ulm university universität
uulm

Universität Ulm | 89069 Ulm | Germany

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

Mobile Application Lab - FancyQuartett Dokumentation

Dokumentationan der Universität Ulm

Vorgelegt von:
Lukas Stöferle, Ferdinand Birk, Marius Kircher
lukas.stoeferle@uni-ulm.de, ferdinand.birk@uni-ulm.de, marius.kircher@uni-ulm.de

Gutachter:
Prof. Dr. Manfred Reichert
Marc Schickler

Betreuer:
Marc Schickler

WS 2015/16

Fassung 20. März 2016

Inhaltsverzeichnis

1

Einleitung

Seit der Ankündigung des weltweit ersten Smartphones durch *Apple Inc.* im Jahre 2007 hat sich sehr viel auf dem Markt für mobile Endgeräte getan. Heute gibt es eine Vielzahl an verschiedenen Smartphones von unterschiedlichen Herstellern, die jeweils mit unterschiedlichen Betriebssystemen ausgestattet sind, wie z.B. *Android* von *Google Inc.*, *iOS* von *Apple Inc.* oder *WindowsPhone* von *Microsoft*. Smartphones sind heute nicht mehr aus unserem Alltag wegzudenken, da sie den Benutzer aufgrund ihrer Mobilität und dank verschiedener Applikationen (kurz „Apps“) unterstützen aber auch unterhalten. Im diesjährigen Anwendungsfachs *Mobile Application Lab* entstand deshalb im Rahmen eines kleinen Softwareprojektes eine Quartett-App, die sich durch eine beliebige Anzahl an auswählbaren Kartendecks von den bereits existierenden Apps im App-Store abheben soll. Hierbei werden Kartendecks per *Representational State Transfer (REST)* von einem Server der Universität Ulm heruntergeladen, die dann auf dem Gerät persistent gespeichert werden und somit auch offline verfügbar sind. Neben den vorgegebenen Anforderungen durfte jedes Team eigene Ideen einbringen, was zu unterschiedlichen Anwendungen führte.

1.1 Motivation

Apps spielen heute, aber auch in Zukunft eine große Rolle im mobilen Software-Bereich. Durch die Programmierung einer Quartett-App in Teams soll mehr Praxiserfahrung in das doch eher theorielastige Studium mit einfließen. Die Quartett-App deckt möglichst viele Bereiche der App-Programmierung ab, wodurch die Teams später in der Lage sind, ihre eigenen Ideen und Anwendungen planen und umsetzen zu können.

1.2 Kontext

Die Quartett-App wird im Auftrag des *Institut für Datenbanken und Informationssysteme* der *Universität Ulm* entwickelt. Dabei sollen die Teams grundlegende Werkzeuge zur App-Programmierung kennenlernen und somit Schritt für Schritt die benötigten Schritte der App-Entwicklung meistern. Durch sorgfältige Planung, wie z.B. mit Hilfe von Mockups oder Diagrammen, soll die spätere Implementierung vereinfacht werden. Neben der Planung und Implementierung der Anwendung sollen die Teams sich mit den Besonderheiten der Zielplattform und der dazugehörigen Smartphones (wie z.B. Datenhaltung, Sensorik oder gestenbasierter Eingabe) auseinander setzen.

2

Anforderungen

In diesem Teil der Dokumentation werden die Anforderungen, welchen die App entsprechen soll, aufgeführt und erläutert. Dabei werden sowohl die Anforderungen unseres Betreuers als auch die eigenen optionalen Anforderungen zusammengefasst.

2.1 Funktionale Anforderungen

2.1.1 Spieltyp

Die Anwendung soll ein reines Single-Player-Spiel sein, d.h. es gibt keine Möglichkeit per Bluetooth- bzw. Internetverbindung mit anderen Geräten zu kommunizieren. Der Benutzer spielt also immer gegen den Computergegner. Es soll lediglich die Möglichkeit geben zu zweit am selben Gerät ein Multiplayer-Spiel (Hotseat) zu spielen, somit können zwei Personen gegeneinander spielen.

2.1.2 Spielmodi

Dem Spieler soll eine Auswahl von drei verschiedenen Spielmodi angeboten werden, welche im Folgenden erläutert werden:

To-The-End Der Spieler, der zum Schluss alle Karten besitzt, hat das Spiel gewonnen. Ein Spieler hat das Spiel verloren, wenn er keine Karten mehr besitzt.

Time Ein Spiel ist durch ein Zeitlimit begrenzt. Wird dieses Limit erreicht, gewinnt der Spieler der die meisten Karten besitzt. Falls beide Spieler gleich viele Karten besitzen sollten wird eine extra Runde gespielt, in welcher dann der Sieger bzw. Verlieren gekürt wird. Hat ein Spieler vor Erreichen des Zeitlimits keine Karten mehr, hat er verloren.

Points Jeder Spieler besitzt ein Punktekonto, welches sich durch gewonnene Duelle erhöht. Erreicht ein Spieler die maximale Punktzahl hat er das Spiel gewonnen. Besitzt ein Spieler keine Karten mehr, bevor die maximale Punktzahl erreicht wird, gewinnt der Spieler der die meisten Punkte besitzt. Die Punkte ergeben sich aus der Qualität der gewählten Attributwerte der aktuellen Karte. Sehr gute Attribute geben nur 1 Punkt, Attributewerte die eine $\pm$5% Differenz zum Median des Wertes im Kartendeck besitzen geben 2 Punkte, die schlechteren Attributwerte ergeben 5 Punkte. Gewinnt der Gegenspieler ein Duell, so bekommt er die Punkte die er für seinen Attriutwert bekommen hat gutgeschrieben. Dieser Spielmodus soll vor allem den Spieler dazu verleiten, schlechte Attributwerte zu wählen.

2.1.3 Einstellungsmöglichkeiten

Zusätzlich zu den drei Spielmodi sollen weitere Einstellungen möglich sein, welche das Spielerlebnis abwechslungsreicher machen sollen. Diese sind optional und können je nach belieben aktiviert bzw. deaktiviert werden.

Maximale Rundenanzahl Ein Spiel kann durch eine maximale Rundenanzahl begrenzt werden. Wird das Rundenlimit erreicht, wird ein Gewinner bzw. Verlierer ermittelt. Wenn es keinen Gewinner gibt wird eine zusätzliche Runde gespielt, in welcher dieser dann ermittelt wird.

Zeitlimit für ein Spielzug Der Spielzug eines Spielers kann durch ein Zeitlimit begrenzt werden. Wählt ein Spieler nach Ablauf der Zeit keinen Attributwert aus, so wird zufällig ein Attributwert der aktuellen Karte ausgewählt.

2.1.4 Gespeichertes Spiel

Während eines Spiels soll immer die Möglichkeit bestehen, das Spiel zu pausieren. Hierbei soll das aktuelle Spiel gespeichert werden, welches dann im Hauptmenü später fortgesetzt werden kann. Das pausierte Spiel soll persistent gespeichert werden, damit es nach einer Schließung der App weiterhin existert.

2.1.5 Statistiken

Es soll die Möglichkeit bestehen, eine kleine Statistik über die vergangenen Spiele einsehen zu können. In dieser sollen die Anzahl der gespielten Spiele und die Anzahl der bisherigen Duelle aufgelistet werden, welche jeweils die Anzahl der Siege bzw. Niederlagen beinhalten.

2.1.6 Galerie

In der Anwendung soll es eine Galerie geben, in welcher die offline und online verfügbaren Kartendecks aufgelistet werden. Zusätzlich soll hier die Möglichkeit bestehen, dass weitere Kartendecks heruntergeladen werden können. Die Ansicht der Kartendecks und der Karten kann zwischen einer Listen- und einer Grid-Ansicht gewechselt werden. Außerdem soll es eine Detailansicht für einzelne Karten geben in welcher diese dann genauer betrachtet werden können. In der Detailansicht soll dann mit Wischgesten nach links bzw. recht die vorherige bzw. nächste Karte angezeigt werden.

2.1.7 Kartendecks

Die App soll in der Lage sein mehrere Kartendecks zu verwalten, mit welchen der Spieler dann spielen kann. Ein Kartendeck besteht aus mindestens 8 und höchstens 72 Karten. Die Karten setzen sich aus einem Titel, mindestens einem Bild und beliebig vielen Attributen zusammen. Ein Attribut besteht aus einem Namen, Wert und der zugehörigen Einheit. Zusätzlich können Attribute optional ein Icon beinhalten. Um unterscheiden zu

können ob der höhere oder niedrigere Attributwert gewinnt wird anhand einer „WhatWins-Variable“ bestimmt.

2.1.8 Serverkommunikation

Der Benutzer kann in der Galerie weitere Kartendecks online beziehen. Hierbei sollen in der Galerie vorerst minimale Vorschaudecks vom REST-Server heruntergeladen und angezeigt werden. Damit kann Datenvolumen gespart werden, was beim mobilen Netzwerk von großer Bedeutung sein kann. Der Benutzer kann dann ein Kartendeck wählen, welches vom Server heruntergeladen und auf dem Gerät offline gespeichert wird.

2.2 Nicht-funktionale Anforderungen

2.2.1 Material Design

Die Anwendung soll sich nach an die Vorgaben zum *Material Design* von *Google Inc.* halten. Die grafische Elemente sollen dadurch einen hohen Wiedererkennungswert haben was wiederum die Usability der Anwendung zu verbessert.

2.2.2 Gestensteuerung

Die Anwendung soll verschiedene gestenbasierte Eingaben unterstützen. Speziell sollen die Gesten „Wischen“ und „Schütteln“ verwendet werden. Gestensteuerung ist eine Besonderheit von mobilen Geräten, daher bietet es sich an derartige Funktionen mit einzubinden.

2.2.3 Animationen

Animationen sollen dazu verwendet werden, um das „Look & Feel“ der Anwendung zu verbessern. Außerdem sollen sie gezielt die Aufmerksamkeit des Benutzers auf bestimmte Ereignisse lenken.

2.2.4 Offline-Szenario

Ist keine Verbindung zum Internet möglich, sollen trotzdem alle Funktionen die nicht zwingend eine Internetverbindung benötigen zur Verfügung stehen.

3

Architektur

Die Software-Architektur unserer App musste sich erster Linie den Besonderheiten des Android-Systems anpassen. Dazu wurden neben den Java APIs auch stark die spezifischen Android APIs benutzt. Deren Grundlagen werden wir unten erläutern.

Um die funktionalen Anforderungen zu erfüllen, wurde ein Datenmodell zur Speicherung während der Laufzeit sowie zur persistenten Speicherung benötigt. Die wichtigsten Datenobjekte bei dem Quartett-Spiel sind *Deck* sowie *Card*, wobei ein Deck aus mehreren Karten und eine Karte aus mehreren Bildern sowie Attributwerden besteht. Auf unsere Realisierung des Datenmodells wird unten näher eingegangen.

Eine zentrale funktionale Anforderung ist die Anbindung an einen REST-Server, von dem neue Kartendecks geladen werden können. Die Architektur benötigt also ein Modul, das HTTP-Anfragen senden, textbasierte Daten empfangen und diese in das interne Datenformat umwandeln kann. Unsere Umsetzung davon werden wir unten erklären.

3.1 Besonderheiten in Android

Eine Android-App setzt sich aus einer oder mehreren *Activities* zusammen. Außerdem ist der App eine *AndroidManifest.xml* Datei zugeordnet, die die enthaltenen Activities deklariert und in eine hierarchische Beziehung zueinander stellt. Activities werden als Java-Klassen implementiert, die von der Klasse *Activity* aus den Android APIs ableiten. Den Einstiegspunkt in die App bildet die *MainActivity*.

3.1.1 Activity Lifecycle

Das Speichermanagement wird vom Android-System im Hintergrund geleistet. Aus Speicherplatz- und Energieeffizienzgründen auf mobilen Geräten haben Activities einen sogenannten Lifecycle. Zu entsprechenden Zeitpunkten werden vom System die Methoden *onCreate()*, *onPause()* usw. aufgerufen (siehe Abbildung 3.1). In diese Methoden wird vom Anwendungsprogrammierer Code eingefügt.

3.1.2 Layouts und Views

Um eine GUI anzuzeigen benötigt die Activity ein Layout. Das Layout wird i.d.R. in *onCreate()* mit dem API *setContentView()* gesetzt. Layouts werden in XML Resourcen spezifiziert. Ein Layout enthält hierarchisch angeordnete *Views* (dazu gehören Buttons, TextViews, Checkboxes, ProgressBars, ImageViews usw.). Views erhalten Attribute, die ihre Größe und Position im Layout und ihr Erscheinungsbild festlegen. Außerdem erhalten sie ein ID-Attribut, mit dem in der Activity auf sie zugegriffen werden kann.

Das API *findViewById()* liefert eine Referenz auf die entsprechende View-Instanz. Diese kann nun programmatisch verändert werden. Views implementieren das Observer-Pattern und ermöglichen damit ereignis-orientierte Programmierung. Die Klasse *View* (bzw. davon abgeleitete Klassen) enthalten die APIs *set[Ereignis]Listener()*. Diesen Methoden werden spezielle Listener-Objekte übergeben, die Methoden enthalten, die bei dem entsprechenden Ereignis aufgerufen werden.

3.1.3 Kommunikation zwischen Activities

Zu einem Zeitpunkt ist i.d.R. nur eine Activity aktiv. Beim Wechsel von einer Activity zu einer anderen müssen jedoch Informationen übergeben werden können. Der Wechsel erfolgt zentral durch das API *startActivity()*. Die Methode erhält ein sogenanntes *Intent*-Objekt. Dieses erhält die Informationen in serialisierter Form. Übergibt man ein Java-Objekt, muss dieses also das *Serializable*-Interface implementieren.

3.1.4 Activities in FancyQuartett

In unserer App verwenden wir 8 Activities, deren Struktur Abbildung 3.2 zeigt. Diese soll hier kurz erläutert werden.

Unsere *MainActivity* besteht aus 3 Fragmenten. Fragmente sind *Sub-Activities*, die in eine Activity eingebettet sind und haben ihren jeweils eigenen Lifecycle. Sie können beispielsweise in einem sogenannten *ViewPager* als Tabs verwendet werden.

Das erste Fragment dient als GUI zum starten eines Spiels. Startet der Benutzer ein neues Spiel, wird zur *NewGameSettingsActivity* gewechselt. Diese dient zum wählen eines Decks und der Spieleinstellungen. Zum wählen eines Decks wird die *NewGameGalleryActivity* gestartet, die alle verfügbaren Decks (sowohl offline als auch online) anzeigt. Nach dem Klick auf Start wird zur *GameActivity* gewechselt. Diese stellt die aktuelle Karte dar und instanziiert die Klasse *GameEngine*, welche die Ausführung der Spiellogik übernimmt.

Das zweite Fragment stellt die Deck-Gallerie dar. Bei der Auswahl eines Decks wird die *CardActivity* gestartet, die alle Karten des Decks in einer *ListView* anzeigt. Wird eine Karte ausgewählt wird die *CardViewerActivity* gestartet, die die Karte mit ihren Bildern und Attributwerten in einem *ViewPager* anzeigt. Durch *Swipen* nach links bzw. nach rechts kann zur vorherigen bzw. nächsten Karte des Decks navigiert werden.

Das dritte Fragment zeigt eine Statistik an und bietet keine Interaktionsmöglichkeiten.

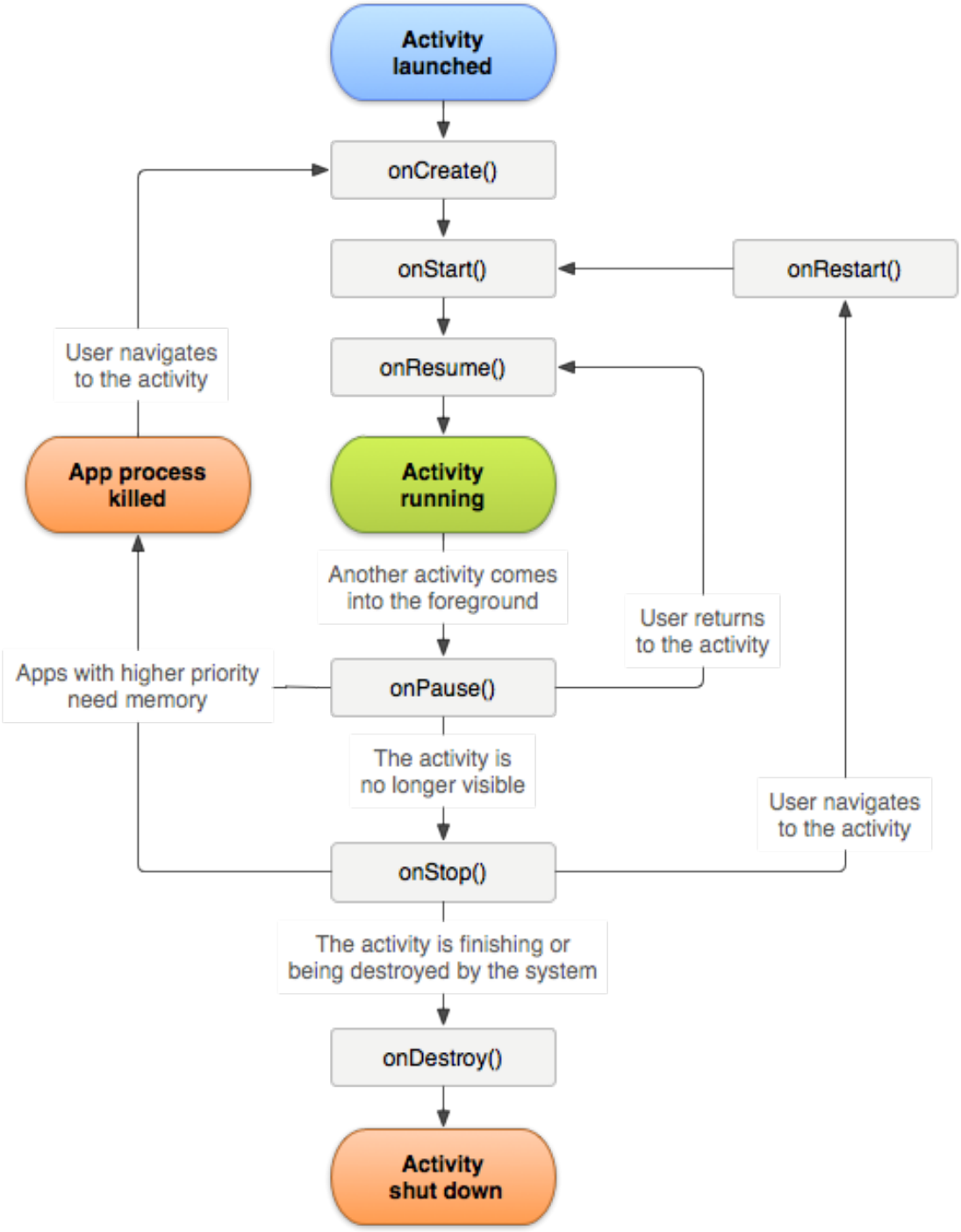

Abbildung 3.1: *Activity Lifecycle*

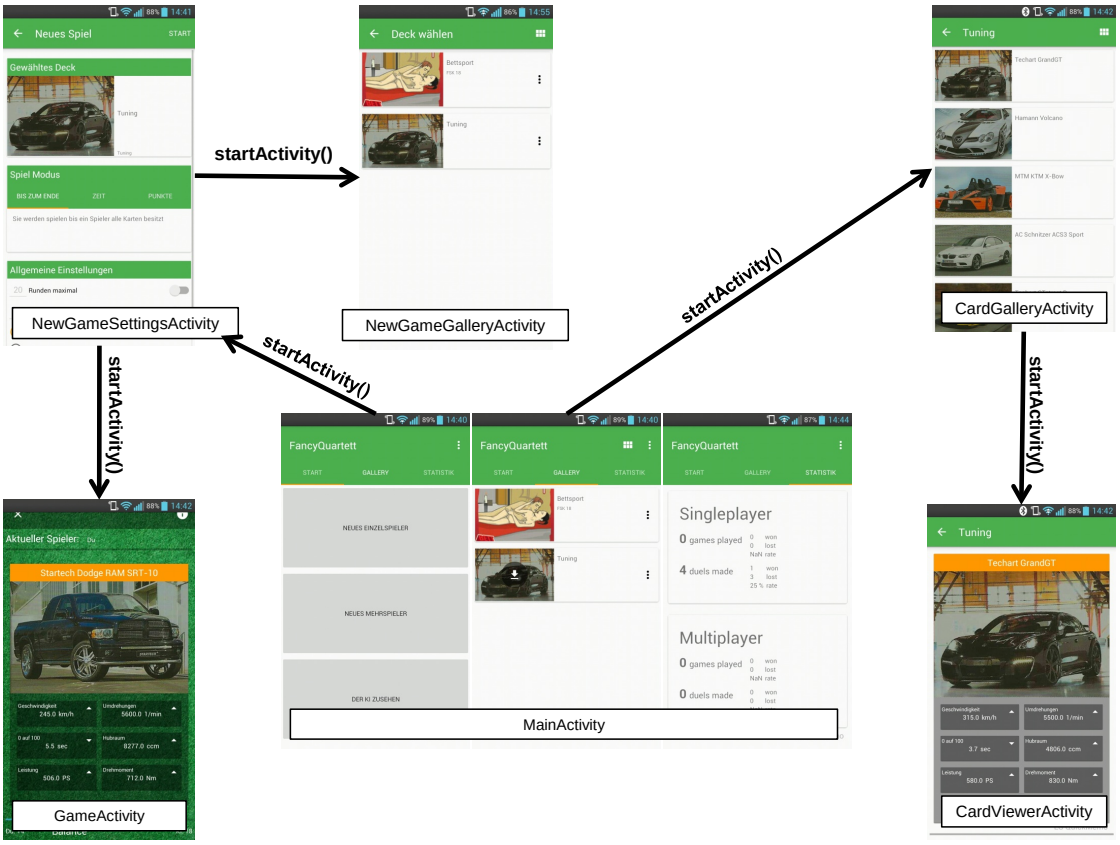

Abbildung 3.2: Die Activities in FancyQuartett

3.2 Datenmodell

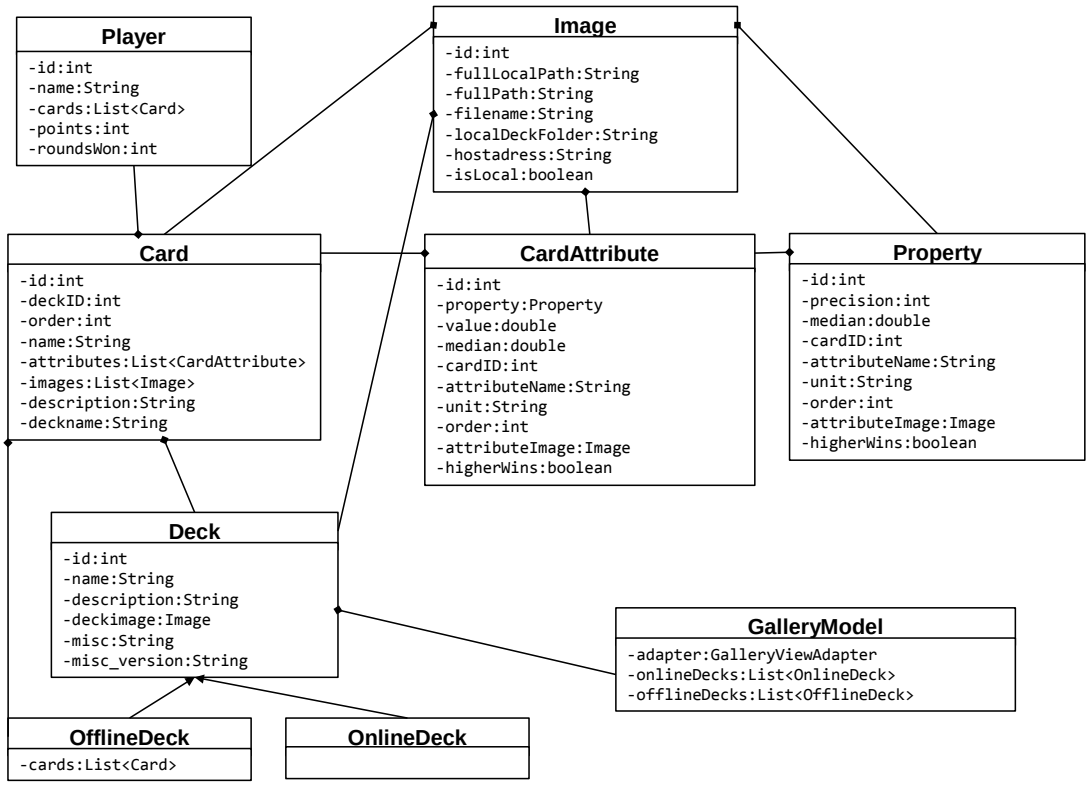

Abbildung 3.3: Datenmodell von FancyQuartett

Das UML-Klassendiagramm in Abbildung 3.3 zeigt unsere Realisierung der Datenbasis von FancyQuartett. In dieser Form werden die Daten zur Laufzeit gehalten.

Die persistente Speicherung erfolgt auf dem Dateisystem. Android weist jeder App einen Bereich im Dateisystem zu. Dort legen wir mittels der Klasse *AssetsInstaller* für jedes heruntergeladene Deck einen Ordner an, in dem wir eine JSON Datei und die zu den Karten gehörenden Bilder speichern. AssetsInstaller erbt von dem Android API *AsyncTask*. Das ermöglicht es, die Ein-/Ausgabe-Operationen in einem seperaten Thread auszuführen, um das GUI nicht zu blockieren.

3.3 Netzwerkfunktionen

Für das Laden von Decks über das Netzwerk wurde uns ein Server mit einer REST-API zur Verfügung gestellt, der Daten im JSON-Format sendet. Von diesem konnten folgende Ressourcen abgefragt werden:

- /**decks** liefert ein Array aus Objects. Die Anzahl der Objects entpricht den verfügbaren Decks. Die Objects enthalten jeweils die für ein Deck relevanten Informationen, die den Attributen im obigen Klassendiagramm in Abbildung 3.3 entsprechen.
- /**decks**/**<ID>** liefert das Deck-Object mit der entprechenden ID.
- /**decks**/**<ID>**/**cards** liefert ein Array aus Objects, die jeweils die ID und den Namen einer Karte enthalten.
- /**decks**/**<ID>**/**cards**/**<ID>** liefert das Card-Object mit der entsprechenden ID. Die Attribute in diesem Object nicht enthalten.
- /**decks**/**<ID>**/**cards**/**<ID>**/**attributes** liefert ein Array aus Objects, die jeweils die für ein Attribut und dessen Wert relevanten Informationen enthalten.

Das Senden der HTTP-Requests wurde durch das Java-API *HttpURLConnection* unterstützt. Durch den Aufruf von *openConnection()* wird ein korrekt geformter HTTP-Request automatisch gesendet und die empfangenen Daten in einen *InputStream* geschrieben. Aus diesem kann zu einem beliebigen späteren Zeitpunkt gelesen werden. Die Instanz von HttpURLConnection muss aber noch zugreifbar sein.

Die APIs im Package *org.json* machen das Handling der empfangenen JSON-Daten einfach. Ein *JSONObject* bzw. *JSONArray* kann direkt aus einem ASCII-String erstellt werden. Die Konvertierung in das interne Datenformat (siehe Abbildung 3.3) erfolgt 1:1 durch Lesen des entsprechenden Wertes aus dem JSONObject und Schreiben in ein gleich aufgebautes Java-Objekt.

Der Download eines gesamten Decks erfolgt sequentiell Karte für Karte. Für jede Karte werden ebenfalls sequentiell, die zugehörigen Bilder heruntergeladen und im Dateisystem gespeichert. Bilder werden zwischen Beendigung des Downloads und Schreiben ins Dateisystem als *Bitmap* gehalten. Ein InputStream kann von der Android-API *Bit-*

mapFactory über die Methode *decodeStream()* in ein Bitmap umgewandelt werden. Für den Download von Decks sorgt die Klasse *DeckDownloader*, die von *AsyncTask* erbt und den Downloadvorgang nebenläufig zum GUI-Thread ausführt.

4

Implementierung

5

Anforderungsvergleich

6

Fazit

Abbildungsverzeichnis

Tabellenverzeichnis