Android-Go-App
Implementierungsbericht

Dennis Bäuml,
Theresa Heine,
Victoria Karl,
Tarek Wilkening

Betreuer:
Erik Burger,
Axel Busch,
Heiko Klare

19. Februar 2017

Inhaltsverzeichnis

1 Einleitung

Die Anwendungslogik bildet die eigentliche Kommunikationsschnittstelle zur onlinen Datenverwaltung. Dazu gibt die Anwendungslogik die Ein/Ausgabedaten der View per Intent an den NetworkIntentService weiter. Dieser kümmert sich darum, dass Anfragen vom Client im Hintergrund bearbeitet und an den Server gesendet werden. Die Antworten vom Server werden von Recievern auf dem Client empfangen. Ist dieser erfolgreich, so speichert die Anwendungslogik auch die Daten in der Datenbank des Clients. Einzelne wichtige Daten wie UserName, UserId und den Gruppenname der Gruppe in der man sich aktuell befindet werden in SharedPreferences gespeichert und weitergegeben.

Auf der android Datenbank gibt es zwei Tabellen. Die eine speichert die Gruppen und den dazugehörigen Treffpunkt und die andere die User und in welchen Gruppen diese Mitglied sind. Dabei werden nur die Gruppen und zugehörigen Mitglieder gespeichert in denen der aktuelle User auch Mitglied ist.

Die Klassen der Anwendungslogik wurden komplett in JAVA implementiert. Auf dem Client wurden die Klassen aufgeteilt in Model, View, Controller und Communication. Auf dem Server wurden die Klassen aufgeteilt in blablabla hier muss Tarek was schreiben Da es sich um eine objektorientierte Programmierung handelt, wurden die einzelnen Funktionen der Anwendungslogik in geeignete Klassen aufgeteilt. So entstanden einige Klassen, die jede einen Teil der Realisierung übernimmt.

Die Klassen der Anwedungslogik wurden in zwei Paketen gruppiert: *.al und *.beans mit jeweiligen Klassen:

2 Pflicht- und Wunschkriterien

In diesem Kapitel gehen wir auf die Pflicht- und Wunschkriterien ein, die zum Einen in der GoApp fertig implementiert sind und zum Anderen nur auf dem Server und/ oder nur auf dem Client funktionieren. Bei dem zweiten Punkt fehlt die Kommunikation zwischen Client und Server. Die Server Requests werden noch nicht aufgerufen und es existieren noch keine Reciever die diese empfangen.

2.1 Umgesetze Pflichtkriterien

1. Benutzer registrieren ist möglich.

2. Eine neue Gruppe erstellen is möglich.

3. Für eine Gruppe kann ein neuer Treffpunkt und dafür Uhrzeit, Datum und ein Zielort festgelegt werden.

4. Für eine Gruppe lässt sich ein Link erstellen und diesen kann man per externen Messenger an Freunde als Einladung versenden.

5. Ein Link lässt sich mit der GoApp öffnen und man wird der Gruppe hinzugefügt.

2.2 Implementierte Funktionen des Clients

Benutzer erstellen/ löschen In der App lassen sich Benutzer erstellen und löschen. Beim Erstellen wird die Gültigkeit des Namens überprüft. Löschen interagiert jedoch noch nicht mit dem Server und findet nur auf der android Datenbank statt.

Benutzer umbenennen (Wunschkriterium) Diese Funktion wird bisher nur auf der android Datenbank ausgeführt ohne Interaktion mit dem Server.

Gruppen erstellen/ löschen Gruppen lassen sich erstellen und werden auch auf dem Server erstellt. Die Anzahl der zu erstellenden Gruppen pro Benutzer ist jedoch noch nicht begrenzt. Genauso wenig wie die maximale Anzahl von Mitgliedern pro Gruppe. Löscht man eine Gruppe, passiert das nur auf dem Client.

Gruppe: Treffpunk festlegen Treffpunkte können von Administratoren gesetzt werden und sind nach einem GroupUpdate für alle Mitglieder sichtbar. GroupUpdate funktioniert noch nicht, wenn Mitglieder die Gruppe verlassen oder die Gruppe gelöscht wurde.

Es funktioniert nur, wenn neue Mitglieder hinzukommen, sich die Administratorrechte geändert haben oder ein Mitglied seinen Namen geändert hat.

Gruppe: Benutzer einladen / entfernen Der empfangene Link wird automatisch mit der GoApp geöffnet und man wird der Gruppe hinzugefügt. Hat man die GoApp nicht installiert kommt man auf eine Website die einen dazu auffordert die GoApp zu installieren. Jeder Link ist nur einmal gültig, also kann nur ein Mitglied zu einer Gruppe hinzufügen. Benutzer entfernen funktioniert bisher nur auf dem Client.

Gruppe: Mitglieder zu Administratoren machen (Wunschkriterium) Der Gruppenadministrator kann ein Gruppenmitglied zum Administrator machen. Dies passiert jedoch nur auf dem Client und nicht mit der Interaktion mit dem Server.

Gruppe: beitreten / verlassen Einer Gruppe kann man über Link beitreten. Das verlassen einer Gruppe funktioniert nur auf dem Client und nicht mit Interaktion mit dem Server. Mitglieder entfernen funktioniert auf dem Client noch gar nicht.

Gruppe: GPS Daten mitteilen / empfangen Funktioniert leider noch nicht. Der GoIntentService selbst läuft aber wir haben es leider nicht geschafft, dass die Standorte abgerufen und angezeigt werden können. Für Standorte die wir nicht über den GoService in die Karte eingefügt haben wurden diese angezeigt und bei mehreren Standorten wurden diese zusammengefasst. Wenn man weiter reinzoomed, dann werden zusammengefasste Positionen wieder getrennt. Das einzige Problem liegt dabei alles miteinander zu verbinden.

Gruppe: Umbenennen (Wunschkriterium) Eine Gruppe kann man auf dem Client umbenennen aber nicht in Interaktion mit dem Server.

3 Abweichungen zum Entwurf

In diesem Kapitel werden alle Änderungen aufgezählt, welche sich in der Implementierungsphase zum Entwurf ergeben haben.

3.1 Serverseitige Abweichungen

3.2 Implementierte Funktionen des Servers

/FS010/ Benutzerkonten verwalten

Über *RegistrationRequest* kann man ein neues Benutzerkonto auf dem Server anlegen. Das Benutzerkonto wird an die Geräte-ID des Smartphones gebunden und in der Datenbank des Servers hinterlegt.
Mit der *DeleteUserRequest* kann der zuvor erstellte Account gelöscht werden.

/FS170/ Umbenennen des Benutzers

Zusätzlich kann der Benutzer über die *RenameUserRequest* seinen Nutzernamen ändern.

/FS030/ Gruppen erstellen / löschen

Ein registrierter Benutzer kann via *CreateGroupRequest* Gruppen auf dem Server anlegen und per *DeleteGroupRequest* wieder löschen.
Der Benutzer ist automatisch in der eigens erstellten Gruppe Admin.

/FS040/ Gruppe: Benutzer einladen / entfernen

Mit der *CreateLinkRequest* kann ein Gruppenadmin einen Einladungslink erstellen. Das im Link enthaltene *'Secret'* wird in der Gruppe auf dem Server hinterlegt.
Über die *KickMemberRequest* kann ein Gruppenmitglied aus der Gruppe entfernt werden.

/FS180/ Gruppe: Mitglieder zu Administratoren machen

Mit der *MakeAdminRequest* können Mitglieder zu Admins befördert werden.

/FS060/ Gruppe: beitreten / verlassen

Ein anderer Benutzer kann mit einer *JoinGroupRequest* einer Gruppe beitreten.
Das zum Beitreten benutzer *'Secret'* wird aus der Gruppe vom Server gelöscht.
Eine Gruppe kann mit der *LeaveGroupRequest* verlassen werden.

/FS090/ Gruppe: Treffpunk festlegen

Mit der *SetAppointmentRequest* kann der Gruppenadmin einen Treffpunkt festlegen. Dieser wird in der Server-Datenbank gespeichert, wo er von anderen Gruppenmitgliedern abgerufen werden kann.

/FS110/ Gruppe: GPS Daten mitteilen / empfangen

Per *BroadcastGpsRequest* kann ein Gruppenmitglied der Gruppe seinen Standort mitteilen. Gleichzeitig erhält er, verpackt in einer *ObjectResponse*, die Standorte der anderen Gruppenmitglieder.

/FS190/ Gruppe: Umbenennen

Analog zur *RenameUserRequest*, kann man mit der *RenameGroupRequest*
Den Gruppennamen ändern (der Name muss eindeutig sein).

/FS130/ Verschlüsselung

Transportverschlüsselung wird über HTTPS garantiert.

3.3 Abweichungen vom Entwurf

3.3.1 Request / Response

Da wird (leider erst nach der Entwurfsphase) festgestellt haben,
dass der Jackson ObjectMapper über Annotations auch Polymorphismus unterstützt,
ist ein Großteil der Logik in die Requests gewandert.
Um uns an das Command-pattern zu halten, haben wir und dazu entschieden,
den Requests eine execute() Methode hinzuzufügen. Somit sind Daten und Logik
zusammen gekapselt. Das Pattern erschien anfangs hinderlich, da man,
sobald sich das Model änderte, alle Requests anpassen musste.
Als das Model jedoch fertig war, wurde es sehr einfach, neue Requests für neue
Aufgaben hinzuzufügen.

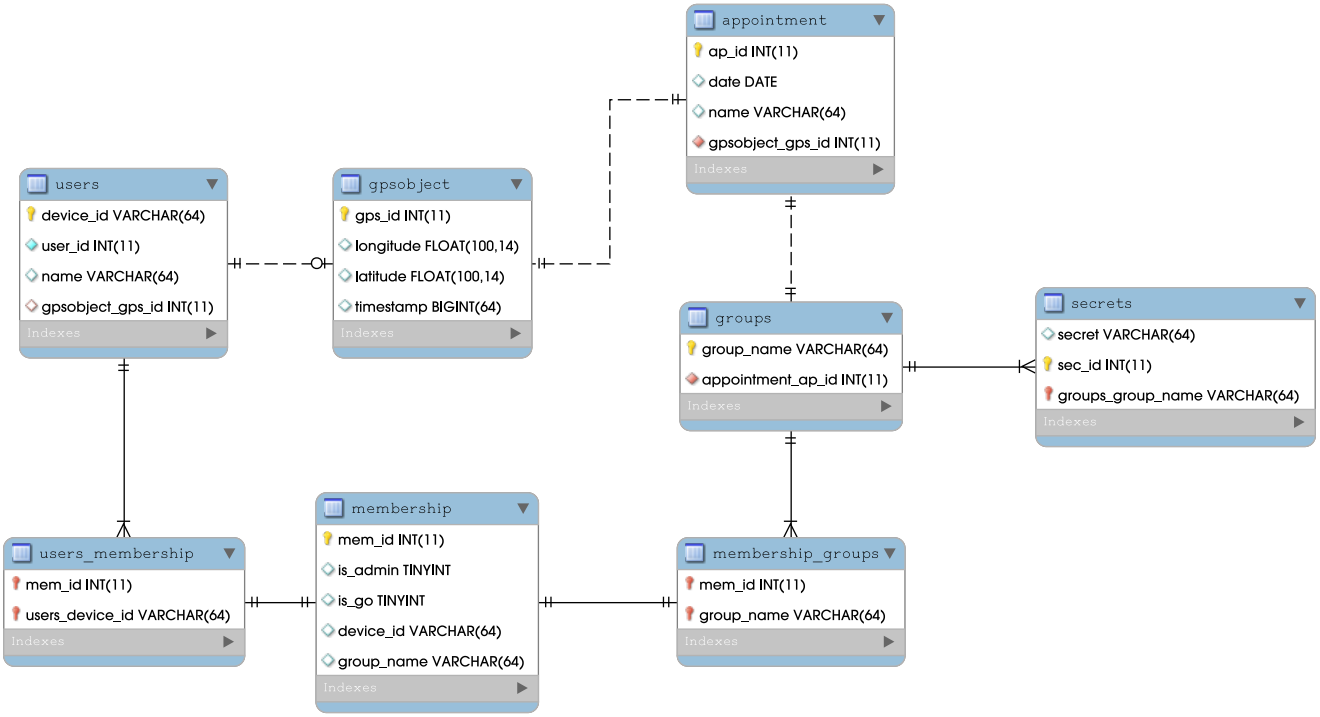
Die Response Klassen waren, von Anfang bis Ende, nur Container für Parameter.

Zum Vereinfachen haben wir sie zu Response und ObjectResponse zusammengefasst.
In einer ObjectResponse können beliebige Objekte die Serializable implementieren, verpackt werden.
Das Serializable Interface ist dabei nur für das Abbilden auf Subklassen zuständig, die der ObjectMapper sonst nicht de-serialisieren könnte.
Die Requests sind immernoch grob in Group- und User-Requests unterteilt.

3.3.2 Servlets

Die Umstrukturierungen des Servers, hinsichtlich der Requests, haben dazu geführt, dass nur noch ein HttpServlet nötig war das MainServlet.
Es serialisiert/deserialisiert Request und Response Objekte bearbeitet Http-Anfragen.
Die GoApp kommuniziert ausschließlich über POST-Requests mit dem Server.
Über eine GET-Anfrage mit beispielsweise einem Webbrowser,
erhält man nur die Aufforderung, die App zu installieren.

3.4 Datenbank-Anbindung Server



Da unser ausgefallenes Gruppenmitglied der Meinung war, dass sich durch die Datenbank nichts an unserem Entwurf ändert, ist es leider auch nicht mit im Entwurf vorhanden. Da das Persistieren einer geschachtelten HashMap nicht ohne weiteres möglich ist, erschien es uns einfacher eine neue Klasse einzuführen, welche die Beziehung zwischen Benutzer und Gruppe modelliert. So entstand eine neue Tabelle Membership und die zugehörige Klasse MemberAssociation. Sie stellt eine ManyToMany Beziehung zwischen Gruppe und Benutzer dar.
OneToOne Beziehungen existieren ausserdem zwischen Group und Appointment, Appointment und GpsObject, sowie SimpleUser und GpsObject.
Die SSecrets"der Gruppen mussten als Collection ebenfalls in eine neue Tabelle ausgelagert werden.

Die Schnittstellen UserManager und GroupManager wurden zu ResourceManager zusammengelegt.
Diese zusätzliche Indirektion hat sich als nützlich erwiesen und macht die Datenbank zudem austauschbar.

Durch die MemberAssociations wurde auch das UserDecorator-Pattern überflüssig.
Darin sind die Attribute gespeichert, die die Beziehung zwischen Benutzer und Gruppe beschreiben (Rechte, Go-Status etc.).

3.5 Clientseitige Abweichungen

communication

package: *.client.communication

1. Alle Antworten vom Server konnten zu Response und ObjectResonse zusammengefasst werden. Der Vorteil von ObjectResponse ist, dass alle möglichen Objekte darin verpackt und verschickt werden können.

2. Zum Serialisieren brauchten wir Wrapper Klassen, weshalb es nun mehrere Serializable Klassen gibt. Dies war uns beim Entwurf noch nicht bewusst und ist erst in der Implementierung aufgefallen.

controller

package: *.client.controller.database

1. Da es weder FeedEntryAllocation noch FeedEntryAppointment mehr gibt, gibt es auch keinen AllocationService und keinen AppointmentService mehr (hießen davor ServiceAllocation und ServiceAppointment).

2. GroupService und UserService haben noch ein paar weitere Methoden erhalten um das Arbeiten mit der android Datenbank zu erleichtern. Zum Beispiel deleteAll() um alle Informationen von der Tabelle zu löschen.

3. insertData() in GroupService und UserService geben nicht mehr einen boolean zurück sondern sind nur void.

package: *.client.controller.objectStructure

1. In den Methoden von AccountHandler und GroupHandler müssen je die Activity in der die Methode aufgerufen werden als Methodenparameter mitgegeben werden, da man sonst keine Request starten kann.

2. In GroupHandler kam ein neuer Constructor hinzu. Bei diesem werden alle Informationen einer bereits existierenden Gruppe umbergeben, damit man diese erstellen kann und die Argumente als Methodenparameter mitgibt.

model

package: *.client.model.database

1. FeedEntryAllocation und FeedEntryAppointment wurden entfernt und in FeedEntryUser und FeedEntryGroup integriert. So ist die Datenbank besser erweiterbar, da weniger Tabellen angepasst werden müssen. Die Standorte von Usern werden nicht mehr auf der android Datenbank gespeichert sondern wenn der GoStatus aktiv ist regelmäßig vom Server abgerufen.

2. DBHelperAllocation, DBHelperAppointment und DBHelperUser existieren nicht mehr sondern es gibt nur noch DBHelperGroup. Das ist ein Fehler, der beim Entwurf nicht aufgefallen ist. Davor waren es mehrere Datenbanken mit je einer Tabelle. Jetzt ist es eine Datenbank mit mehreren Tabellen.

package: *client.model.objectStructure

1. Der GroupClient wird nicht mehr über eine groupId sondern nur über seinen Namen identifiziert.

2. In SimpleUser wird im Constructor nicht mehr die deviceId sondern die userId übergeben, da die deviceId direk abgerufen wird und die userId vom Server übermittelt wird.

3. UserDecoratorClient hat noch ein zusätzliches Attribut dafür erhalten, ob dasjenige Mitglied Admin oder nur GroupMember ist, um diese Information aus dem Object auslesen zu können um es in der Datenbank zu speichern.

4. In GroupClient müssen allen Methoden die Activity mitgegeben werden, damit diese einen Request an den Server starten können. Die meisten anderen Methodenparameter sind dabei weggefallen. Die getMember() wurde zu getMemberType() geändert und gibt nun einen boolean statt einem UserDecoratorClient zurück.

5. In AppointmentDate wurde das Format von Datum und Uhrzeit von Date zu String geändert, da so unnötige Typumwandlungen im Code vermieden werden können.

view

package: *client.view

1. Die Base Activity wurde komplett neu erstellt. Ist zuständig für den Navigation-Drawer und alle Verknüpfungen, die dieser erstellt. Die Ausführung von diesem haben wir im Entwurf aus zeitlichen/gesundheitlichen Gründen nicht rechtzeitig kommuniziert bekommen. Die GroupActivity erbt von dieser.

2. Die Methode showDatePickerDialog(Viwe) (analog für Time) des DatePickerFragment und TimePickerFragment sind ins GroupAppointmentFragment verschoben worden, wo sie nun private sind. Diese öffnen das jeweilige Fragment in dem Uhrzeit und Datum gewählt werden können.

3. GroupActivity: onBackPressed():void, wurde @Overrid.Die Methode ist dafür da, wenn man in einem Fragment A ist und in einanderes Fragment B weitergeleitet wird, dass durch das klicken des zurück Kopfes man nicht in die letzte Activity kommt, sondern dass man von dem Fragment B wieder zurück in das Vorherige (hier A) kommt. Bezüglich dem Wechseln der Fragments gab es oft Schwierigkeiten, deshalb wurde diese Methode überschrieben.

   onStart():void, wurde @Override. Hier wurde die Response bezüglich dem beitreten eines Nutzers in eine Gruppe mittels klicken des Links.

   onDestroy():void, wurde @Override .In der onDestroy wird der Receiver für Join-Group mit Link, also der Receiver der in der onStart Methode implementiert wird, wird hier geschlossen, damit dieser nicht andere Respones abfängt.

4. In GroupAppointmentFragment, GroupMapGoFragment, GroupMembersFragment, GroupnameCreateFragment und UsernameRegistrationFragment wurden onAttach und onDetach hinzugefügt, da diese für die Reciever der Responses des Servers benötigt werden.

5. GroupMapFragment:

   +group:GroupClient, wurde protected, damit die davon erbenden Fragments GroupMapGoFragment und GroupMapNotGoFragment auf die Variable zugreifen können, und diese verändern

   +setMyLocation(boolean):void, wurde hinzugefügt Wird in der GroupMapGoFragment gerufen, damit der eigene Standort ermittelt wird. Da der eigene Standort in die Karte eingezeichnet werden muss, muss die Methode in der GroupMapFragment implementiert werden und wird nur in der GroupMapGoFragment gerufen.

Die Vererbung erlaubt es nicht, auf die Map in der Oberklasse zu bearbeiten, um somit das einzeichnen des Standortes zu vereinfachen und effektiv zu machen, haben wir uns auf diese Methode geeinigt.

+setMyGroupmemberLocation(LinkedList<GpsObject>):void, wurde hinzugefügt. Die Methode implementiert die Möglichkeit eine Liste von GpsObjects in Clustern zusammenzufassen und diese dann als blaue Kreise mit einer Zahl, welche der Anzahl der zusammengefassten Standorte repräsentiert, in die Karte einzeichnet. Die Methode sorgt auch, dass die Punkte welche zusammengefasst sind, beim Rein und Raus Zoomen wieder zusammengefasst beziehungsweise getrennt in der Karte eingezeichnet werden.

+onResume():void, wurde hinzugefügt Die Methode implementier nur, dass wenn man zurück in das Fragment kommt, die Karte von OpenStreetMap die default Einstellungen übernehmen soll, falls keine hinterlegt wurden. startService:void, wurde protected, damit das davon erbende Fragment GroupMapGoFragment diese implementieren kann

6. GroupMapGoFragment startService():void, wurde protected, damit von diesem Fragment aus der GoIntentService gestartet werden kann

7. GroupMembersFragment +onCreateOptionsMenu(Menu, MenuInflater):void, wurde hinzugefügt

8. MemberAdapter extends RecyclerView, damit der Client nicht alle Mitglieder auf einmal laden muss +static class PersonViewHolder extends RecyclerView.ViewHolder Hier wird klar getrennt wie was formatiert wird. In dem RecyclerView kann man über CardView einstellen wie jedes Element formatiert und angezeigt wird. In dem Adapter werden sowohl die notwendigen Daten geholt, ausgetauscht und jedes Element formatiert.

9. PlacePickerFragment extends Fragment wurde komplett hinzugefügt. Ursprünglicher Plan war es, diesen über das GroupMapFragment laufen zu lassen. Aber aus Gründen der stilistischen objektorientierten Implementierung, haben wir dafür eine neue Klasse erstellt

10. UsernameActivity +onBackPressed():void, wurde hinzugefügt

4 Verzögerugen bei der Implementierung

Zu Beginn der Implementierungsphase waren wir noch relativ gut im Zeitplan. Leider hat sich das relativ schnell in eine andere Richtung entwickelt. Zum Einen hatten wir gehofft, dass Matthias sich um Hibernate kümmert und weitere Aufgaben auf dem Server übernimmt. Zum Anderen hatten wir auch erwartet, dass die View weniger Arbeit wäre und Android Studio weniger Probleme machen würde. Bei ungefähr der Hälfte der Implementierungsphase wurde dann auch noch ein Gruppenmitglied krank und konnte die rechte Hand kaum noch verwenden und die Woche darauf wurde ein anderes Mitglied krank, ohne dass wir etwas neues von unserem fünften Gruppenmitglied wussten. Auch wenn wir uns nicht darauf verlassen wollten hatten wir natürlich gehofft, dass sich Matthias an unserem Projekt noch beteiligen würde. Was wir in der letzten Woche total unterschätzt hatten war die Interaktion zwischen Server und Client. Da Matthias nun offiziell nicht mehr Mitglied unserer Gruppe ist, musste Hibernate noch implementiert werden und solange konnten wir auch die Reciever im Client nicht testen. Als wir soweit waren, dass die ersten Requests and den Server gesendet werden konnten ist uns aufgefallen, dass Reciever alle Responses abfangen die in die App eingehen. Das hat natürlich zu vielen Problemen geführt die leider nicht alle behoben werden konnten. Wir hatten für einen sehr großen und uns allen unbekannten Teil (also wie genau Client und Server interagieren) zu wenig Zeit, um ein ordentliches Ergebnis liefern zu können. Hat ein Teil funktioniert, hat ein anderer nicht mehr funktioniert und da es am Anfang hieß, dass Matthias sich um das Testen des Codes kümmern sollte, haben wir kaum Unit Test implementiert, was natürlich die Systeminteraktion erschwert. Fehler zu finden war sehr aufwändig. Und wenn der Server aktualisiert wurde konnte man solange natürlich nicht testen, ob die Reciever funktionieren und mit der Response umgehen können.

Also zusammengefasst lief es am Anfang ganz gut und wir kamen voran aber die einzelnen Teile die jeder für sich implementiert hat dann zu einem Großen zusammen zu fügen hat uns sehr viele Nerven und Zeit gekostet.

5 Unit Tests

Wegen der Aufgabenverschiebung, entstanden durch den Wegfall eines Gruppenmitgliedes, blieb leider keine Zeit mehr sich um Unittestszu kümmern. Dementsprechend konnten Unit Tests nur für den GroupServer geschrieben werden. Dadurch dass wir teilweise nicht genau wussten, wie wir z.B. die android Datenbank testen können, da wir diese simulieren hätten müssen und das gleichzeitig auch noch mit den Activites zusammenhängt, haben wir useren Code nur durch Ausprobieren von verschiedenen Eingaben getestet. Es werden auch kaum Exceptions oder ungültige Eingaben abgefangen. In erster Linie haben wir versucht unsere App bei korrekten Eingaben zum Laufen zu bekommen.