1.4 Jobs

Statt Logik-Operationen in *Fragments* oder *Activities* zu absolvieren, nutzt *psyq* das zentrale Job-Verwaltungs-Framework *Android Priority Job Queue*. Durch dessen Einsatz können sämtliche komplexeren Datenbank- und Netzwerkanfragen ausgegliedert und dadurch auch wiederverwendet werden, wenn verschiedene aufrufende Komponenten dieselbe Aktion – und damit den identischen Job – auslösen möchten.

Eine in sich geschlossene Aktion wird hierbei durch einen Job abgebildet. Dieser Job muss von `com.path.android.jobqueue.Job` erben und ruft in seinem Konstruktor den Konstruktor seiner Superklasse mit den entsprechenden Werten und Eigenschaften auf, die dieser Job konkret benötigt (Listing 6.i).

```
1  public UpdateAllDecksCompleteJob() {
2        // arguments could also be passed
3        // via constructor if required
4
5        super(new Params(Priority.HIGH)
6              .persist().requireNetwork());
7  }
```

Listing 6.i: Konstruktor eines Jobs

Der Job muss außerdem folgende Methoden überschreiben [34] (Listing 6.j):

```
1  @Override
2  public void onAdded() {
3        // The job was added. Inform the UI
4        // that the job will eventually run.
5  }
6  @Override
7  public void onRun() throws Throwable {
8        // The jobs logic goes here.
9  }
10 @Override
11 protected boolean shouldReRunOnThrowable
12       (Throwable throwable) {
13       // An error occurred in onRun.
14       // Return value determines whether this job
15       // should retry running (true) or abort (false).
16 }
17 @Override
18 protected void onCancel() {
19       // Job has exceeded retry attempts or
20       // shouldReRunOnThrowable() has returned false.
21 }
```

Listing 6.j: Zu überschreibende Methoden eines Jobs

In der Methode `onAdded()` können kurze Operationen, die zwischen dem Hinzufügen des Jobs zur *Queue* und der eigentlichen Aufgabe des Jobs liegen, getätigt werden. Hier wird z. B. eine Rückmeldung per *Event* (siehe Kap. 6.1.5) an die entsprechenden View-Komponenten gesendet, die diese über die erfolgreiche Erzeugung des Jobs informiert. Nachfolgend kann z. B. eine kurze Statusmitteilung wie „Starte Update“ gezeigt oder ein *Progress-Spinner* eingeblendet werden.

`onRun()` beinhaltet nun das Kernstück eines Jobs, also seine wesentliche Aufgabe. An dieser Stelle werden z. B. Datenbankabfragen, die Speicherung von Spielständen oder API-Aufrufe zur User-Authentifizierung sowie für das Update der Decks erledigt.

`shouldReRunOnThrowable(Throwable throwable)` gibt an, ob der Job bei einem Fehler wiederholt oder schlicht verworfen werden soll.

Da in `onCancel()` beschrieben werden kann, was bei einem etwaigen Abbruch des Jobs passieren soll, wurde auch diese Methode überschrieben und mit entsprechenden Events zur Handhabung des Abbruchs gearbeitet. So können Abbrüche von verschiedenen Aufrufern individuell an entsprechender Stelle behandelt werden, ohne dass der Job alle Einzelheiten zur Behandlung benötigt.

Jobs können von allen beliebigen Programm-Komponenten einer App aufgerufen werden, also z. B. auch von einem kurzlebigen `AlertDialog`. Durch die konsequente Verwendung von *Events* zur Verteilung von Erfolgs- und Fehlermeldungen in *psyq* kann der Aufrufer demnach auch schon vom System entfernt worden sein (z. B. durch die *Garbage Collection*), wenn der Job endet. Aufwendige *Callback*-Konstruktionen, die sich teils auch riskant hinsichtlich Speicherlecks zeigen würden, können somit entfallen.

Alle wiederverwendbaren, in sich gekapselten oder äußerst komplexen Aufgaben in *psyq* werden als *Jobs* umgesetzt und befinden sich im Package `de.akln.psyq.jobs`.

6.1.5 Events

Für die Kommunikation innerhalb der App nutzt *psyq* die *EventBus*-Bibliothek von *greenrobot*. Diese stellt sämtlichen Programmkomponenten einen zentralen Kanal zum Informationsaustausch bereit.
Hierfür muss zunächst eine entsprechende Event-Klasse angelegt werden. Diese dient der Identifizierung des jeweiligen Events sowie dem etwaigen Transport komplexer Objekte. Beispielhaft zeigt Listing 6.k die Klasse `GetTaskFailedEvent`.

```
1   public class GetTaskFailedEvent {
2
3       String reason;
4
5       public GetTaskFailedEvent(String reason) {
6           this.reason = reason;
7
8       }
9
10      public String getReason() {
11          return reason;
12      }
13  }
```

Listing 6.k: Event-Klasse GetTaskFailedEvent

Wird nun das in diesem Beispiel erwähnte Event von einer Komponente, wie z. B. dem aktuell arbeitenden Job auf den EventBus gepostet (siehe Listing 6.l), ist dies für alle registrierten Beobachter (siehe Listing 6.m) sichtbar.

```
1 EventBus.getDefault()
2 .post(new GetTaskFailedEvent("job was cancelled"));
```

Listing 6.l: Schreiben eines Events auf den EventBus

```
1 @Override
2 protected void onCreate(Bundle savedInstanceState) {
3     super.onCreate(savedInstanceState);
4
5     // ...
6
7     //register EventBus
8     EventBus.getDefault().register(this);
```

Listing 6.m: Activity registriert sich beim EventBus

Alternativ können sich Beobachter auch mittels `registerSticky(this)` für *Sticky Events* registrieren. Diese speziellen Events bleiben bestehen, auch wenn sie direkt nach ihrer Erzeugung keinen Abnehmer gefunden haben.

Zum Behandeln von Events muss die entsprechende Klasse die Methode `onEvent(Event mEvent)` implementieren, deren einziger Parameter das erwartete Event ist, z. B. `GetTaskFailedEvent mEvent`.
Die Behandlung, die hier von der Klasse `ChooseOneTaskFragment` übernommen wird, ist in Listing 6.n zu sehen. In diesem Fall wird `onEventMainThread` implementiert, eine Methode die für Änderungen im UI-Thread vorgesehen ist.

```
1 public void onEventMainThread(final
2       PlayTaskFailedEvent mEvent) {
3     // ... show error etc
4
5     // restore on click listener that has been set to
6 null at first selection
7     answerListView.setOnItemClickListener(this);
8 }
```

Listing 6.n: Behandlung eines Fehler-Events

6.1.6 Herausforderungen

6.1.6.1 Realm vs. Gson

Zur lokalen Speicherung der Quiz- und Spieldaten verwendet *psyq* die Bibliothek *Realm* (siehe auch Kap 5.7.2). Für ihren Einsatz müssen die *POJO*-Klassen, also Model-Klassen von speichernden Objekten, von `RealmObject` erben. Primitive Klassenvariablen werden ebenso unterstützt wie Objekte, die ihrerseits von `RealmObject` erben. Listen werden unterstützt – allerdings lediglich in der eigenen Implementierung `RealmList<T extends RealmObject>`. Getter und Setter werden von *Realm* intern überschrieben.

Wie eingangs bereits erwähnt, leidet die verwendete Bibliothek *Realm* aufgrund ihres geringen Alters und wegen ihres außergewöhnlichen Prinzips noch an einigen "Kinderkrankheiten". So ist es z. B. nicht erlaubt, Objekte, die von `RealmObject` erben, also Model-Objekte, an Methoden verschiedener *Threads* zu übergeben. Zudem dürfen Model-Objekte ausschließlich wie in Listing 6.o erzeugt werden.

```
1 | Realm realm = new Realm(context);
2 | Deck deck = realm.create(Deck.class);
```

Listing 6.o: Instanziierung von Realm-Objekten

Eine Instanziierung mittels gewöhnlichem Konstruktor und anschließender Speicherung in *Realm* ist bisher nicht erlaubt.
Eine solche Instanziierung ist für die Erstellung von Objekten durch *Gson* jedoch essentiell.
Aufgrund dieser Hindernisse und der damit einhergehenden Problematik wird in *psyq* der folgende Umweg gewählt, bis eine zufriedenstellende Lösung seitens *Realm* realisiert wird.
Listen existieren in den Models doppelt – einmal als `RealmList`, einmal als reguläre `List`. Die Objekte werden nach der JSON-*Deserialisierung* durch *Gson* klassisch erzeugt. Hierfür werden die regulären `Lists` verwendet, damit die entstandenen Objekte über Thread-Grenzen hinweg weitergegeben werden können. Anschließend werden die Objekte über eine Helferklasse rekursiv und innerhalb eines Threads in *Realm*-kompatible Objekte überführt, Listen als `RealmLists` abgebildet und gespeichert. Dieser Schritt ist in einigen Fällen aktuell unumgänglich. Diese Umgehungslösung ist allerdings nur solange nötig, bis *Realm* die reguläre Instanziierung von Model-Objekten und

idealerweise auch das Weitergeben von Objekten über Thread-Grenzen hinweg erlaubt. Man arbeite gar an einer vollständigen Lösung, um JSON-Daten direkt in *Realm* deserialisieren und speichern zu können. In Version 0.76.0 vom 19. Dezember 2014 ist dies erstmals rudimentär möglich [56].

Die angesprochene Problematik sowie deren Übergangslösung führten zu einem weiteren, nur schwer rückzuverfolgenden Problem: Auf diversen *Samsung*-Geräten trat unter Android 4.4.2 und Android 4.4.4 ein interner Fehler auf, der die App sogleich beendete. Nach wenig aussagekräftigen Log-Meldungen und entsprechend langer Fehlersuche konnte die Ursache schließlich ausgemacht werden: Eine `RealmList` ist im Gegensatz zu einer regulären `List` nicht `Serializable` und konnte daher nicht von *Gson* verarbeitet werden. Abhilfe schaffte ein spezifisch konfiguriertes `Gson`-Objekt, das nicht-annotierte Felder explizit von der Serialisierung und Deserialisierung ausschließt, wie in Listing 6.p gezeigt:

```
1  Gson gson = (new GsonBuilder())
2          .excludeFieldsWithoutExposeAnnotation()
3          .create();
```

Listing 6.p: Gson-Objekt, welches nicht annotierte Felder der Model-Klasse ignoriert

Dieses spezifisch erzeugte Gson-Objekt muss anschließend mittels `setConverter` in der *Retrofit*-Konfiguration, wie in Listing 6.q ersichtlich, eingebracht werden.

```
1  restAdapter = new RestAdapter.Builder()
2        .setEndpoint(Const.API_ENDPOINT)
3        .setLogLevel(RestAdapter.LogLevel.FULL)
4        // set the custom gson converter to
5        // avoid RealmList (de)serialization
6        .setConverter(new GsonConverter(gson))
7        .setClient(new OkClient(okHttpClient))
8        .build();
9  setCustomConverter(new GsonConverter(gson));
```

Listing 6.q: Konfiguration zur Verwendung des erstellten Gson-Objekts

6.2 API

Die API stellt die Schnittstelle zwischen App und Serverkomponente dar. Ihr Ziel ist die Gewährleistung des Austausches aller nötigen Daten bei

gleichzeitiger Minimierung des Datenverkehrs, d. h. es soll möglichst wenig *Overhead* entstehen.
Im Folgenden werden die dafür notwendigen Punkte genauer beschrieben.

6.2.1 JSON-Format

Das von der Programmierschnittstelle genutzte Format zur Datenübertragung soll leichtgewichtig, standardisiert und flexibel sein. Eine proprietäre Eigenentwicklung scheidet schon allein wegen der fehlenden Universalität aus. Obwohl XML (eXtensible Markup Language) durch Standards definiert ist und eine geeignete Möglichkeit der Datenhierarchie bietet, hat dieses Format einen Nachteil: den *Overhead*, der u. a. durch die schließenden *Tags* entsteht und so auch zusammen mit dem Inhalt wächst.
Dies ist einer der Gründe, weshalb Entwickler wie auch Diensteanbieter in ihren APIs inzwischen immer öfter auf das schlanke Datenaustauschformat JSON (JavaScript Object Notation) setzen.
Ein JSON-Dokument ist im Allgemeinen aus verschachtelten Objekten und Arrays aufgebaut. Objekte beinhalten Schlüssel-Wert-Paare, wobei der Schlüssel eine textuelle Bezeichnung ist und der Wert eine primitive Eigenschaft, ein Array oder selbst ein Objekt sein kann. Arrays enthalten eine geordnete Liste von (schlüssellosen) Werten.

Beispielhaft sei in Listing 6.r die Antwort im JSON-Format gezeigt, die der Server beim API-Aufruf von `getAllDecksComplete` an die App zurückschickt:

```
1  [
2      {
3          "id": "1",
4          "title": "My Testdeck",
5          "description": "Einige Testfragen",
6          "status": null,
7          "image": "company.png",
8          "start_questionnaire_id": null,
9          "ownTask": [
10             {
11                 "id": "1",
12                 "deck_id": "1",
13                 "pre_questionnaire_id": null,
14                 "ownAnswer": [
15                     {
16                         "id": "1",
17                         "text": "10000",
18                         "correct": "1",
19                         "task_id": "1"
20                     },
21                     {
22                         "id": "2",
23                         "text": "7000",
24                         "correct": "0",
25                         "task_id": "1"
26                     },
27                     {
28                         "id": "3",
29                         "text": "5000",
30                         "correct": "0",
31                         "task_id": "1"
32                     },
33                     {
34                         "id": "4",
35                         "text": "12000",
36                         "correct": "0",
37                         "task_id": "1"
38                     }
39                 ],
40                 "ownQuestion": [
41                     {
42                         "id": "1",
43                         "text": "Wie viele Studenten
44 studieren an der Uni Ulm ca.?",
45                         "task_id": "1",
46                         "image": null
47                     }
48                 ]
49             },
50             ...
51             ]
52         },
53         ...
54 ]
```

Listing 6.r: JSON-Antwort des API-Aufrufs getAllDecksComplete

6.2.2 Rückgabedaten

Um die Daten für den entsprechenden Aufruf liefern zu können, muss die API-Komponente diese zunächst aus der MySQL-Datenbank extrahieren und korrekt miteinander verknüpfen.
Wegen der feingranularen Datenbankstruktur sind aufwendige Abfragen mit komplexen JOIN-SQL-Statements die erwartete Folge. Dies kann jedoch durch die Verwendung von *RedBeanPHP* optimiert werden. Die PHP-ORM-Bibliothek kommt ebenfalls bei der Erzeugung der Daten im Backend zum Einsatz (siehe Kap. 6.3.1) und "kennt" die Beziehung der Datentypen zueinander. So kann die Abfrage zumindest die zugehörigen "Kind"-Objekte mitliefern, ohne auf umständliche JOIN-Statements für jede verknüpfte Tabelle angewiesen zu sein. Aufwendigere Statements oder außergewöhnliche Datenbankoperationen benötigen nach wie vor den Einsatz von klassischem SQL.
Die in Kap. 6.3.1 erwähnte Namenskonvention bzgl. der Datenbanktabellenspalten von *RedBeanPHP* birgt allerdings auch Stolpersteine: So wird z. B. logischerweise jede Tabelle mit einem abgefragten Deck verknüpft, die eine Spalte mit Namen `deck_id` besitzt. Da auch Tabellen wie `PlayedDeck` so auf die zugrundeliegende Deck-ID verweisen, kommt es hier zu Komplikationen. Beim Update würden in diesem Fall nicht nur sämtliche Decks mit ihren enthaltenen Tasks, Questions und Answers zurückgegeben werden, sondern auch sämtliche PlayedDecks mit den entsprechenden PlayedTasks und GivenAnswers. Dies stellt nicht nur einen immensen unnötigen Datenverkehr, sondern auch ein erhebliches Datenschutz-Leck dar. Da *RedBeanPHP* leider nur sehr begrenzte entsprechende Möglichkeiten bietet, bestimmte Tabellen aus- oder nur definierte einzuschließen, wurde die Problematik per PHP gelöst. Nachdem die ORM-Bibliothek alle Daten wie beschrieben zurückliefert, filtert die PHP-API diese gründlich, bevor sie sie weitergibt. Auf diese Weise gelangen nur die gewünschten Daten im JSON-Format zu den Clients.

6.3 Backend

Das Backend stellt den Autoren die Schnittstelle zur Pflege von Inhalten und zur Präsentation von Statistiken zur Verfügung. Eine funktional solide technische Umsetzung sowie eine optisch ansprechende und vielseitig

verwendbare Benutzeroberfläche sind daher ausschlaggebend für den Nutzen und den Komfort des Backends.
Im Folgenden werden einige interessante Charakteristika in diesem Zusammenhang beschrieben.

6.3.1 Daten-Verwaltung

Nachfolgend wird gezeigt, wie das in Kap. 5.6.1 bereits erwähnte ORM *RedBeanPHP* hilfreich eingesetzt werden kann. Listing 6.s veranschaulicht zunächst die nötige Einbindung der Bibliothek sowie den Verbindungsaufbau zur verknüpften Datenbank. Hierbei ist besonders auf die Angabe zur Zeichenkodierung zu achten, welche sicherstellt, dass auch Umlaute und Sonderzeichen entsprechend korrekt kodiert übermittelt werden.

```
1 | R::setup('mysql:host=localhost;dbname=database_rb;
2 | charset=UTF8','database_name','dbpassword');
```

Listing 6.s: Setup von RedBeanPHP

Anschließend können beliebige Datensätze angelegt werden. Für jeden Datentyp wird ein Eintrag in der jeweiligen MySQL-Tabelle erzeugt. Sollte diese noch nicht existieren wird sie automatisch erstellt. Attribute des Objekts werden auf Tabellenspalten abgebildet.
Die zuvor erwähnten Relationen verschiedener Datentypen zueinander müssen mittels Namenskonvention kenntlich gemacht werden. Listing 6.t zeigt, wie ein Task durch die Angabe von `ownAnswer` mehrere Answers besitzen kann. Die für die Abbildung dieser 1:n-Beziehung in diesem Fall nötige Spalte `task_id` in der `Answer`-Tabelle erstellt *RedBeanPHP* automatisch und fügt den passenden Wert ein.

```
1  R::begin();
2
3  $answers = array();
4  foreach ( $_POST['taskAnswers'] as $taskAnswer ) {
5        // only add filled answers
6        if(trim($taskAnswer[text])!==''') {
7              $ans = R::dispense('answer');
8              $ans->text = $taskAnswer[text];
9              $isCorrect = 0;
10             if(isset($taskAnswer[correct]) &&
11 $taskAnswer[correct] == 1) {
12                   $isCorrect = 1;
13             }
14             else {
15                   $isCorrect = 0;
16             }
17             $ans->correct = $isCorrect;
18             R::store($ans);
19             $answers[] = $ans;
20       }
21 }
22
23 $tsk = R::dispense('task');
24 $tsk->deckId = $_POST['taskDeckId'];
25 $tsk->preQuestionnaireId =
26 $_POST['preQuestionnaireId'];
27 $tsk->ownQuestionList[] = $qst;
28 $tsk->ownAnswerList = $answers;
29 $id = R::store($tsk);
30
31 R::commit();
```

Listing 6.t: PHP: Erstellen mehrerer Antworten für einen Task

7 Walkthroughs

7.1 Szenario 1: Anlegen eines Decks im Backend

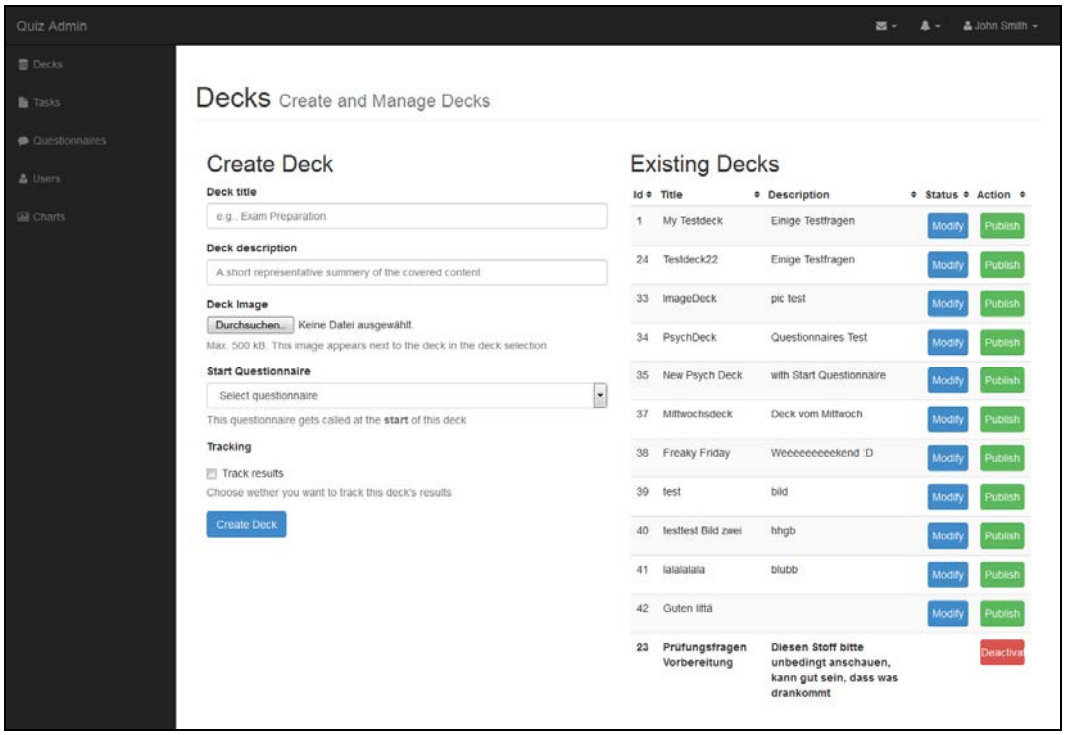

Abb. 7.a: Übersicht der Decks im Backend

Im Backend erhält der Autor zunächst einen Überblick über alle Decks (Abb. 7.a).

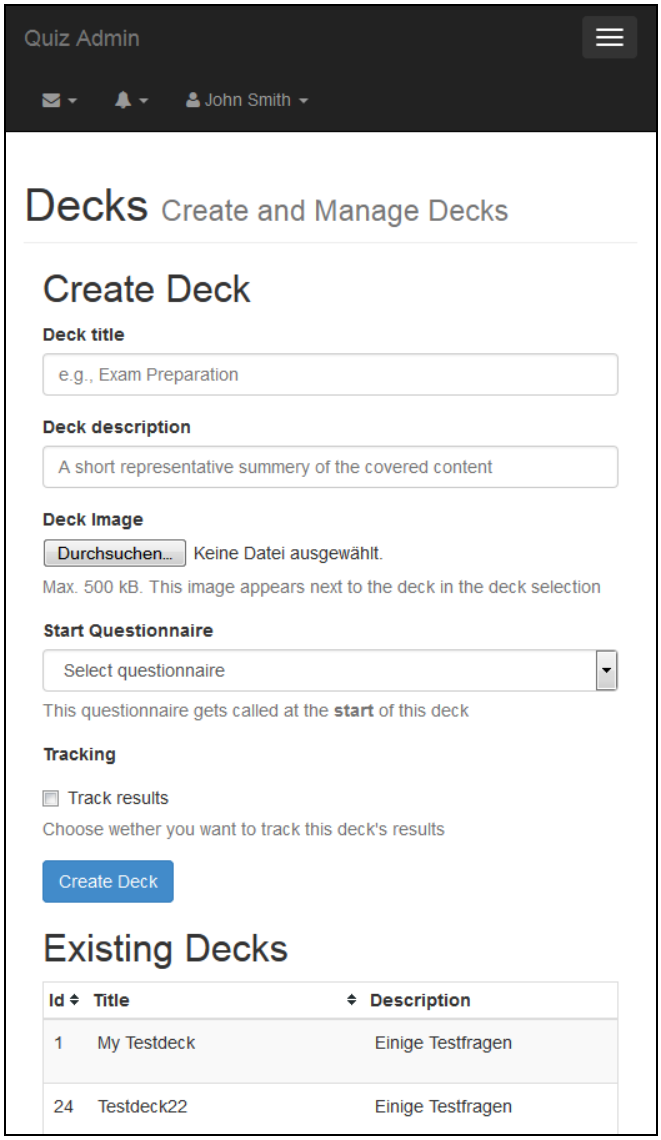

Abb. 7.b: Mobile Ansicht der Deck-Übersicht im Backend

Die Pflege der Daten kann auch auf kleineren Bildschirmen bzw. auf Mobilgeräten erfolgen. Das *Responsive Design* passt sich automatisch an, so dass die Deck-Übersicht auch auf Smartphones adäquat präsentiert wird (Abb. 7.b).

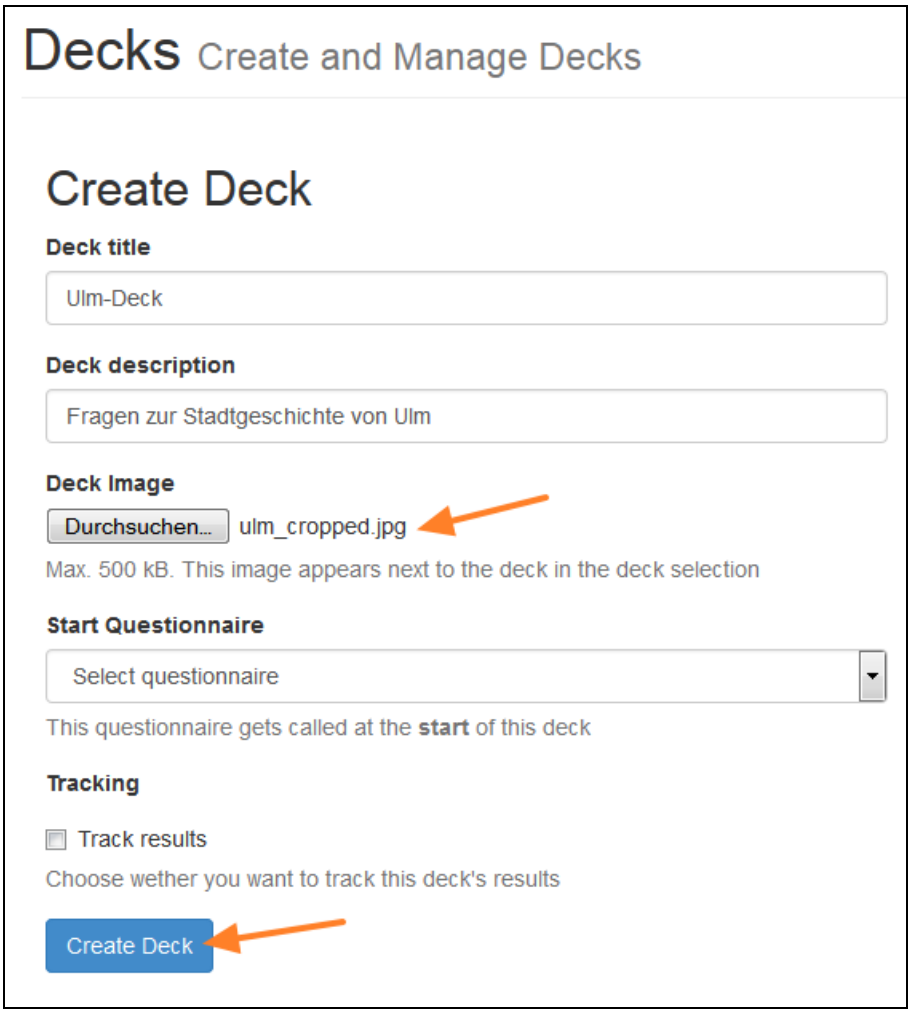

Abb. 7.c: Erstellen eines neuen Decks

Soll ein neues Deck angelegt werden, muss der Autor dafür zunächst einen Titel wählen. Beschreibung und Deck-Image sind optionale Attribute (Abb. 7.c).

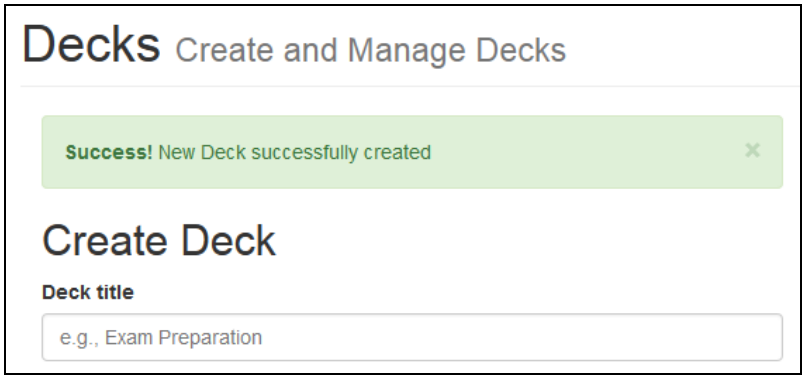

Abb. 7.d: Deck erfolgreich erstellt

Wurde das Deck erfolgreich erstellt, teilt das System dem Autor dies umgehend mit (Abb. 7.d).

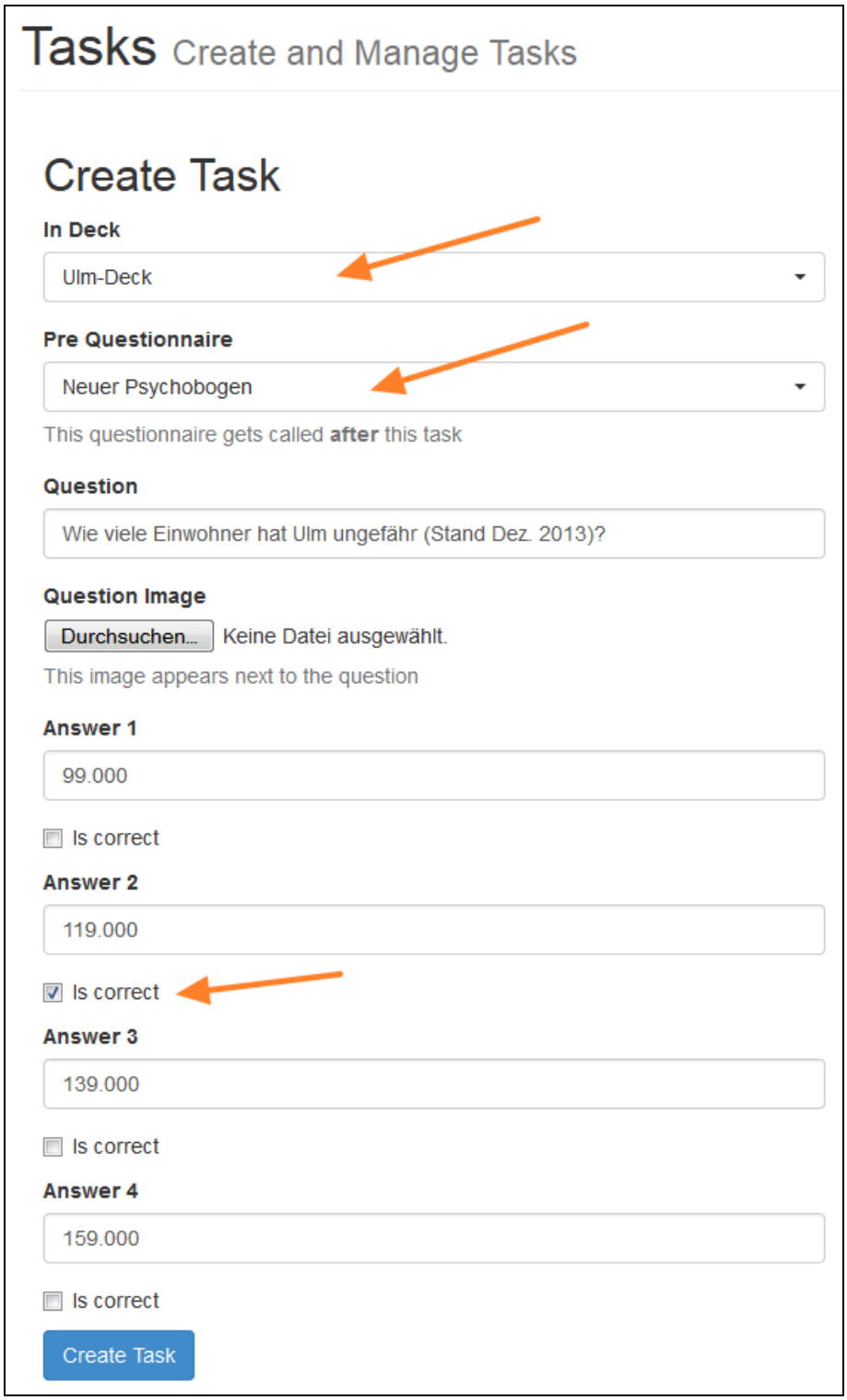

Abb. 7.e: Erstellen eines neuen Tasks

Nun gilt es, einen Task für das neu erstellte Deck zu erzeugen. Die Zuordnung zum gewünschten Deck erfolgt über ein Auswahlmenü. Zudem kann dem Deck ein psychologischer Fragebogen nachgeschaltet werden. Nachfolgend muss der Autor eine Frage und verschiedene Antwortmöglichkeiten formulieren, von denen er eine als korrekt markiert (Abb. 7.e) und kann den Task anschließend erzeugen.

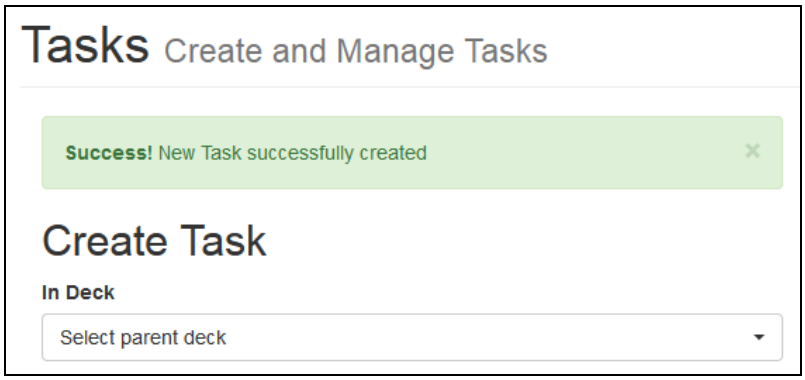

Abb. 7.f: Task erfolgreich erstellt

Auch dieser Schritt wird durch eine Erfolgsmeldung bestätigt, die dem Autor das nötige Feedback vom System liefert.

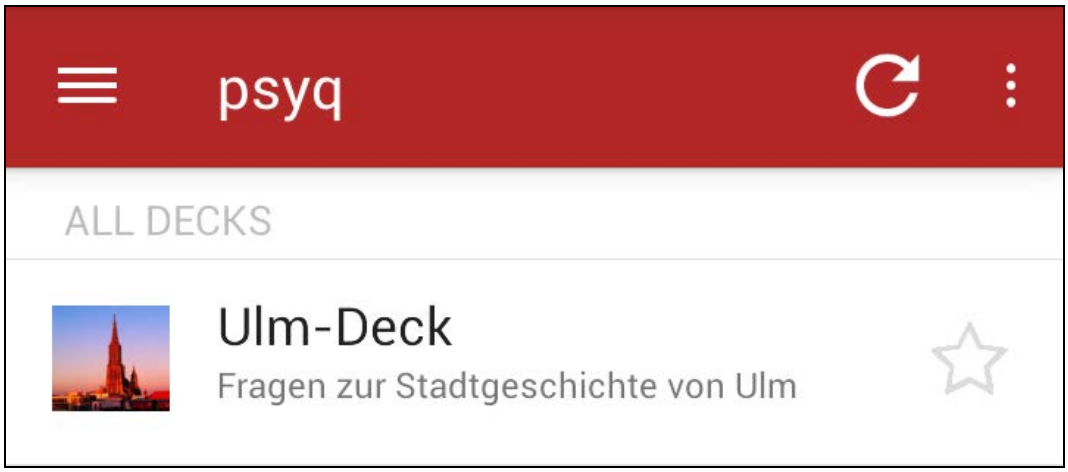

Abb. 7.g: Neu erstelltes Deck in der App

Nach einem Update erscheint das erstellte Deck in der App (Abb. 7.g) und kann beliebig gespielt werden (Abb. 7.h).

Abb. 7.h: Spielen des neu erstellten Decks

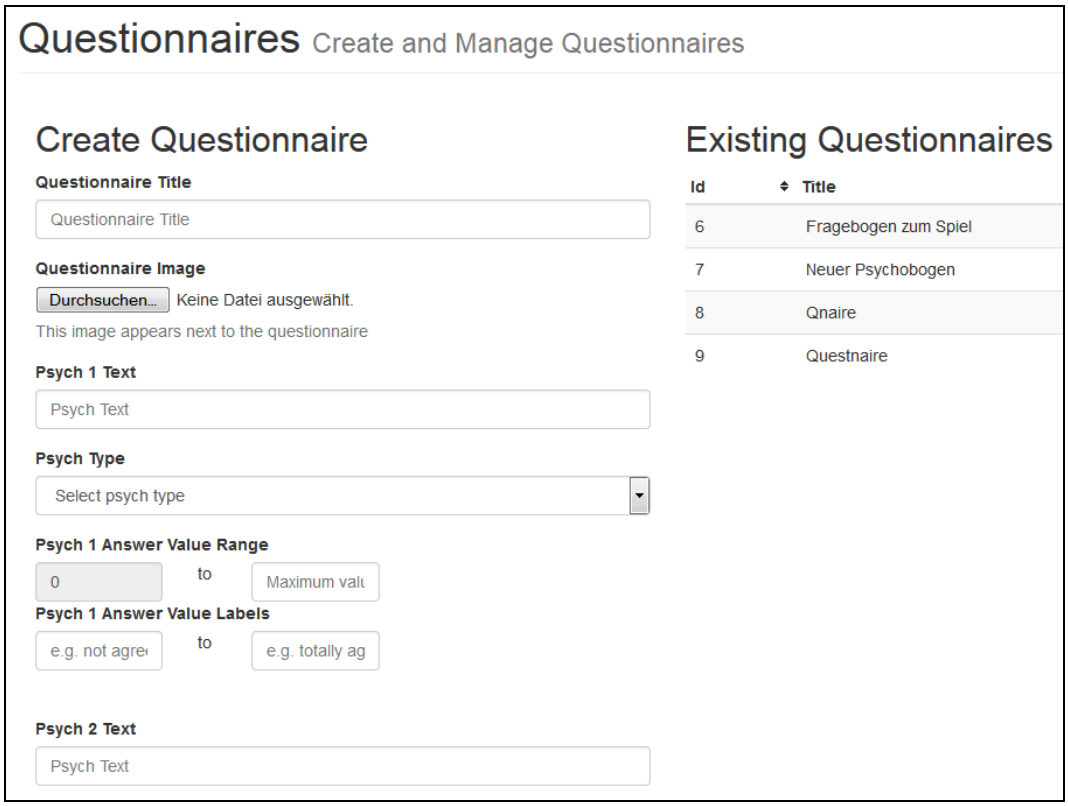

Abb. 7.i: Erstellen eines Questionnaires mit Psychs

Questionnaires können im Backend auf ähnliche Weise wie Decks und Tasks erstellt werden. Hervorzuheben sind hier die Möglichkeiten des Autors, aus den verschiedenen *Psych-Types* zu wählen und die Range-Werte sowie -beschriftungen frei zu bestimmen.

7.2 Szenario 2: Spielen eines Decks mit der App

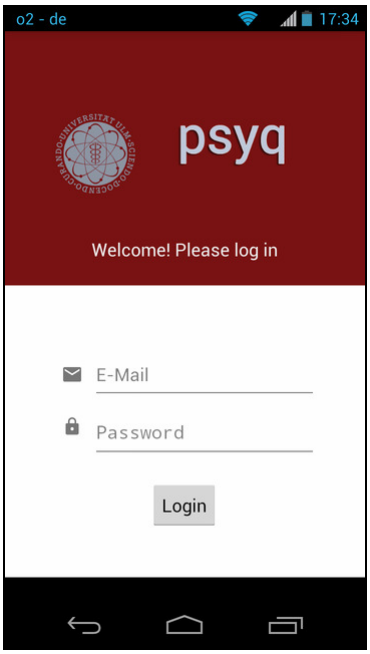

Abb. 7.j: Startbildschirm der App

Nach dem Start der App wird dem Spieler die Login-Ansicht gezeigt (Abb. 7.j).

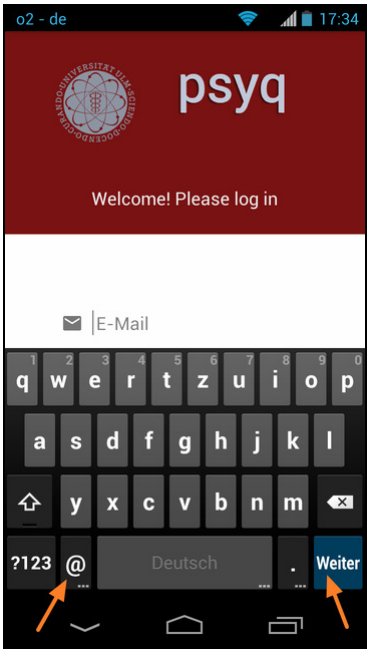

Abb. 7.k: Login-Funktion: E-Mail-Eingabe

Beim Einloggen wird der Spieler unterstützt (Abb. 7.k), indem die Input-Felder einen Hinweis enthalten, die Ansicht beim Klick in das Feld automatisch scrollt und die Tastatur auf eine E-Mail-Adresse vorbereitet ist. Zudem kann mühelos in das nächste Feld navigiert werden.

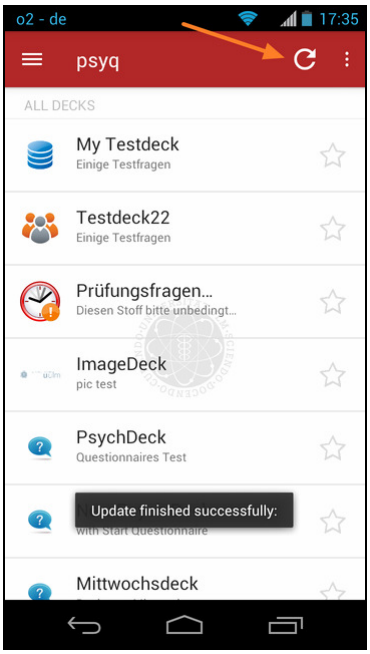

Abb. 7.l: AllDecks-Ansicht direkt nach Update

Nach dem initialen Update erhält die App die nötigen Decks vom Server (Abb. 7.l) und informiert den Spieler darüber. Dieser kann nun ein Deck auswählen, um ein Spiel zu beginnen.

Abb. 7.m: Quiz-Ansicht: ChooseOne

Die im *ChooseOne-Game-Mode* präsentierte Frage (Abb. 7.m) enthält zufällig angeordnete Antwortmöglichkeiten und erwartet eine Auswahl des Spielers.

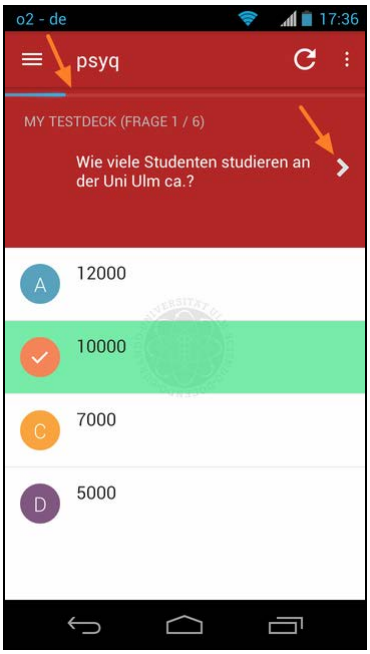

Abb. 7.n: Quiz-Ansicht: ChooseOne: Korrekte Lösung

Hat der Spieler die Frage korrekt beantwortet (Abb. 7.n), so wird dies ebenso hervorgehoben wie der Hinweis auf den nächsten Task. Zudem erhöht sich der visuell dargestellte Fortschritt innerhalb des Decks.

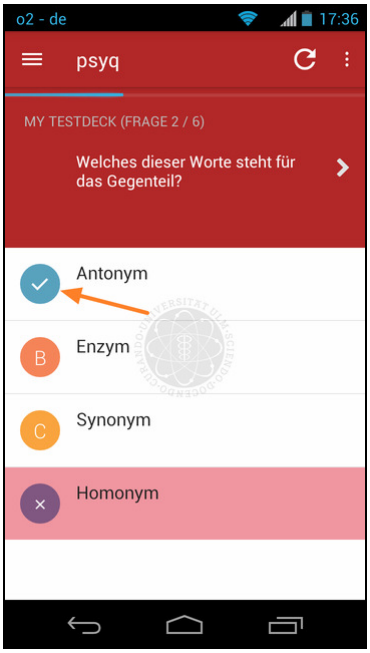

Abb. 7.o: Quiz-Ansicht: ChooseOne: Falsche Lösung

Bei einer falschen Antwort wird der Spieler hingegen nicht nur über diese Tatsache informiert, sondern auch über die Lösung, welche korrekt gewesen wäre (Abb. 7.o).

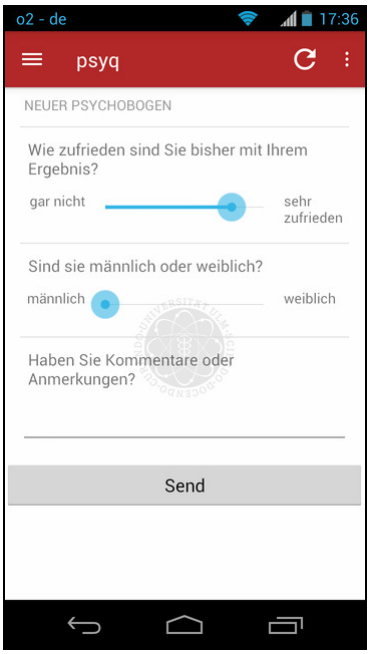

Abb. 7.p: Eingeschobener Fragebogen

Ist ein Fragebogen zwischengeschaltet, wird die entsprechende Ansicht dafür geladen und dynamisch mit den Questionnaire-Inhalten gefüllt. Dann kann der Fragebogen vom Spieler ausgefüllt werden (Abb. 7.p).

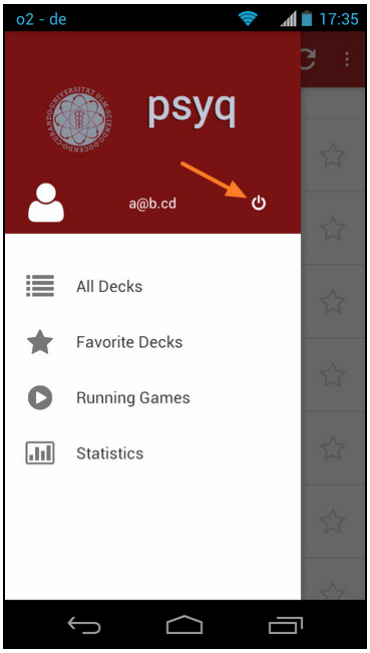

Abb. 7.q: NavigationDrawer mit Menü, Profil-Info und Logout-Funktion

Über das Menü-Icon oder eine Slide-Geste vom linken Bildschirmrand gelangt der Spieler jederzeit in das Menü der App (Abb. 7.q). Dieses beinhaltet die Hauptmenüpunkte, Kurzinformationen zum eingeloggten Spieler und die Logout-Funktionalität.

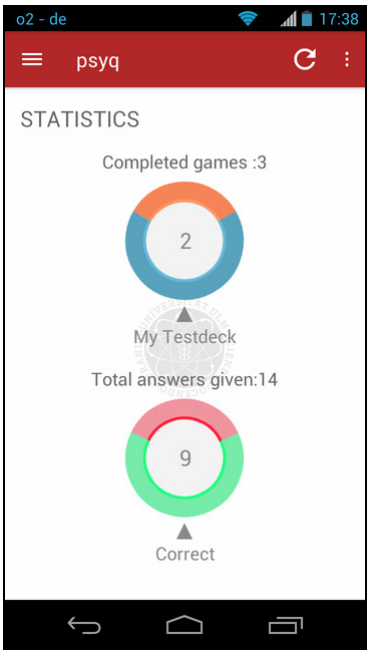

Abb. 7.r: Individuelle Statistik-Ansicht

Möchte der Spieler über seine Resultate informiert werden, kann er diese in der Statistik-Ansicht (Abb. 7.r) betrachten, welche er über das App-Menü erreicht. Hier präsentieren sich die Ergebnisse aller von diesem Spieler beendeten Quizze grafisch aufbereitet.

8 Anforderungsabgleich

Dieses Kapitel dient dem Abgleich und der Analyse der in Kap. 3.2 gesteckten Anforderungen an *psyq*.
Neben einer Beschreibung enthält jede Anforderung einen Verweis, der angibt, ob sie erfüllt wurde. Ist dies der Fall, wird zudem auf das entsprechende Kapitel verwiesen, in dem dies nachzulesen ist.

Tabelle 8.a und Tabelle 8.b geben einen Einblick darüber, wie die gewünschten Anforderungen erreicht wurden.

Nr.	Anforderungsbeschreibung	Zuord-nung	Art		Analyse
			funktional	nicht-fktl.	
1	Anlegen von Spielern, Decks, Tasks, Questionnaires und Psychs im Backend durch Autoren	Backend	x		Anforderung erfüllt (siehe Kap 7.1)
2	Bilder-Upload für Decks, Tasks und Spieler	Backend	x		Anforderung erfüllt (siehe Kap 7.1)
3	Freie Formulierungen für Fragentexte, Antworten, Psych-Texte und Min.-/Max.-Labels durch Autoren	Backend & App	x		Anforderung erfüllt (siehe Kap 7.1)
4	Globale Statistikanzeige im Backend	Backend	x		Anforderung erfüllt (siehe Kap 6.3.1)
5	Plattformunabhängiges Browser-basiertes Backend	Backend	x		Anforderung erfüllt (siehe Kap 5.7.6)
6	Einfache Bedienung des Backends auch für technisch weniger versierte Autroen	Backend		x	Anforderung erfüllt (siehe Kap 7.1)
7	Vielfältige Einstellmöglichkeiten bzgl. Decks und Questionnaires durch Autoren	Backend & App		x	Anforderung erfüllt (siehe Kap 7.1)
8	Flexible Questionnaire-Kopplung an Decks oder Tasks	B'end, DB, API & App	x		Anforderung erfüllt (siehe Kap 5.2)
9	Unterstützung bei der Dateneingabe durch konsistente Bezeichnungen, Hinweise und Beispiel-Vorgaben	Backend		x	Anforderung erfüllt (siehe Kap 7.1)
10	Feingranulare Datenbankstruktur, um Erweiterungen zu ermöglichen	DB & Backend		x	Anforderung erfüllt (siehe Kap 5.2.1)
11	Datenabruf über die API in einem einheitlichen Format für optimale Wiederverwendbarkeit	API	x		Anforderung erfüllt (siehe Kap 6.2.1)
12	Update vom Server liefert der App die aktuellen Daten	API & App	x		Anforderung erfüllt (siehe Kap 6.2.2)
13	Login- und Logout-Funktionalität zur eindeutigen Zuordnung von Spielergebnissen	App & API	x		Anforderung erfüllt (siehe Kap 6.1.3)
14	Starten / Fortsetzen von Decks an der zuletzt aktiven Position nach einer Unterbrechung	App	x		Anforderung erfüllt (siehe Kap 6.1.3)

Tabelle 8.a: Abgleich der Anforderungen - Teil 1

Nr.	Anforderungsbeschreibung	Zuord-nung	Art		Analyse
			funktional	nicht-fktl.	
15	Fortschrittsanzeige, um Spielern eine visuelle Rückmeldung über ihre Position innerhalb des Decks zu geben	App		x	Anforderung erfüllt (siehe Kap 7.2)
16	Antwortmöglichkeiten eines Tasks bei jedem Aufruf erneut zufällig mischen, um den Lerneffekt zu steigern	App	x		Anforderung erfüllt (siehe Kap 7.2)
17	Visuelle Rückmeldung zu korrekten und falschen Antworten	App	x		Anforderung erfüllt (siehe Kap 7.2)
18	Favorisieren von Decks zur besseren Auffindbarkeit	App	x		Anforderung erfüllt (siehe Kap 5.2.2)
19	Markante Anzeige bereits laufender Spiele	App		x	Anforderung erfüllt. (siehe Kap 5.2.2)
20	Spiel-Ende erzwingen, um einheitliche Vergleichsdaten zu erhalten	App	x		Anforderung erfüllt. Spiel wird an voriger Stelle fortgesetzt
21	Resultate der Quizze und der Fragebögen automatisch an den Server übermitteln	App & API	x		Anforderung erfüllt (siehe Kap 5.7.5)
22	Individuelle Statistik innerhalb der App für Spieler	App	x		Anforderung erfüllt (siehe Kap 7.2)
23	Kompatibilität der App mit unterschiedlichsten Android-Versionen und -Geräten	App		x	Anforderung erfüllt (siehe Kap 3.1)
24	Intuitive Bedienung der App und Einhaltung von Design-Richtlinien zur optimal unterstützten App-Bedienung	App		x	Anforderung erfüllt (siehe Kap 6.1.1)
25	Speichern von Nutzer- und Spielspezi-fischen Daten (z. B. Spielfortschritt) über App- und Geräte-Neustart hinweg	App	x		Anforderung erfüllt (siehe Kap 5.2.2. und Kap 6.1.3)
26	Spielerspezifische Datenhaltung, um mehrere Anwender pro Gerät zu ermöglichen	App	x		Anforderung erfüllt (siehe Kap 5.2.2)
27	Flexible modulare Architektur der App zur einfacheren Wartung, Anpassbarkeit und Erweiterbarkeit	App		x	Anforderung erfüllt (siehe Kap 5.4)
28	Einheitliche Verwendung konsistenter Begriffe und stimmige Reaktionen des Systems	App & Backend		x	Anforderung erfüllt (siehe Kap 7)

Tabelle 8.b: Abgleich der Anforderungen - Teil 2

9 Ausblick und Fazit

9.1 Mögliche Erweiterungen

Durch die Konstruktion als flexibles Rahmen-System sind den Inhalten in *psyq* bzgl. Themengebieten, Formulierungen und Umfängen kaum Grenzen gesetzt. Auf funktionaler Ebene gibt es ebenfalls vielfältige Möglichkeiten für Erweiterungen und neue Konzepte.

Dieses Kapitel geht auf Ideen für die Weiterentwicklung und Optimierung ein, welche in dieser Arbeit angedacht und zum Teil auch bereits vorbereitet wurden.

Einige dieser Entwürfe enthalten Hinweise, die sich auf die Architektur oder die Implementierung beziehen. Diese sind in Kap. 5 und Kap. 6 detailliert beschrieben.

9.1.1 Weitere Game-Modes

Ein außerordentlich großes Potential für die Erweiterung und Variation von *psyq* stellen zusätzliche Game-Modes dar.
Die Realisierung eines Multiple-Choice-Tests bietet sich als naheliegende Möglichkeit an, um den Spieler zusätzlich zu fordern.

Weiterhin denkbar ist das Eintippen der richtigen Lösung. Hier sollte jedoch auf eine gewisse Toleranz hinsichtlich Schreibweise, Schreibfehlern etc. geachtet werden.
Auch eine sprachgestützte Version, die Fragen vorlesen oder gesprochene Antworten annehmen kann, ist möglich, bedarf allerdings zusätzlicher Arbeit an der App-Komponente.

Die Serverseite muss für solche zusätzlichen Game-Modes lediglich in sehr überschaubarem Umfang angepasst werden.

Die Programmierung wurde für derartige weiteren Spielvarianten bewusst offen gehalten. So könnten Tasks eine oder mehrere IDs enthalten, die den jeweils möglichen Game-Modes entsprechen. Bei mehreren Möglichkeiten könnte einer der Modes zufällig ausgewählt werden. Jeder implementierte Game-Mode erfordert ein entsprechendes Fragment, das die dafür benötigte Präsentation und Interaktion übernimmt. Im Fall des Multiple-Choice-Modes wäre hier also ein `MultipleChoiceTaskFragment` nötig, das dem implementierten `ChooseOneTaskFragment` ähneln würde.

Je nach Formulierung der Fragen und Wahl der korrekten Antwortmöglichkeiten ist auch ein "Fallback-Game-Mode" vorstellbar, der verwendet wird, wenn der Spieler eine veraltete App-Version benutzt. Auf diese Weise könnte der Anwender neu erschienene Quizze dennoch spielen und so zumindest die Wissens-Inhalte erlernen. Da der gespielte Game-Mode in den Resultaten vermerkt ist, lassen sich die Ergebnisse danach filtern bzw. die Statistiken entsprechend anpassen.

9.1.2 Spielen gegen andere Spieler und Punktevergleich

Um sowohl den sozialen Gedanken zu stärken als auch den spielerischen Ehrgeiz durch Wettbewerb zu erhöhen, ist ein Vergleich mit anderen Spielern vorstellbar. Von der globalen Highscore-Liste innerhalb der App über einen direkten Vergleich mit befreundeten Spielern bis hin zum abwechselnden Spielen derselben Quizze ist hier nahezu jede Art der Wettbewerbsgestaltung vorstellbar.
Während die generelle Bestenliste kein großes technisches – ggf. aber ein datenschutzrechtliches – Problem bildet, stellt das synchrone Spielen mit dauerndem Punkteabgleich eine größere Herausforderung dar. Push-

Nachrichten über Googles *GCM* könnten hierfür einen interessanten Ansatz bieten.

9.1.3 Fragen vorschlagen

Sobald Kunden von Diensten oder Teilnehmer von Umfragen und Quizzes selbst am jeweiligen Projekt mitwirken können, indem sie Verbesserungsvorschläge einbringen und Inhalte generieren, wird sich vermutlich ihr Interesse am Projekt steigern und für den Anbieter ein Mehrwert entstehen [57].
Die Partizipation bringt sowohl Spielern als auch Autoren einen Mehrwert. Auf diese Weise ist das Zustandekommen neuer Inhalte möglich, die zuvor womöglich übersehen, umständlich formuliert oder zu grob behandelt wurden. Aus Spielersicht entstehen hier völlig andere Eindrücke und daher auch alternative Ideen zur Abfrage.

Zudem könnte die Mitarbeit durch virtuelle Belohnungen attraktiver gemacht werden. Möglich wären z. B. die Einführung eines grafisch aufgewerteten Benutzerbildes (Avatar), verschiedenfarbig und hervorgehoben formatierte Spielernamen oder ein zusätzliches Einheitensystem ("Community-Punkte"). Online-Foren-Systeme zeigen bereits seit vielen Jahren, dass diese Form der virtuellen Vergütung gut angenommen wird. Eine Erhöhung der primären Punktezahl ist allerdings fragwürdig, da die Punkte zum Vergleich der Lernleistung gedacht sind und gute Spieler, die sich aber weniger einbringen, dadurch benachteiligt würden.

9.1.4 Nutzung vorhandener Logins

Services wie *OpenID* bieten universelle Accounts, welche von anderen Services als Zugang genutzt werden können. Dafür akzeptieren diese den *OpenID*-Login und ersparen dem Nutzer somit das Anlegen eines weiteren separaten Accounts. Auch der Login über *Google*, *Twitter* oder v. a. *Facebook* ist – insbesondere bei Spielen – eine gern gesehene Alternative zum gesonderten Benutzerkonto. Zudem kann die Verwendung des *Facebook*-Logins weitere Features wie z. B. das mühelose Posten von Ergebnissen oder die automatische Suche nach Nutzern der App innerhalb der Kontaktliste ermöglichen. Allerdings ist jede solche Verknüpfung kritisch bezüglich des Datenschutzes und der eventuell möglichen Datenfusion zu betrachten. Gerade bei sensiblen

Quiz-Inhalten und insbesondere bei persönlichen psychologischen Fragen ist diese Art der Vernetzung nicht uneingeschränkt empfehlenswert.

9.1.5 Antwortzeit

Eine zusätzliche Erweiterungsidee stellt eine zeitliche Limitierung für das Auswählen einer Anwort dar. So könnte eine andere Form der Erinnerung an die korrekte Antwort trainiert werden, als wenn beliebig viel Zeit für die Beantwortung zur Verfügung steht.
Zum einen ist ein *Timeout* vorstellbar, vor dessen Eintreten der Spieler seine Antwort geben muss.
Zum anderen könnte die für die Antworten benötigte Zeit gemessen werden und in die Punkteberechnung einfließen.
Beide Verfahren sind pro Task oder für ein komplettes Deck denkbar.
Weiterhin wäre zu entscheiden, ob der Spieler diese Einschränkung kennen und sie stets wahrnehmen soll, wie z. B. durch einen eingeblendeten Countdown in numerischer oder grafischer Form.

9.1.6 Userspezifische Decks

psyq ist als Rahmen-System konstruiert. Eine Individualisierung enthält typischerweise eine Sammlung von thematisch ähnlichen Decks, je nach den von den Autoren eingepflegten Daten. Alle Spieler dieser Individualisierung erhalten beim folgenden Update genau diese Decks in ihrer App.
Eine mögliche Erweiterung sähe vor, dass bestimmte Decks exklusiv für ausgewählte Nutzer oder Nutzergruppen zur Verfügung gestellt werden. Auf diese Weise könnten fortgeschrittene Inhalte nur fortgeschrittenen Spielern oder thematisch exotische Decks nur den relevanten Spielergruppen präsentiert werden.
Dies könnte durch weniger Einträge in der Deck-Übersicht zu einer aufgeräumteren Benutzeroberfläche sowie zu verringertem Datenverkehr und dadurch zu schnelleren Updates führen.

9.1.7 Gruppierung oder Tagging von Decks

Bei einer Vielzahl von Decks oder einer Menge verschiedener abgedeckter Themengebiete liegt es nahe, Decks zu ordnen, um weiterhin eine übersichtliche Benutzeroberfläche zu gewährleisten.

Dies könnte z. B. über hierarchische Kategorieebenen realisiert werden, die entweder weitere Ebenen oder Decks enthalten können. Um eine Tiefe von mehr als zwei Ebenen zu erreichen, ist die Implementierung verschachtelter `ExpandleListViews` nötig [58], welche allerdings unter Android nicht vorgesehen ist und möglicherweise zu Problemen führt.
Zudem erhöht sich so die Anzahl der notwendigen Schritte für den Spieler, um vom App-Start zum eigentlichen Quiz zu gelangen. Dies verringert die Benutzerfreundlichkeit. Ebenfalls geklärt werden müsste in diesem Fall, ob Decks mehreren (Unter-)Kategorien zugeordnet werden dürfen.

Ein weiterer Ansatz zur Strukturierung der Inhalte sind Schlagwörter ("Tags"). Bei dieser Herangehensweise werden verschiedene Tags angelegt und anschließend den Decks zugewiesen, woraufhin diese nach ihren Tags gefiltert werden können. Die so entstehenden Abfragemöglichkeiten durch Kombination mehrerer (und evtl. auch durch Ausschluss anderer) Tags sind nahezu grenzenlos und weitaus flexibler als das starre Gerüst der festgelegten hierarchischen Kategorisierung. Eine Tag-Cloud könnte häufig vergebene oder beliebte Tags hervorheben und somit als interaktives Filterwerkzeug fungieren.
Allerdings ist eben diese Filterung mit willkürlich vielen Tags auf kompakt dimensionierten Smartphone-Displays eine große Herausforderung. Hier müsste nicht nur die Funktionalität an sich, sondern auch eine gute *Usability* gewährleistet werden. Schon das Anzeigen einer bedienbaren TagCloud zusammen mit den Live-Filterergebnissen stellt eine nicht zu unterschätzende Aufgabe dar. Um hierfür die Menge der anzuzeigenden Tags gering halten zu können, wäre es denkbar, bestimmte Tags ihrerseits zu kategorisieren bzw. eine *XOR*-Bedingung zwischen bestimmten Tags zu definieren.

9.1.8 Suche nach Decks

Das Auffinden eines gewünschten Decks könnte durch eine ausgereifte Suche innerhalb der App optimiert werden. So ist nicht nur die Suche nach Titel, Beschreibung und ID eines Decks denkbar, sondern z. B. auch das Durchsuchen der Fragentexte, um das entsprechende Deck zu finden.
Des Weiteren könnte es langfristig von Vorteil sein, Decks auf dem Server zu durchsuchen. Dies wäre v. a. in einem Szenario interessant, in dem die App-Komponente beim Update nicht mehr alle verfügbaren Decks lädt, sondern der Spieler deren Downloads manuell veranlassen muss.

9.1.9 Teams

Die Gruppierung von Spielern zu Teams kann einen Mehrwert für die einzelnen Spieler bringen. Vorstellbar ist neben dem Punkte-Vergleich innerhalb der Gruppe auch ein Team-Modus. So könnten z. B. Team-Ziele definiert werden, die dann von allen Mitgliedern gemeinsam zu erreichen wären.
Dieses Konzept müsste allerdings nicht nur in Datenbank, Backend und App umgesetzt, sondern auch bei der Berechnung der Statistiken und Auswertungen bedacht und angemessen behandelt werden.

9.1.10 Unterstützung weiterer Plattformen

Wie bereits erwähnt, ist die Architektur der API in keiner Weise an die Plattform *Android* gebunden. So sind Client-Apps auf weiteren mobilen und nichtmobilen Plattformen durchaus denkbar.
Auch eine universelle Web-Anwendung ist theoretisch möglich.

Die lokale Datenhaltung, die Kommunikation der Komponenten etc. ist allerdings nicht immer eins-zu-eins übertragbar. So würde sich die persistente Datenhaltung innerhalb der Web-Anwendung wohl durchaus komplex gestalten. In diesem Zusammenhang sei auch der Gedanke an einen "Slim-Client" genannt, der lediglich die Präsentation der Daten übernimmt und die Eingaben des Nutzers weitergibt, selbst jedoch nicht über den Spielverlauf entscheidet und keine Speicherung von Daten, Berechnung von Statistiken oder anderweitige logische Operationen übernimmt. Ein solcher Client müsste jedoch ständig online sein und der zugehörige Server müsste über eine erhöhte Leistung verfügen.

9.2 Fazit

Das vorliegende App-System *psyq* baut auf der Android-Plattform auf und zeigt, wie spielerische Wissensvermittlung in angemessener Weise mit psychologischen Fragebögen verknüpft werden kann. Besonders im mobilen Kontext sind Aspekte wie geringer Datenverkehr, eine schnelle Koordination der Komponenten und eine strukturierte Oberfläche herauszuheben.
Die Perspektiven, welche die Android-Plattform den App-Entwicklern in diesem Zusammenhang bietet, gestalten sich vielfältig. Darüber hinaus erweitert der

Einsatz von etablierten Programm-Bibliotheken dieses Spektrum noch einmal erheblich. Die immer neuen Ideen von Programmierern und Anwendern ziehen stets zusätzliche Herausforderungen nach sich. Diese erfordern kreative Kombinationen und Modifikationen bewährter Methoden oder gänzlich innovative Ansätze auf technischer Ebene.

Die Planung des Konzepts von *psyq* und die detaillierte Vorbereitung der Entwicklungsphase erforderten einen relativ langen Zeitraum. Diese gründlichen Vorarbeiten erwiesen sich aber als äußert hilfreich für die technische Umsetzung. Es wurde sowohl auf die passende Auswahl der eingesetzten Tools als auch auf deren Zweckmäßigkeit und Flexibilität geachtet. Eine ausführliche Recherche und Analyse verschiedener Lösungsansätze konnte hier negative Überraschungen vermeiden.

Einen Großteil der Konzeptionszeit nahm die Entwicklung des möglichst flexibel strukturierten Aufbaus von *psyq* in Anspruch. Die Umsetzung dieses Ansatzes gestaltete sich teils aufwendig, konnte jedoch zu jeder Zeit durch die konsequente Struktur und die entstehende Modularität überzeugen.

Das so entwickelte Gesamtsystem zeigt anschaulich, dass ein variables Rahmen-System für unterschiedliche App-Individualisierungen eine ideale Lösung für solche Fälle bietet, in denen die Autoren weder ein hohes technisches Verständnis aufweisen noch eine direkte Verbindung zu den Programmierern oder "Technikern" herstellen müssen.

Die Implementierung mit dem Ziel einer möglichst hohen Kompatibilität und dem gleichzeitigen Anspruch, moderne Design-Richtlinien umzusetzen, stellte sich als herausfordernd und lehrreich dar. *psyq* demonstriert aber anschaulich, dass diese Verbindung bei konsequenter Umsetzung erfolgreich möglich ist. Durch den Einsatz anspruchsvoller Techniken und den modularen Aufbau entstand so ein nützliches und vielseitig einsetzbares Gesamtsystem.

Abbildungsverzeichnis

Abb. 3.a: Fragmentierung der Android-Versionen (Stand: 5. Januar 2015) [18] 14
Abb. 3.b: Ausschnitt der *LogCat*-Ansicht, gefiltert nach dem *Tag* "bus" 15
Abb. 3.c *psyq*-Anwendungsfalldiagram 21
Abb. 5.a: Vereinfachte Basis-Architektur von psyq 29
Abb. 5.b Datenstruktur innerhalb von *psyq* 30
Abb. 5.c Datenbankstruktur von *psyq* 32
Abb. 5.d: Speicherung von Rahmendaten - Schritt 1 34
Abb. 5.e: Speicherung von Rahmendaten - Schritt 2 34
Abb. 5.f Informationsfluss: Client-Server-Kommunikation 35
Abb. 5.g Klassen- und Paketstruktur von *psyq* 37
Abb. 5.h Dialogstruktur der App-Komponente 39
Abb. 5.i Performance-Vergleich (Inserts pro Sekunde): Realm, SQLite, OrmLite, greenDAO [29] 42
Abb. 5.j: Erzeugung eines POJOs durch Gson 43
Abb. 5.k Ablauf der Kommunikation über den EventBus [32] 43
Abb. 6.a: Beispiel zu Google-Vorgaben bzgl. einheitlicher Abstände (Angaben in dp) [47] und deren Umsetzung in *psyq* 48
Abb. 6.b: App-Menü via NavigationDrawer 50
Abb. 6.c Unterschiedliche Umsetzung des NavigationDrawers in den Google-Apps [52] 51
Abb. 6.d Vorteile durch die Verwendung von Fragments [55] 53
Abb. 7.a: Übersicht der Decks im Backend 69
Abb. 7.b: Mobile Ansicht der Deck-Übersicht im Backend 70
Abb. 7.c: Erstellen eines neuen Decks 71
Abb. 7.d: Deck erfolgreich erstellt 71
Abb. 7.e: Erstellen eines neuen Tasks 72
Abb. 7.f: Task erfolgreich erstellt 73
Abb. 7.g: Neu erstelltes Deck in der App 73
Abb. 7.h: Spielen des neu erstellten Decks 73
Abb. 7.i: Erstellen eines Questionnaires mit Psychs 74
Abb. 7.j: Startbildschirm der App 75
Abb. 7.k: Login-Funktion: E-Mail-Eingabe 75
Abb. 7.l: AllDecks-Ansicht direkt nach Update 76
Abb. 7.m: Quiz-Ansicht: ChooseOne 76
Abb. 7.n: Quiz-Ansicht: ChooseOne: Korrekte Lösung 77

Abb. 7.o: Quiz-Ansicht: ChooseOne: Falsche Lösung 77
Abb. 7.p: Eingeschobener Fragebogen 78
Abb. 7.q: NavigationDrawer mit Menü, Profil-Info und Logout-Funktion 78
Abb. 7.r: Individuelle Statistik-Ansicht 79

Listingverzeichnis

Listing 6.a: Deaktivierung der ActionBar 47
Listing 6.b: ToolBar als SupportActionBar 47
Listing 6.c: Anpassung des Titelabstands in SupportActionBar 49
Listing 6.d: Transparente Grafik um den geforderten Abstand herzustellen 49
Listing 6.e: Layout activity_main.xml der MainActivity 54
Listing 6.f: Behandlung des GetStoredUserFailedEvents in der MainActivity 54
Listing 6.g: Speicherung des Logins mittels SnappyDB 57
Listing 6.h: Abfragen des User-Objekts 58
Listing 6.i: Konstruktor eines Jobs 59
Listing 6.j: Zu überschreibende Methoden eines Jobs 59
Listing 6.k: Event-Klasse GetTaskFailedEvent 60
Listing 6.l: Schreiben eines Events auf den EventBus 61
Listing 6.m: Activity registriert sich beim EventBus 61
Listing 6.n: Behandlung eines Fehler-Events 61
Listing 6.o: Instanziierung von Realm-Objekten 62
Listing 6.p: Gson-Objekt, welches nicht annotierte Felder der Model-Klasse ignoriert 63
Listing 6.q: Konfiguration zur Verwendung des erstellten Gson-Objekts 63
Listing 6.r: JSON-Antwort des API-Aufrufs getAllDecksComplete 65
Listing 6.s: Setup von RedBeanPHP 67
Listing 6.t: PHP: Erstellen mehrerer Antworten für einen Task 68

Tabellenverzeichnis

Tabelle 3.a: Anforderungen - Teil 1 19
Tabelle 3.b: Anforderungen - Teil 2 20
Tabelle 8.a: Abgleich der Anforderungen - Teil 1 81
Tabelle 8.b: Abgleich der Anforderungen - Teil 2 82

Literaturverzeichnis

[1] Lewandowski, Matt. Help For Choosing Your Mobile Application Analytics Platform. http://www.degdigital.com/blog/Help-For-Choosing-Your-Mobile-Application-Analytics-Platform (zuletzt besucht am 13.01.2015)

[2] Ruf-Leuschner, M. and Pryss, R. and Liebrecht, M. and Schobel, J. and Spyridou, A. and Reichert, M. and Schauer, M.: Preventing further trauma: KINDEX mum screen - assessing and reacting towards psychosocial risk factors in pregnant women with the help of smartphone technologies. In: XIII Congress of European Society of Traumatic Stress Studies (ESTSS) Conference, June 5-9, 2013, Bologna, Italy.

[3] Isele, D. and Ruf-Leuschner, M. and Pryss, R. and Schauer, M. and Reichert, M. and Schobel, J. and Schindler, A. and Elbert, T.: Detecting adverse childhood experiences with a little help from tablet computers. In: XIII Congress of European Society of Traumatic Stress Studies (ESTSS) Conference, June 5-9, 2013, Bologna, Italy.

[4] Schobel, J. and Ruf-Leuschner, M. and Pryss, R. and Reichert, M. and Schickler, M. and Schauer, M. and Weierstall, R. and Isele, D. and Nandi, C. and Elbert, T.: A generic questionnaire framework supporting psychological studies with smartphone technologies. In: XIII Congress of European Society of Traumatic Stress Studies (ESTSS) Conference, June 5-9, 2013, Bologna, Italy.

[5] Crombach, A. and Nandi, C. and Bambonye, M. and Liebrecht, M. and Pryss, R. and Reichert, M. and Elbert, T. and Weierstall, R.: Screening for mental disorders in post-conflict regions using computer apps - a feasibility study from Burundi. In: XIII Congress of European Society of Traumatic Stress Studies (ESTSS) Conference, June 5-9, 2013, Bologna, Italy.

[6] Schlee, W. and Herrmann, J. and Pryss, R. and Reichert, M. and Langguth, B.: How dynamic is the continuous tinnitus percept? In: 11th International Tinnitus Seminar, 21-24 May, Berlin. 2014.

[7] Schlee, W. and Herrmann, J. and Pryss, R. and Reichert, M. and Langguth, B.: Moment-to-moment variability of the auditory phantom perception in chronic tinnitus. In: 13th Int'l Conf on Cochlear Implants and Other Implantable Auditory Technologies, June 18-21, Munich. 2014.

[8] Elmes, Damien. Anki - powerful, intelligent flashcards. http://ankisrs.net/docs/manual.html (zuletzt besucht am 13.01.2015)

[9] Repetico GmbH. Karteikarten online lernen - Karteikarten Software - Repetico. https://www.repetico.de/ (zuletzt besucht am 13.01.2015)

[10] Repetico. Häufig gestellte Fragen zur Karteikarten App Repetico. http://www.karteikartenapp.de/karteikarten-app-repetico-faq/ (zuletzt besucht am 13.01.2015)

[11] Geiger, P. and Schickler, M. and Pryss, R. and Schobel, J. and Reichert, M.: Location-based Mobile Augmented Reality Applications: Challenges, Examples, Lessons Learned. In: 10th Int'l Conference on Web Information Systems and Technologies (WEBIST 2014), Special Session on Business Apps, April 3-5, 2014, Barcelona, Spain.

[12] Pryss, R. and Mundbrod, N. and Langer, D. and Reichert, M.: Supporting medical ward rounds through mobile task and process management. Information Systems and e-Business Management. 2014.

[13] Schobel, J. and Schickler, M. and Pryss, R. and Maier, F. and Reichert, M.: Towards Process-Driven Mobile Data Collection Applications: Requirements, Challenges, Lessons Learned. In: 10th Int'l Conference on Web Information Systems and Technologies (WEBIST 2014), Special Session on Business Apps, April 3-5, 2014, Barcelona, Spain.

[14] Schobel, J. and Schickler, M. and Pryss, R. and Nienhaus, H. and Reichert, M.: Using Vital Sensors in Mobile Healthcare Business Applications: Challenges, Examples, Lessons Learned. In: 9th Int'l Conference on Web Information Systems and Technologies (WEBIST 2013), Special Session on Business Apps, 8 - 10 May 2013, Aachen, Germany.

[15] Pryss, R. and Langer, D. and Reichert, M. and Hallerbach, A.: Mobile Task Management for Medical Ward Rounds - The MEDo Approach. In: 1st Int'l Workshop on Adaptive Case Management (ACM'12), BPM'12 Workshops, 2 September 2012, Tallinn, Estonia.

[16] Pryss, R. and Tiedeken, J. and Kreher, U. and Reichert, M.: Towards Flexible Process Support on Mobile Devices. In: Proc. CAiSE'10 Forum - Information Systems Evolution, June 2010, Hammamet, Tunisia.

[17] Robecke, A. and Pryss, R. and Reichert, M.: DBIScholar: An iPhone Application for Performing Citation Analyses. In: CAiSE Forum-2011, June 22-24, 2011, London, UK.

[18] Google Inc. Dashboards. https://developer.android.com/about/dashboards/index.html (zuletzt besucht am 13.01.2015)

[19] Google Inc. Migrating to Android Studio. http://developer.android.com/sdk/installing/migrate.html (zuletzt besucht am 13.01.2015)

[20] Schrader, C.: Educational Computer Games and Learning: Three Empirical Studies of Games' Impact on Virtual Presence, Cognitive Load and Learning Outcomes. 2012.

[21] Schrader, C. and Bastiaens, Th. J.: Educational computer games and learning: The relationship between design, cognitive load, emotions and outcomes. In: Journal of Interactive Learning Research, 23(3), 251-271. 2012.

[22] Schrader, C. and Bastiaens, Th. J.: The influence of virtual presence: Effects on experienced cognitive load and learning outcomes in educational computer games. In: Journal of Computers in Human Behavior, 28, 648-658. 2012.

[23] Schrader, C. and Bastiaens, Th. J.: Learning in Educational Computer Games for Novices: The Impact of Implementation and Delivery of Support Devices on Virtual Presence, Cognitive Load and Learning Outcomes. In: The International Review of Research in Open and Distance Learning, 13 (3), 206-227. 2012.

[24] Schrader, C. & Bastiaens, Th. J.: Relations between the tendency to invest in virtual presence, actual virtual presence and learning outcomes in educational computer games. In: International Journal of Human-Computer Interaction, 28 (12), 775-783. 2012.

[25] Mooij, Gabor de. RedBeanPHP4. http://redbeanphp.com/ (zuletzt besucht am 13.01.2015)

[26] Otto Mark, Thornton Jacob. Bootstrap. http://getbootstrap.com/ (zuletzt besucht am 13.01.2015)

[27] Google, Inc. Support Library. http://developer.android.com/tools/support-library/index.html (zuletzt besucht am 13.01.2015)

[28] Realm. Realm for Android – Realm: a mobile database that replaces Core Data & SQLite. http://realm.io/news/realm-for-android/ (zuletzt besucht am 13.01.2015)

[29] http://realm.io/assets/benchmarks-android/benchmarks-android.001.png auf http://realm.io/news/realm-for-android/ (zuletzt besucht am 13.01.2015)

[30] Google, Inc. google-gson - A Java library to convert JSON to Java objects and vice-versa. https://code.google.com/p/google-gson/ (zuletzt besucht am 13.01.2015)

[31] greenrobot. EventBus. https://github.com/greenrobot/EventBus (zuletzt besucht am 13.01.2015)

[32] https://raw.githubusercontent.com/greenrobot/EventBus/master/EventBus-Publish-Subscribe.png auf https://github.com/greenrobot/EventBus/blob/master/EventBus-Publish-Subscribe.png (zuletzt besucht am 13.01.2015)

[33] greenrobot. EventBus Comparison. https://github.com/greenrobot/EventBus/blob/master/COMPARISON.md (zuletzt besucht am 13.01.2015)

[34] Path, Inc. Android Priority Job Queue (Job Manager). https://github.com/path/android-priority-jobqueue (zuletzt besucht am 13.01.2015)

[35] Jake Wharton. Butter Knife. http://jakewharton.github.io/butterknife/ (zuletzt besucht am 13.01.2015)

[36] Jake Wharton. NineOldAndroids. https://github.com/JakeWharton/NineOldAndroids/ (zuletzt besucht am 13.01.2015)

[37] Nabil Hachicha. SnappyDB. https://github.com/nhachicha/SnappyDB (zuletzt besucht am 13.01.2015)

[38] EsotericSoftware. Kryo. https://github.com/EsotericSoftware/kryo (zuletzt besucht am 13.01.2015)

[39] Square, Inc. Retrofit. https://github.com/square/retrofit (zuletzt besucht am 13.01.2015)

[40] Square, Inc. OkHttp. https://github.com/square/okhttp (zuletzt besucht am 13.01.2015)

[41] Google, Inc. HttpURLConnectionImpl.java. https://android.googlesource.com/platform/external/okhttp/+/b76bc42e8b906eb1d11f3ed3468ecde454891fe8/src/main/java/com/squareup/okhttp/internal/http/HttpURLConnectionImpl.java (zuletzt besucht am 13.01.2015)

[42] Tarasevich, Sergey. Universal Image Loader. https://github.com/nostra13/Android-Universal-Image-Loader (zuletzt besucht am 13.01.2015)

[43] Khare, Amulya. TextDrawable. https://github.com/amulyakhare/TextDrawable (zuletzt besucht am 13.01.2015)

[44] Cech, Paul. EazeGraph. https://github.com/blackfizz/EazeGraph (zuletzt besucht am 13.01.2015)

[45] Google, Inc. Design. http://developer.android.com/design/index.html (zuletzt besucht am 13.01.2015)

[46] Nurik, Roman. Material Design on Android Checklist. http://android-developers.blogspot.de/2014/10/material-design-on-android-checklist.html (zuletzt besucht am 13.01.2015)

[47] http://material-design.storage.googleapis.com/publish/v_2/material_ext_publish/0Bx4BSt6jniD7TVRfaDhNM1luSU0/layout_metrics_keylines_mobile11.png auf http://www.google.com/design/spec/layout/metrics-keylines.html#metrics-keylines-keylines-spacing (zuletzt besucht am 13.01.2015)

[48] Google, Inc. Navigation Drawer. http://developer.android.com/design/patterns/navigation-drawer.html (zuletzt besucht am 13.01.2015)

[49] Kilian, A.: Metadaten-basierter Aufgabenplaner "meTask" für den mobilen Einsatz am Beispiel des Android Frameworks. Bachelorarbeit, Universität Ulm. 2011.

[50] Google, Inc. Navigation drawer - Patterns. http://www.google.com/design/spec/patterns/navigation-drawer.html (zuletzt besucht am 13.01.2015)

[51] Spradlin, Liam. The Many Faces Of Google's Hamburger Navigation Drawer. http://www.androidpolice.com/2014/10/30/google-turns-design-inconsistency-ten-latest-round-navigation-drawers/ (zuletzt besucht am 13.01.2015)

[52] Screenshot mehrerer Bilder auf [51].

[53] Whitwam, Ryan. Play Store v5.1.11 Rolling Out With Full-Height Navigation Drawer, New My Account Page With Order History, And More. http://www.androidpolice.com/2014/12/04/play-store-v5-1-11-rolling-out-with-full-height-navigation-drawer-new-my-account-page-and-more-apk-download/ (zuletzt besucht am 13.01.2015)

[54] Google, Inc. Supporting Multiple Screens. http://developer.android.com/guide/practices/screens_support.html#NewQualifiers (zuletzt besucht am 13.01.2015)

[55] http://developer.android.com/images/fundamentals/fragments.png auf http://developer.android.com/guide/components/fragments.html (zuletzt besucht am 13.01.2015)

[56] Realm. Realm Java 0.76. http://realm.io/news/realm-java-0.76/ (zuletzt besucht am 13.01.2015)

[57] Wecht, C.: Frühe aktive Kundenintegration in den Innovationsprozess. Dissertation, Universität St. Gallen. 2006.

[58] Google, Inc. ExpandableListView. http://developer.android.com/reference/android/widget/ExpandableListView.html (zuletzt besucht am 13.01.2015)

Glossar

- **Backend**
  Eingabeoberfläche zur Pflege des Inhalts; oft Browser-basiert mit Formularfeldern.
- **ChooseOne**
  Mit *psyq* eingeführter Begriff; beschreibt den Spielmodus, bei dem mehrere Antwortmöglichkeiten vorgegeben sind, von denen eine korrekt ist.
- **Featurephone**
  Erweiterte Mobiltelefone mit Basis-Funktionalitäten bzgl. Multimedia und Social Networks; Internetzugang, aber keine nachträglichen Apps.
- **Flag**
  Boolesche Variable; kann den Zustand 0 oder 1 annehmen und einen ja-oder-nein-Status repräsentieren, wie z. B. `isCorrect` im Zusammenhang mit einer *Answer*.
- **Flashcard**
  Beschreibt eine digitale Karteikarte, auf deren einer Seite sich ein Begriff, eine Fragestellung, ein Bild o. Ä. befindet, während die andere Seite die Lösung, Erklärung oder Übersetzung darstellt.
- **Overhead**
  Fällt an, wenn Nutzdaten transportiert oder verarbeitet werden und beschreibt die dafür zusätzlichen Ressourcen hinsichtlich der Verwaltung, z. B. zusätzlich benötigter Speicher oder eine höhere Bandbreite. Je länger z. B. die Tag-Bezeichnung in einem XML-Dokument ist, desto größer wird - bei unveränderten Nutzdaten – der erzeugte Overhead.
- **Rahmen-System**
  Steht für ein System, welches das passende technische und organisatorische Gerüst bietet, um eine bestimmte Funktionalität zu bieten – unabhängig vom nachträglich eingepflegten Inhalt.
- **Tap / Touch-Tap**
  Beschreibt die Eingabe auf Touchscreens durch kurzes Berühren eines bestimmten Bereiches (entgegen der Geste). Entspricht einem Mausklick auf klassischen Computersystemen und wird im Emulator auch so imitiert.

CD-Inhalt

Auf der beiliegenden CD-ROM befinden sich folgende Daten

- psyq (Quellcode inkl. Ordnerstruktur, Grafiken, ausführbare APK-Datei, ...)
- Diese Dokumentation (PDF)
- In dieser Dokumentation verwendete Grafiken und Screenshots (verschiedene Formate)
- Ursprüngliches Mockup (PDF)
- Verknüpfungen zu Online-Quellen

Erklärung

Name: Andreas Kilian

Matrikelnummer: 650858

Hiermit erkläre ich, dass ich diese Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Ulm, den ....

Andreas Kilian