**Deckblatt**

**Inhaltsverzeichnis** Seite

1. Die Idee hinter der App

2. Vorteile der App gegenüber herkömmlicher Methoden

3. Die App und ihre Zustände

3.1 Übersicht

4. Die Funktionen der App und deren Umsetzung

4.1 Login und Registartion

4.2 Das Steckbriefkonzept

4.3 Das Freunde-System und die Chat Funktion

5 Der Lebenszyklus einer App

5.1 Der Lebenszyklus einer Activity

5.2 Activity-Stack

6 Aufbau des Servers

6.1 SQL

6.1.1 Struktur der Datenbank

6.1.2 PreparedStatements und Statements

6.2 Jetty

6.2.1 Starten des Webservers

6.2.2 XML

7 Server-Client Kommunikation

7.1 Client

7.1.2 Request und Response

7.2 Server

7.1.2 Request und Response

8. Danksagung

9. Literaturverzeichnis

10. Eidesstattliche Erklärung

Verschlüsselung /Er Diagramm/ Zyklen/ neuer aufruf

**1. Die Idee hinter der App**

Ein großer Teil der Jugendlichen hat ab dem 14ten Lebensjahr schon einmal, zumindest einen, Anfängerkurs für Standardtänze belegt. Bevor das Belegen eines solchen Kurses jedoch möglich ist, muss erst ein passender Tanzpartner gefunden werden. Die Vermittlung kann Bspw. über den Ausbildungsplatz wie z.B. Gymnasien, Realschulen oder Unis organisiert sein. Das Finden eines Tanzpartners über den Freundeskreis ist auch sehr weit verbreitet. Für Jugendliche, bei denen genannte Methoden entweder nicht zur Verfügung standen oder erfolglos waren, insbesondere infolge einer Knappheit an tanzwilligen Jungen, gab es an der Tanzschule, an der ich einst lernte, eine weitere Vermittlungsmethode.

Über eine Pinnwand mit öffentlich aushängenden Steckbriefen konnten die Suchenden Kontaktdaten austauschen.

Das Ziel meiner Tanzschul App ist es die Funktionen dieser Pinnwand in eine Server-Client basierende App zu integrieren, diese zu erweitern und neue Features hinzuzufügen.

Wobei der Server die Pinnwand darstellt, auf der die Steckbriefe der User gespeichert werden.

Kombiniert mit einem User-System, damit Nutzer ihre Steckbriefe leichter anpassen löschen oder erstellen können. Der Client kann dann die gespeicherten Steckbriefe vom Server abrufen und danach auslesen. Oder auch Änderungen an den Steckbriefen vornehmen indem er dem Server neue Daten sendet

**2. Vorteile der App gegenüber herkömmlicher Methoden**

Die Tanzschul App übernimmt das Prinz einer wie vorhin beschrieben Pinnwand nicht nur, sie bringt viele Vorteile und Verbesserungen mit sich.

Dank eines Servers werden alle wichtigen Daten, die die App zum Funktionieren benötigt online abgespeichert. Das führt dazu, dass die Applikation jederzeit und von überall verwendet werden kann.

Steckbriefe können schnell und bequem von unterwegs erstellt oder bei Bedarf angepasst werden. Zudem können Änderungen an der „Pinnwand“ des Servers jederzeit und von überall von der App abgerufen werden.

Alle wichtigen Daten werden serverseitig in einer SQL Datenbank abgespeichert, was praktisches Filtern der zum Client gesendeten Steckbriefe, nach den Wünschen des Users, ermöglicht.

Zudem ist bei meiner App die Kontaktaufnahme erleichtert. Rufnummern anderer User können

im Geräte eigenen Telefonbuch abgespeichert oder in die Zwischenablage kopiert werden.

Für Nutzer, die lieber eine Nachricht hinterlassen wollen gibt es einen App eigenen Chat.

**3. Die App und ihre Zustände**

Die Tanzschul App weist für jede ihre Zustände eine eigene Activity auf, diese ist dafür zuständig die Benutzeroberfläche zu definieren und alle Daten die für potentielle Benutzer Interaktionen benötigt werden zu verwalten.[[1]](#footnote-1)

Ein Konstrukt aus mehreren Activities hat mehrere Vorteile gegenüber einem Neuzeichnen der [GUI](https://developer.android.com/guide/topics/ui/index.html) (Graphical User Interface) innerhalb einer einzigen [Activity-Klasse](https://developer.android.com/reference/android/app/Activity.html), für jeden Wechsel des Zustands. Bei letzterem müssen nämlich mehrere Layouts innerhalb einer Klasse implementiert werden.

Das führt dazu, dass der Code insgesamt komplizierter gestaltet ist, um die verschiedenen Zustände der App umzusetzen und voneinander abzugrenzen zu können. Beispielsweise muss bei jedem Zustandswechsel unnötiger Ressourcenspeicher freigegeben und Verweise auf nichtmehr zeichenbare GUI-Elemente gelöscht werden.

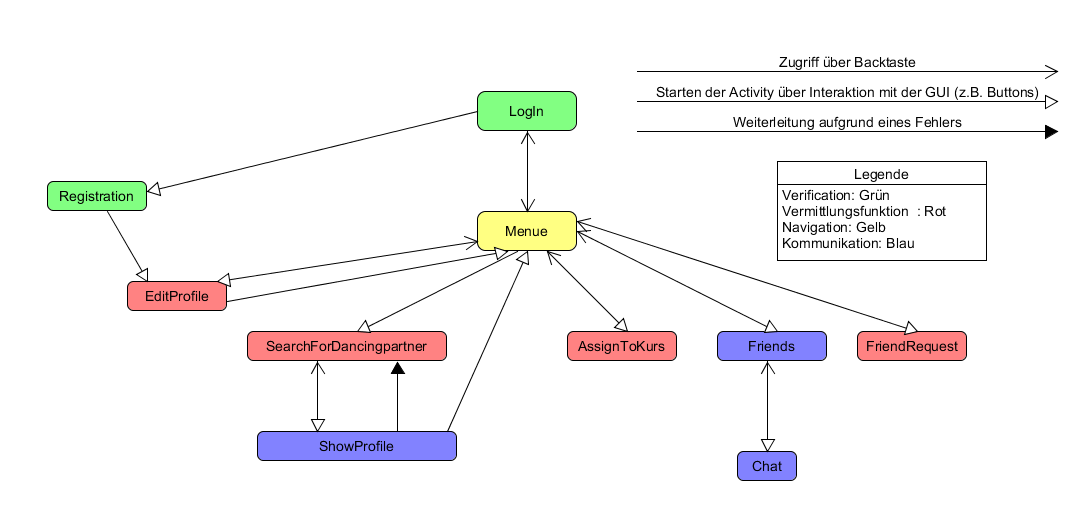
Des Weiteren führte die Umsetzung von mehreren Benutzeroberflächen in einer Klasse zu unnötig langem Code und würde somit die Verständlichkeit und Übersichtlichkeit dessen stark verschlechtern.

Zu guter Letzt müssten bei der neuzeichnenden Variante sehr viele Attribute in der Activity-Klasse und so im Arbeitsspeicher des Geräts abgespeichert werden die nur in bestimmten Zuständen der App überhaupt Verwendung finden.

Beim Verwenden mehrerer Activitys ist die Abgrenzung untereinander deutlich einfacher. Denn in diesem Fall kann der vordefinierte [Lebenszyklus](https://developer.android.com/training/basics/activity-lifecycle/index.html) einer Activity verwendet werden. Dieser bring Methoden wie *onPause*, *onStop* oder *onDestroy* mit sich in denen nicht mehr verwendet Ressourcen freigegeben werden können. Diese werden aufgerufen, wenn die Activity, neugezeichnet, temporär in den Hintergrund verschoben oder beendet wird. Beim Starten einer neuen Activity können wichtige Informationen über Intents weitergegeben werden.

Informationen die auch nach dem Beenden der App vorhanden sein sollen können extern z.B. in sog. [SharedPreferrences](https://developer.android.com/reference/android/content/SharedPreferences.html) gespeichert werden.

**3.1 Überblick über die Zustände der App**

****

Diese Grafik zeigt die in der Tanzschul App vorhandenen Aktivities und so auch alle Zustände. Die Pfeile veranschaulichen, durch welche Art von User Interaktion der Zustandswechsel ausgelöst wurde und so auch eine neue Aktiviy in den Vordergrund rückt.

Alle grün markierten Aktivitys sind dienen zur Nutzer Verifikation. Jeder User hat eine individuelle ID die nur er selbst kennt, diese kann er bei den grün markierten Activities zugesandt bekommen.

Die gelb markierte Activity dient zur Navigation zwischen den unterschiedlichen Funktionen der App. Die rot markierten Acivitys sind sozusagen das Herz der App sie sind sie Implementieren die Funktionen die zur eigentlichen Vermittlung beitragen.

Zur direkten oder indirekten Interaktion zwischen Usern dienen die blau gefärbten Activitys.

**4. Die Funktionen der App und ihre Umsetzung**

Im Vorausgehenden wurde die Strukturelle Anordnung der Aktivities veranschaulicht. In diesem Teil wird auf die Aufgaben und die Entstehungsgedanken der einzelnen Activities eingegangen. Zu aller erst jedoch einmal zur so genannten *ConnectedActivity*. Sie ist eine abstrakte Klasse die von allen Activitys erweitert werden soll, die auf die Internetressourcen des Endgeräts zugreifen oder deren Existenz überprüfen. Das sind zu dem jetzigen Stand der App alle Activitys.

Die Methoden der *ConnectedActivity* sollendie Implementierung der Server-Client Kommunikation erleichtern, dazu zählt ein einheitliches Fehlerbehandlungssystem.

Nun noch genaueres zur Fehler Behandlung bevor wir zu den eigentliche Activitys kommen, es gibt im Großen und Ganzen zwei Typen von Fehlern, die in meinem Konzept erfasst werden können. Zum einen Verbindungsfehler, diese können durch eine nicht vorhandene oder zu langsame der Internetverbindung des Geräts oder durch ein Nichtantworten des Servers ausgelöst werden. In diesem Fall wird die Verbindung des Endgeräts geprüft und eine passende Fehlermeldung ausgelesen je nachdem ob die Activity generell auf eine Internetressource zugreifen kann oder nicht.

Die zweite Möglichkeit stellen Fehler Codes die vom Server in einer Response erhalten werden dar. Bei Eingabefehlern wird je nach Fehlercode eine passende Fehlermeldung ausgelesen. Wohingegen bei unbekannten oder fatalen Fehlern zur Login-Activity zurückgekehrt wird um mögliche Folgefehler zu vermeiden.

Anfangs bestand die Idee beide Arten von Fehlern in einer Methode zu verarbeiten. Jedoch wird die Methode *onConnectionError()*, die für das Behandeln von Verbindungsfehlern zuständig ist meist von den Kinderklassen übernommen wohingegen die *onError()* Methode in einigen Fällen zur individuellen Reaktion auf Fehlercodes überschrieben werden muss.

**4.1 Login Funktion**

Im ersten Konzept meiner App war keine Login-Funktion oder gar eine Account-System geplant. Schnell wurde jedoch klar, dass es ohne ein solches schwierig werden würde, Funktionen wie einen Chat zu implementieren oder die „Pinnwand“ vor Spam zu schützen.

Auf Basis dieser Überlegungen entstand die LogIn-Activity, mit dieser kann der Nutzer seine individuelle Identifikationsnummer (ID) im Austausch seiner passenden Kombination von E-Mail und Passwort vom Server abrufen. Die ID muss bei jedem Request zur Überprüfung und Identifikation des Clients an den Server gesandt werden. Nutzer die noch keinen Account besitzen werden über den „Registrieren“-Button zur Registration-Activity weitergeleitet.

Besonders wichtig war es mir hier, den Login für den User bequem zu gestalten, deshalb wurde eine Funktion hinzugefügt mit der man E-Mail und Kennwort speichern kann. Um das zu realisieren, müssen die beiden Parameter extern gespeichert werden, da nach dem Schließen der App alle nicht gespeicherten Informationen verloren gehen.

Dabei bieten sich mehrere Möglichkeiten, beispielsweiße könnte man eine eigene SQLite Datenbank anlegen um die Werte zu speichern. Das würde jedoch einen unnötigen Programmier -und Rechenaufwand bedeuten. Für das Speichern einer kleinen Anzahl von sog. „key-values“ bieten sich die SharedPreferences an, die genau dafür ausgelegt sind wenige Datensätze primitiven Datentyps oder Strings in Datenpaaren zu speichern.[[2]](#footnote-2) Möchte man komplexe Datentypen speichern ist eine SQLite Datenbank besser geeignet.[[3]](#footnote-3)

Kommen wir zur Registration-Activity der schwierigste Teil des Codes liegt darin, die vom Nutzer eingegebenen Daten auf theoretische Richtigkeit zu überprüfen und passende Fehlermeldungen auszulesen. Letztendlich habe ich die Überprüfung mit einem *boolean array* umgesetzt, das für jede zu erfüllender Bedingung ein item aufweist, nur wenn alle Items des Arrays den Wert true haben wird dem Sever eine Anfrage zum Erstellen eines neuen Accounts zugesandt. Dabei ist anzumerken, dass sowohl Client als auch Server seitig eine passende Überprüfung der Gültigkeit einer E-Mail fehlt.

**4.2 Das Steckbriefkonzept**

Jeder Nutzer besitzt ein Profil, dieses enthält alle persönlichen Daten die über den Nutzer gespeichert werden, es kann jederzeit über die EditProfile-Activity angepasst werden. Über die SearchForDancinpartner-Activity kann der Nutzer die Steckbriefe nach Kursstufe und dem Wochentag an dem der gewünschte Tanzkurs stattfinden soll filtern. Ich habe für die Stufenbezeichnungen der Grundkurse von der Tanzschule Fischer Ingolstadt, an der ich einst selbst Schüler war übernommen. Die höheren Stufen entsprechen dem einheitlichen Medaillensystem der [ADTV](http://www.adtv.de/) geprüften Tanzschulen. Der Wochentag bezeichnet den Tag in dem die wöchentlichen stattfindenden Kurse gehalten werden.

Die Steckbriefe (nur die des anderen Geschlechts) anderer User werden vom Server abgerufen und in einer ListView angezeigt. Klickt man auf eine der ListView-Items, kann man das Profil des Nutzers der diesen Steckbrief erstellt hat einsehen. Möchte man Steckbriefe von sich selbst für bestimmte Kurse hinzufügen oder löschen, kann man das in der AssingToKurs-Activity getan werden.

**4.3 Das Freunde-System und die Chatfunktion**

dsfdf

**3. Der Server**

Auf dem Server werden alle möglichen Daten über die User, Kurse, Admins und deren Beziehungen untereinander gespeichert. Öffentliche Daten werden auf Anfrage an die Nutzer gesendet.

**3.1 Die Serverstruktur**

**6. Danksagungen**

Abschließend möchte ich allen Seminarmitgliedern für ihre Offenheit und Hilfsbereitschaft danken. Insbesondere Tim Möschel der mir seine App als Serverseitiges Vorbild zeigte und Dominik Okwieka, Leander Dreier, Oskar Loeprecht und Klaus Hackner die mir jeder Zeit mit Rat beiseite standen.

**7. Literaturverzeichnis**

ANDROID Open Source Project: *Activity*, https://developer.android.com/reference/android/app/Activity.html, aufgerufen am 22.10.2016

ANDROID Open Source Project: *Saving Key-Value Sets*, https://developer.android.com/training/basics/data-storage/shared-preferences.html, aufgerufen am 22.10.2016

RACHITA NANDA Androidexterous: *Difference between sqlite and shared preferences in android,* https://blograchita.wordpress.com/2013/05/26/difference-between-sqlite-and-shared-preferences-in-android/ am 23.10.2016

**8. Eidesstattliche Erklärung**

Ich erkläre, dass ich die Seminararbeit nicht ohne fremde Hilfe angefertigt und nur die im Literaturverzeichnis aufgeführten Quellen und Hilfsmittel verwendet habe.

Kösching, den 10. November 2016

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Abgabe nach den Ferien 8.11.16

**2. Das Grundprinzip der Tanzschulapp**

Jeder neue Nutzer muss sich vor der Nutzung der App registrieren und einen neuen Account erstellen. Mit diesem kann er sich dann sich dann einloggen um auf weitere Funktionen der App zugreifen. Jeder Nutzer hat ein eigenes Profil dass er nach seinen Vorstellungen anpassen kann. Wenn der User, Nutzern des anderen Geschlechts mitteilen möchte, dass er nach einen Tanzpartner für einen bestimmten Kurs sucht kann er sich für diesen suchend eintragen. Als Alternative dazu kann er sich Nutzer die das bereits getan haben nach Kursstufe und Wochentag sortiert vom Server abrufen. Anschließen kann er das Profil jedes ausgegeben Benutzers einsehen. Im Profil Tab werden alle vom fremden Nutzer angegeben Informationen angezeigt. Insofern man nun Kontakt aufnehmen möchte hat man mehrere verschiedene Möglichkeiten das zu tun darunter auch ein App eigener Chat.

Da die Kurse nicht in jeder Saison zum gleichen Zeitpunkt sattfinden muss es für die Inhaber einer Tanzschule, die meine App verwendet, möglich sein neue Kurse hinzuzufügen und veraltete zu löschen. Zu diesem Zweck habe ich noch eine Admin App programmiert.

**8. Das Grundprinzip der Tanzschul App**

1. Vgl. ADROID, *Activity* [↑](#footnote-ref-1)
2. Vgl. RACHITA NANDA [↑](#footnote-ref-2)
3. Vgl. ANDROID, *Saving Key-Value Sets* [↑](#footnote-ref-3)