Android-Go-App Entwurfsdokument

Dennis Bäuml,
Matthias Dressel,
Theresa Heine,
Victoria Karl,
Tarek Wilkening

Betreuer:
Erik Burger,
Axel Busch,
Heiko Klare

15. Januar 2017

Inhaltsverzeichnis

1 Grobentwurf **3**
- 1.1 Architekturstil . . . 4
  - 1.1.1 Clientseitig . . . 4
  - 1.1.2 Serverseitig . . . 4
- 1.2 Paketdiagramm . . . 4

2 Feinentwurf **5**
- 2.1 Clientseitig . . . 6
  - 2.1.1 ClientView . . . 6
  - 2.1.2 ClientModel . . . 24
    - 2.1.2.1 Database . . . 25
    - 2.1.2.2 ObjectStructure . . . 30
  - 2.1.3 ClientController . . . 38
    - 2.1.3.1 Database . . . 39
    - 2.1.3.2 ObjectStructure . . . 42
- 2.2 Serverseitig . . . 43
  - 2.2.1 Servlets . . . 43
  - 2.2.2 Kommunikation . . . 47
  - 2.2.3 Server-Modell . . . 52

3 Sequenzdiagramme **60**

4 Änderungen zum Pflichtenheft **64**

1 Grobentwurf

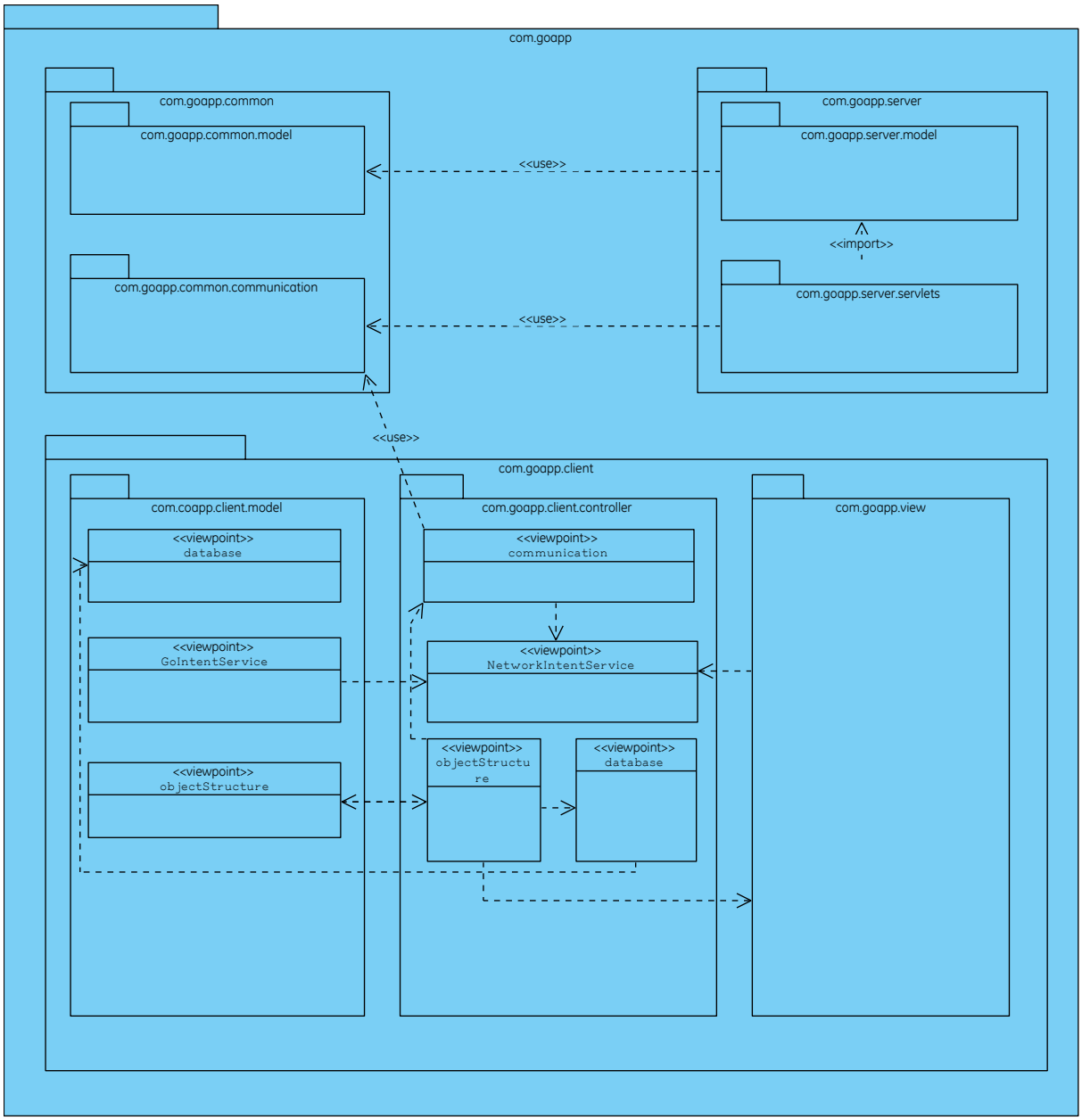


Abbildung 1: Paketdiagramm Gesamt

1.1 Architekturstil

1.1.1 Clientseitig

–> MVC nicht ganz eindeutig Model und Controler wegen Activities und Fragments überschneiden sich

1.1.2 Serverseitig

1.2 Paketdiagramm

2 Feinentwurf

Im folgenden Kapitel erhalten Sie eine Übersicht über die auf dem Client und dem Server verwendeten Klassen, welche die GoApp umsetzen sollen.

Um eine bessere Vorstellung davon zu bekommen, was unter sich immer wiederholenden Variablen verstanden wird, möchte ich diese an dieser Stelle vorstellen und erklären:

1. GroupId: Eine GroupId ist eine neunstellige Integerzahl, welche mit einer 1 beginnt. Jede GroupId existiert genau einmal und macht es möglich, Gruppen eindeutig zu identifizieren.

2. UserId: Eine UserId ist wie die GruppenId eine neunstellige Integerzahl, welche nicht mit einer 1 beginnt. Jede UserId existiert genau einmal und macht es möglich, Benutzer eindeutig zu identifizieren.

3. GroupName: Ein Gruppenname kann bis zu 30 Zeichen lang sein und aus Groß- und Kleinbuchstaben bestehen und Sonderzeichen beinhalten. Jeder Gruppenname muss eindeutig sein.

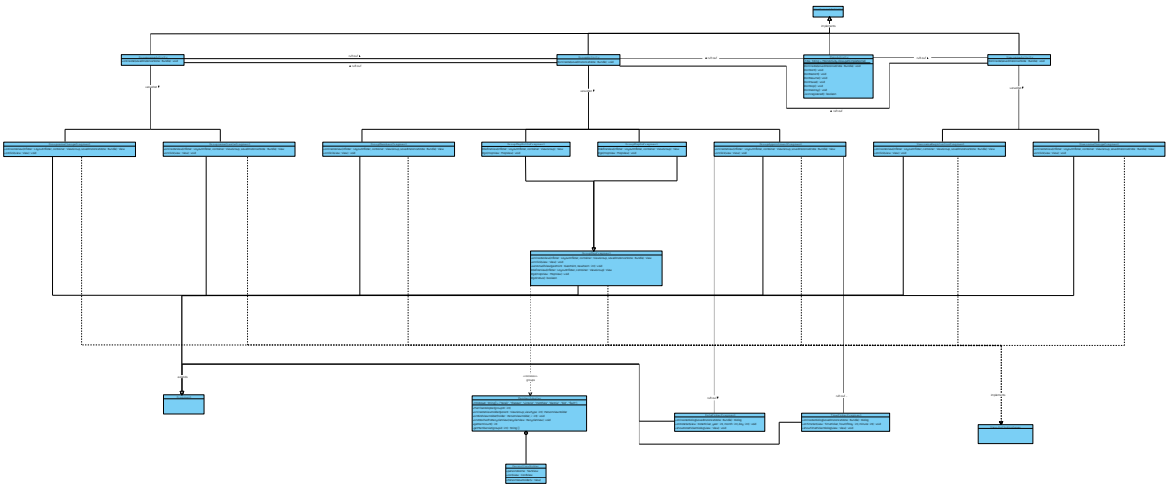
4. UserName: Ein Benutzername kann wie der Gruppenname bis zu 30 Zeichen lang sein und aus Groß- und Kleinbuchstaben bestehen. Im Gegensatz zur Gruppe muss dieser Name nicht eindeutig sein.

5. date: Das Datum eines Treffpunktes wird in dem Format dd.MM.yyyy dargstellt.

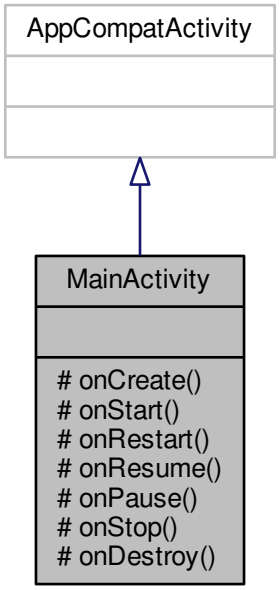
6. time: Die Uhrzeit eines Treffpunktes wird in dem Format MM:HH dargestellt.

2.1 Clientseitig

2.1.1 ClientView



1. **MainActivity**



Aufgrund der MVC Struktur der App ist die MainActivity die Hauptactivity des View Teil. Beim Öffnen der App auf dem Client wird diese als erste Activity geöffnet. Sie hat vor allem eine managende Funktion: sie überpfüft ob dieser Client schon registriert ist oder nicht. Sie erbt von der AppCompatActivity und implementiert dementsprechen auch deren Methoden.

Methoden

a) protected onCreate(Bundle savedInstanceState)

   Erweitert die onCreate Methode der AppCompatActivity indem sie, wenn der Client schon registriert ist, erst an die UsernameActivity weiterleitet. Ist er jedoch schon registriert, so leitet sie direkt an die zuletzt aufgerufene GroupActivity weiter.

b) protected onStart()

   Erweitert die onStart() Methode der AppCompatActivity

c) protected onRestart()

   Erweitert die onRestart() Methode der AppCompatActivity

d) protected onResume()

   Erweitert die onResume() Methode der AppCompatActivity

e) protected onPause()

   Erweitert die onPause() Methode der AppCompatActivity

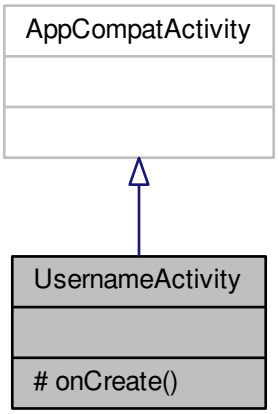
f) protected onStop()

   Erweitert die onStop() Methode der AppCompatActivity

g) protected onDestroy()

   Erweitert die onDesroy() Methode der AppCompatActivity

2. **UsernameActivity**



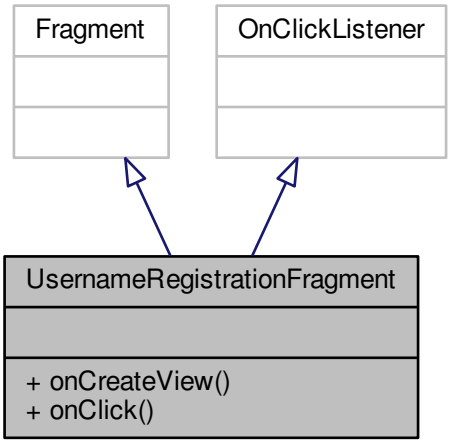
Die UsernameActivity ist für das benennen des Benutzernamens zuständig. Sie wird sowohl zum ersten Start der App von der MainActivity aufgerufen, als auch von von der GroupActivity, wenn der Benutzer auf den Benutzernamen tippt. Sie enthält das UsernameChangeFragment und das UsernameRegistrationFragment. Sie erbt von der AppCompatActivity und implementiert dementsprechen auch deren Methoden.

Methoden

a) protected onCreate(@Nullable Bundle savedInstanceState)

Erweitert die onCreate Methode der AppCompatActivity mit dem laden des UsernameChangeFragments.

3. **UsernameRegistrationFragment**



Das UsernameRegistrationFragment ist dafür zuständig einen neuen User auf dem Server anzulegen. Es wird auf jedem Client in den username_container der UsernameActivity geladen und somit nur einmalig aufgerufen, bis sich der Benutzer registriert hat. Es legt die erste Ansicht fest, die ein Benutzer sieht, wenn er die App das erste mal öffnet, bzw. wenn er die App öffnet und sich noch nie registriert hat. Es erbt von Fragment und implementiert den View.onClickListener. Dementsprechend implementiert sie auch deren Methoden.

Methoden

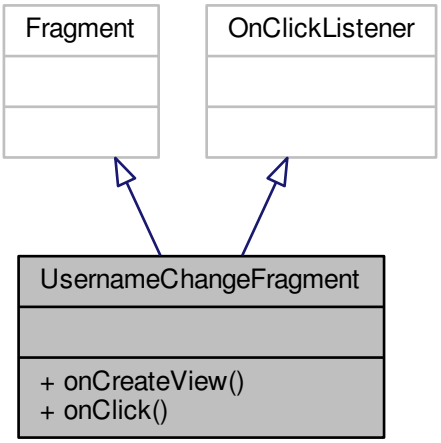
a) public onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState): View

   Erweitert die onCreateView Methode des Fragments mit der gewünschten Ansicht, die der View übergeben wird und fürgt dem OnClickListener den Button hinzu. Diese Methode gibt die aktuelle View zurück.

b) public onClick(View view)

   Implementiert die onClick Methode des OnClickListeners, so dass beim Klicken auf den Next-Button überprüft wird, ob der gewünschte Benutzername zugelassen ist. In diesem Fall legt er einen neuen User an und leitet an eine leere GroupActivity weiter.

4. **UsernameChangeFragment**



Das UsernameChangeFragment ist dafür zuständig den Username zu ändern. Es wird von der UsernameActivity in den username_container geladen. Es legt die Ansicht fest, die ein Benutzer sieht, wenn er seinen Benutzernamen ändern möchte. Es erbt von Fragment und implementiert den View.onClickListener. Dementsprechend implementiert es auch deren Methoden.

Methoden

a) public onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState): View

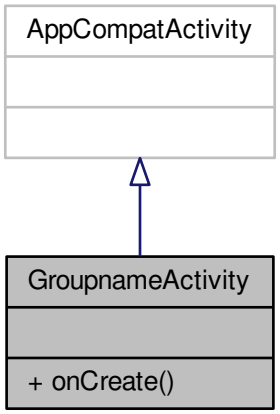
   Erweitert die onCreateView Methode des Fragments mit der gewünschten Ansicht, die der View übergeben wird und fügt dem OnClickListener den Button hinzu, wenn der Benutzer die App nicht zum ersten Mal öffnet. In diesem Fall lädt es das UsernameRegistrationFragment in den username_container der UsernameActivity. Diese Methode gibt die aktuelle View zurück.

b) public onClick(View view)

   Implementiert die onClick Methode des OnClickListeners, so dass beim Klick auf dem Next-Button überprüft wird, ob der gewünschte neue Benutzerna-

me zugelassen ist. In diesem Fall ändert er diesen und leitet an die zuletzt aufgerufene GroupActivity weiter.

5. **GroupActivity**



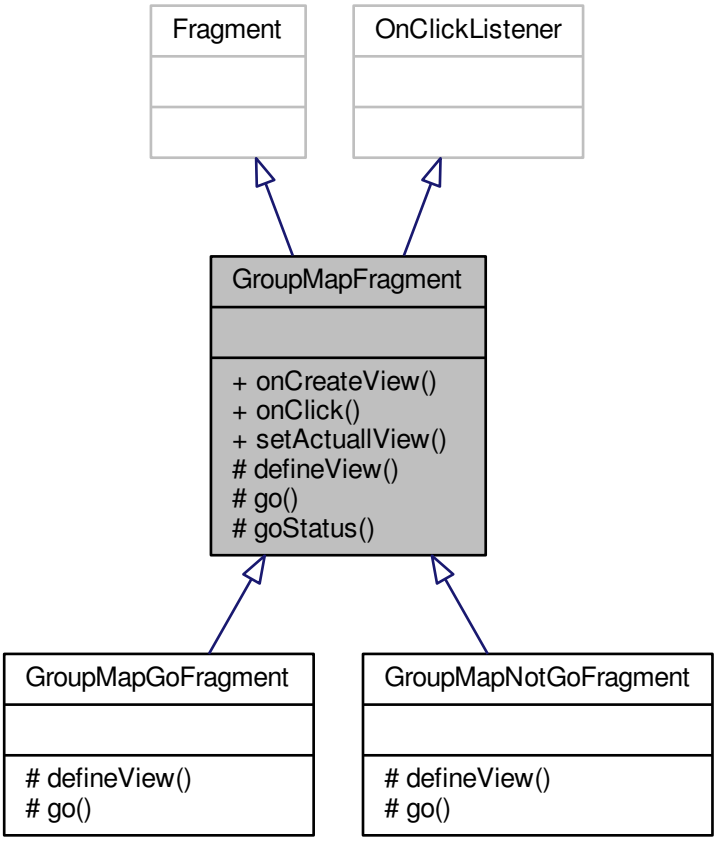
GroupActivity ist für das Management der Gruppe zuständig. Sie ist die Activity, die die meiste Zeit geöffnet ist. Von allen anderen Activitys aus kann sie aus geöffnet werden. Sie enthält das GroupMapFragment, das GroupMapGoFragment und das GroupmembersFragment. Sie erbt von der AppCompatActivity und implementiert dementsprechen auch deren Methoden.

Methoden

a) protected onCreate(@Nullable Bundle savedInstanceState

   Erweitert die onCreate Methode der AppCompatActivity mit dem laden des GroupMapFragments in den group_container .

6. **GroupMapFragment**



Das GroupMapFragment ist dafür zuständig, die Map-Ansicht anzuzeigen. Allerdings hat es lediglich die Funktion die Oberklasse für das GroupMapGoFragment und das GroupMapNotGoFragment zu sein, wird also niemals direkt aufgerufen. Es erbt von Fragment und implementiert den View.onClickListener. Dementsprechend implementiert es auch deren Methoden.

Methoden

a) public onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState): View

   Erweitert die onCreateView Methode des Fragments mit der gewünschten Ansicht, indem es defineView() aufruft und den controller der MapView definiert.

Außerdem fügt es dem OnClickListener die Button hinzu.Diese Methode gibt die aktuelle View zurück.

b) protected defineView(LayoutInflater inflater, ViewGroup container): View

Gibt die gewünschte View zurück. Allerdings ist diese Methode hier lediglich ein Platzhalter für detaillierte Methoden.

c) public onClick(View view)

Implementiert die onClick Methode des OnClickListeners, so dass er beim Klick auf den Gruppennamen das GroupMembersFragment in den group_container der GroupActivity lädt, beim Klick auf das Datum, wenn man Gruppenadministrator ist, das GroupAppointmentFragment in den group_container der GroupActivity lädt und beim Klick auf den Go-Button die go() Methode aufruft.

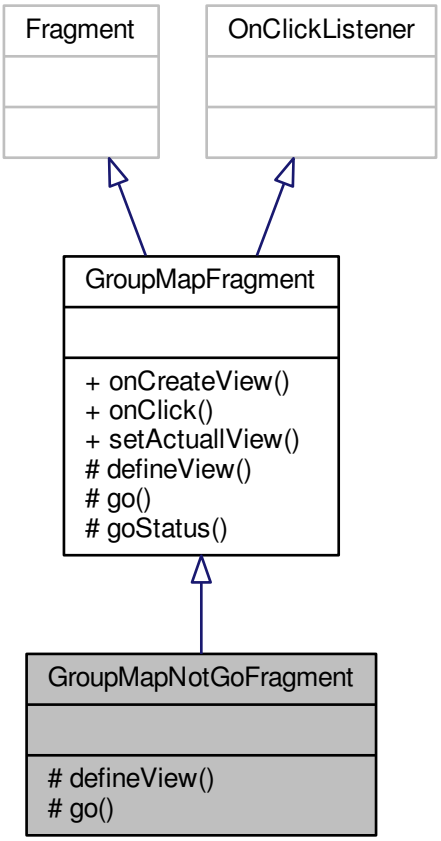
d) protected go(MapView mapView)

Definiert was passiert, wenn der Benutzer den Go-Button drückt. Allerdings ist diese Methode hier lediglich ein Platzhalter für detaillierte Methoden.

e) public setActualView(IGeoPoint geoPoint, int newZoom)

Speichert die aktuelle Zoom- und Fokus-Einstellung eines anderen GroupMapFragments

7. **GroupMapNotGoFragment**



Das GroupMapNotGoFragment ist dafür zuständig, die Map-Ansicht anzuzeigen, wenn der Go-Button nicht gedrückt ist. Es wird von der GroupActivity in den group_container geladen, immer dann wenn eine Gruppe aufgerufen wird. Es erbt von dem GroupMapFragment und implementiert dementsprechen auch dessen Methoden.

Methoden

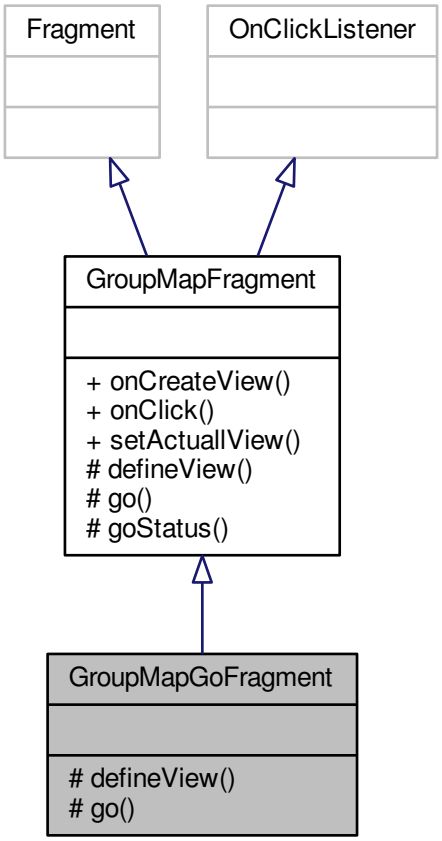
a) protected defineView(LayoutInflater inflater, ViewGroup container): View

   Gibt die gewünschte View zurück. Wenn jedoch der Go-Button gedrückt ist, lädt es das GroupMapGoFragment in den group_container der GroupActivity.

b) protected go(MapView mapView)

Lädt das GroupMapGoFragment in den group_container der GroupActivity.

8. **GroupMapGoFragmen**



Das GroupMapGoFragment ist dafür zuständig, die Map-Ansicht anzuzeigen, wenn der Go Button gedrückt ist. In diesem Fall wird es von dem GroupMapNotGo-Fragment in den group_container geladen, wenn der Benutzer go drückt, bzw. go gedrückt hat. Es erbt von dem GroupMapFragment und implementiert dementsprechend auch dessen Methoden.

Methoden

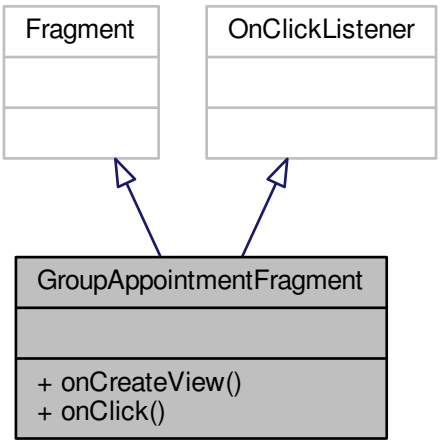
a) protected defineView(LayoutInflater inflater, ViewGroup container): View

   Gibt die gewünschte View zurück.

b) protected go(MapView mapView)

   Lädt das GroupMapNotGoFragment in den group_container der GroupActivity.

9. **GroupAppointmentFragment**



Das GroupAppointmentFragment ist dafür zuständig ein neues Treffen zu erstellen. Es wird von einem GroupFragment in den group_container geladen, wenn der Gruppenadministrator auf den Appointment-Button drückt. Es erbt von Fragment und implementiert den View.onClickListener. Dementsprechend implementiert es auch deren Methoden.

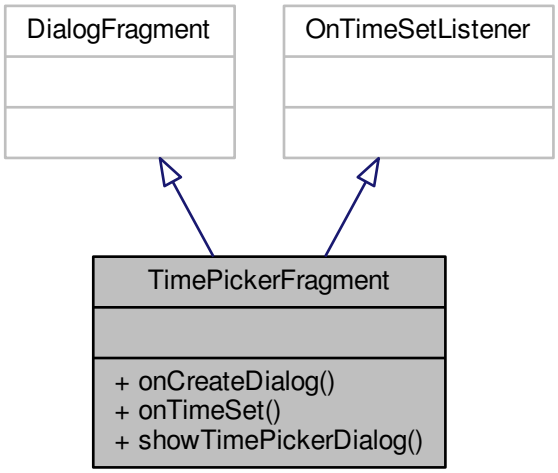
Methoden

a) public onCreateView(LayoutInflater inflater, @Nullable ViewGroup container, @Nullable Bundle savedInstanceState): View Gibt die gewünschte View zurück und fügt dem On.Click.Listener die Button hinzu.

b) public onClick(View view)

Implementiert die onClick Methode des OnClickListeners, so dass er beim Klick auf den Gruppennamen das GroupMembersFragment in den group_container der GroupActivity lädt, beim Klick auf das Datum das GroupMapGoFragment, bzw. das GroupMapNotGoFragment in den group_container der GroupActivity lädt, beim Klick auf die Time-, Date-, bzw. Place-Button das TimePickerFragment, DatePickerFragment, bzw. das GroupMapFragment lädt damit der Gruppenadministrator die Daten für ein Treffen auszuwählen und beim Klick auf den Next-Button dieses Appointment zu bestätigen kann. Dieses wird dann als nächstes Treffen in der Gruppe gespeichert.

10. **TimePickerFragment**



Das TimePickerFragment ist dafür zuständig eine Uhrzeit auszuwählen und diese zurückzugeben. Es wird von dem GroupAppointmentFragment geladen, wenn der Benutzer den Time-Button drückt. Es erbt vom DialogFragment und implementiert den TimePickerDialog.OnTimeSetListener. Dementsprechend implementiert es auch deren Methoden.

Methoden

a) public onCreateDialog(Bundle savedInstanceState): Dialog

Öffnet den TimePickerDialog

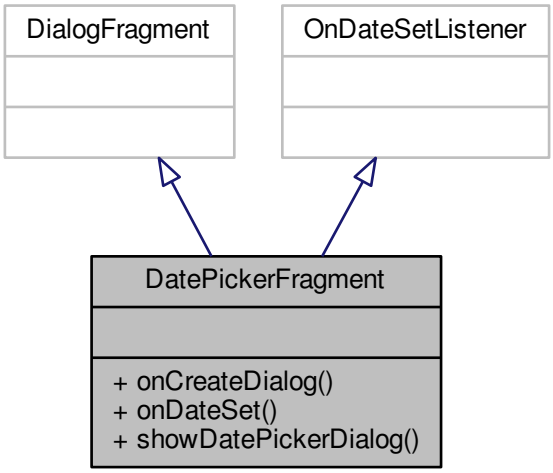
b) public onTimeSet(TimePicker view, int hourOfDay, int minute)

Speichert die gewählte Zeit in einem gesonderten Appointment

c) public showTimePickerDialog(View view)

Macht den TimePickerDialog sichtbar

11. **DatePickerFragment**



Das DatePickerFragment ist dafür zuständig ein Datum auszuwählen und dieses zurückzugeben. Es wird von dem GroupAppointmentFragment geladen, wenn der Benutzer den Date-Button drückt. Es erbt vom DialogFragment und implementiert den DatePickerDialog.OnDateSetListener. Dementsprechend implementiert es auch deren Methoden.

Methoden

a) public onCreateDialog(Bundle savedInstanceState): Dialog

Öffnet den DatePickerDialog

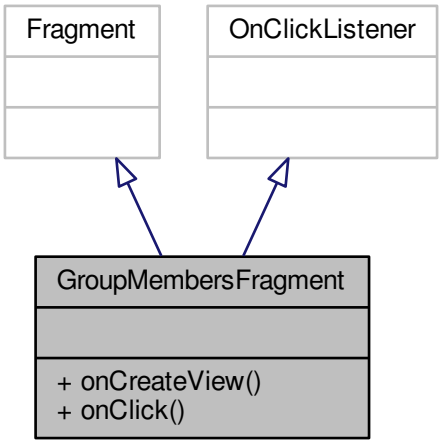
b) public onDateSet(DatePicker view, int year, int month, int day)

Speichert das gewählte Datum in einem gesonderten Appointment

c) public showDatePickerDialog(View view)

Macht den DatePickerDialog sichtbar

12. **GroupMembersFragment**



Das GroupMembersFragment ist dafür zuständig die Gruppenmitglieder zu verwalten. Es wird von einem GroupFragment in den group_container geladen, wenn der Benutzer auf den Groupname-Button drückt. Es erbt von Fragment und implementiert den View.onClickListener. Dementsprechend implementiert es auch deren Methoden.

Methoden

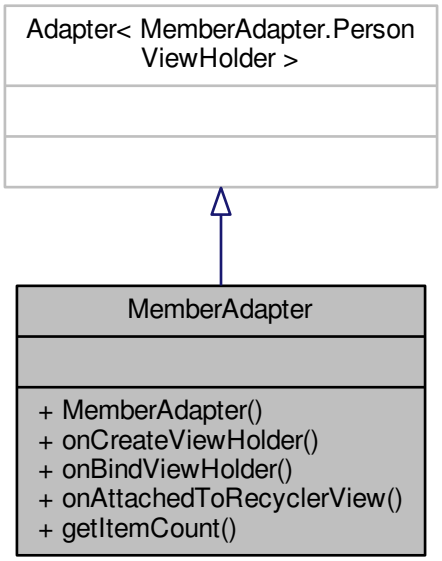
a) public onCreateView(LayoutInflater inflater, @Nullable ViewGroup container, @Nullable Bundle savedInstanceState): View

Erweitert die onCreateView Methode des Fragments mit der gewünschten Ansicht, je nachdem ob der Benutzer Gruppenadministrator dieser Gruppe ist oder nicht. Außerdem kreiert es eine RecyclerView, ruft den MemberAdapter auf um alle Gruppenmitglieder anzeigen zu können und fügt der OnClickListener die Button hinzu.

b) public onClick(View view)

Implementiert die onClick Methode des OnClickListeners, so dass er beim Klick auf den Gruppennamen das GrouMapNotGoFragment, bzw. das GroupMapGoFragment in den group_container der GroupActivity lädt, beim Klick auf das Datum das GroupAppointmentFragment, in den group_container der GroupActivity lädt und beim Klick auf den AddMember-Button den SendLinkDialog lädt.

13. **MemberAdapter**



Der MemberAdapter hat die Aufgabe die Liste der Mitglieder in die RecyclerView zu laden. Er wird von dem GroupMembersFragment aufgerufen. Er erbt von RecyclerView.Adapter<MemberAdapter.PersonViewHolder> und implementiert dementsprechend auch dessen Methoden. Außerdem kreiert er die public static class PersonViewHolder, die von RecyclerView.ViewHolder erbt und dafür zuständig ist den Inhalt der RecyclerView zu speichern.

Methoden

a) public MemberAdapter(int groupID)

   Stellt einen passenden Konstruktor bereit

b) public PersonViewHolder onCreateViewHolder(ViewGroup parent, int viewType)

   Erstellt eine neue View, wenn sie vom LayoutManager aufgerufen wird

c) public onBindViewHolder(PersonViewHolder holder, int i)

   Ersetzt den Inhalt der View, wenn sie vom LayoutManager aufgerufen wird

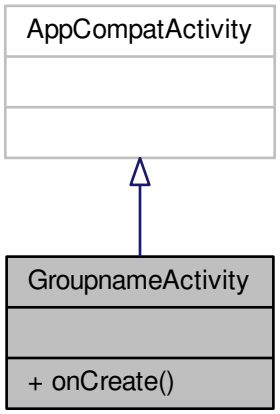
d) public onAttachedToRecyclerView(RecyclerView recyclerView)

   Ruft die selbe Methode der Oberklasse auf

e) public getItemCount(): int

   Gibt die Größe von dem Datensatz zurück, wenn sie vom LayoutManager aufgerufen wird

14. **GroupnameActivity**



Die GroupnameActivity ist für das Verwalten der Gruppe zuständig. Sie wird von

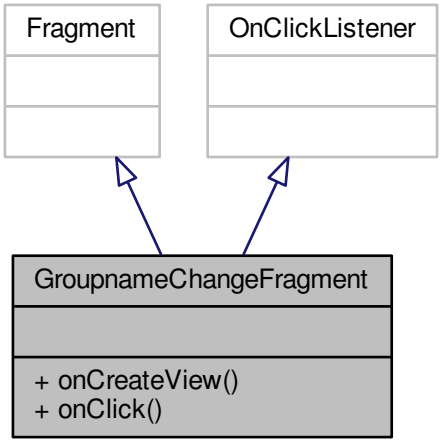
einem GroupFragment aufgerufen, sowohl wenn der Benutzer den CreateGroup-Button drückt, als auch wenn ein Gruppenadministrator den Namen seiner Gruppe umbenennen will. Sie enthält das GroupnameChangeFragment und das GroupnameCreateFragment. Sie erbt von der AppCompatActivity und implementiert dementsprechen auch deren Methoden.

Methoden

a) protected onCreate(@Nullable Bundle savedInstanceState)

   Erweitert die onCreate Methode der AppCompatActivity mit dem laden des GroupnameCreateFragments in den groupname_container.

15. **GroupnameChangeFragment**



Das GroupnameChangeFragment ist dafür zuständig den Gruppenname zu ändern. Es wird von dem GroupnameCreateFragment in den groupname_container geladen. Es legt die Ansicht fest, die ein Benutzer sieht, wenn er einen Gruppennamen ändern möchte. Es erbt von Fragment und implementiert den View.onClickListener. Dementsprechend implementiert es auch deren Methoden.

Methoden

a) public onCreateView(LayoutInflater inflater, ViewGroup container, Bundle sa-

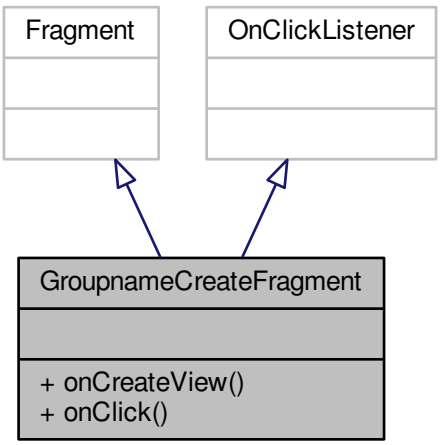
vedInstanceState): View

Erweitert die onCreateView Methode des Fragments mit der gewünschten Ansicht, die der View übergeben wird, fügt der View den alten Gruppennamen und dem OnClickListener den Button hinzu. Diese Methode gibt die aktuelle View zurück.

b) public onClick(View view)

Implementiert die onClick Methode des OnClickListeners, so dass beim Klick auf dem Next-Button überprüft wird, ob der gewünschte neue Gruppenname zugelassen ist. In diesem Fall ändert er diesen und leitet an dessen GroupActivity weiter.

16. **GroupnameCreateFragment**



Das GroupnameCreateFragment ist dafür zuständig eine neue Gruppe auf dem Server anzulegen. Es legt die Ansicht fest, die ein Benutzer sieht, wenn er den CreateGroup-Button gedrückt hat. Es erbt von Fragment und implementiert den View.onClickListener. Dementsprechend implementiert sie auch deren Methoden.

Methoden

a) public onCreateView(LayoutInflater inflater, ViewGroup container, Bundle sa-

vedInstanceState): View

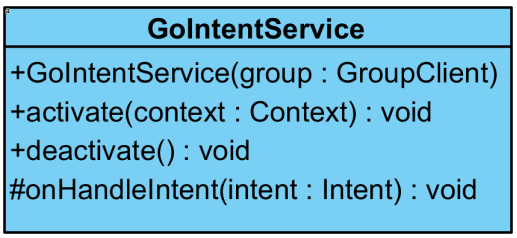
Erweitert die onCreateView Methode des Fragments mit der gewünschten Ansicht, die der View übergeben wird und fürgt dem OnClickListener den Button hinzu, wenn der Benutzer nicht eine bestehende Gruppe ausgewählt hat, in der er Administrator ist. In diesem Fall läd sie das GroupnameCreaterFragment in den groupname_container. Diese Methode gibt die aktuelle View zurück.

b) public onClick(View view)

Implementiert die onClick Methode des OnClickListeners, so dass beim Klicken auf den Next-Button überprüft wird, ob der gewünschte Gruppenname zugelassen ist. In diesem Fall legt er eine neue Gruppe an und leitet an dessen GroupActivity weiter.

2.1.2 ClientModel

GoIntentService



Der GokIntentService kümmert sich um Netzwerkanfragen die den GoStatus, also Anfragen in regelmäßigen Zeitintervallen an den Server sendet und Koordinaten übermittelt. Dieser Service läuft im Hintergrund ab, um die Benutzeroberfläche nicht zu behindern.

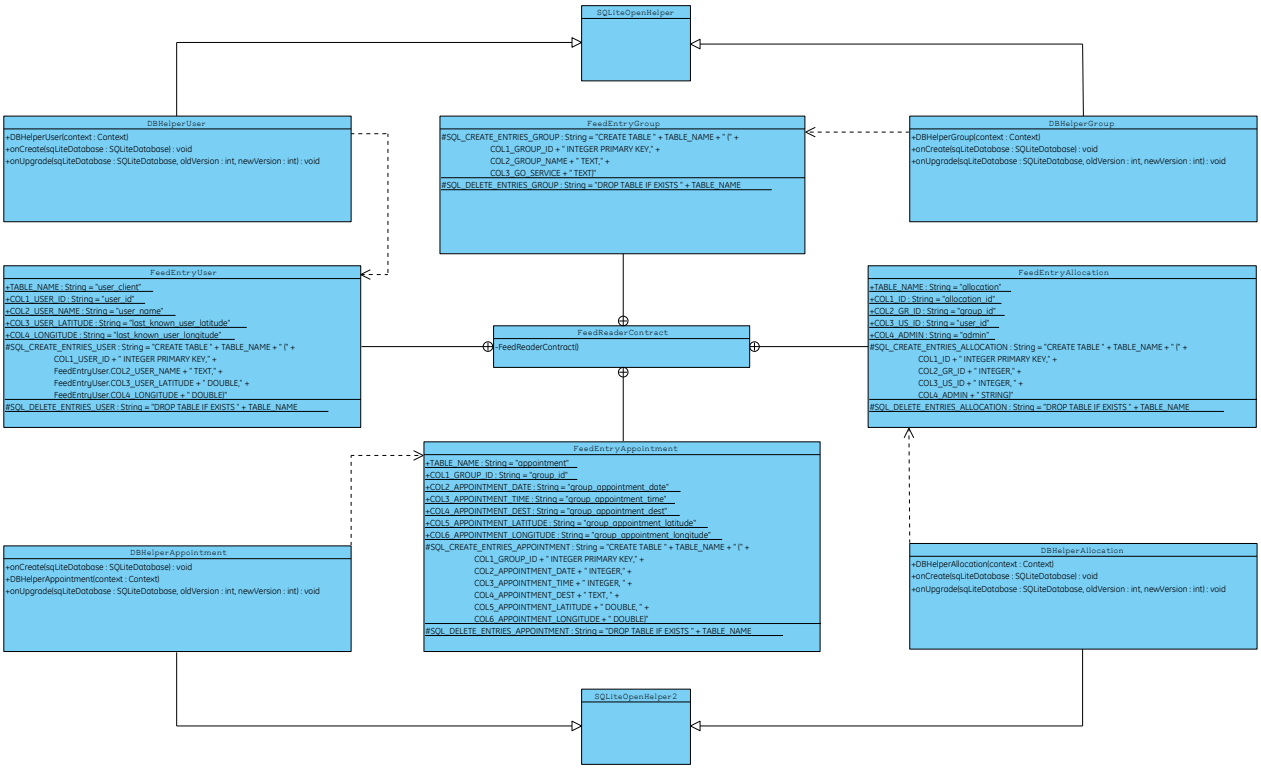
1. public GoIntentService(group: Group) Konstruktor für den IntentService.

2. public activate(context: Context):void GoIntetnService wird aktiviert, wenn der GoStatus aktiviert wurde.

3. public deactivate():void GoIntentService wird deaktiviert, wenn der GoStatus deaktiviert wurde.

4. protected onHandleIntent(intent: Intent): void Für jede Netzwerkanfrage wird ein neuer Thread gestartet und es wird verhindert, dass Anfragen beim Drehen des Bildschirms verloren gehen.

2.1.2.1 Database

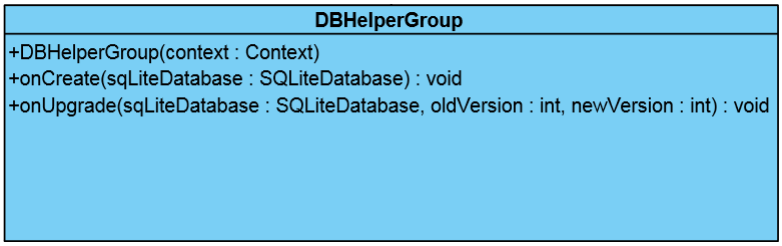


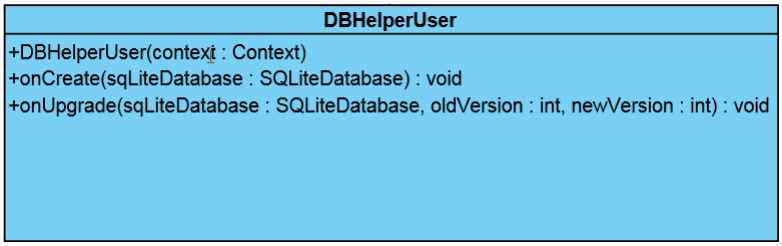
DBHelper
Die vier nachfolgenden DBHelper erben ihren Konstruktor und ihre Methoden von SQLiteOpenHelper.

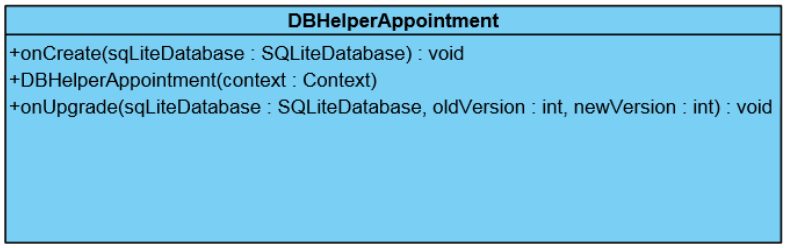
1. public DBHelperGroup(context: Context)
   Der Konstruktor definiert dabei den Namen und die Versionsnummer der Datenbank.

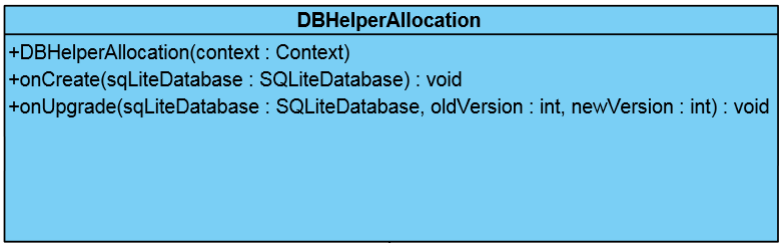
2. public onCreate(sqLiteDatabase: SQLiteDatabase)
   Die SQLiteDatenbank wird mit den in FeedReaderEntry definierten Spalten aufgebaut, wenn sie das erste Mal aufgerufen wird.

3. public onUpgrade(sqLiteDatabase: SQLiteDatabase, oldVersion: int, newVersion: int)
   Diese Methode wird verwendet, wenn man die Datenbank verändert hat. Dieser wird dann eine neue Versionsnummer zugeteilt.

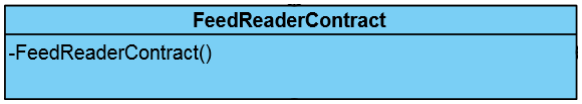








FeedReaderContract

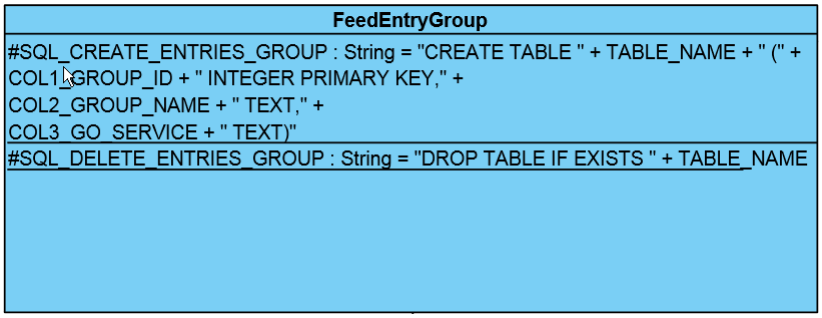


Die FeedReaderContract Klasse definiert in statischen Innenklassen wie die Tabellen der Datenbank aufgebaut sind. Jede der nachfolgenden FeedEntry Klassen implementiert dabei das Interface BaseColumns.

1. public static final CREATE ENTRIES:String
   Legt die Spalten wenn die Tabelle generiert wird in genau dieser Reihenfolge an.

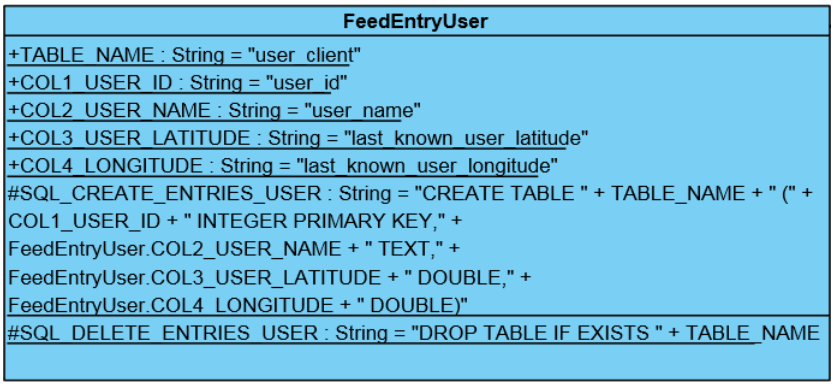
2. public static final DELETE ENTRIES:Strning
   Löscht die zuvor definierten Spalteneinträge.

FeedEntyGroup



Die Innenklasse FeedEntryGroup definiert den Namen und die Spalten der Tabelle, welche die Gruppen auf dem Client speichert. Dabei steht in der ersten Spalte die eindeutige Gruppen ID (welche die Zeilen eindeutig unterscheidbar macht), in der zweiten Spalte der eindeutige Gruppenname und in der dritten Spalte, ob der GoService des aktuellen Benutzers für diese Gruppe aktiviert oder deaktiviert ist. Wenn eine Gruppe gelöscht wird, dann wird auch ihr Eintrag in der Datenbank gelöscht.

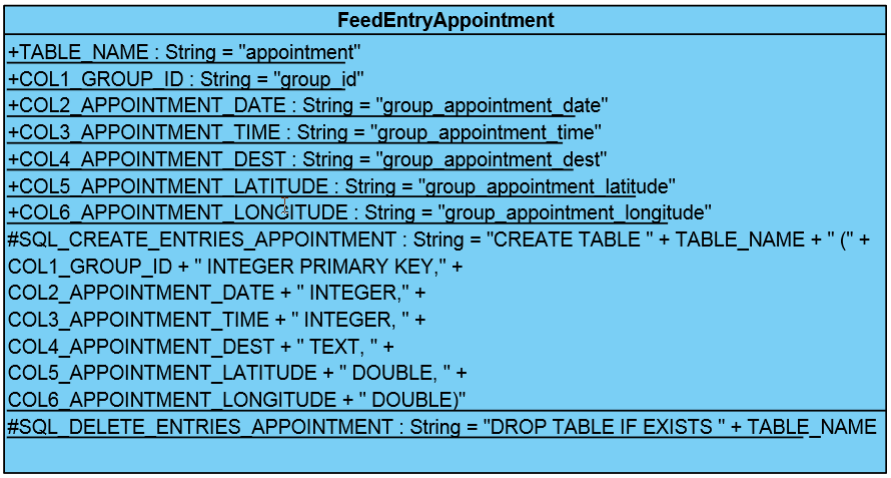
FeedEntyUser



Die Innenklasse FeedEntryUser definiert den Namen und die Spalten der Tabelle, welche die Benutzer auf dem Client speichert. Dabei steht in der ersten Spalte die eindeutige Benutzer ID (welche die Zeilen eindeutig unterscheidbar macht), in der zweiten Spalte der Benutzername und in der dritten und vierten Spalte steht je ein Wert der zuletzt

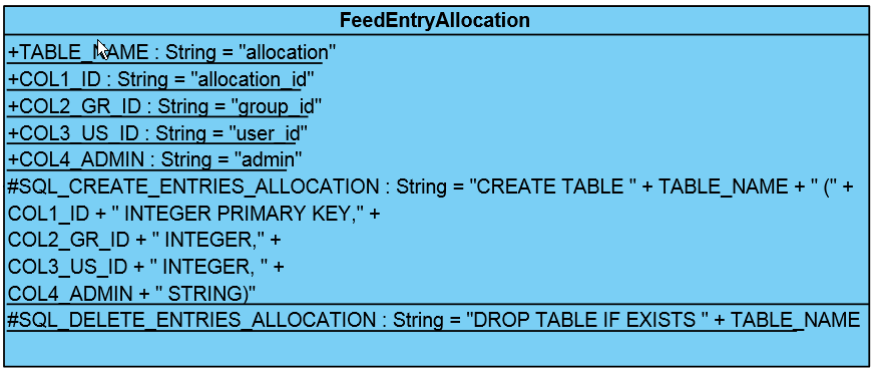
bekannten Gps Daten des jeweiligen Nutzers. Es werden nur Benutzer gespeichert mit denen der aktuelle Benutzer in mindestens einer gemeinsamen Gruppe ist.

FeedEntyAppointment



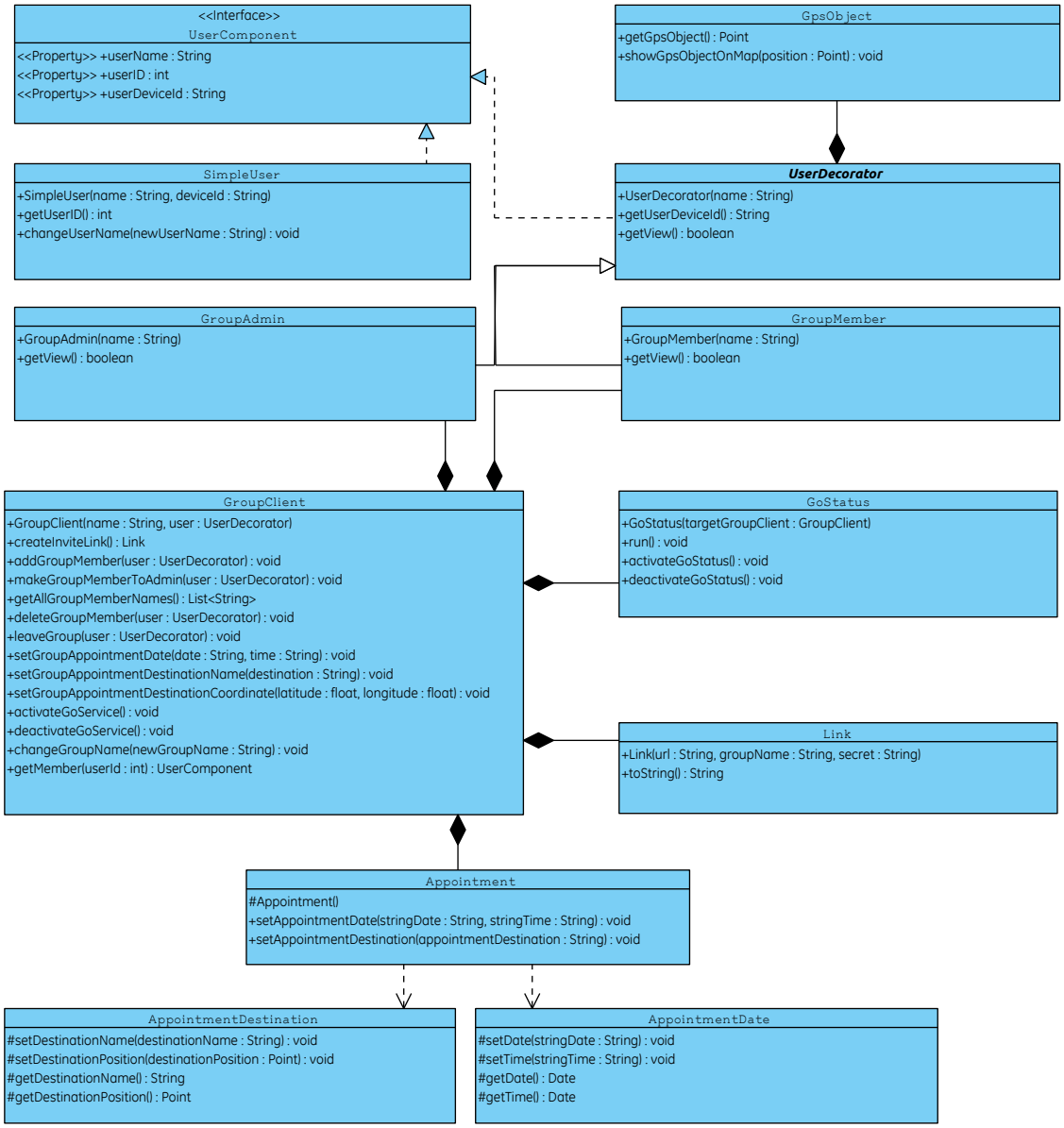
Die Innenklasse FeedEntryAppointment definiert den Namen und die Spalten der Tabelle, welche die Treffpunkte zu jeder Gruppe speichert. Jede Gruppe hat dabei nur eine Zeile, welche den Treffpunkt definiert. Dabei steht in der ersten Spalte die Gruppen id (welche die Zeilen eindeutig unterscheidbar macht), in der zweiten Spalte steht das Datum und in der dritten die Uhrzeit des Treffpunktes. In der vierten Spalte steht der Name des Zielortes und in der fünften und sechsten steht jeweils ein Wert der Gps Daten des Zielortes. Wenn sich der Treffpunkt der Gruppe ändert, dann werden die Werte des alten Treffpunktes überschrieben. Sollte die die Gruppe des dazugehörigen Treffpunktes gelöscht werden, dann wird auch der Eintrag in dieser Tabelle gelöscht.

FeedEntyAllocation

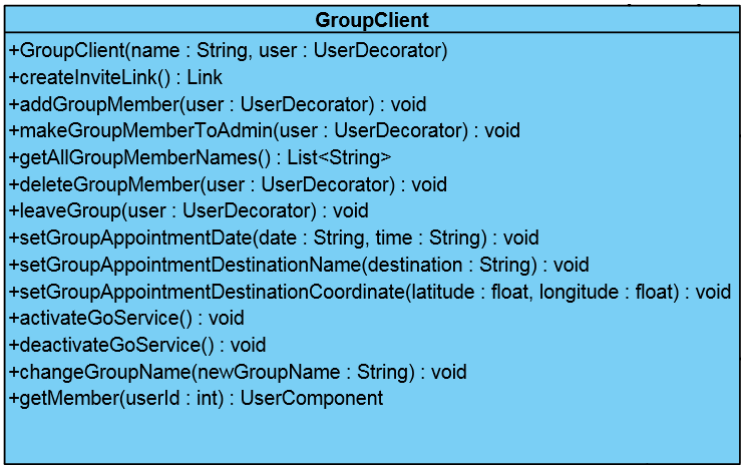


Die Innenklasse FeedEntryAllocation definiert den Namen und die Spalten der Tabelle, welche die jeweiligen Mitglieder jeder Gruppe speichert, in der der aktuelle Benutzer Mitglied ist. Zu jedem Mitglied wird vermerkt, ob dieses Administratorrechte hat. Dabei steht in der ersten Spalte die Allocation id (welche die Zeilen eindeutig unterscheidbar macht und automatisch hochgezählt), in der zweiten Spalte steht die Gruppen id, in der dritten Spalte die Benutzer id und in der vierten Spalte, ob dieser Benutzer Gruppenadministrator ist oder nicht. Gruppen id wird für jedes Gruppenmitglied vermerkt, also kommt mehrmals vor, sobald mehr als ein Benutzer Mitglied dieser Gruppe ist.

2.1.2.2 ObjectStructure



GroupClient



Die Klasse GroupClient definiert wie eine Gruppe auf dem Client aufgebaut ist und welche Funktionalität ihr zur Verfügung steht.

1. public GroupClient(name: String, user: UserDecorator)
   Konstruktor der einer neuen Gruppe einen eindeutigen Gruppennamen zuweist und den Ersteller als Gruppenadministrator festlegt.

2. public createInviteLink():Link
   Um ein Gruppenmitglied hinzuzufügen, muss dieses über einen eindeutigen Link beitreten. Wird der Link ausgeführt wird man entweder zu GoApp weitergeleitet (hat App bereits installiert) oder wird dazu aufgefordert diese zu installieren. Der Link ist nicht mehr gültig sobald das Mitglied hinzugefügt wurde. Man muss Administrator sein um diese Funktion verwenden zu können.

3. public addGroupMember(user: UserDecorator)
   Ein Mitglied wird der Gruppe hinzugefügt.

4. public makeGroupMemberToAdmin(user: UserDecorator)
   Ein Gruppenmitglied wird vom Administrator zum Gruppenadministrator gemacht.

5. public getAllGroupMemberNames():List<String>
   Alle Namen der Mitglieder einer Gruppe werden zurückgegeben.

6. public deleteGroupMember(user:UserDecorator)
   Ein Gruppenmitglied wird aus der Gruppe durch den Administrator entfernt.

7. public leaveGroup(user:UserDecorator)
   Ein Mitglied verlässt die Gruppe.

8. public activateGoService()
   Der aktuelle Benutzer aktiviert seinen GoStatus und übermittelt seine GPS Daten an die Gruppe (GPS muss eingeschalten sein).

9. public deactivateGoService()
   Der aktuelle Benutzer deaktiviert seinen GoStatus und wird nicht mehr verfolgt.

10. public changeGroupName(newGroupName:String)
    Der Gruppenadministrator benennt die Gruppe zu einem anderen eindeutigen Namen um.

11. public getGroupID():int
    Die Gruppen Id wird zurückgegeben

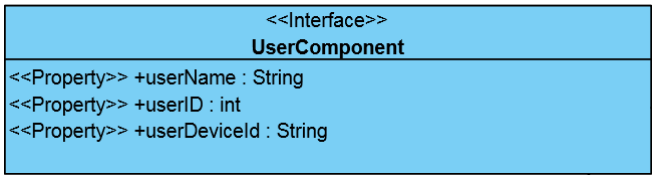
12. public getGroupName():String
    Der Gruppenname wird zurückgegeben

13. public getGoStatus():GoStatus
    Der GoStatus wird zurückgegeben

14. public getAppointment():Appointment
    Das Appointment wird zurückgegeben

15. public getMember(userId: int):UserComponent
    Der Typ des aktuellen Benutzers einer Gruppe (Gruppenmitglied oder Administrator)wird zurückgegeben, um die Ansicht der Gruppe auf seine ihm zugänglichen Funktionen zu beschränken/ erweitern.

UserComponent



Dieses Interface ist die erste Komponente des Decorator pattern und definiert grundlegende Funktionen des Benutzers die jederzeit aufgerufen werden können.

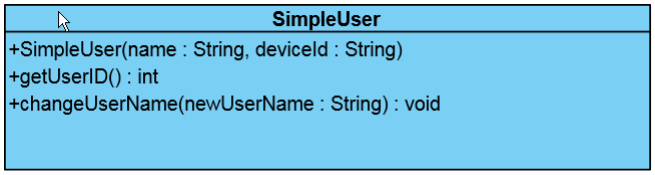
Mit getUserId() kann der Benutzer von anderen Gruppenmitgliedern unterschieden werden, da der Benutzername nicht eindeutig sein muss, die Id aber schon. Mit getUserDeviceId() kann der Benutzer auf dem Server angelegt werden. Jede Gerätenummer kann nur einmal auf dem Server vorkommen, um Missbrauch der GoApp vorzubeugen.

1. public getUserName():String
   Benutzername des aktuellen Benutzers kann in der GoApp visualisiert werden.

2. public getUserID():int
   Mit der Benutzer Id kann sich der aktuelle Benutzer unter den anderen Gruppenmitglieder identifizieren.

3. public getUserDeviceID():String
   Die Device Id wird auf dem Server zusammen mit den anderen zwei Werten gespeichert, um einem Endgerät nur einen Benutzer zu erlauben, um Missbrauch zu vermeiden.

SimpleUser



Die SimpleUser Klasse implementiert das UserComponent Interface. Der Benutzer ist ein SimpleUser Objekt, wenn er nicht in der Gruppenansicht ist und so weder einem Gruppenadministrator noch einem Gruppenmitglied entspricht.

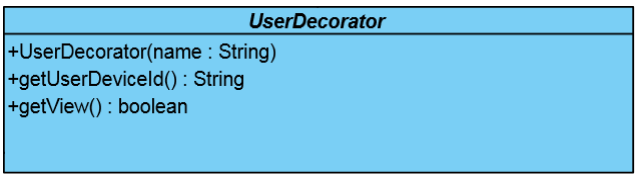
1. public getUserName():String
   Siehe UserComponent.

2. public getUserID():int
   Siehe UserComponent.

3. public getUserDeviceID():String
   Siehe UserComponent.

4. public changeUserName(newUserName: String)
   Der aktuelle Benutzer kann seinen Benutzernamen nachträglich ändern.

UserDecorator



Die UserDecorator Klasse ist eine abstrakte Klasse, die wie SimpleUser das UserComponent Interface implementiert.

1. public getUserName():String
   Siehe UserComponent.

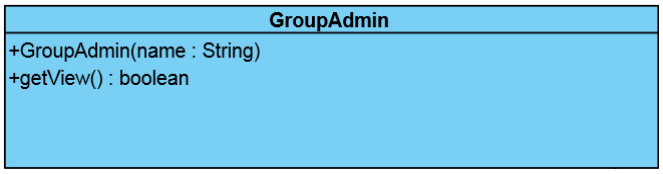
2. public getUserID():int
   Siehe UserComponent.

3. public getUserDeviceID():String
   Siehe UserComponent.

4. public getView():boolean
   Mit dieser Methode wird bestimmt welche Gruppenansicht der aktuelle Benutzer sieht (die für ein Gruppenmitglied oder die für einen Gruppenadministrator mit zusätzlichen Funktionen).

5. public getGpsObject():GpsObject
   Die Gps Daten des aktuellen Benutzers werden zurück gegeben.

GroupAdmin

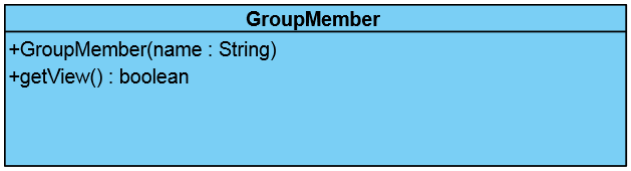


Der Gruppenadministrator unterscheidet sich vom Gruppenmitglied in sofern, dass ihm weitaus mehr Funktionen zur Verfügung stehen, wie Mitglieder hinzufügen/ entfernen und Treffpunkte festlegen.

1. public GroupAdmin(name: String)
   Konstruktor für einen Gruppenadministrator.

2. public getView():boolean
   Siehe UserDecorator.

GroupMember

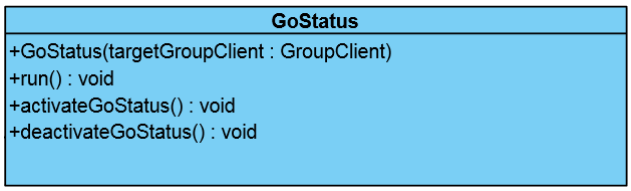


Das Gruppenmitglied unterscheidet sich vom Gruppenadministrator in sofern, dass ihm die Gruppe betreffend kaum Funktionen zur Verfügung stehen.

1. public GroupMember(name: String)
   Konstruktor für einen Gruppenmitglied.

2. public getView():boolean
   Siehe UserDecorator.

GoStatus



Der GoStatus bestimmt darüber in welchen Abständen die Position des aktuellen Benutzers an die anderen Gruppenmitglieder übermittelt wird.

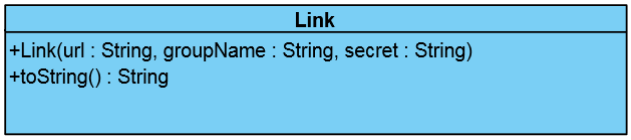
1. public GoStatus(targetGroupClient: GroupClient)
   Konstruktor für den GoStatus wird für jede Gruppe für den aktuellen Benutzer definiert.

2. public run()
   Periodisch bei aktivierten GoStatus der Standort an die Gruppenmitglieder weitergeleitet.

3. public activateGoStatus()
   GoStatus und Gps Tracking aktivieren.

4. public deactivateGoStatus()
   Gps Tracking deaktivieren.

Link

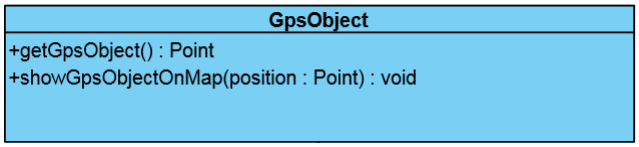


Mit dem Link kann man andere Mitglieder

1. public Link(url: String, groupName: String , secret: String)
   Konstruktor für einen Link. Ein Gruppenlink besteht aus den gelisteten drei Komponenten. Durch den Gruppennamen lässt sich das Mitglied welches den Link betätigt zu derjenigen Gruppe hinzufügen und das sercret verhindert, dass der Link leicht regenerierbar ist.

2. public toString():String
   Der Link wird wieder in seine einzelne Komponenten sortiert.

GpsObject

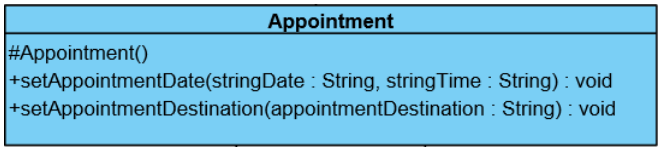


Das GpsObject gibt an in welcher Form die Gps Daten übermittelt werden.

1. public getGpsObject():Point
   Der Konstruktor für das GpsObject definiert ein neuen Point der aus einem Längen und einem Breitengrad besteht.

2. public getTimestamp():String
   Timestamp gibt zurück, wie alt der zuletzt ermittelte Standort ist.

Appointment



Appointment fasst alle Informationen zusammen die es über den Treffpunkt zu speichern gibt.

1. protected Appointment()
   Konstruktor für ein Appointment welches beim Erstellen einer Gruppe default Werte für Datum, Uhrzeit und Ort definiert.

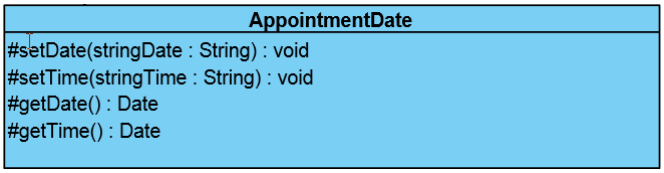
2. public setAppointmentDate(String stringDate, String stringTime)
   Datum und Uhrzeit des Treffens ändern.

3. getAppointmentDate():AppointmentDate
   Datum und Uhrzeit des Appointments zurückgeben.

4. public setAppointmentDestination(String appointmentDestination)
   Über den Namen/Adresse oder über die Position auf der Karte einen Zielort setzen.

5. public getAppointmentDestination():AppointmentDestination
   Den Zielortes des Treffpunktes zurückgeben.

AppointmentDate



AppointmentDate legt fest wie Datum und Uhrzeit formatiert werden.

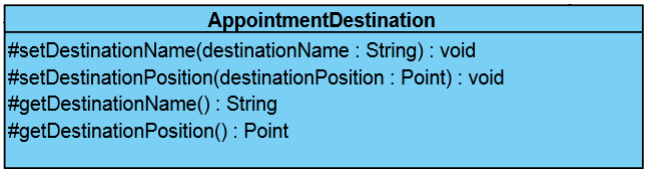
1. protected setDate(stringDate: String)
   Ein Datum festlegen.

2. protected setTime(stringTime: String)
   Eine Uhrzeit festlegen.

3. protected getDate():Date
   Das Datum zurückgeben.

4. protected getTime():Date
   Die Uhrzeit zurückgeben.

AppointmentDestination



AppointmentDestination legt fest wie Name und Position des Zielortes definiert sind.

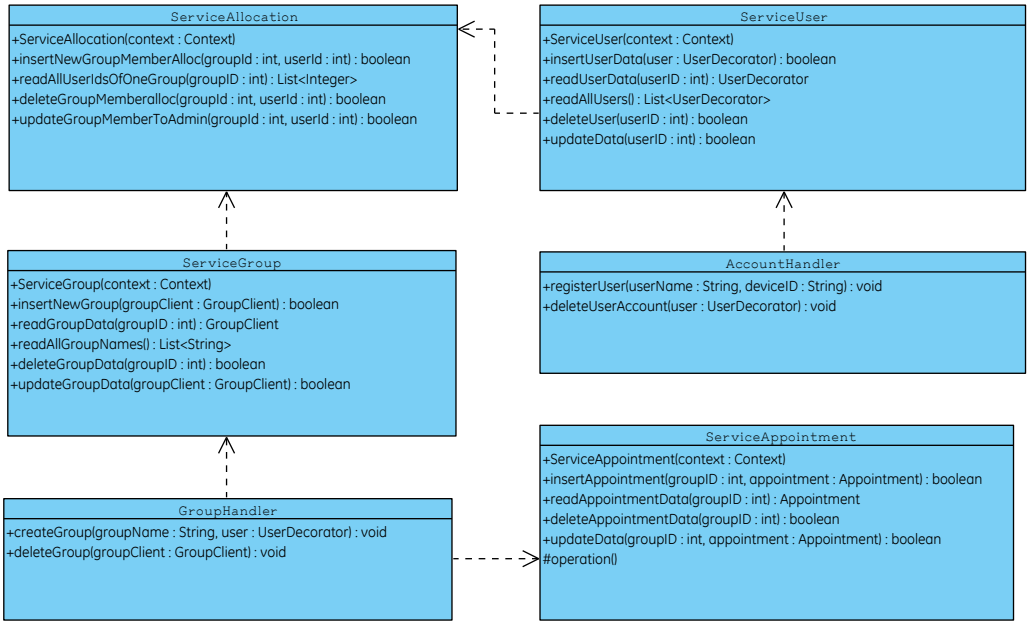
1. protected setDestinationName(destinationName:String)
   Den Zielort über den Namen/ die Adresse bestimmen.

2. protected setDestinationPosition(destinationPosition:Point)
   Den Zielort über eine Position auf der Karte bestimmen.

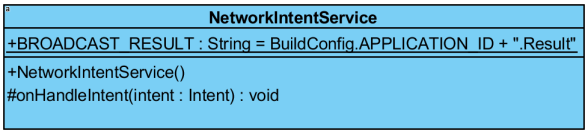
3. protected getDestinationName():String
   Den Zielortnamen zurückgeben.

4. protected getDestinationPosition():Point
   Die Zielortposition zurückgeben.

2.1.3 ClientController



NetworkIntentService



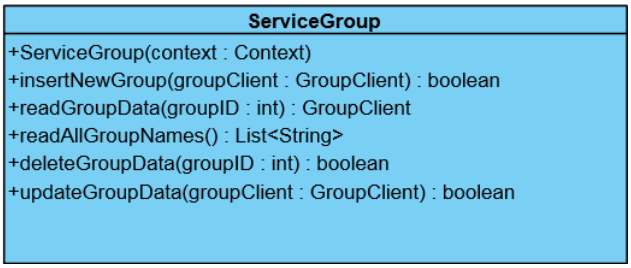
Der NetworkIntentService kümmert sich um Netzwerkanfragen, läuft aber im Hintergurnd ab um die Benutzeroberfläche nicht zu behindern.

1. public NetworkIntentService() Konstruktor für den GoIntentService.

2. protected onHandleIntent(intent: Intent) Für jede Netzwerkanfrage wird ein neuer Thread gestartet und es wird verhindert, dass Anfragen beim Drehen des Bildschirms verloren gehen.

2.1.3.1 Database

Services
Die folgenden Service Klassen stellen die grundlegenden Funktionen zur Verfügung, mit welchen auf die Daten der in den Innenklassen von FeedReaderContract definierten Tabellen der Datenbank zugegriffen werden kann. Dabei können Daten hinzugefügt (insert), gelesen (read), gelöscht (delete) und verändert (update) werden. Weitere Details dazu, wie die jeweilige Datenbanken aufgebaut sind, finden sie in FeedEntryGroup, FeedEntryUser, FeedEntryAppointment, FeedEntryAllocation.



In der Datenbank auf die ServiceGroup zugreift, werden die Gruppen gespeichert und ob der aktuelle Nutzer der Go-App für diese spezielle Gruppe seinen Go-Button aktiviert hat oder nicht.

1. public ServiceGroup(context: Context)
   Konstruktor, dem den Kontext der Applikation mitgegeben wird.

2. public insertNewGroup(groupClient: GroupClient):boolean
   Eine neue Gruppe wird der Datenbank hinzugefügt und der GoStatus des Erstellers

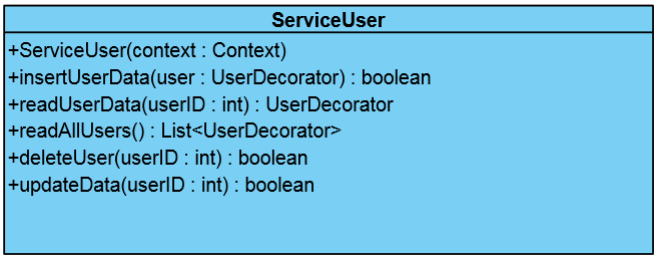
auf false gesetzt.

3. public readGroupData(groupID: int):GroupClient
   Informationen über Gruppenname und GoStatus lesen.

4. public readAllGroupNames():List<String>
   Alle Gruppen in denen der aktuelle Benutzer Mitglied ist werden zurückgegeben.

5. public deleteGroupData(groupID: int):boolean
   Eine Gruppe wird aus der Datenbank gelöscht.

6. public updateGroupData(groupClient: GroupClient):boolean
   Gruppendaten werden aktualisiert, wenn sich der Gruppenname oder der GoStatus ändert.



In der Datenbank auf die ServiceUser zugreift ist der Name und die zuletzt bekannte Position des Benutzers gespeichert, welche bei neuen Informationen angepasst werden.

1. public ServiceUser(context: Context)
   Konstruktor, dem den Kontext der Applikation mitgegeben wird.

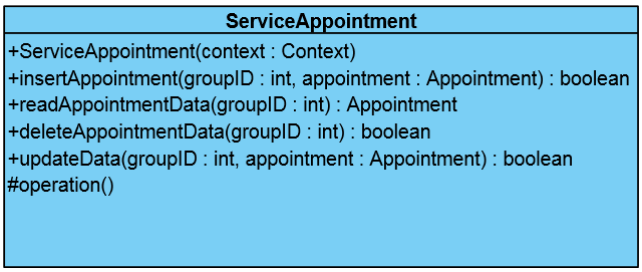
2. public insertUserData(UserDecorator user):boolean
   Ein neuer Benutzer wird der Datenbank hinzugefügt, sobald dieser mit dem aktuellen Benutzer in einer Gruppe ist.

3. public readUserData(userID: int):UserDecorator
   Inforamtionen über Benutzername und Position eines Benutzers können gelesen werden.

4. public readAllUsers():List<UserDecorator>
   Alle Benutzernamen mit denen der aktuelle Benutzer in einer Gruppe ist lesen.

5. public deleteUser(userID: int):boolean
   Einen Benutzer von der Datenbank entfernen.

6. public updateData(userID: int):boolean
   Benutzerdaten anpassen, wenn sich Name oder Position geändert haben.



In der Datenbank auf die ServiceAppointment zugreift, wird zu jeder Gruppe das aktuelle Treffen gespeichert und verändert, sobald ein neues existiert.

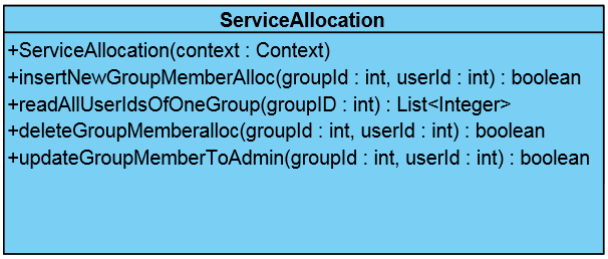
1. public ServiceAppointment(context: Context)
   Konstruktor, dem den Kontext der Applikation mitgegeben wird.

2. public insertAppointment(groupID: int, appointment: Appointment):boolean
   Wird aufgerufen wenn eine neue Gruppe erstellt wird. Dabei wird ein default Appointment initialisiert.

3. public readAppointmentData(groupID: int):Appointment
   Informationen über Datum, Uhrzeit, Zielortname und Zielortposition können abgerufen werden.

4. public deleteAppointmentData(groupID: int):boolean
   Wenn eine Gruppe gelöscht wird, wir auch das dazugehörige Appointment gelöscht.

5. public updateData(groupID: int, appointment: Appointment):boolean
   Das Appointment wird angepasst, sobald es geändert wird.



In der Datenbank auf die ServiceAllocation zugreift, wird die Verbindung zwischen Gruppen und Gruppenmitgliedern bzw. Administratoren gespeichert.

1. public (context: Context):ServiceAllocation
   Konstruktor, dem den Kontext der Applikation mitgegeben wird.

2. public insertNewGroupMemberAlloc(groupID: int , userID: int):boolean
   Wenn ein neues Mitglied einer Gruppe hinzugefügt wird, dann wird diese Verbindung hier eingetragen.

3. public readAllUserIdsOfOneGroup(groupID: int):List<Integer>
   Um alle Mitglieder einer Gruppe zu bekommen, müssen dafür alle Benutzer Id's mit derselben Gruppen Id zurückgegeben werden.

4. public deleteGroupMemberAlloc(groupid: int, userId: int):boolean
   Wird ein Gruppenmitglied aus einer Gruppe entfernt, so wird auch die Verbindung von diesem zur Gruppe gelöscht.

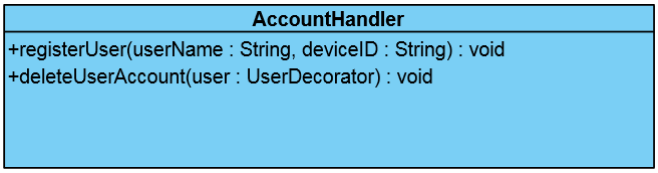
5. public deleteAllGroupMemberAlloc(groupId: int):boolean
   Wird eine Gruppe gelöscht, so werden alle Verbindungen zu Benutzern dieser Gruppe gelöscht.

6. public updateGroupMemberToAdmin(groupId: int , userID: int):boolean
   Wird ein Gruppenmitglied zum Administrator gemacht, dann wird dies in der Datenbank angepasst.

2.1.3.2 ObjectStructure

In Account- und GroupHandler werden allgemeine Funktionen definiert, die nicht selbst vom GroupClient oder von Unterklassen von UserComponent ausgeführt werden können, sondern eine übergeordnete Position benötigen.

AccountHandler



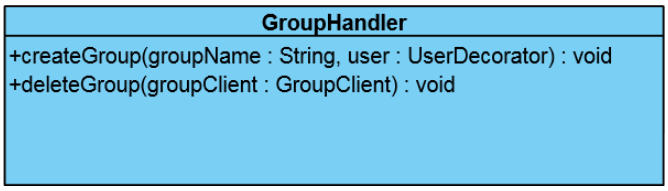
Die AccountHandler Klasse handelt das Registrieren und löschen vom aktuellen Benutzer ab.

1. public registerUser(userName: String, deviceID: String)
   Ein neuer Benutzer registriert sich mit einem Namen und seiner Gerätenummer.

Dieser wird zunächst als SimpleUser Objekt angelegt und mit der Gerätenummer auf dem Server gespeichert, um Missbrauch zu verhindern.

2. public deleteUserAccount(user: UserDecorator)
   Löscht ein Benutzer seinen Account, wird er von der Datenbank gelöscht.

GroupHandler



Die GroupHandler Klasse handelt das Erstellen und Löschen von Gruppen ab.

1. public createGroup(groupName: String, user: UserDecorator)
   Eine neue Gruppe wird erstellt, mit einem eindeutigen Gruppennamen und mit dem Ersteller als Administrator.

2. public deleteGroup(GroupClient groupClient)
   Eine Gruppe wird gelöscht und alle ihre Mitglieder entfernt.

2.2 Serverseitig

2.2.1 Servlets

Die Tomcat-Servlets bilden zusammen einen Webservice in Anlehnung an einen REST-Webservice (Representational State Transfer).
Im MVC-Paradigma würden sie in die Kategorie Controller fallen.
Die Servlets sind in verschiedene Aufgabenbereiche unterteilt. Alle Servlets erben von BaseServlet, was wiederum von HttpServlet erbt. Die genaue Art der Anfrage wird über den QueryString der HttpServletRequest bestimmt.
Alle Anfragen werden in Form von JSON über die HTTP-POST methode übergeben.
In der doPost() Methode der Servlets werden sie bearbeitet. die doGet() Methode wird bei einer HTTP-GET request aufgerufen.
Sie liefert Informationen über den Zustand des Servlets, bzw. leitet den Nutzer weiter, falls er die Anfrage von einem Webbrowser aus sendet.
Über einen POST erhaltene Daten werden als JSON String interpretiert und via Jackson API auf Request-Objekte abgebildet.
Innerhalb des Request-Objekts sind Benutzer und Gruppen nur durch ID und Name

spezifiziert. Die zugehörigen Objekte werden aus der Datenbank angefordert. Name und ID dienen dabei als Schlüssel.

Das GroupServlet als Beispiel zur server-internen Bearbeitung von Anfragen:

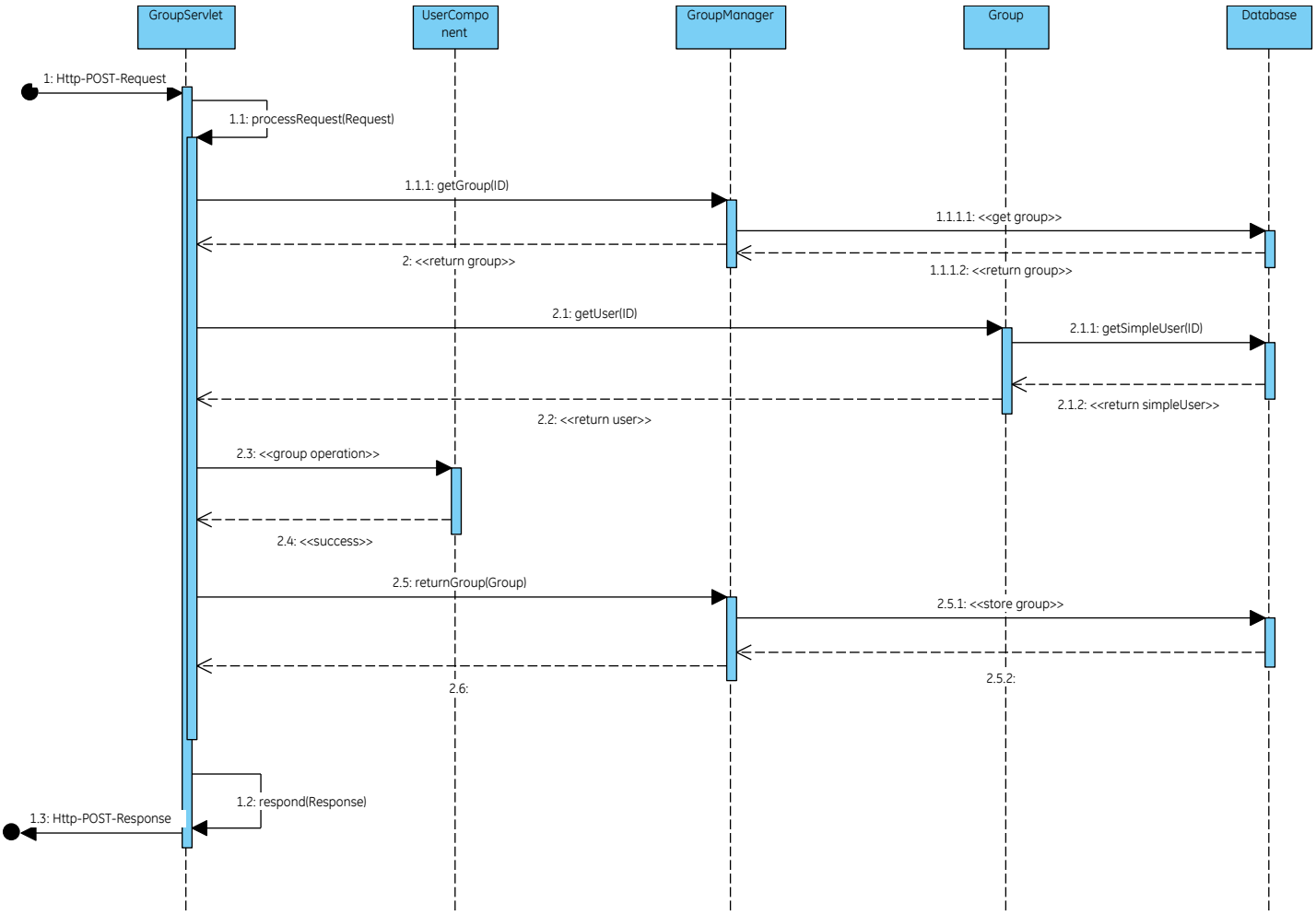


Abbildung 2: Sequenzdiagramm GroupServlet

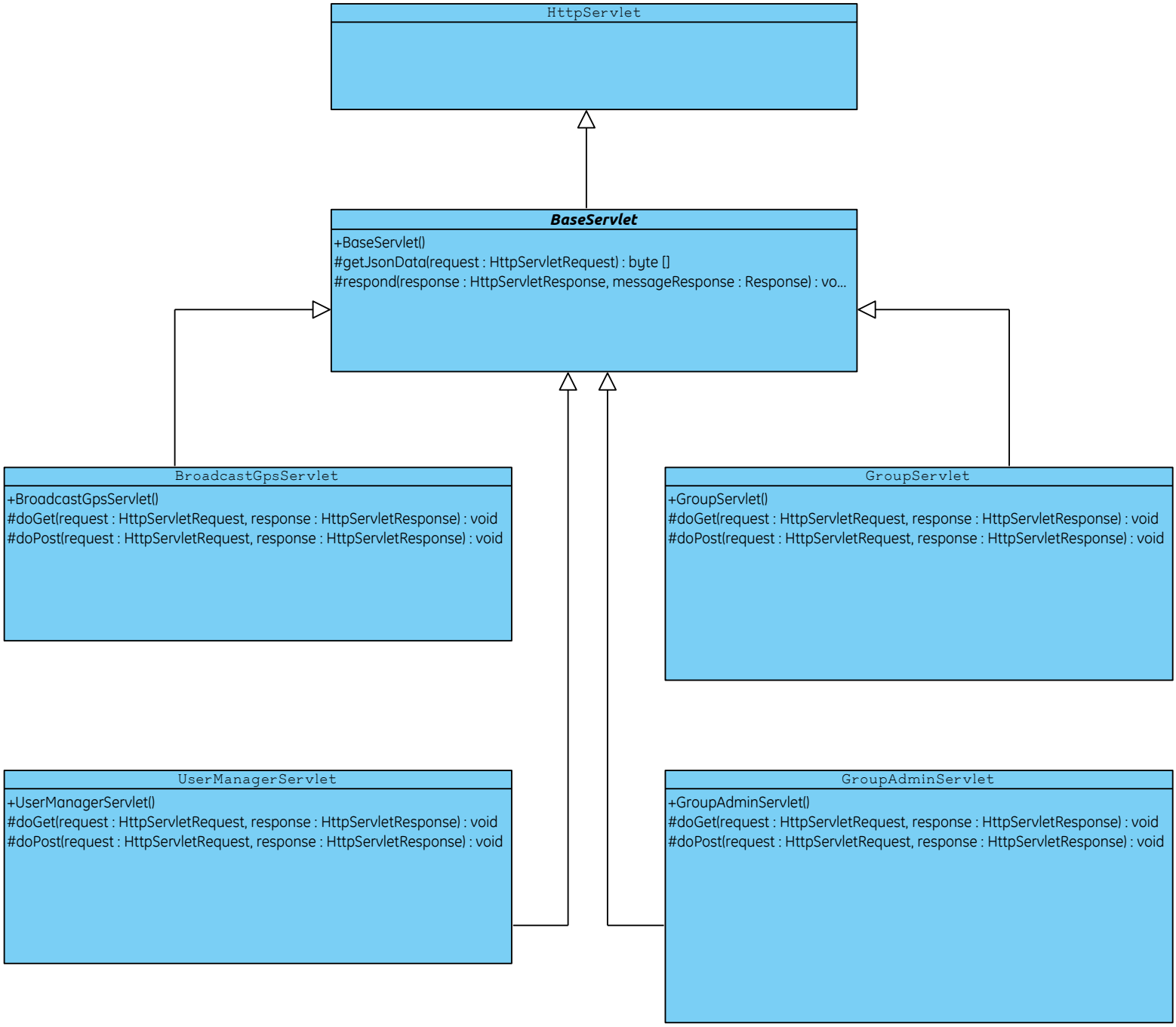


Abbildung 3: Vererbungshierarchie der Servlets

Klasse GroupServlet

Zuständig für alle Nutzer-Anfragen bezüglich Gruppen.
Hier werden nur Parameter der Gruppe abgefragt. Administrative Anfragen werden vom GroupAdminServlet bearbeitet.
Von einem Gruppenmitglied abgefragt, werden beispielsweise aktueller Treffpunkt, aktuelle Gruppenmitglieder oder aktueller Gruppenname.
Die GroupServlet Klasse dient als eine Schnittstelle zu der Datenbank auf dem Server.
Bildet die Anfrage auf ein GroupRequest Objekt ab.

Klasse UserManagerServlet

Das UserManagerServlet ist für alle Benutzeroperationen zuständig. Dazu gehören: Benutzeraccount anlegen, Benutzername ändern, Benutzer löschen.
Damit ist das UserManagerServlet eine weitere Schnittstelle zur Datenbank.
Bildet die Anfrage auf ein UserRequest Objekt ab.

Klasse GroupAdminServlet

Im GroupAdminServlet werden alle Anfragen eines Gruppen-Administrators bearbeitet.
Dabei wird die Zielgruppe und der Nutzer in einem Request-Objekt übergeben.
Geprüft wird, ob der Benutzer Admin der Zielgruppe ist. Ist dies der Fall,
wird über den QueryString die gewünschte Operation auf der Gruppe ausgeführt.
Bildet die Anfrage auf ein GroupAdminRequest Objekt ab.

Klasse BroadcastGpsServlet

In diesem Servlet werden von Gruppenmitgliedern gesendete GPS-Daten empfangen und in der Datenbank gespeichert. Ist der GoStatus für ein Mitglied gesetzt,
können andere Mitglieder ihn Abrufen.
Bildet die Anfrage auf ein BroadcastGpsRequest Objekt ab.

2.2.2 Kommunikation

Zum Datenaustausch zwischen Client und Server haben wir zwei Klassen erstellt:
Request und Response.
Davon erben die jeweils aufgabenspezifischen Request/Response-Klassen.

Klasse Request

Die abstrakte Request Klasse besteht hauptsächlich aus gettern und settern,
diese werden unter anderem von der Jackson API zum Serialisieren benötigt.
Dabei dient die Klasse ausschließlich als container und enthält keine Programmlogik.
Wichtig ist, dass nur ID's bzw. Namen in Form von String/Integer übergeben werden,
dadurch wird die Größe des resultierenden JSON-Strings möglichst klein gehalten.

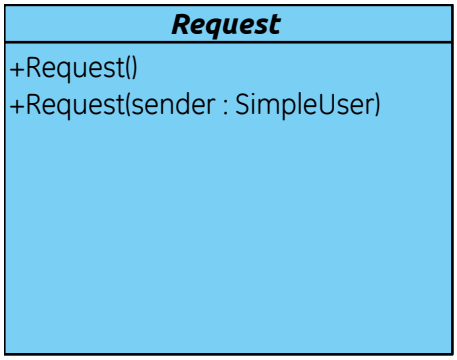


Abbildung 4: Klassendiagramm Request

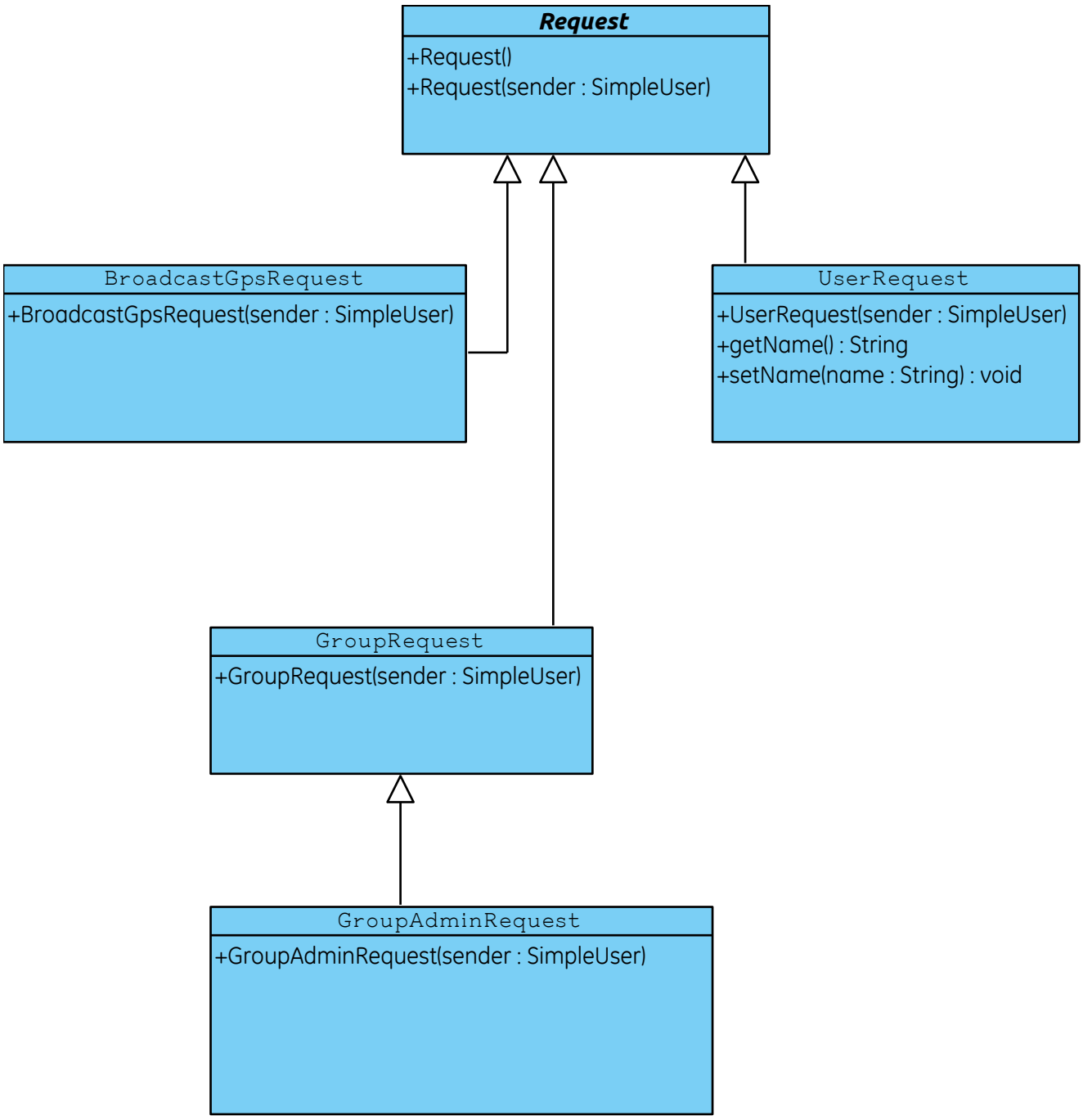


Abbildung 5: Vererbungshierarchie Request

Klasse UserRequest

UserRequest ist eine Anfrage um einen neuen Benutzeraccount anzulegen,
bzw. falls schon vorhanden, den Benutzernamen zu ändern.

Klasse GroupRequest

Alle gruppenbezogenen Anfragen. Benötigt werden Benutzer-ID, Name der entsprechenden Gruppe, und ein QueryString, der die Operation beschreibt, die auf der Gruppe ausgeführt werden soll.
Dieser Anfragetyp kann nur von einem Gruppenmitglied gesendet werden.
Diese Anfrage kann gesendet werden um den aktuellen Stand einer Gruppe abzufragen.
Änderungen sind zum Beispiel wenn ein neuer Treffpunkt gesetzt wurde,
ein Benutzer der Gruppe beigetreten ist, oder ein Benutzer die Gruppe verlassen hat.

Klasse GroupAdminRequest

GroupAdminRequest ist eine erweiterung der normalen GroupRequest. Sie enthält zusätzlich alle Anfragen eines Gruppenadministrators. Dazu zählen: Gruppe löschen, Gruppe umbenennen, Mitglieder einladen, etc.
Die Klasse hält die Variablen für jede mögliche Administrator-Anfrage.

Klasse BroadcastGpsRequest

Anfrage um die eigenen GPS-Daten an die Gruppe zu verteilen, oder GPS-Daten der anderen Gruppenmitglieder abzufragen.
Die eigenen GPS-Daten sind nicht zwingend notwendig, da das anfragende Mitglied seinen
Status auf 'Go' setzen kann, ohne die eigenen GPS-Daten preiszugeben.

Klasse Response

Als Antwort auf eine Request sendet der Server eine Response an den Client.
Die Response enthält alle angeforderten Daten bzw. manipulierte Daten und
ob die Operation, die der Nutzer angefragt hat, erfolgreich war.
Auf einfache Anfragen, wie beispielsweise den eigenen Benutzernamen zu ändern,
wird mit einer einfachen Response geantwortet, die dem Nutzer sagt,
ob der Name erfolgreich geändert wurde.

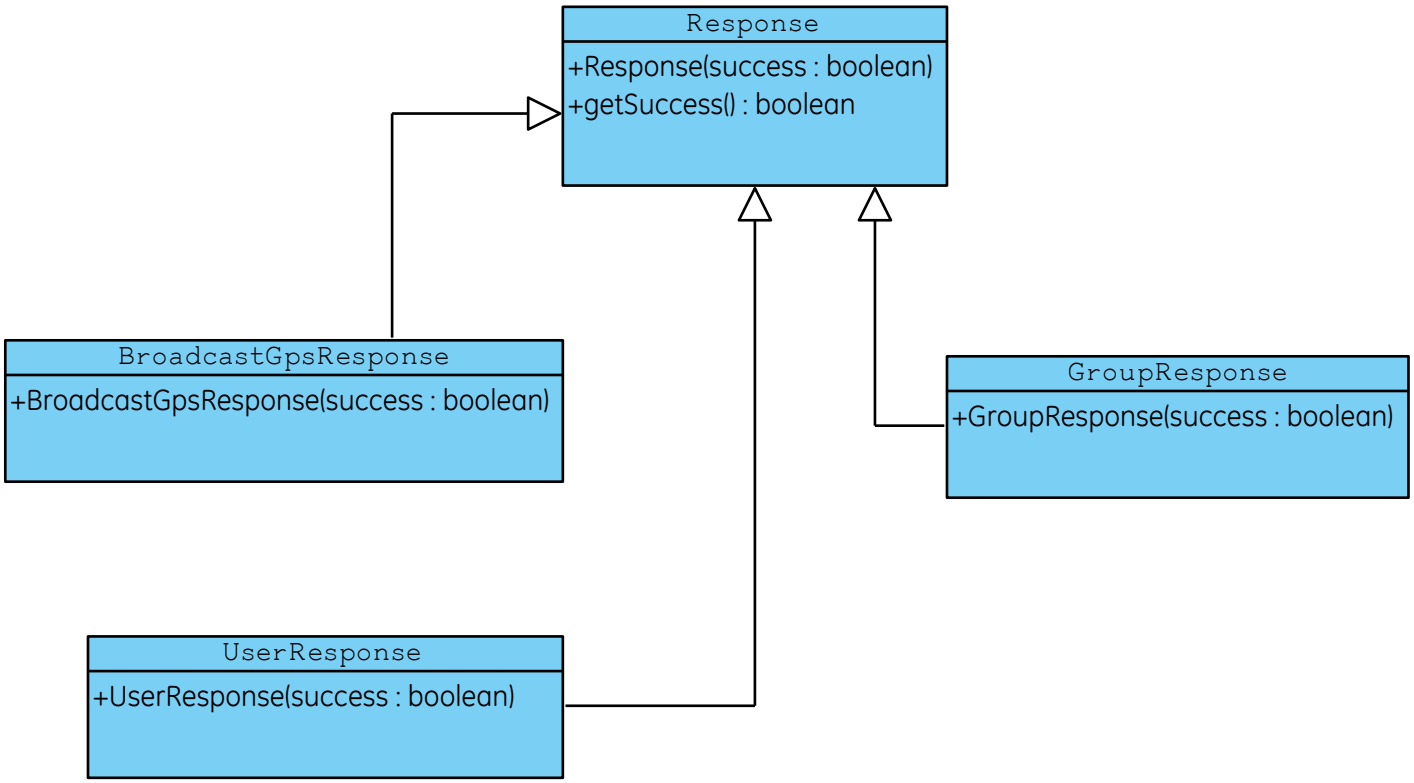


Abbildung 6: Vererbungshierarchie Response

Klasse UserResponse

Anwort auf eine Nutzername ändern Anfrage oder eine Registrieren Anfrage.
Bei der Registrierung wird eine eindeutige Benutzer-ID zurückgegeben.
Im Falle einer Benutzername-ändern-Anfrage wird nur der Status zurückgegeben.

Klasse GroupResponse

Antwort auf eine Gruppen-Anfrage oder eine Administrator-Anfrage.
Ein GroupResponse-Objekt enthält dabei die, nach der Operation, aktuelle Gruppe.
Bei einer Update-Anfrage wird ebenfalls mir einer GroupResponse geantwortet.

Klasse BroadcastGpsResponse

Als Antwort auf eine BroadcastGpsRequest, enthält die BroadcastGpsResponse die GPS-Daten
der anderen Gruppenmitglieder, die bereits 'Go' gedrückt haben, also ihre Daten bereits an den Server gesendet haben.

2.2.3 Server-Modell

Das Modell des Servers unterscheidet sich in wesentlichen Punkten zu dem des Clients: Operationen auf Gruppen oder Benutzern lösen keine Anfragen an einen Server aus, sondern arbeiten direkt mit der Schnittstelle zur Datenbank.
Das Client-Modell stellt praktisch die 'Fernbedienung' für das Server-Modell dar.

Hibernate

Die Schnittstelle unserer Datenbank stellt Hibernate dar. Dazu werden in den Klassen GroupAdminServer, GroupMemberServer und SimpleUser (Also alles was gespeichert werden muss), Annotations gesetzt, die Hibernate zum Abbilden benötigt. Anfragen an die Datenbank werden von GroupManager und UserManager gesteuert.

Klasse UserManager

Der UserManager verwaltet Benutzeraccounts und bietet eine allgemeine Schnittstelle zur Datenbank. Er hat zwei Methoden um ein Benutzer-Objekt aus der Datenbank anzufordern: getUserById(int ID) - durchsucht die Datenbank nach einem User mit gegebener Nutzer-ID und getUserByDevId(String devID) - durchsucht die Datenbank nach einem Benutzer mit passender Geräte-ID.
Darüber werden die meisten Benutzer identifiziert, die eine Anfrage senden.
Die Nutzer-ID ist dazu gedacht, dass Gruppenmitglieder nicht die Geräte-ID der anderen Mitglieder erfahren.
createUser(String Name) - Erzeugt einen Neuen Benutzer mit gegebenem Namen.
deleteUser(String devId) - Löscht den Benutzer mit der gegebenen Geräte-ID.

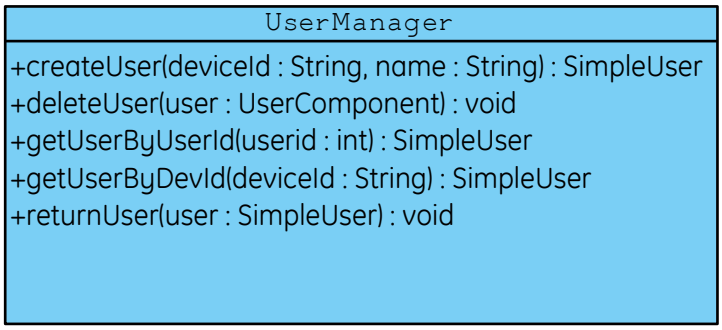


Abbildung 7: Klassendiagramm UserManager

Klasse GroupManager

Gleichermaßen wir der UserManager, bietet auch der GroupManager eine Schnittstelle zur Datenbank. Er bietet die Fabrikmethode createGroup() zum Erstellen neuer Gruppen. Die Methode legt dazu einen neuen Eintrag in der Datenbank an und liefert das neue Group-Objekt zurück.

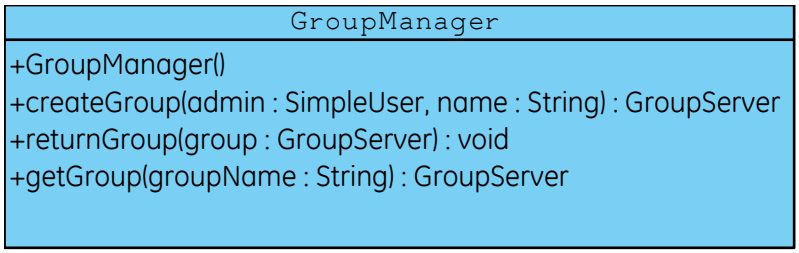


Abbildung 8: Klassendiagramm GroupManager

Klasse GroupServer

Serverseitige Darstellung einer Gruppe. Operationen werden auf der Gruppe nicht direkt ausgeführt. Nach dem Erstellen der Gruppe, laufen alle Operationen über den Gruppenadministrator. über getMember(ID) bekommt man das Mitglied, dass man in der aktuellen Gruppe darstellt. Also entweder GroupMember oder GroupAdmin. In der Fabrikmethode getMember(ID) wird dem UserDecoratorServer das Gruppen-Objekt übergeben, an dass dieser dann gebunden ist.

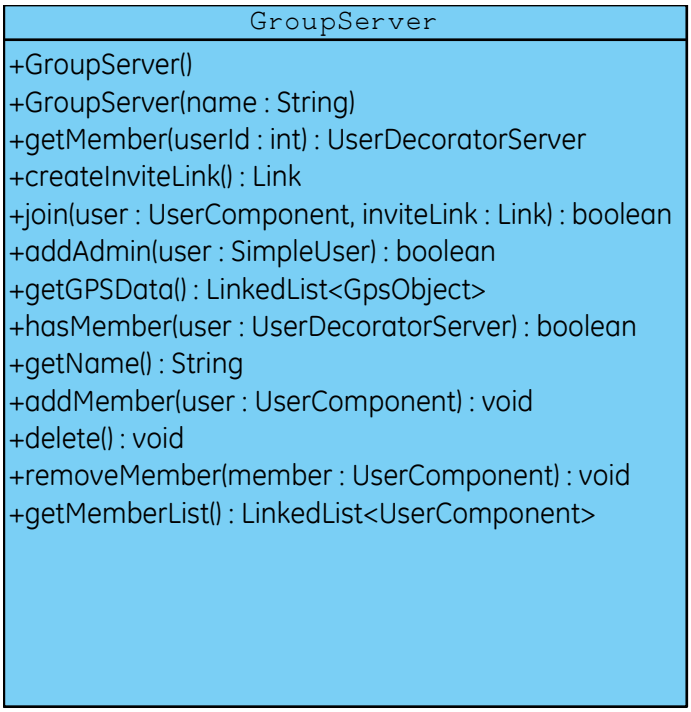


Abbildung 9: Klassendiagramm GroupServer

Klasse LinkGenerator

Die Aufgabe des LinkGenerators besteht darin, zu einer gegebenen Gruppe einen Einladungslink zu generieren. Dabei wird an die Basis-URL des GroupServlets der Gruppenname und ein zufällig generierter String angehängt.

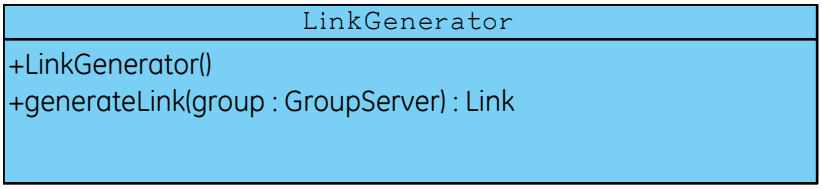


Abbildung 10: Klassendiagramm LinkGenerator

Klasse Link

Da die Klasse Link aus dem common package für Client bereits dokumentiert ist,
sind hier nur noch ein paar Anmerkungen. Ein Link vom Administrator einer Gruppe erzeugt und ist an die Gruppe gebunden.
Diese Klasse repräsentiert eine Gruppeneinladung. über die Getter kann man unabhängig den Gruppennamen und das Geheimnis abfragen.
Hat man als Administrator den Link vom Server erhalten, wird über die toString() Methode die endgültige URL erstellt. Diese kann nun über andere Wege versendet werden. Öffnet man eine zugesendete URL mit der GoApp, wird daraus eine GroupRequest erzeugt, mit der ein anderer Nutzer der Gruppe beitreten kann.
Das Geheimnis wird im zugehörigen Gruppen-Objekt gespeichert.

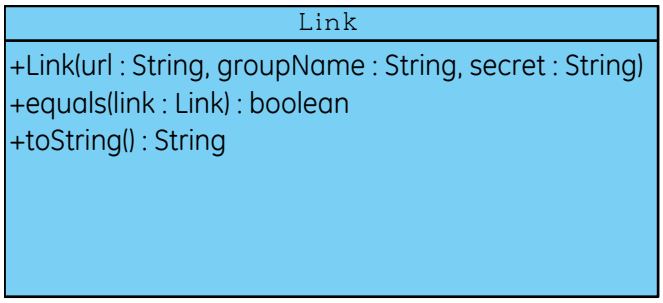


Abbildung 11: Klassendiagramm LinkGenerator

Klasse UserDecoratorServer

Das serverseitige Gegenstück zum UserDecorator. Er implementiert das gleiche Interface wie der UserDecorator auf dem Client, nur dass er mit GroupServer arbeitet.
Im UserDecoratorServer sind sowohl die Methoden des Admins, wie auch die des einfachen GroupMembers vorhanden. Die Idee dahinter ist, dass man nicht verhindern kann, dass ein normaler GroupMember eine Admin-Anfrage sendet.
Die Implementierung der beiden Klassen unterscheidet dann das Verhalten.
(Siehe GroupAdminServer und GroupMemberServer)

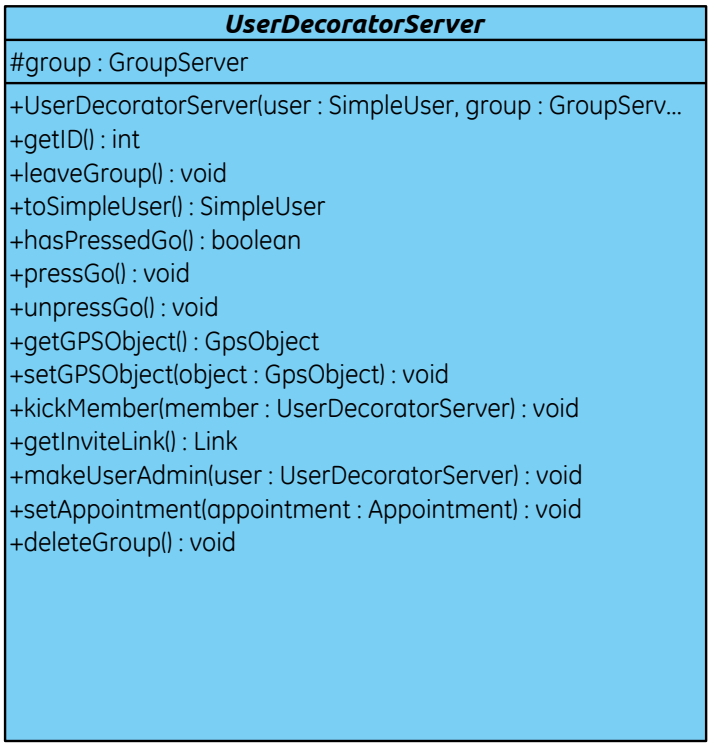


Abbildung 12: Klassendiagramm UserDecoratorServer

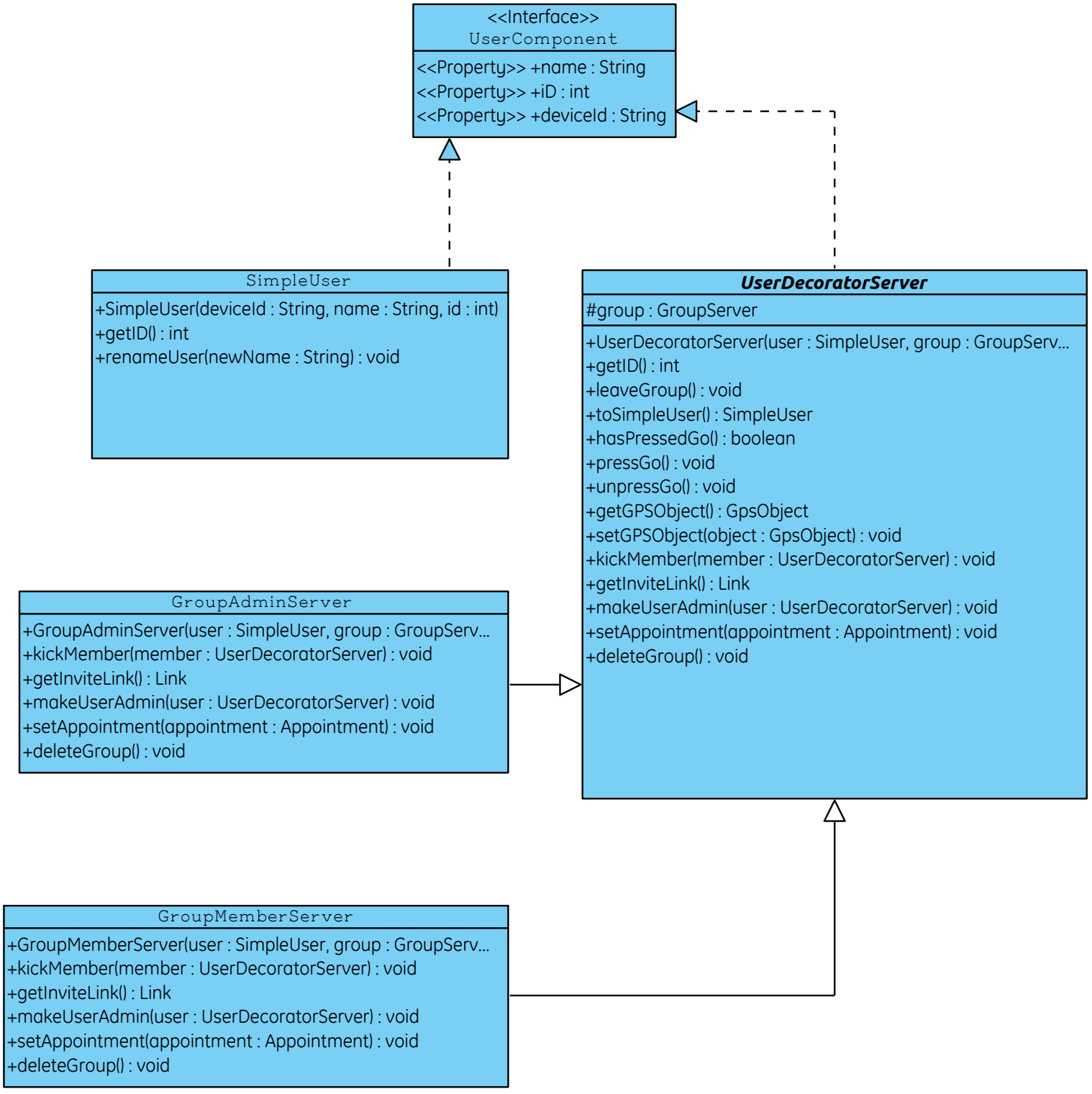


Abbildung 13: Klassenhierarchie UserDecoratorServer

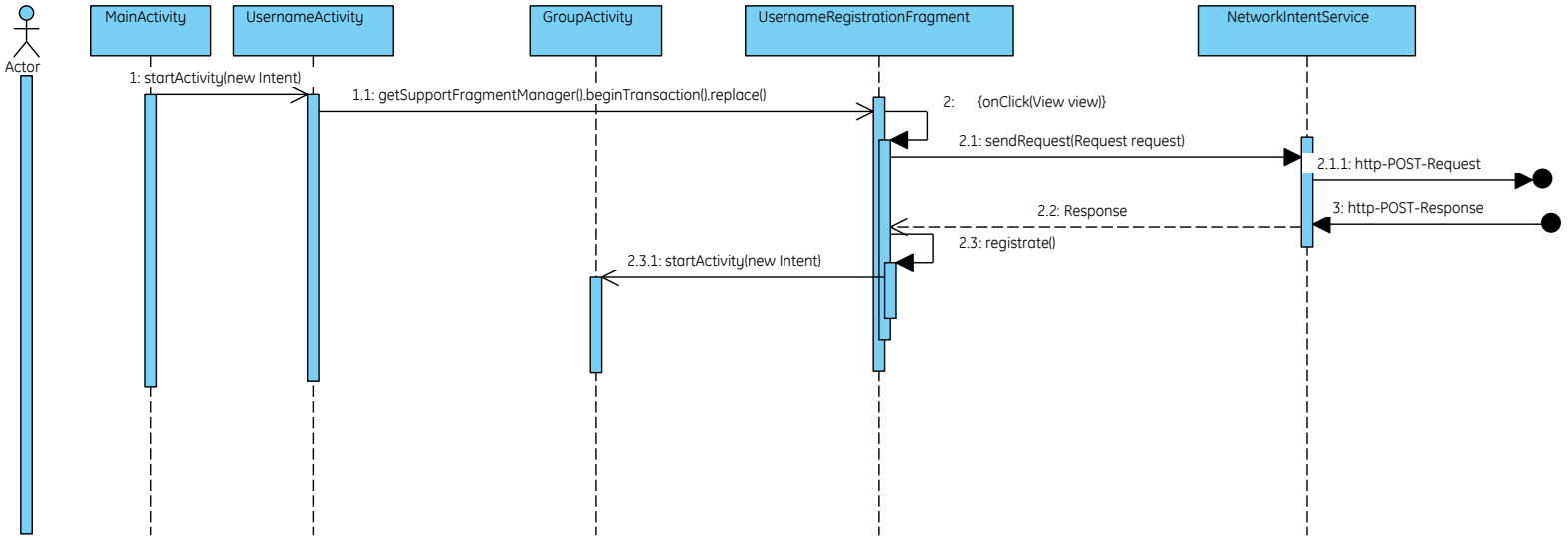
Klasse GroupMemberServer

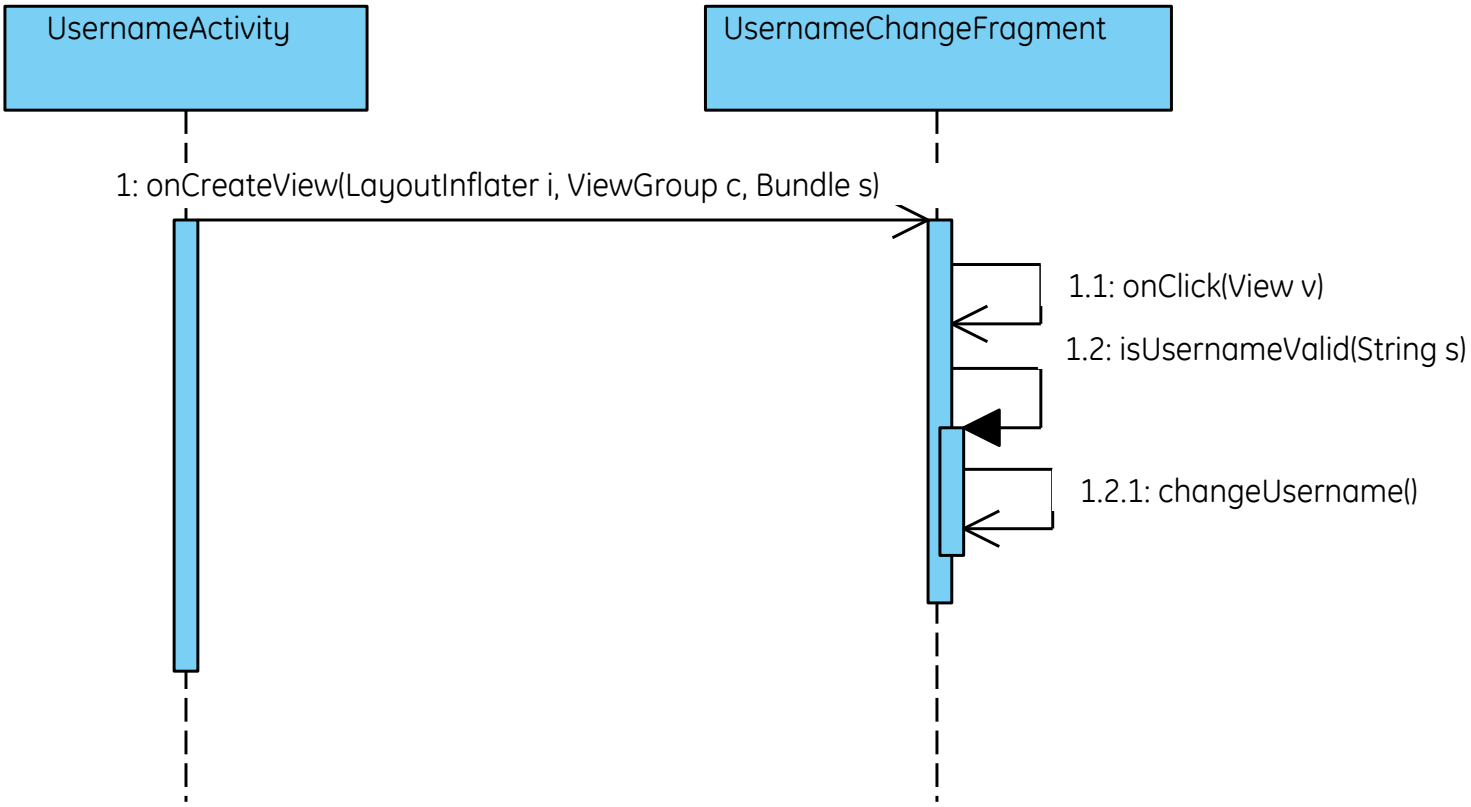
Die serverseitige Klasse für Gruppenmitglieder. Die einfache Implementierung eines Gruppenmitglieds ohne besondere Rechte.
Die Klasse GroupMemberServer implementiert auch die Methoden des Administrators, falls ein skrupelloses Gruppenmitglied eine Admin-Anfrage sendet, wird dabei jedoch eine Fehlermeldung ausgelöst.

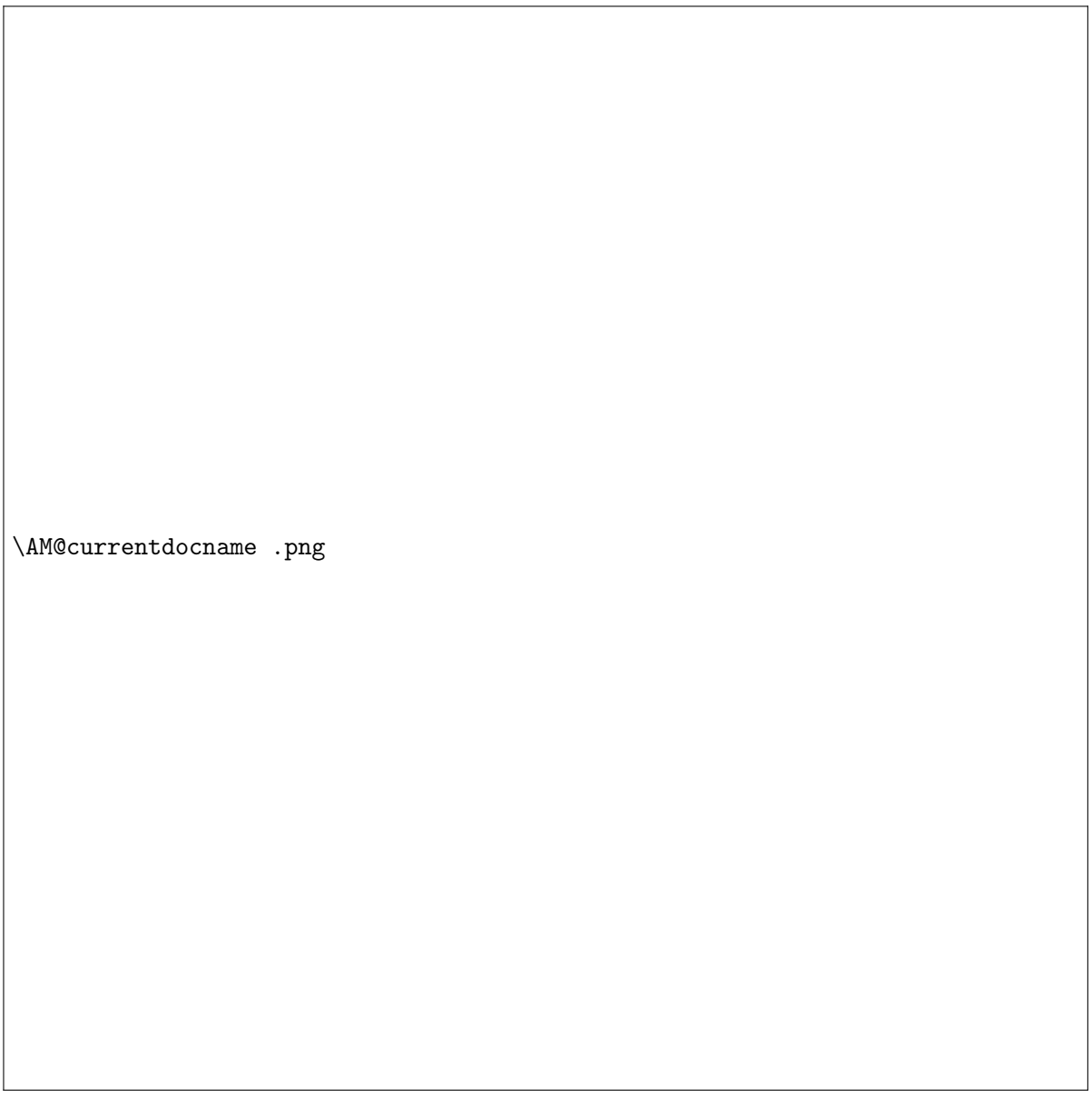
Klasse GroupAdminServer

Die Gruppen-Administrator Klasse auf dem Server steuert alle Gruppenoperationen, das heißt, wenn man über group.getUser(ID) ein Objekt der Klasse GroupAdminServer zurückbekommt, kann man darüber die Gruppe 'group' kontrollieren.
Als Indirektion ruft der Die Operationen sind an die Gruppe gebunden, auf der man getUser(ID) aufruft.

3 Sequenzdiagramme


Actor
MainActivity
UsernameActivity
GroupActivity
UsernameRegistrationFragment
NetworkIntentService
1: startActivity(new Intent)
1.1: getSupportFragmentManager().beginTransaction().replace()
2: {onClick(View view)}
2.1: sendRequest(Request request)
2.1.1: http-POST-Request
2.2: Response
3: http-POST-Response
2.3: registrate()
2.3.1: startActivity(new Intent)


UsernameActivity
UsernameChangeFragment
1: onCreateView(LayoutInflater i, ViewGroup c, Bundle s)
1.1: onClick(View v)
1.2: isUsernameValid(String s)
1.2.1: changeUsername()



.png

4 Änderungen zum Pflichtenheft

Die folgenden Kriterien haben wir während der Entwurfsphase aus unseren Wunschkriterien gestrichen:

1. Benutzer und Gruppen nach langer Inaktivität löschen, um Gruppennamen wieder verwenden zu können und um unnötigen Speicherverbrauch zu reduzieren.

2. Benachrichtigung bei Gruppenaktivität, also wenn ein neues Treffen festgelegt wurde oder sich jemand schon auf den Weg zum Treffpunkt gemacht hat.

3. Anzeigen einer Zahl, wie viele Gruppenmitglieder schon den Go Button gedrückt haben.

Die folgenden Dinge gehören zu unserem Entwurf, welche wir jedoch aus Zeitgründen, wegen nicht zur Verfügung stehender Gruppenmitglieder unsere PSE Gruppe, leider nicht hinbekommen haben.

1. Leider konnten wir nicht weiter auf unsere Entwurfsmuster eingehen, wobei es sich v.a. um das Decorator pattern für den Benutzer handelt.

2. Der Grobentwurf unseres Entwurfhefts ist etwas kürzer als erhofft.

3. Wir konnten auch nicht darauf eingehen, ob unsere Pakete unabhängig voneinander testbar sind.

4. Wir hätten gerne etwas übersichtlicher unsere Funktionen aus dem Pflichtenheft mit denen im Entwurfsheft verknüpft.

«««< HEAD ======= -Benutzer und Gruppen nach Inaktivität löschen -Benachrichtigung bei Gruppenaktivität »»»> lastminute