



ulm university universität  
**uulm**

**Universität Ulm** | 89069 Ulm | Germany

**Fakultät für  
Ingenieurwissenschaften,  
Informatik und  
Psychologie**  
Institut für Datenbanken  
und Informationssysteme

# Mobile Application Lab

Ausarbeitung zur App an der Universität Ulm

**Vorgelegt von:**

Fabian Fischbach, Luis Beaucamp und Tim Stenzel

**Gutachter:**

Marc Schickler

**Betreuer:**

Marc Schickler

2017

Fassung 31. März 2017

© 2017 Fabian Fischbach, Luis Beaucamp und Tim Stenzel

This work is licensed under the Creative Commons. Attribution-NonCommercial-ShareAlike 3.0 License. To view a copy of this license, visit <http://creativecommons.org/licenses/by-nc-sa/3.0/de/> or send a letter to Creative Commons, 543 Howard Street, 5th Floor, San Francisco, California, 94105, USA.

Satz: PDF- $\text{\LaTeX}$  2 $\epsilon$

# Inhaltsverzeichnis

<b>1</b>	<b>Einleitung</b>	<b>1</b>
1.1	Motivation und Problemstellung . . . . .	1
1.2	Zielsetzung . . . . .	1
1.3	Struktur der Arbeit . . . . .	2
<b>2</b>	<b>Grundlagen</b>	<b>3</b>
2.1	Quartettspiel . . . . .	3
2.2	Mobile Plattform . . . . .	4
2.3	Frameworks . . . . .	5
<b>3</b>	<b>Anforderungsanalyse</b>	<b>7</b>
3.1	Funktionale Anforderungen . . . . .	7
3.2	Nichtfunktionale Anforderungen . . . . .	10
<b>4</b>	<b>Konzept, Entwurf und Implementierung</b>	<b>12</b>
4.1	Implementierungsdetails . . . . .	12
4.2	Besonderheiten . . . . .	18
4.2.1	Deckcreator und -editor . . . . .	19
4.2.2	Down- und Upload von Decks . . . . .	21
4.3	Architektur . . . . .	24
4.3.1	Datenmodell und -speicherung . . . . .	24
4.3.2	Klassen- und Activity-Architektur . . . . .	25
4.3.3	Arbeitsaufteilung . . . . .	26
4.4	Schwierigkeiten während der Implementierung . . . . .	26
<b>5</b>	<b>Anforderungsabgleich</b>	<b>28</b>
5.1	Funktionale Anforderungen . . . . .	29
5.2	Nichtfunktionale Anforderungen . . . . .	31
<b>6</b>	<b>Zusammenfassung und Ausblick</b>	<b>32</b>

# 1

## Einleitung

In dieser Dokumentation wird die Entwicklung einer App im Rahmen des Anwendungsfaches Mobile Application Lab an der Universität Ulm und die dadurch erstellte App in Form einer Quartett App vorgestellt.

### 1.1 Motivation und Problemstellung

Die Problemstellung wurde uns im Rahmen dieses Projektes schon gegeben, da wir uns auf die Entwicklung einer Quartett App für Smartphones konzentrieren sollten. Das Prinzip des beliebten Kartenspiels soll für ein Smartphone entwickelt werden und so zu jeder Zeit und an jedem Ort auch ganz ohne physische Karten spielbar sein.

Beim Anschauen des aktuellen Marktes für Quartett Apps fällt schnell die Vielzahl an verschiedenen Apps auf. Diese erweisen nach genauerem Betrachten jedoch erhebliche Mängel auf. So sind manche von ihnen sehr veraltet und funktionieren nicht mehr richtig auf neueren Smartphone Modellen. Auch entsprechen diese inhaltlich nicht unseren Vorstellungen von einer guten Quartett App. Sie sind sehr beschränkt, was die verschiedenen Spielmodi angeht und beschränken sich meistens auf ein einziges Kartendeck oder Decks aus einem Themengebiet.

### 1.2 Zielsetzung

Da auf dem Markt eine Nachfrage besteht wollen wir diese ausnutzen und eine eigene Quartett Anwendung erstellen. Diese wollen wir auf Basis von Android und Java ent-

## 1 Einleitung

wickeln. Dabei geht es uns primär darum, den Umgang mit den neuen Techniken zu erlernen und eine Grunderfahrung im Programmieren von Android Anwendungen zu erlangen, sodass wir diese nach Abschließen des Projektes beherrschen können.

Inhaltlich möchten wir eine Quartett App entwickeln, die sich als Singleplayer wie ein richtiges Quartett spielen lässt. Sie soll verschiedene Spielmodi haben, welche frei konfigurierbar sein sollen. Zudem soll die Auswahl an Decks breit gefächert sein, was durch einen Deckcreator und einer Onlinefunktion zum Up- und Downloaden realisiert werden soll. Der Anwendung soll zudem die Möglichkeit bieten, alle Karten anzugucken und laufende Spiele zu unterbrechen. Dabei soll die App benutzerfreundlich sein und schön aussehen, sowie auf den neusten aber auch auf älteren Android Smartphones lauffähig sein.

### 1.3 Struktur der Arbeit

In dieser Dokumentation werden zuerst einmal grundlegend die Quartett Spielregeln erklärt sowie Android vorgestellt und unsere verwendeten Frameworks präsentiert. Den Hauptteil bildet die Implementierung, welche unsere App im Allgemeinen und mit ihren Besonderheiten vorstellt. Außerdem wird darin unsere Architektur gezeigt und wir erläutern Schwierigkeiten, die wir während der Implementierungsphase hatten. Abschließend gibt es einen Anforderungsabgleich und einen Ausblick auf die Zukunft des Projektes.



Abb. 1.1: Quartett42 Logo

# 2

## Grundlagen

### 2.1 Quartettspiel

Zum Quartettspielen sind natürlich einige Regeln notwendig, die im Folgenden erklärt werden.

Gespielt wird Eins gegen Eins, Spieler gegen Computer. Zuerst wählt der Spieler den Schwierigkeitsgrad (leicht, mittel oder schwer) des Computers, dann den Spielmodus und das Limit für das Spielende (Zeit-, Runden- oder Punktemodus und entsprechend die Spielzeit, Runden- oder Punkteanzahl). Danach wird ein Deck gewählt, gemischt, Computer und Spieler bekommen jeweils die Hälfte der Karten verdeckt auf einem Stapel, bei dem immer nur die oberste Karte sichtbar ist und es wird zufällig bestimmt wer mit dem ersten Zug beginnen darf. Ein Zug sieht immer folgendermaßen aus: Der Spieler, der am Zug ist, wählt ein Attribut (Beispiel: in einem Autoquartett die max. Geschwindigkeit) und nennt den entsprechenden Wert. Der andere Spieler nennt nun ebenfalls seinen Wert (Beispiel von oben: max. Geschwindigkeit) und die Werte werden verglichen. Für jedes Attribut wurde vor einem Spiel festgelegt ob für dieses ein höherer oder niedrigerer Wert gewinnt. Der Vergleich wird also durchgeführt. Der Spieler mit dem besseren Wert gewinnt den Vergleich, bekommt beide Karten der Runde unter seinen Stapel und darf im nächsten Vergleich das Attribut wählen. Bei einem Unentschieden behält jeder seine Karten, legt sie unter seinen Stapel und der Spieler, der das Attribut gewählt hat, wählt auch das Nächste. Das Spiel ist zu Ende, wenn entweder ein Spieler keine Karten mehr auf seinem Stapel hat oder das Limit des gewählten Spielmodus erreicht ist (Zeit abgelaufen, alle Runden ausgespielt oder Punktelimit erreicht).

## 2 Grundlagen

Dies sind die normalen Regeln und Spielmodi für ein Quartettspiel. In unserer App gibt es aber zusätzlich neben dem normalen Modus (bei dem jeweils, wie festgelegt, der höhere oder niedrigere Wert den Vergleich gewinnt) auch noch den sog. Insanemodus, bei dem jeweils nicht der vorher festgelegte höhere oder niedrigere Wert gewinnt, sondern genau umgekehrt. Damit es im Spiel aber nicht zu Verwirrungen kommt ist immer neben jedem Attribut ein Pfeil, der angibt, ob höher oder niedriger besser ist. Egal ob der normale Modus oder Insanemodus gewählt wird, kann man sich noch zusätzlich entscheiden, ob man den Expertenmodus spielen möchte oder nicht. Der Expertenmodus ist aber nichts für Anfänger, denn es werden die Werte einer Karte durch '?' ersetzt, sodass man das Deck bzw. die Karten schon ein bisschen besser kennen muss um hier erfolgreich zu sein.

### 2.2 Mobile Plattform

Android ist ein mobiles Betriebssystem, also für Smartphones und Tablets, das von Google entwickelt wurde und auf Linux basiert. Die App-Entwicklung ist geprägt durch einzelne Aktivitäten (ein angezeigter Screen ist eine Aktivität), die miteinander kommunizieren und in ihrer 'Lebenszeit' ein vorgegebenes Zustandsmodell 2.1 durchlaufen.

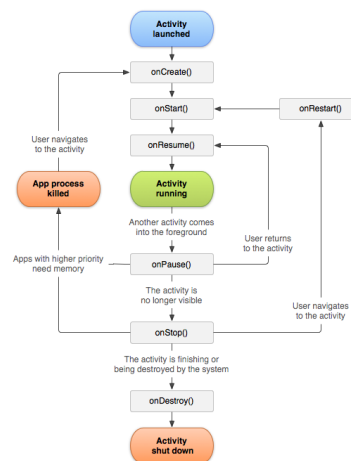


Abb. 2.1: Android Zustandsmodell<sup>1</sup>

<sup>1</sup><http://www.javatpoint.com/images/androidimages/Android-Activity-Lifecycle.png>

## 2 Grundlagen

Dieses Zustandmodell ist auch anfangs einer der Nachteile von Android, da es nicht so leicht zu verstehen ist und uns auch ein paar Probleme bereitet hat. Nachdem wir uns aber im Laufe der App-Entwicklung immer mehr mit Android vertraut gemacht haben, war auch das Modell kein Problem mehr, sondern eher ein Vorteil, da es sehr logisch und durchdacht ist. Eine weitere Schwierigkeit, die während der Entwicklung aufgetreten ist, ist, dass es so viele verschiedene Android-Versionen und Geräte gibt. Da wir unsere App für so viele Versionen wie möglichen entwickeln wollten, kamen deshalb auch mal das ein oder andere Problem auf, wie z. B. dass manche Libraries oder Frameworks erst ab bestimmten Versionen verfügbar sind oder die vielen verschiedenen Geräte alle unterschiedlichen Seiten- und Größenverhältnisse haben.

Die Vorteile von Android überwiegen aber auf jeden Fall, vor allem, wenn man sich damit tiefer beschäftigt und eingearbeitet hat. Einer der größten Vorteile ist die sehr gute Dokumentation von Android durch die das Einlesen in die Möglichkeiten und Funktionen recht leicht ist. Auch die weltweite Verbreitung und Beliebtheit von Android ist hier ein Vorteil, da es sehr viele Entwickler gibt und so jedes Problem schon einmal aufgetreten ist und daher auch zu den allermeisten auch Lösungen oder Workarounds bekannt sind. Außerdem wird Android stetig Weiterentwickelt, weshalb immer mehr möglich ist und der Umgang mit bestimmten Funktionen, wie z. B. der Zugriff auf Gerätefunktionen wie Kamera oder Galerie immer leichter wird. Weitere Vorteile für uns sind die Vertrautheit mit Java und die Einfachheit von Android Studio.

### 2.3 Frameworks

Frameworks sind eine Art Gerüst oder Rahmen, die eingesetzt werden um das Programmieren zu vereinfachen und die geschriebenen Zeilen zu verringern.

Wir haben in unserer App drei Frameworks als Hilfen genutzt:

- Picasso<sup>2</sup>: Erlaubt einfaches Handling (z. B. Größentransformationen) von Bildern in oftmals einer Zeile

---

<sup>2</sup><http://square.github.io/picasso/>



## 2 Grundlagen

- MPAndroidCharts<sup>3</sup>: Ermöglicht die Erstellung von Diagrammen (in unserem Fall Tortendiagramme für die Statistiken)
- Floating Action Button<sup>4</sup>: Ein fancy Menü, das schön ein- und ausgeklappt werden kann

---

<sup>3</sup><https://github.com/PhilJay/MPAndroidChart>

<sup>4</sup><https://github.com/Clans/FloatingActionButton>

# 3

## Anforderungsanalyse

In diesem Kapitel werden die ursprünglich definierten funktionalen und nichtfunktionalen Anforderungen tabellarisch dargestellt. Jede der Anforderungen hat einen eindeutigen Identifikator (ID), einen Titel (TITEL), und eine Beschreibung (BES).

### 3.1 Funktionale Anforderungen

<b>ID:</b>	<b>FA1</b>
<b>TITEL:</b>	Startmenü
<b>BES:</b>	Nach dem Start der Anwendung sieht der Benutzer ein Startmenü mit den Einträgen „Spiel starten“, „Einstellungen“, „Spielregeln“, „Deckübersicht“ („Rangliste“), („Infos“)

<b>ID:</b>	<b>FA2</b>
<b>TITEL:</b>	Einstellungen
<b>BES:</b>	Es gibt eine Möglichkeit, die Spieleinstellungen zu ändern. Diese umfassen Schwierigkeitsgrad, Soundeffekte (an/aus) und Spielmodus (rundenbasiert/-zeitbasiert/Kartenbasiert)

<b>ID:</b>	<b>FA3</b>
<b>TITEL:</b>	Spielregeln
<b>BES:</b>	Es gibt eine Möglichkeit, die Spielregeln anzuzeigen.

### 3 Anforderungsanalyse

<b>ID:</b>	<b>FA4</b>
<b>TITEL:</b>	Infoseite
<b>BES:</b>	Es gibt eine Möglichkeit, eine Infoseite (mit Copyright- und weiteren Informationen) anzuzeigen

<b>ID:</b>	<b>FA5</b>
<b>TITEL:</b>	Deckübersicht
<b>BES:</b>	Der Benutzer hat die Möglichkeit eine Liste mit allen verfügbaren Decks anzuzeigen. Beim Auswählen eines Decks kann der Benutzer durch die Karten des Decks scrollen. Dabei sieht er bei jeder Karte das Bild und die Attribute. Außerdem kann der Benutzer auf der Deckübersichts-Seite neue Decks hinzufügen. Diese Decks dienen als „Erweiterung“ und der Benutzer kann sie sich herunterladen.

<b>ID:</b>	<b>FA6</b>
<b>TITEL:</b>	Spiel starten
<b>BES:</b>	Der Benutzer hat vom Startmenü aus die Möglichkeit, das Spiel zu starten. Dafür öffnet sich ein Dialog, auf dem der Benutzer das Deck auswählt und festlegt, ob er gegen einen anderen Spieler oder gegen einen Computer spielen will. Danach werden die Karten auf die beiden Spieler verteilt und es wird zufällig bestimmt, welcher Spieler beginnt.

<b>ID:</b>	<b>FA7</b>
<b>TITEL:</b>	Spielablauf
<b>BES:</b>	Während des Spiels sieht der Spieler pro Runde nur seine „oberste Karte im Stapel“. In einer Runde werden die Werte des erstgewählten Attributs verglichen. Nach Auswahl eines Attributs vom ersten Spieler werden die zu vergleichenden Werte beider Spieler angezeigt und das gewinnende Attribut markiert. Bei einem Gleichstand werden die Karten beider Spieler unter den jeweils eigenen Stapel gelegt.

### 3 Anforderungsanalyse

<b>ID:</b>	<b>FA8</b>
<b>TITEL:</b>	Spielstand anzeigen
<b>BES:</b>	Während des Spiels wird dauerhaft der Spielstand (Anzahl Karten Spieler 1 : Anzahl Karten Spieler 2) angezeigt.

<b>ID:</b>	<b>FA9</b>
<b>TITEL:</b>	Spielzeit anzeigen
<b>BES:</b>	Beim zeitbasierten Modus wird neben dem Spielstand auch die verbleibende Rundenzeit und die verbleibende Gesamt-Spielzeit angezeigt.

<b>ID:</b>	<b>FA10</b>
<b>TITEL:</b>	Spielende
<b>BES:</b>	<p>Das Spiel ist vorbei, wenn</p> <ul style="list-style-type: none"><li>• [alle Modi] ein Spieler alle Karten hat (Dieser Spieler hat gewonnen)</li><li>• [Zeitbasiert] die Zeit abgelaufen ist (Der Spieler mit den meisten Karten hat gewonnen)</li><li>• [Rundenbasiert] die vorher ausgewählte Anzahl an Runden gespielt worden sind (Spieler mit den meisten Karten hat gewonnen)</li></ul>

### 3 Anforderungsanalyse

<b>ID:</b>	<b>FA11</b>
<b>TITEL:</b>	Speicherung der Daten
<b>BES:</b>	<p>Die Daten des Spiels (Decks, Karten und ihre Attribute) werden in einer Datenbank gespeichert. Dabei gilt für jedes Deck:</p> <ul style="list-style-type: none"><li>• Die Anzahl an Karten soll eine gerade Zahl zwischen 16 und 64 sein.</li><li>• Die Anzahl an Attributen soll eine Zahl zwischen 5 und 10 sein.</li><li>• Für jedes Attribut wird spezifiziert, ob ein höherer Wert oder ein niedriger Wert gewinnt.</li></ul>

## 3.2 Nichtfunktionale Anforderungen

<b>ID:</b>	<b>NFA1</b>
<b>TITEL:</b>	Entwicklungssprache und -Umgebung
<b>BES:</b>	Die Anwendung wird in Java mit der Entwicklungsumgebung „Android Studio“ entwickelt und soll auf Geräten mit Betriebssystemen ab Android [4.1] funktionieren.

<b>ID:</b>	<b>NFA2</b>
<b>TITEL:</b>	Reaktionszeit
<b>BES:</b>	Die Reaktionszeit der Anwendung soll zu jeder Zeit maximal 1,5 Sekunden betragen

<b>ID:</b>	<b>NFA3</b>
<b>TITEL:</b>	Persistenter Spielzustand
<b>BES:</b>	Während eines Spiels soll die App minimiert werden können (z.B. durch den Home Button), ohne dass der Spielzustand verloren geht. D.h., ein Spiel soll bei erneutem Öffnen fortgesetzt werden können.

### 3 Anforderungsanalyse

<b>ID:</b>	<b>NFA4</b>
<b>TITEL:</b>	Benutzbarkeit
<b>BES:</b>	Die Anwendung soll intuitiv bedienbar sein. D.h., die Buttons und andere Auswahlmöglichkeiten sollen eine ausreichende Größe haben und wie erwartet reagieren.

<b>ID:</b>	<b>NFA5</b>
<b>TITEL:</b>	Offline-Modus
<b>BES:</b>	Die Anwendung soll stets auch das Spielen ohne Internetanbindung ermöglichen.

# 4

## Konzept, Entwurf und Implementierung

### 4.1 Implementierungsdetails

Wir erklären in diesem Kapitel die einzelnen Funktionen unserer App genauer, unterstützen sie an Hand von Screenshots und zeigen vergleichend auch die Mockups die zu Beginn des Projekts erstellt wurden, um dem Kunden seine Vorstellungen visualisiert präsentieren zu können.

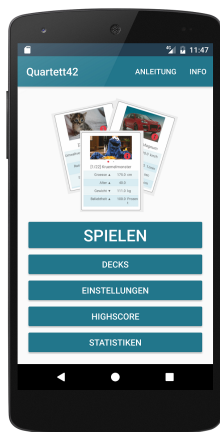


Abb. 4.1: Hauptmenü der App



Abb. 4.2: Mockup Hauptmenü

Nach dem öffnen der App befindet man sich zu Beginn im Hauptmenü, welches in Abbildung 4.1 zu sehen ist. Für das Hauptmenü wir, wie auch für die gesamte App, haben wir ein ansprechendes Farbklima gewählt. Designmäßig haben wir uns für ein einfaches und nicht zu verspieltes Design entschieden. Wir wollten eine App, bei der schnell alles ersichtlich ist und ohne Vorwand alles bedient werden kann und die Elemente

#### 4 Konzept, Entwurf und Implementierung

besitzt, die Benutzer bereits von anderen Android Anwendungen kennen. Wenn möglich, wiederholen sich die selben Elemente und Designaspekte innerhalb der App.

Durch den Menüpunkt 'SPIELEN' landet der Spieler auf einem Bildschirm, in welchem er ein Deck wählen kann und bei Bedarf die Einstellungen anpassen kann, wie in Abbildung 4.3 zu sehen ist. Diese sind frei wählbar und alle Kombinationen sind möglich. Die Einstellungen werden lokal auf dem Smartphone gespeichert und können beim nächstenmaligen Starten direkt übernommen werden, ohne sie jedes mal neu zu ändern. Das ermöglicht einen schnellen Start ins Spiel.

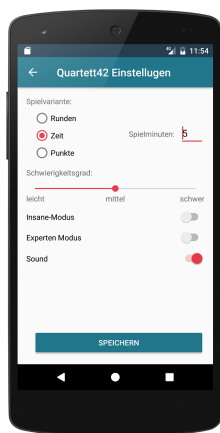


Abb. 4.3: Einstellungsmenü der App

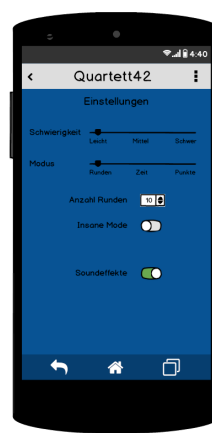


Abb. 4.4: Mockup Einstellungen

Nun ist das eigentliche Spiel gestartet. Der Spieler sieht jeweils nur seine erste oberste Karte, wie in Abbildung 4.5 abgebildet ist. Für jedes Attribut wird neben dem Wert (beziehungsweise einem Fragezeichen im "Expertenmodus") auch die jeweilige Einheit und die Siegesvariante (höher oder niedriger gewinnt) angegeben. Zwischen den Bildern einer Karte kann durch Swipe-Gesten gewechselt werden und zu jedem Bild kann, falls vorhanden, die passende Information durch drücken auf den Info-Button angezeigt werden. Wenn der Spieler am Zug ist, sind seine Attribute aktiv und er kann ein Attribut für den Vergleich auswählen. Für seinen Zug hat der Spieler unbegrenzt Zeit, außer im Zeitmodus, in welchem die Zeit für einen Zug auf 30 Sekunden beschränkt ist, damit der Spieler sich nicht auf seinen bisherigen Erfolgen ausruhen kann. Die Zeit für seinen Zug wird ihm angezeigt und zudem werden die letzten 10 Sekunden durch ein akusti-



#### 4 Konzept, Entwurf und Implementierung

sches Signal verdeutlicht, sofern in den Einstellungen die Sounds aktiviert sind. In allen Spielvarianten wird dem Spieler während des Spiels der aktuelle Zwischenstand (durch Punkte oder Karten) und die verbleibende Spielzeit(verbleibende Karten/Punkte/Zeit) angezeigt.

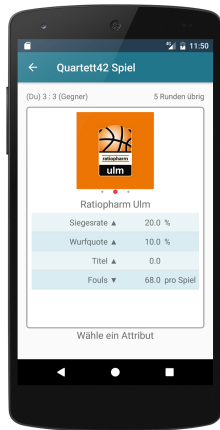


Abb. 4.5: Kartenansicht im Spiel



Abb. 4.6: Mockup Karte im Spiel

Ist der Gegner am Zug, sieht die Kartenansicht des Spielers gleich aus, aber seine Attribute sind inaktiv und nicht auswählbar. Nun muss der Spieler eine gewisse Zeit warten, bis der Computer sein Attribut für den Vergleich ausgewählt hat. Diese Zeit kann er nutzen, um seine Karte zu betrachten. Im Schwierigkeitsgrad "leicht" wählt der Computer ein zufälliges Attribut für den Vergleich aus. Im Schwierigkeitsgrad "mittel" wählt er unter einer zufälligen Menge an Attributen, welche ungefähr die Mächtigkeit der Hälfte aller Attribute hat, den besten Wert aus. Im Schwierigkeitsgrad "schwer" wählt der Computer unter allen Attributen zu einer gewissen Wahrscheinlichkeit den besten Wert aus. Diese Schwierigkeitsgrade sorgen dafür, dass der Spieler zwar einen Anstieg der Schwierigkeit des Computers spürt, jedoch immer eine Chance zu gewinnen hat. Zudem bleibt durch die Zufallswerte der Zug des Computergegners unvorhersehbar.

Nach jedem gespielten Zug sieht der Spieler den Vergleichsbildschirm mit seiner Karte und der Karte seines Gegners, wie in Abbildung 4.7 dargestellt (Wir sind hier insofern vom Mockup abgewichen, dass das Bild einer Karte mit dem gewählten Attribut angezeigt wird, statt nur alle Attribute anzuzeigen, da es so attraktiver aussieht). Dort kann wieder

#### 4 Konzept, Entwurf und Implementierung

bei beiden Karten zwischen den Bildern durch Swipe-Bewegungen gewechselt werden und die jeweiligen Informationen durch Drücken des Info-Buttons angesehen werden. Der gewählte Wert beider Spieler wird angezeigt und der Gewinner hervorgehoben. Der Gewinner bekommt beide Karten, bei einem Unentschieden behält jeder Spieler seine Karte. Zudem wird der Punktestand aktualisiert. Im Kartenmodus und im Zeitmodus ist jede Karte genau ein Punkt wert. Im Punktemodus berechnen sich die Punkte für den Gewinner durch den prozentualen Unterschied beider Werte. So kann der Spieler mit ein und dem selben Attribut einer Karte unterschiedlich viele Punkte machen, je nachdem welche Karte der Gegner gerade besitzt. Außerdem ist es dadurch möglich, dass ein Spieler mit weniger Karten als der Gegner gewinnt, weil er durch geschickte Spielzüge mehr Punkte gesammelt hat.

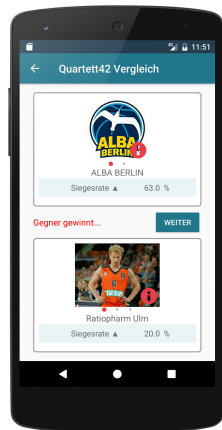


Abb. 4.7: Kartenvergleich



Abb. 4.8: Mockup Kartenvergleich

Der Spieler hat jederzeit die Möglichkeit, das laufende Spiel zu unterbrechen. Dazu wird ein Spiel automatisch zwischengespeichert, wenn der Spieler das laufende Spiel oder die App verlässt. Dieses kann der Spieler zu einem späteren Zeitpunkt fortsetzen. Wenn er ein neues Spiel beginnen möchte wird er gefragt, ob er lieber das alte Spiel fortsetzen will oder ein neues beginnen und das alte überschreiben möchte.

Steht ein Gewinner fest, wird der Spielend-Bildschirm angezeigt. Auf diesem steht der Gewinner, der Endstand und die gesammelten Punkte des Gewinners, wie in Abbildung

4.9 zu sehen.

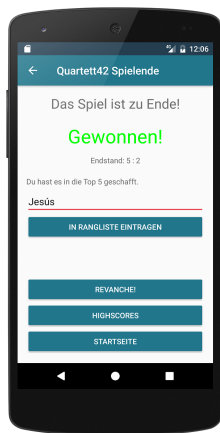


Abb. 4.9: Endscreen der App



Abb. 4.10: Mockup Endscreen

Ist der Spieler der Gewinner, wird anhand dieser Punkte entschieden, ob der Spieler es in die Rangliste geschafft hat, und falls ja, auf welche Position. Die Punkte berechnen sich durch eine Kombination aus den gesammelten Punkten im Spiel, dem Schwierigkeitsgrad des Spiels, dem Spielmodus (normal, Insanemodus oder Expertenmodus) und der Anzahl der eingestellten Runden/Zeit/Punkte. Dadurch bleibt die Rangliste immer fair und kann nicht durch vereinfachte Einstellungen oder eine erhöhte Anzahl an Spielrunden beeinflusst werden. Der Spieler kann sich hier direkt mit seinem Namen in die Rangliste eintragen, wobei der jeweils letzte Name gespeichert wird. Außerdem kann er von diesem Bildschirm direkt die Ranglisten (Abbildung 4.11) einsehen oder eine Revanche starten, welche ein Spiel mit dem gleichen Deck und den gleichen Einstellungen startet.

Neben dem Spiel ansich befinden sich die meisten Implementierungsdetails unserer Anwendung im Menüpunkt 'GALERIE'. Dort hat der Spieler eine Übersicht aller Decks, die auf dem Smartphone vorhanden sind. Dieses Menü wurde durch ein Grid-Layout-Adapter erstellt und findet sich an vielen Stellen in unserer App wieder. Es erlaubt neben einer einfachen und immer gleich großen und gleich formatierten Darstellung der Bilder auch ein einfaches Scrollen und Anzeigen von Deckinformationen durch drücken auf

#### 4 Konzept, Entwurf und Implementierung

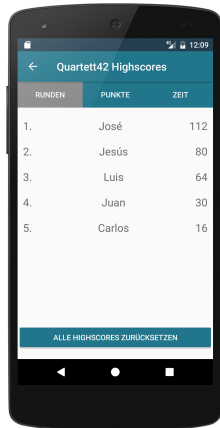


Abb. 4.11: Rangliste der App

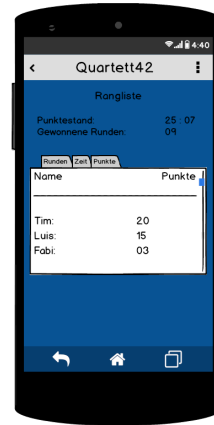


Abb. 4.12: Mockup Rangliste

den Info-Button. Ein Bild davon gibt es in Abbildung 4.13. Im Menü kann der Spieler die Karten eines Decks angucken, was ungefähr ähnlich aussieht wie die Kartenansicht während eines Spiels in Abbildung 4.5. Zudem kann er hier die Namen der Decks umbenennen, die einzelnen Werte und Bilder mit dazugehörigen Beschreibungen einzelner Karten der Decks ändern sowie neue Karten zu Decks hinzufügen oder Karten löschen. Außerdem kann ein Deck auch gelöscht werden oder über den Deckcreator ein völlig neues Deck mit neuen Attributen erstellt werden. Neue Decks können dann auf den Deckstore hochgeladen werden, sodass es für andere Nutzer zur Verfügung steht. Über diesen können wiederum auch neue Decks von anderen Nutzern auf das Smartphone herunter geladen werden. Auf diese zuletzt genannten Funktionen werden wir in den Besonderheiten genauer eingehen.

Neben der Rangliste gibt es auch noch einige Statistiken in unserer App, wie zum Beispiel die Anzahl der gespielten Spiele und die dabei gewonnen und verlorenen Spiele für die verschiedenen Spielmodi. Diese werden mit Hilfe der Bibliothek MPAndroidCharts als Piechart dargestellt und werden nach jedem bis zum Ende durchgespielten Spiel aktualisiert. Ein Bild davon gibt es in Abbildung 4.15.

## 4 Konzept, Entwurf und Implementierung

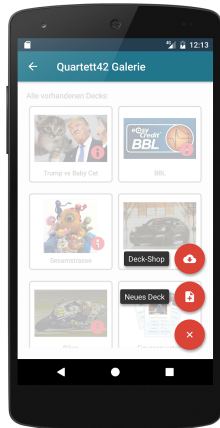


Abb. 4.13: Die Galerie der App



Abb. 4.14: Mockup Galerie



Abb. 4.15: Die Statistiken der App

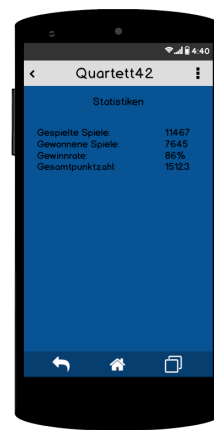


Abb. 4.16: Mockup Statistiken

## 4.2 Besonderheiten

Im Vergleich zu anderen Quartett Apps gibt es bei unserer Quartett42 App, neben den verschiedenen Spielmodi und Decks, besonders zwei Funktionen, die so bisher noch nicht verfügbar waren. Diese sind zum einen unser Deckcreator und -editor und zum anderen der Deck Down- und Upload.



Abb. 4.17: Erster Schritt im Deckcreator

### 4.2.1 Deckcreator und -editor

Der Deckcreator erlaubt das Erstellen neuer Decks oder das Bearbeiten vorhandener Decks. Dadurch kann jeder Spieler ein komplett neues eigenes Deck nach seinen Wünschen erstellen. Über die Galerie gelangt der Spieler in den Deckcreator. Dort muss er erst einmal allgemeine Angaben zu seinem gewünschten Deck angeben, wie in Abbildung 4.17 dargestellt. Dort muss ein Name für das Deck eingegeben werden, wobei der Creator ein Deck nur erstellt, wenn dieser Name nicht bereits vergeben ist. Beschreibung und Bild sind optional, wobei bei keinem angegebenen Bild ein Standardbild verwendet wird. Das Bild kann entweder aus der Fotogalerie des Smartphones gewählt werden oder direkt aufgenommen werden, wobei es in beiden Fällen direkt auf eine akzeptable Größe verkleinert wird. Zudem müssen eine beliebige Anzahl an Attributen festgelegt werden, und für jede Attribut, wann es gewinnt, und welche Einheit es hat. Hier müssen verschiedene Überprüfungen statt finden. So darf kein Attribut zwei mal verwendet werden und es dürfen nirgends unerlaubte Zeichen oder leere Eingaben vorkommen.

#### 4 Konzept, Entwurf und Implementierung

Sind alle Eingaben korrekt, wird das Deck zwischen gespeichert und der Nutzer zum zweiten Schritt weiter geleitet.

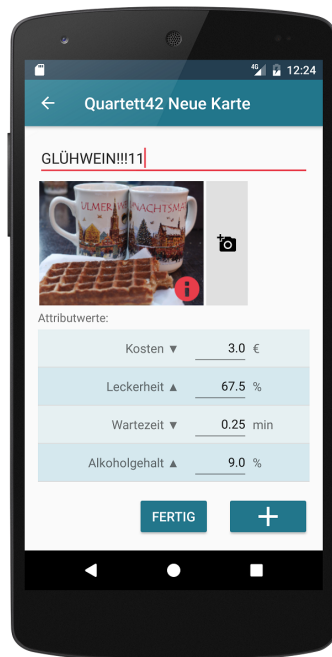


Abb. 4.18: Zweiter Schritt im Deckcreator

Im zweiten Schritt geht es um die Erstellung einzelner Karten und deren Werte. Wenn der Benutzer ein bereits vorhandenes Deck bearbeiten will, gelangt er direkt in diesen Schritt. Ein Bild davon kann man in Abbildung 4.18 sehen. Für jede Karte muss ein Name eingegeben werden, welcher pro Deck wieder nur einmal vorkommen darf. Zudem kann eine beliebige Anzahl an Bildern wie beim Deckbild und dazu passende Beschreibungen angegeben werden. Für jedes Attribut der Karte muss ein Wert angegeben werden. Auch hierbei muss wieder alles auf gültige Eingaben überprüft werden und es dürfen keine leeren Eingaben vorkommen. Der Spieler kann jederzeit eine neue Karte hinzufügen und eine alte Karte löschen. Zudem kann er zwischen allen Karten hin und her wechseln. Dabei wird das Deck bei jedem Wechsel zwischengespeichert. Damit ein Deck spielbar ist, muss es mindestens zwei Karten besitzen. Ist der Nutzer fertig mit dem Erstellen oder Bearbeiten eines Decks, kann er es speichern und ansehen oder spielen. Zudem besteht die Möglichkeit, den Erstellvorgang zu unterbrechen und zu einer anderen Zeit fortzusetzen.

### 4.2.2 Down- und Upload von Decks

Über den Deckstore können neue Decks runter und hoch geladen werden. Das ermöglicht das Teilen von selbst erstellten Decks mit anderen und führt dazu, dass immer wieder neue Decks herunter geladen werden können. Die Speicherung der Decks findet auf einem Server des Institutes für Datenbank und Informationssysteme der Universität Ulm statt. Die Daten Decks werden dabei als JSON-Strings gespeichert.

Über die Galerie gelangt man in den Deckstore, in welchem dem Spieler alle Decks aus dem Deckstore angezeigt werden, welche er noch nicht selbst auf sein Smartphone herunter geladen hat. Diese erfragt die Anwendung durch einen Http-Request an den Server, welcher eine Liste alle auf dem Server vorhandenen Decks zurück liefert. Zudem wird für jedes Deck noch das Deckbild und die Deckbeschreibung herunter geladen und gecached. Außerdem erhält der Nutzer beim Starten der App eine Notifikation in der Statusleiste, falls neue Decks im Deckstore verfügbar sind. Die Darstellung im Deckstore erfolgt wieder über den Grid-View-Adapter, ähnlich zu Abbildung 4.13. Der Benutzer kann nun ein Deck auswählen, welches er herunterladen möchte. Der Nutzer sieht während dem gesamten Ladevorgang einen Fortschrittsbalken, welcher den Fortschritt in Prozent berechnet und darstellt. Für das Deck wird dann zuerst die allgemeine Deckinformation heruntergeladen sowie eine Liste aller Karten. Dann wird jede einzelne Karte heruntergeladen und für jede Karte alle dazugehörigen Bilder und Werte. Für jede dieser Daten wird ein Http-Request an den Server gesendet, welcher die Daten in Form eines JSON-Strings zurück gibt. Da nicht nur unsere Anwendung sondern mehrere Anwendungen Decks auf den Server hochladen können, welche nicht unbedingt formal korrekt sein können, wird in jedem Schritt überprüft, ob das Deck alle Anforderungen erfüllt. Dazu gehören zum Beispiel eine Mindestanzahl an Karten und Attributen, keine unerlaubten Zeichen und leere Angaben sowie keine falschen Bilddateien. Letzteres wird durch ein Standardbild ersetzt, bei allen anderen Fehlern wird der Download abgebrochen, um unnötige Wartezeiten zu vermeiden. Bilder, die zu groß sind, werden automatisch verkleinert. Nach dem vollständigen Download aller Daten und der Kontrolle wird das Deck zusammen gebaut und in den lokalen Speicher gespeichert. Hierbei wird bevorzugt versucht, die Daten auf einem externen Speicher zu



#### *4 Konzept, Entwurf und Implementierung*

speichern, um nicht den internen Speicher zu belasten. Eine Grafik, wie der Download funktioniert, sieht man in Abbildung 4.19.

In der Galerie hat der Benutzer die Möglichkeit, ein Deck direkt auf den Server hochzuladen. Zuerst wird durch eine Anfrage an den Server überprüft, ob das Deck schon im Deckstore vorhanden ist. Ist dies der Fall, kann das selbe Deck nicht erneut hochgeladen werden. Fall nicht, bekommt der Benutzer wieder eine Ladeanzeige mit Fortschrittsbalken zu sehen. Bevor das Deck hochgeladen wird, wird auch dieses zur Sicherheit auf Fehler überprüft, welche aber im Normalfall dank der Fehlerüberprüfung bei der Deckerstellung nicht vorkommen dürfen. Dann wird das Deck in viele einzelne JSON-Strings zerlegt und dem Server über einen Http-Request das Deck mit seinen Informationen bekannt gegeben. Danach wird wie beim Download, jede Karte und mit ihm seine Werte und Bilder als JSON-String über einen extra Http-Request gesendet, wobei die Bilder in einen Base64-String konvertiert und versendet werden. Nach dem vollständigen Upload ist das Deck für alle Benutzer über den Deckstore verfügbar. Eine Grafik, wie der Upload funktioniert, sieht man in Abbildung 4.19. Sollte der Benutzer keine Internetverbindung, wird sowohl der Download als auch der Upload abgebrochen.

#### 4 Konzept, Entwurf und Implementierung

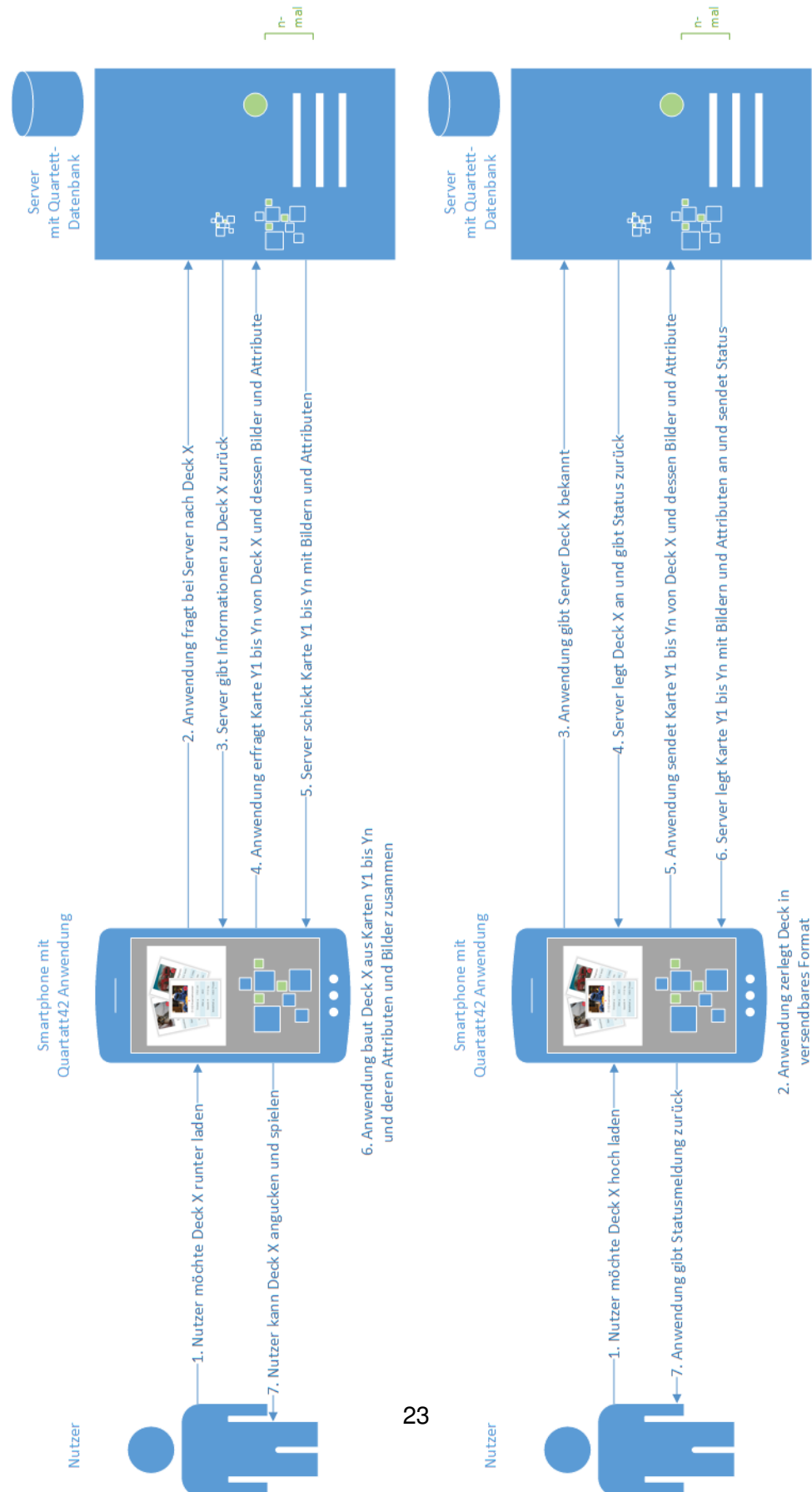


Abb. 4.19: Links: Downloadmodell, rechts: Uploadmodell

## 4.3 Architektur

### 4.3.1 Datenmodell und -speicherung

Für die Speicherung der Daten wollten wir ein möglichst einfaches, schnelles, aber auch platzsparendes Verfahren verwenden. Deswegen fiel die Wahl schnell auf eine Speicherung der Daten im JSON-Format. JSON bietet einige Vorteile, wie zum Beispiel die kompakte Speicherung, eine einfache Handhabung, da die Daten in Javascript-Syntax gespeichert werden und beliebige Erweiterbarkeit. Außerdem ermöglicht die Datenspeicherung in JSON einige Vorteile, die durch klassische Datenbanken nicht gewährleistet sind wie die Speicherung von Listen und Arrays. Aus diesem Grund entspricht unser Datenmodell nicht ganz den Chen-Notation, ist aber an diese angelehnt.

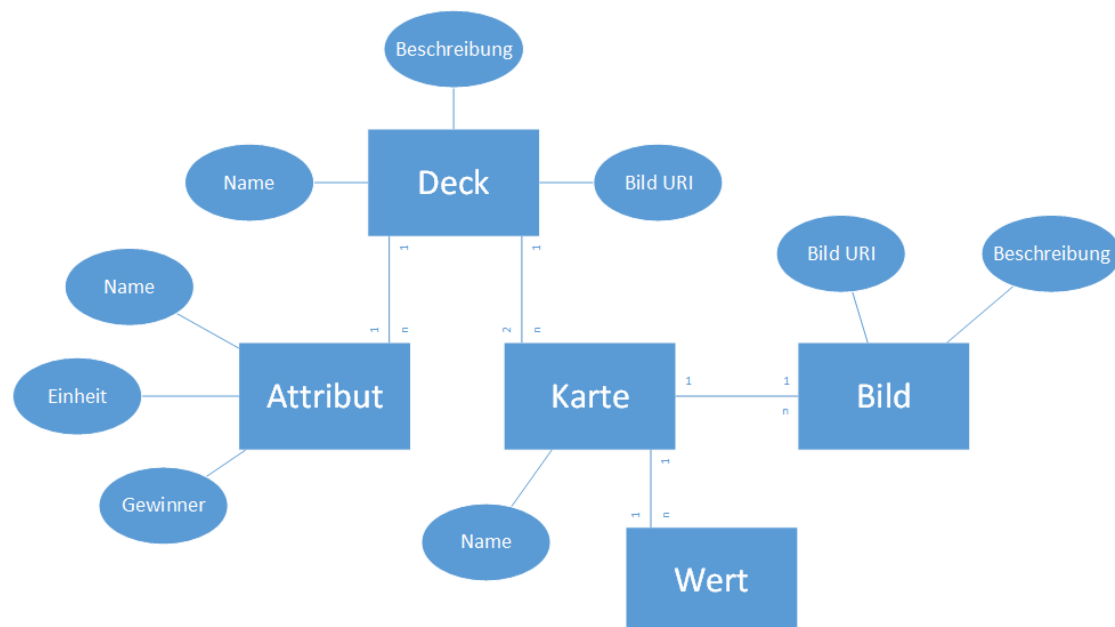


Abb. 4.20: Datenmodell

Gespeichert wird eine Liste von Decks. Jedes Deck enthält einen eindeutigen Namen und eine Beschreibung, sowie URI zu einem Bild. Außerdem enthält jedes Deck zwischen vier und zehn Attributen, welche einen für ein Deck eindeutigen Namen, eine Einheit und die Gewinnrichtung angeben. Dies erspart einen Overhead, der entstehen würde,

#### 4 Konzept, Entwurf und Implementierung

wenn man diese Informationen bei jeder einzelnen Karte mitangeben müsste. Zu jedem Deck gehört eine Liste von mindestens zwei Karten. Diese haben jeweils neben einem für dieses Deck eindeutigen Namen ein oder mehrere Bilder, welches aus einer URI zu einem Bild und einer Beschreibung besteht. Außerdem enthält jede Karte gleich viele Werte, wie das Deck Attribute. Mit diesem einfachen Modell können sämtliche Daten gespeichert werden. Alle Einstellungen und laufenden Spiele werden direkt auf dem Smartphone über SharedPreferences gespeichert. Sämtliche Bilder werden direkt als Bitmap gespeichert und nur deren URI in den JSON-Daten verlinkt. Dabei werden die Bilder schon beim Speichern auf eine akzeptable Größe runter skaliert. Wenn möglich, wird ein externer Speicherplatz einem internen vorgezogen.

##### 4.3.2 Klassen- und Activity-Architektur

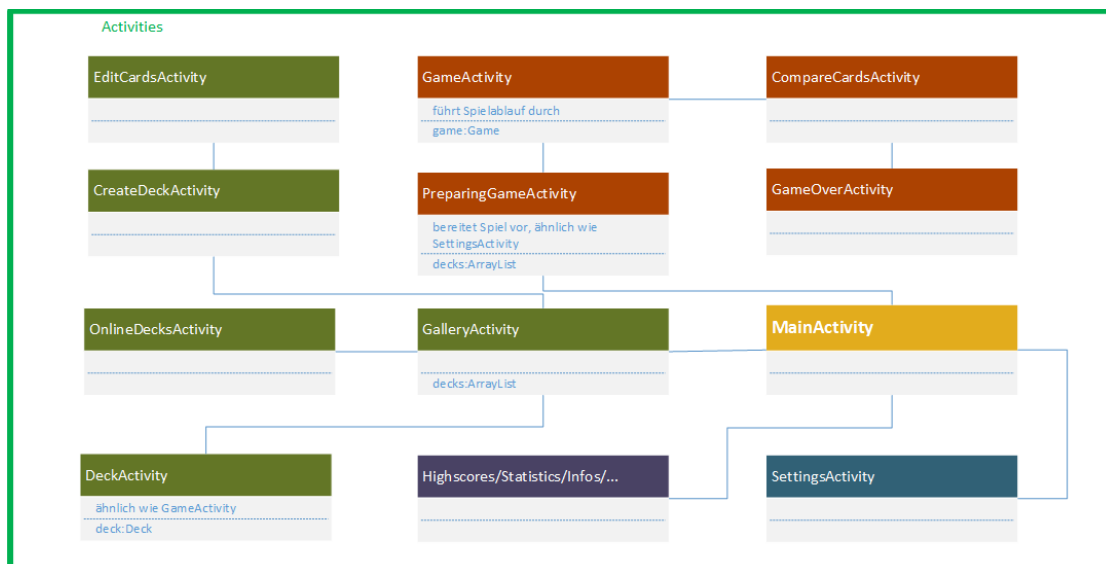


Abb. 4.21: Activitymodell (vereinfacht)

Die Architektur unserer Klassen entspricht mit kleinen Änderungen so ziemlich dem Datenmodell in Abbildung 4.20 weswegen wir es hier nicht noch einmal extra abbilden. Was die Activities angeht, haben wir hier ein vereinfachtes Modell unserer Activities. Dieses beinhaltet alle Activities, welche durch eine Oberfläche für den Nutzer sichtbar

und bedienbar sind. Es kann also gleichermaßen auch als GUI-Modell verstanden werden.

Zwischen den Activities gibt es natürlich noch weitere Verzweigungen. Zudem gibt es eine ganze Reihe von Hilfsklassen und -activities, die Hintergrundaufgaben durchführen, wie zum Beispiel das Handeln der Bilder, das Lesen und Schreiben der Daten im Speicher, die Kommunikation mit dem Server sowie das Durchführen der gesamten Spielelogik.

### **4.3.3 Arbeitsaufteilung**

Während der Implementation wurden die Aufgaben so aufgeteilt, dass jeder aus dem Team immer für eine bestimmte Aufgabe zuständig war und diese durch alle Schichten durch erledigen musste. Bei großen Aufgaben, wie dem Spielablauf, wurden die Aufgabe zudem in kleinere Schritte aufgeteilt. Da die App aus mehreren unabhängigen Bereichen besteht, konnten die Aufgaben gut eingeteilt werden, ohne dass es zu Behinderungen kommt. Wenn gerade keine Arbeit für eine Person angefallen war, beschäftigte diese sich mit Verbesserungen, Testen von Funktionen oder mit der Weiterentwicklung des Designs.

## **4.4 Schwierigkeiten während der Implementierung**

In diesem Abschnitt möchten wir noch auf die größten Probleme eingehen, die wir während der Implementation hatten, und wie wir diese gelöst haben. Generell muss man dazu sagen, dass es insgesamt doch zu deutlich weniger Problemen gekommen ist, als wir anfangs vermutet hatten. Und wenn Probleme aufkamen, hat man dank der ausführlichen Dokumentation und dank vielen Hilfen im Internet relativ schnell die Lösung dafür gefunden.

- Ein großes Problem war der Umgang mit Bildern, da es dabei sehr viel zu beachten gibt. Dank der unterschiedlichen Bildgrößen und -formate gab es Probleme, diese immer richtig und in der gleichen Größe darzustellen. Auch das Speichern brachte

#### *4 Konzept, Entwurf und Implementierung*

verschiedene Probleme mit sich, bis wir den richtigen Speicherort und Speicherart gefunden haben, um nicht zu viel Platz zu verschwenden. Lösen konnten wir das Problem mit dem oben erwähnte Framework Picasso. Dieses nimmt einen sehr viel Arbeit im Umgang mit Bildern ab, wie das automatische Laden, Skalieren und Anzeigen von Bildern. Sehr lange Methoden konnten so durch einen Einzeiler ersetzt werden.

- Ein weiteres Problem brachte der Umgang mit der Handygalerie, der Kamera und auch mit dem Abspielen von Sounds mit sich. An sich geht das alles sehr einfach mit Android. Die Probleme ergaben sich aber dann beim Testen auf den eigenen Handys und besonders am Emulator, da manche Funktionen hier nicht richtig funktionierten oder anders reagierten als erwartet. Erst nach reichlichem Ausprobieren funktionierte es auf allen Geräten.
- An Stellen, an denen viele Zustände gespeichert werden müssen, wie beim Deckcreator, kam es zu Schwierigkeiten, diese zu speichern. Da der Nutzer jederzeit alle Werte ändern aber auch gleichzeitig zwischen allen Karten hin- und herwechseln können soll, führt das dazu, dass die Daten schlussendlich nach jeder einzelnen Interaktion im Deckcreator zwischengespeichert und überprüft werden müssen.
- Auch der Datenaustausch zwischen den Activities hat uns Probleme bereitet, da diese außer bei primitiven Datentypen nicht trivial ist. Das lösten wir, indem wir versucht haben, den Datenaustausch zwischen den einzelnen Activities auf das Mindeste zu reduzieren und alle größeren Daten und Objekte aus dem Hintergrund zu laden.
- Die wahrscheinlich meiste Zeit haben wir aber mit der GUI und dem Anordnen der Elemente auf der GUI-Ebene verbracht. Trotz den Möglichkeiten, den GUI-Builder zu benutzen, sah die GUI oft nicht aus wie gewünscht. Zudem gab es das Problem, dass eine GUI, die auf einem Smartphone gut aussah, auf einem anderen Smartphone ganz verschoben war oder Teile nicht sichtbar waren. Wenn man eine Anwendung für alle verschiedenen Bildschirmformate anpassen will, kann das viel Zeit und Nerven in Anspruch nehmen. Wir haben das ganze durch ein nicht all zu kompliziertes Design und durch viele wiederkehrende Designaspekte gelöst.

# 5

## Anforderungsabgleich

Es folgt ein tabellarischer Vergleich zwischen den spezifizierten Anforderungen (vgl. Kapitel 3) und den implementierten Funktionen. Es wird bei jeder Anforderung aus Kapitel 3 beurteilt, inwieweit diese in der Implementierung erfüllt ist (Ja/Teilweise/Nein). Eine Zuordnung zu den Einträgen aus Kapitel 3 ist anhand der ID möglich.

Die Funktionen, die implementiert wurden, jedoch nicht im Voraus als Anforderungen spezifiziert wurden, (Statistiken, Deckeditor, etc.), werden in diesem Vergleich nicht berücksichtigt.

## 5.1 Funktionale Anforderungen

ID	Erfüllt	Kommentar
FA1	Ja	-
FA2	Ja	-
FA3	Ja	-
FA4	Ja	-
FA5	Ja	Bei der Galerieansicht der Karten eines Decks war geplant, dass der Benutzer per Swipe nach links / rechts zur vorherigen / nächsten Karte wechselt. Jedoch war die Swipe-Funktionalität schon für das ansehen der verschiedenen Kartenbilder implementiert, was zu Konflikten hätte führen können. Das Wechseln der Karte wurde daraufhin über einfache "Links"- und "Rechts"-Buttons umgesetzt.
FA6	Teilw.	Wir entschlossen uns früh, keinen Spieler-vs.-Spieler-Modus umzusetzen, und konzentrierten uns stattdessen auf Funktionalitäten wie das Erstellen, Ändern, oder Teilen von Decks. Der Rest der beschriebenen Funktionen wurde implementiert.
FA7	Ja	Statt der direkten Hervorhebung des Attributs befindet sich beim Vergleichs-Bildschirm die eigene Karte immer unten (also auf der "Seite" des Benutzers), und die Karte des Gegners immer oben. In der Mitte erscheint ein farbiger Text, der den Gewinner der Runde bekanntgibt. (vgl. Abb. 4.7)
FA8	Ja	-
FA9	Ja	-
FA10	Ja	-
FA11	Teilw.	Wir entschieden uns nach einiger Recherche dazu, eine JSON-Datei zum Persistieren der Daten zu benutzen. Auf Anwenderebene ändert dies nichts. Um dem Benutzer mehr Freiheiten zu lassen, entschieden wir uns zudem bewusst gegen eine Beschränkung der Karten- / Attributanzahl. Beim Hochladen eines Decks in den Deckstore werden die Decks hinsichtlich dieser Beschränkungen geprüft.



## 5.2 Nichtfunktionale Anforderungen

ID	Erfüllt	Kommentar
NFA1	Ja	-
NFA2	Ja	die maximale Reaktionszeit wurde nicht genau gemessen, liegt jedoch weit unter dem gewählten Wert von 1,5 Sekunden.
NFA3	Ja	Der Spielstand kann außerdem (über den "Zurück"-Button) explizit gespeichert werden, wodurch die Anwendung auch komplett beendet werden kann, ohne dass der Spielstand verloren geht.
NFA4	Ja	Diese Anforderung ist schwer beurteilbar, jedoch hielten wir uns stets möglichst an die Android-Richtlinien für Benutzeroberflächen, und alle Bereiche der Anwendung sind leicht bedienbar.
NFA5	Ja	Die Anwendung benötigt nur bei Benutzung des Deckstores (Upload / Download) eine Internetanbindung.

# 6

## Zusammenfassung und Ausblick

Dieses Kapitel ist noch ein kleines Fazit zur App-Entwicklung und zum Anwendungsfach allgemein.

Auch wenn der Einstieg in die App-Entwicklung mit dem EiTimer nicht ganz leicht war, haben wir dort schon gemerkt, dass uns die Entwicklung liegt und Spaß bereitet. Wir haben gerade durch den EiTimer und auch unsere Quartett42-App viel gelernt, was wir im zweiten Teil des Anwendungsfachs natürlich anwenden und weiter vertiefen wollen.

In unseren Augen ist uns unsere App ziemlich gut gelungen, sodass die im Appstore veröffentlicht werden könnte, da es gerade bei Quartettspielen eine Marktlücke gibt. Natürlich gäbe es noch einige Stellen an der sie weiter verbessert werden könnte, wie z. B. im Design oder beim Hinzufügen von weiteren Funktionen (Spieler vs. Spieler, Online/Bluetooth-Spiel, ...).

Was unser Team angeht, haben wir beschlossen auch im zweiten Teil des Anwendungsfachs gemeinsam wieder eine App zu entwickeln, für die wir auch schon einige Ideen gesammelt haben.

- Zusammenfassung: gute App geworden und wir haben viel gelernt ... - Ausblick was die App angeht: App eigentlich fertig aber könnte an manchen Stellen noch optimiert werden (Design, weitere Funktionen,...) - Ausblick was das Team angeht: erlerntes Wissen in das Entwickeln neuer Apps umsetzen

# Abbildungsverzeichnis

1.1	Quartett42 Logo . . . . .	2
2.1	Android Zustandsmodell . . . . .	4
4.1	Hauptmenü der App . . . . .	12
4.2	Mockup Hauptmenü . . . . .	12
4.3	Einstellungsmenü der App . . . . .	13
4.4	Mockup Einstellungen . . . . .	13
4.5	Kartenansicht im Spiel . . . . .	14
4.6	Mockup Karte im Spiel . . . . .	14
4.7	Kartenvergleich . . . . .	15
4.8	Mockup Kartenvergleich . . . . .	15
4.9	Endscreen der App . . . . .	16
4.10	Mockup Endscreen . . . . .	16
4.11	Rangliste der App . . . . .	17
4.12	Mockup Rangliste . . . . .	17
4.13	Die Galerie der App . . . . .	18
4.14	Mockup Galerie . . . . .	18
4.15	Die Statistiken der App . . . . .	18
4.16	Mockup Statistiken . . . . .	18
4.17	Erster Schritt im Deckcreator . . . . .	19
4.18	Zweiter Schritt im Deckcreator . . . . .	20
4.19	Links: Downloadmodell, rechts: Uploadmodell . . . . .	23
4.20	Datenmodell . . . . .	24
4.21	Activitymodell (vereinfacht) . . . . .	25