

# Entwurf Go-App

Rumen Donchev, Kansei Hara, Grischa Hauser, Tanja Müller, Paula Klein, Iris Landerer

19.06.2016

## Inhaltsverzeichnis

<b>1 Softwarearchitektur mit Systemkomponenten</b>	<b>7</b>
<b>2 Entwurfsentscheidungen</b>	<b>10</b>
2.1 Client . . . . .	10
2.1.1 Model-View-Controller-Architektur . . . . .	10
2.1.2 Klassendiagramm . . . . .	14
2.1.3 Activity . . . . .	15
2.1.4 Service . . . . .	17
2.1.5 URI_Builder . . . . .	20
2.1.6 HttpAppClient . . . . .	22
2.1.7 Converter . . . . .	23
2.1.8 Receiver . . . . .	24
2.1.9 Database-Handler . . . . .	25
2.2 Server . . . . .	26
2.2.1 Klassendiagramm . . . . .	26
2.2.2 Objektdiagramm . . . . .	28
2.2.3 Servlets . . . . .	30
2.2.4 DataAccessObjects . . . . .	32
2.2.5 Converter . . . . .	34
2.2.6 Datenbank . . . . .	36
<b>3 Client ↔ Server RESTful API</b>	<b>38</b>
3.1 HTTP-Protokoll . . . . .	38
3.2 Statuscodes . . . . .	39
3.3 Class GroupServlet . . . . .	39
3.3.1 Neue Gruppe erstellen . . . . .	39
3.3.2 Gruppeninformationen anzeigen . . . . .	39
3.3.3 Gruppe bearbeiten . . . . .	40
3.3.4 Gruppe löschen . . . . .	41
3.4 Class GroupUserManagementServlet . . . . .	41
3.4.1 Benutzer zu einer Gruppe hinzufügen . . . . .	41
3.4.2 Liste der Gruppenmitglieder ausgeben . . . . .	42
3.4.3 Gruppenmitglied zum Admin (ab)wählen . . . . .	43

3.4.4	Gruppenmitglied löschen . . . . .	43
3.5	Class GroupsServlet . . . . .	44
3.5.1	Liste der Gruppen anzeigen . . . . .	44
3.6	Class LoginServlet . . . . .	45
3.6.1	Neuen Benutzer in der Go-App registrieren . . . . .	45
3.6.2	Überprüft, ob der Benutzer registriert ist . . . . .	45
3.6.3	In die Go-App einloggen . . . . .	46
3.7	Class LogoutServlet . . . . .	46
3.7.1	Aus der Go-App ausloggen . . . . .	46
3.8	Class LoginFilterServlet . . . . .	47
3.8.1	Prüft, ob der Benutzer auf die angeforderte Ressource zugreifen darf . . . . .	47
3.9	Class NotificationsServlet . . . . .	47
3.9.1	Liste von Nachrichten ausgeben . . . . .	47
3.9.2	Nachrichtenanzeigen an-/ausschalten . . . . .	48
3.10	Class UserServlet . . . . .	48
3.10.1	Neuen Benutzer erstellen . . . . .	48
3.10.2	Benutzerinformationen anzeigen . . . . .	49
3.10.3	Benutzerinformationen ändern . . . . .	49
3.10.4	Benutzer löschen . . . . .	50
3.11	UsersServlet . . . . .	50
3.11.1	Liste aller Benutzer ausgeben . . . . .	50
3.12	Class GPSServlet . . . . .	51
3.12.1	GPS-Daten updaten . . . . .	51
3.13	Class MeetingServlet . . . . .	51
3.13.1	Termin erstellen . . . . .	51
3.13.2	Termininfos . . . . .	52
3.13.3	Termin ändern . . . . .	53
3.13.4	Termin löschen . . . . .	54
3.14	Class MeetingsServlet . . . . .	55
3.14.1	Liste mit allen Terminen ausgeben . . . . .	55
3.15	Class MeetingParticipantManagementServlet . . . . .	56
3.15.1	Liste von Teilnehmern hinzufügen . . . . .	56
3.15.2	Teilnehmer anzeigen . . . . .	56
3.15.3	Zusage ändern . . . . .	57
3.15.4	Teilnehmer aus dem Termin löschen . . . . .	57
<b>4</b>	<b>Klassen des Clients</b>	<b>59</b>
4.1	Package kit.edu.pse.goapp.client.datamodels . . . . .	59
4.2	Package kit.edu.pse.goapp.client.activity . . . . .	59
4.2.1	public class AppCompatActivity . . . . .	59
4.2.2	public interface View.OnClickListener . . . . .	59
4.2.3	Interface PopupMenu.OnMenuItemClickListener . . . . .	59
4.2.4	public class GroupsActivity extends AppCompatActivity implements View.OnClickListener implements PopupMenu.OnMenuItemClickListener MyResultReceiver.Receiver . . . . .	60

4.2.5	public class GroupMemberActivity extends AppCompatActivity implements View.OnClickListener implements PopupMenu.OnMenuItemClickListener MyResultReceiver.Receiver . . . . .	60
4.2.6	public class SettingsActivity extends AppCompatActivity implements View.OnClickListener implements PopupMenu.OnMenuItemClickListener MyResultReceiver.Receiver . . . . .	60
4.2.7	public class CreateNewGroupActivity extends AppCompatActivity implements View.OnClickListener implements PopupMenu.OnMenuItemClickListener MyResultReceiver.Receiver . . . . .	61
4.2.8	public class MeetingParticipantActivity extends AppCompatActivity implements View.OnClickListener implements PopupMenu.OnMenuItemClickListener MyResultReceiver.Receiver . . . . .	61
4.2.9	public class AboutActivity extends AppCompatActivity implements View.OnClickListener implements PopupMenu.OnMenuItemClickListener MyResultReceiver.Receiver . . . . .	61
4.2.10	public class MapActivity extends AppCompatActivity implements View.OnClickListener implements PopupMenu.OnMenuItemClickListener MyResultReceiver.Receiver . . . . .	62
4.2.11	public class LoginActivity extends AppCompatActivity implements View.OnClickListener implements PopupMenu.OnMenuItemClickListener MyResultReceiver.Receiver . . . . .	62
4.2.12	public class MeetingListActivity extends AppCompatActivity implements View.OnClickListener implements PopupMenu.OnMenuItemClickListener MyResultReceiver.Receiver . . . . .	63
4.2.13	public class NewMeetingActivity extends AppCompatActivity implements View.OnClickListener implements PopupMenu.OnMenuItemClickListener MyResultReceiver.Receiver . . . . .	63
4.2.14	public abstract class ResultReceiver . . . . .	63
4.2.15	public class ServiceResultReceiver extends ResultReceiver implements Receiver . . . . .	64
4.3	Package kit.edu.pse.goapp.client.service . . . . .	64
4.3.1	public abstract class IntentService . . . . .	64
4.3.2	public class LoginService extends IntentService . .	65
4.3.3	public class LogoutService extends IntentService . .	65
4.3.4	public class LoginFilterService extends IntentService	66
4.3.5	public class UserService extends IntentService . .	66
4.3.6	public class UsersService extends IntentService . .	67
4.3.7	public class GroupService extends IntentService . .	67
4.3.8	public class GroupsService extends IntentService . .	68
4.3.9	public class GroupUserManagementService extends IntentService	68
4.3.10	public class GPS_Service extends IntentService . .	69
4.3.11	public class MeetingService extends IntentService . .	69
4.3.12	public class MeetingsService extends IntentService . .	70
4.3.13	public class MeetingParticipantManagementService extends IntentService . . . . .	70
4.3.14	public class NotificationService extends IntentService	71

4.4	Package kit.edu.pse.goapp.client.uri_builder . . . . .	71
4.4.1	public abstract class ClientURI_Builder . . . . .	71
4.4.2	public class URI_LoginBuilder . . . . .	72
4.4.3	public class URI_LogoutBuilder . . . . .	72
4.4.4	public class URI_LoginFilterBuilder . . . . .	72
4.4.5	public class URI_GroupBuilder . . . . .	73
4.4.6	public class URI_GroupsBuilder . . . . .	73
4.4.7	public class URI_GroupUserManagementBuilder . . .	73
4.4.8	public class URI_MeetingBuilder . . . . .	74
4.4.9	public class URI_MeetingsBuilder . . . . .	74
4.4.10	public class URI_MeetingParticipantManagementBuilder	75
4.4.11	public class URI_UserBuilder . . . . .	75
4.4.12	public class URI_UsersBuilder . . . . .	75
4.4.13	public class URI_GPS_Builder . . . . .	76
4.4.14	public class URI_NotificationsBuilder . . . . .	76
4.5	Package kit.edu.pse.goapp.client.httpappclient . . . . .	76
4.5.1	public class HttpResponse . . . . .	76
4.5.2	public class HttpAppClient . . . . .	77
4.6	Package kit.edu.pse.goapp.client.converter . . . . .	78
4.6.1	Interface Converter<T> . . . . .	78
4.6.2	Class NotificationConverter implements Converter<T>	78
4.6.3	Class GroupMemberConverter implements Converter<T>	79
4.6.4	Class GroupConverter implements Converter<T> . . .	80
4.6.5	Class GPS_Converter implements Converter<T> . . .	81
4.6.6	Class UserConverter implements Converter<T> . . . .	82
4.6.7	Class ParticipantConverter implements Converter<T>	83
4.6.8	Class MeetingConverter implements Converter<T> . .	83
4.7	Package kit.edu.pse.goapp.client.receiver . . . . .	84
4.7.1	Abstract class BroadcastReceiver . . . . .	84
4.7.2	Class AlarmReceiver extends BroadcastReceiver . . .	85
4.7.3	Class DeviceBootReceiver extends BroadcastReceiver	85
4.8	Package kit.edu.pse.goapp.client.databasehandler . . . . .	85
4.8.1	abstract class SQLiteOpenHelper . . . . .	85
4.8.2	Class DataBaseHandler extends SQLiteOpenHelper . .	86
<b>5</b>	<b>Klassen des Servers</b>	<b>87</b>
5.1	Package kit.edu.pse.goapp.server.datamodels . . . . .	87
5.1.1	Abstract class Meeting . . . . .	87
5.1.2	Class Event extends Meeting . . . . .	88
5.1.3	Class Tour extends Meeting . . . . .	88
5.1.4	Class MeetingCenter . . . . .	89
5.1.5	Class Participant . . . . .	89
5.1.6	Enumeration MeetingConfirmation . . . . .	90
5.1.7	Class User . . . . .	90
5.1.8	Class Group . . . . .	91
5.1.9	Class GPS . . . . .	92

5.1.10	Class Notification . . . . .	93
5.2	Package kit.edu.pse.goapp.server.daos . . . . .	94
5.2.1	Interface GroupDAO . . . . .	94
5.2.2	Class GroupDAOImpl implements GroupDAO . . . . .	94
5.2.3	Interface GroupMemeberDAO . . . . .	95
5.2.4	Class GroupMemberDAOImpl implements GroupMemberDAO	95
5.2.5	Interface MeetingDAO . . . . .	96
5.2.6	Class MeetingDAOImpl implements MeetingDAO . . . . .	96
5.2.7	Interface ParticipantDAO . . . . .	97
5.2.8	Class ParticipantDAOImpl implements ParticipantDAO	97
5.2.9	Interface UserDAO . . . . .	97
5.2.10	Class UserDAOImpl implements UserDAO . . . . .	98
5.2.11	Interface GPS.DAO . . . . .	99
5.2.12	Class GPS.DAOImpl implements GPS.DAO . . . . .	99
5.2.13	Interface NotificationDAO . . . . .	99
5.2.14	Class NotificationDAOImpl implements NotificationDAO	100
5.3	Package kit.edu.pse.goapp.server.converter . . . . .	100
5.3.1	Interface Converter<T> . . . . .	100
5.3.2	Class NotificationConverter implements Converter<T>	101
5.3.3	Class GroupMemberConverter implements Converter<T>	101
5.3.4	Class GroupConverter implements Converter<T> . . .	102
5.3.5	Class GPS_Converter implements Converter<T> . . .	103
5.3.6	Class UserConverter implements Converter<T> . . .	104
5.3.7	Class ParticipantConverter implements Converter<T>	105
5.3.8	Class MeetingConverter implements Converter<T> . .	106
5.4	Package kit.edu.pse.goapp.server.database . . . . .	107
5.4.1	Class Group . . . . .	107
5.4.2	Class GroupMembers . . . . .	107
5.4.3	Class User . . . . .	107
5.4.4	Class Meeting . . . . .	108
5.4.5	Class Participant . . . . .	108
<b>6</b>	<b>Beschreibung charakteristischer Abläufe</b>	<b>109</b>
6.1	Allgemeiner Ablauf eines Requests des Clients . . . . .	109
6.2	Ablauf der Methode getAllMeetings() . . . . .	111
6.3	Ablauf der Methode deleteGroup(int groupId) . . . . .	113
6.4	Ablauf der Methode addAdmin(int groupId, int UserId, boolean adminStatus) bei Aufruft einer Errors . . . . .	114
6.5	Ablauf der Registrierung in das System . . . . .	117
6.6	Ablauf des Logins in das System . . . . .	118
6.7	GPS mithilfe des AlarmReceivers . . . . .	120

<b>7 Änderungen gegenüber dem Pflichtenheft</b>	<b>122</b>
7.1 Gruppeneinladungen . . . . .	122
7.2 Änderung der Prioritäten der funktionalen Anforderungen . . . . .	122
7.2.1 Gruppenverwaltung . . . . .	122
7.2.2 Terminverwaltung . . . . .	123
7.2.3 Lokalisierung . . . . .	124
7.2.4 Benutzerverwaltung . . . . .	125
<b>8 Anhang: Vollständiges großformatiges Klassendiagramm</b>	<b>126</b>

# 1 Softwarearchitektur mit Systemkomponenten

Dieser Abschnitt stellt einen Überblick über die einzelnen Komponenten der Software dar. Genaue Beschreibungen der dabei auftretenden Komponenten befinden sich bei den Entwurfsentscheidungen (Kapitel 2) und bei den Klassenbeschreibungen (Kapitel 3/4/5).

Die App verwendet eine REST-Architektur. Es gilt generell die Anforderung, dass alle Eigenschaften der Client-Server-Architektur gelten. Dadurch ergeben sich Portabilität, Skalierbarkeit und unabhängige Entwicklung der Komponenten als Vorteil. Ein weiteres Grundprinzip ist die Zustandslosigkeit. Jede REST-Nachricht enthält alle Informationen, die für den Server bzw. Client notwendig sind, um die Nachricht zu verstehen. Dies führt zu Skalierbarkeit, Einfachheit und Transparenz. Außerdem ist das System mehrschichtig. Dadurch reicht es, dem Anwender lediglich eine Schnittstelle, was Einfachheit und gute Entwickelbarkeit zur Folge hat, anzubieten. Dahinterliegende Ebenen können verborgen bleiben und somit die Architektur insgesamt vereinfacht werden.

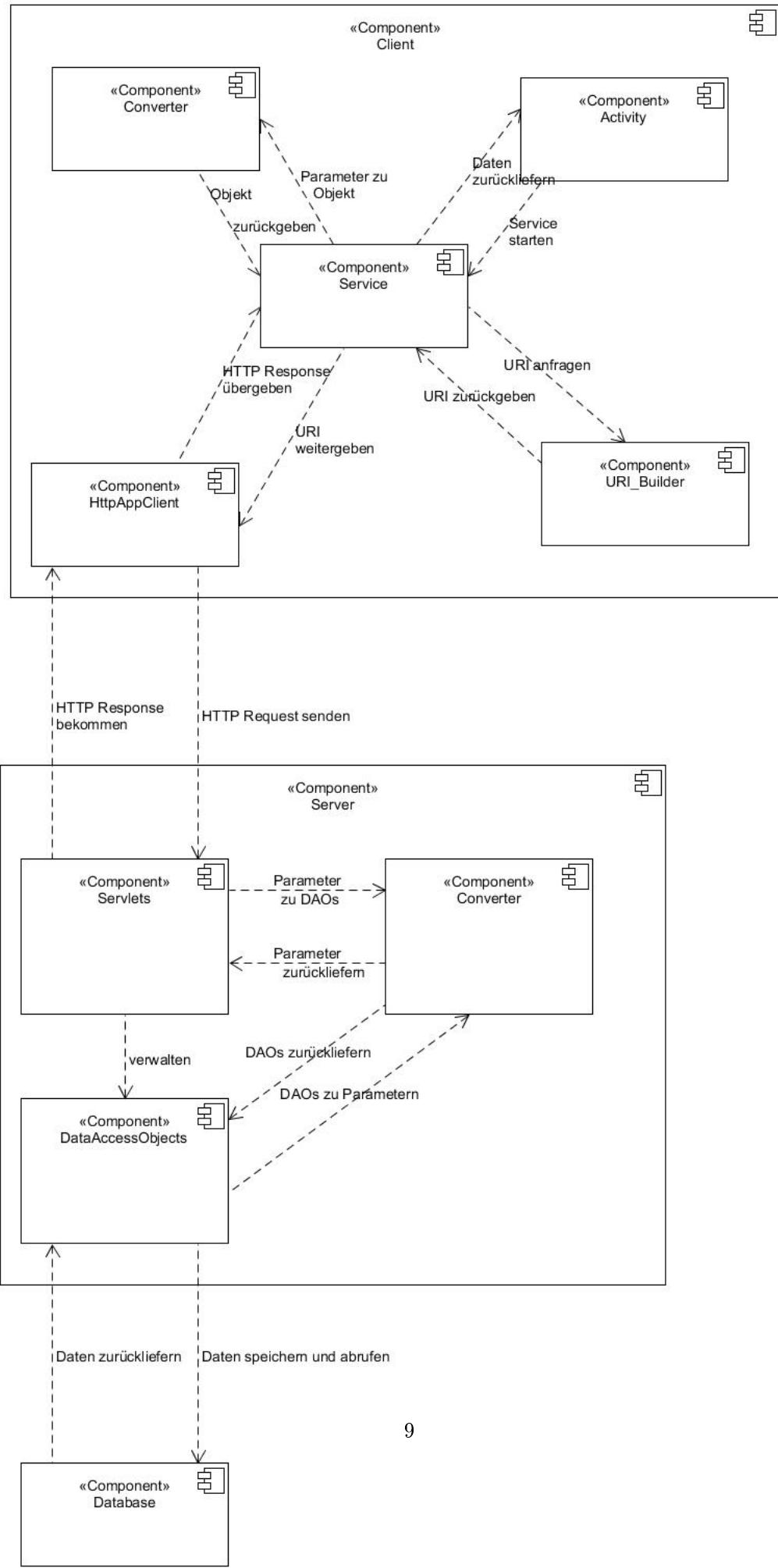
Zur Umsetzung der REST-Architektur verwendet die App das HTTP-Protokoll. Gründe dafür gibt es viele: Einerseits ist das Protokoll etabliert, dennoch vergleichsweise einfach aufgebaut und zu guter Letzt auch in so gut wie jeder Firewall offen. Außerdem lässt sich einfach eine SSL-Verschlüsselung hinzufügen (HTTPS).

Im Client wird zusätzlich die Model-View-Controller-Architektur verwendet, da die Architektur einen flexiblen Programmumentwurf ermöglicht, welcher die Wiederverwendbarkeit der einzelnen MVC-Module gewährleistet und außerdem die Gesamtkomplexität reduziert. Bestehende Systeme können leicht erweitert werden, indem neue MVC-Module hinzugefügt werden.

Auf der Serverseite verwenden wir eine MySQL-Datenbank, da diese uns eine große Flexibilität bereitstellt und des weiteren zeichnet sich eine MySQL-Datenbank durch ihre Performance, Skalierbarkeit und ihrer Akzeptanz gegenüber paralleler Zugriffe aus.

Einen Überblick über den Ablauf des Entwurfs wird im Folgenden gegeben:  
Auf der Activity wird eine Methode ausgeführt und diese startet daraufhin den für die Methode zuständigen Service. Der Service erstellt eine Instanz von URI\_Builder und fragt daraufhin die URI an, die die Adresse für das zugehörige Servlet im Server angibt und außerdem die Parameter, die an den Server geschickt werden sollen, hält. Der Service schickt dann die URI mithilfe einer Instanz des HttpAppClient mit einem HTTP Request, der die Parameter der Anfrage als JSON String enthält, zu dem jeweiligen Servlet im Server. Dort wird der Request verarbeitet und dem Converter übergeben, der den JSON String zu einem Objekt parst, nämlich dem DataAccessObject (DAO). Das Servlet verwaltet nun das DAO. Das DAO stellt eine Schnittstelle zur Datenbank dar und ruft Daten ab oder speichert sie in der Datenbank. Wenn benötigt, liefert die Datenbank Daten an das DAO zurück. Diese werden dann wiederum mithilfe des Converters zu einem JSON String serialisiert und an das Servlet zurückgeliefert. Das Servlet schickt den JSON String zusammen mit einem HTTP Statuscode, der angibt, ob der Request erfolgreich durchgeführt wurde, in einer

HTTP Response an den Client zurück. Dort wird der JSON String im Converter wiederum geparsst und die Objekte an den Service zurückgegeben, der dann die Daten an die Activity übermittelt.



## 2 Entwurfsentscheidungen

### 2.1 Client

#### 2.1.1 Model-View-Controller-Architektur

Die Clientseite von unserem Projekt, wird eine Android-App sein, die in Java implementiert ist. Da die Clientseite keine Logik ausführen muss und nur als Schnittstelle zwischen Server und User benutzt wird, haben wir uns für das Model-View-Controller-Architekturmuster entschieden.

##### Model

Die Objektmodelle entsprechen denen des DataModels und sind identisch zu denen auf dem Server, da wir mit den gleichen Objekten arbeiten.

##### View

Die View bestehen aus einer oder mehreren XML-Dateien für jede Activity und den zugehörigen Activity-Klassen. Die Activities nehmen die Benutzerinteraktionen entgegen und stellen diese in den XML-Dateien dar. Die Methode `onClickListener()` ist für die Interaktion zuständig.

##### Controller

Unsere Services sind für die Koordination der Kommunikation zwischen der View und dem Server zuständig. Fast jeder Service schickt oder holt Information vom Server und übergibt diese an die Activities weiter. Eine Activity kann zu einem oder mehreren Services in Bezug stehen..

Im Folgenden werden die MVCs beschrieben:

##### LoginActivity

View	Model	Controller
LoginActivity	User	LoginService

Benutzer Login oder Registrierung

Kommunikation:

Beim Login fordert die Activity den Benutzernamen und das Passwort vom Benutzer an und erzeugt dann einen LoginService. Bei der Registrierung fordert die Activity einen Benutzernamen, Passwort und Google-Account und erzeugt ebenfalls einen LoginService.

##### MapActivity

View	Model	Controller
MapActivity	Meeting, GPS, MeetingCenterParticipant (bei Touren)	MeetingParticipantManagementService

Zeigt den Treffpunkt auf der Karte an:

Bei einer Tour wird die größte Teilnehmeransammlung auf der Karte angezeigt mit den Teilnehmern, die sich darin befinden.

Bei einer Veranstaltung werden alle Teilnehmer in einem bestimmten Umkreis angezeigt.

Kommunikation:

Erzeugt einen MeetingService und bekommt die Termininformationen zurück, die die GPS-Daten und die Teilnehmer, die sich im Zentrum befinden, beinhalten.

### **MeetingListActivity**

View	Model	Controller
MeetingListActivity	Meetings(Event und Tour)	MeetingsService

Diese Activity zeigt eine Liste von Terminen und die jeweiligen Termininformationen an. In dieser Activity kann man auch einer Terminanfrage ab- oder zusagen.

Kommunikation:

Um die Liste von Terminen anzuzeigen, wird ein MeetingsService erzeugt, der eine Liste von Participant-Objekten zurückliefert. Beim Zu- oder Absagen eines Termins wird dafür ein MeetingService erzeugt, der diese Information an den Server weiterleitet.

### **MeetingParticipantsActivity**

View	Model	Controller
MeetingParticipantsActivity	Participant	MeetingParticipantManagemtService

Zeigt eine Liste von zugesagten Teilnehmern eines Termins an

Kommunikation:

Erstellt ein MeetingParticipantManagemtService, der eine Liste von Participant-Objekten zurückliefert.

### **NewMeetingActivity**

View	Model	Controller
NewMeetingActivity	Meetings(Event und Tour)	MeetingsService

In dieser Activity kann man einen neuen Termin erstellen.

Kommunikation:

Beim erstellen das Treffen wird einen MeetingService erzeugt.

### **GroupsActivity**

View	Model	Controller
GroupsActivity	Group	GroupsService

Zeigt eine Liste von Gruppen an und Gruppen können hier gelöscht werden.

Kommunikation:

Um die Liste von Gruppen anzuzeigen, wird ein GroupsService erstellt, der eine Liste von Group-Objekten zurückliefert. Um eine Gruppe zu löschen, wird ein GroupService erzeugt.

### **GroupMembersActivity**

View	Model	Controller
<b>GroupMemberActivity</b>	<b>User, GroupUserManagement</b>	<b>UsersService, GroupUserManagementService</b>

Zeigt alle Gruppenmitglieder an, Gruppen können hier verlassen werden, Administratoren können Mitglieder suchen und hinzufügen oder sie entfernen.  
Kommunikation:

Für die Liste wird ein GroupUserManagementService erzeugt, der eine Liste von Teilnehmern zurückliefert.

Um eine Gruppe zu verlassen oder Mitglieder hinzuzufügen oder zu löschen, wird ein GroupUserManagementService erzeugt.

Für das Suchen eines Users wird ein UsersService erzeugt, der eine Liste von User-Objekten als Vorschläge zurückgibt.

### **CreateNewGroupActivity**

View	Model	Controller
<b>CreateNewGroupActivity</b>	<b>Group, User</b>	<b>GroupService, UsersService</b>

Hier kann man eine neue Gruppe erstellen und Benutzer zu dieser Gruppe hinzufügen.

Kommunikation:

Um eine neue Gruppe zu erstellen, wird ein GroupService erstellt.

Für die Suche nach Benutzern wird ein UsersService erstellt, der eine Liste von allen Suchtreffern als User-Objekte zurückgibt.

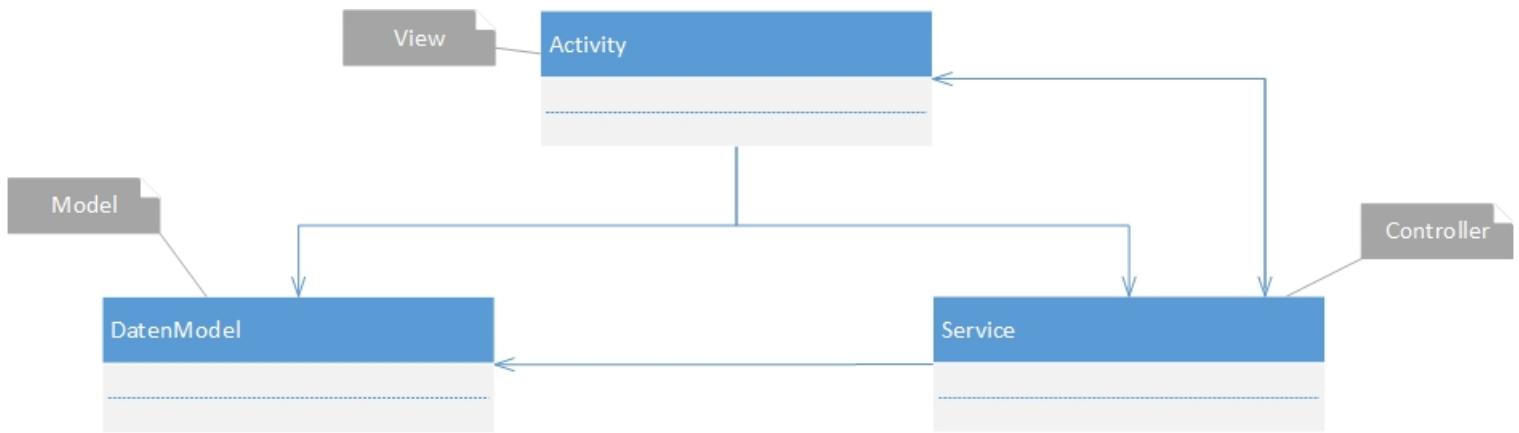
### **SettingsActivity**

View	Model	Controller
<b>SettingsActivity</b>	<b>Notification</b>	<b>NotificationsService</b>

Anzeigen und Ändern der Einstellungen (Standort anzeigen und Benachrichtigung ein/ausschalten)

Kommunikation:

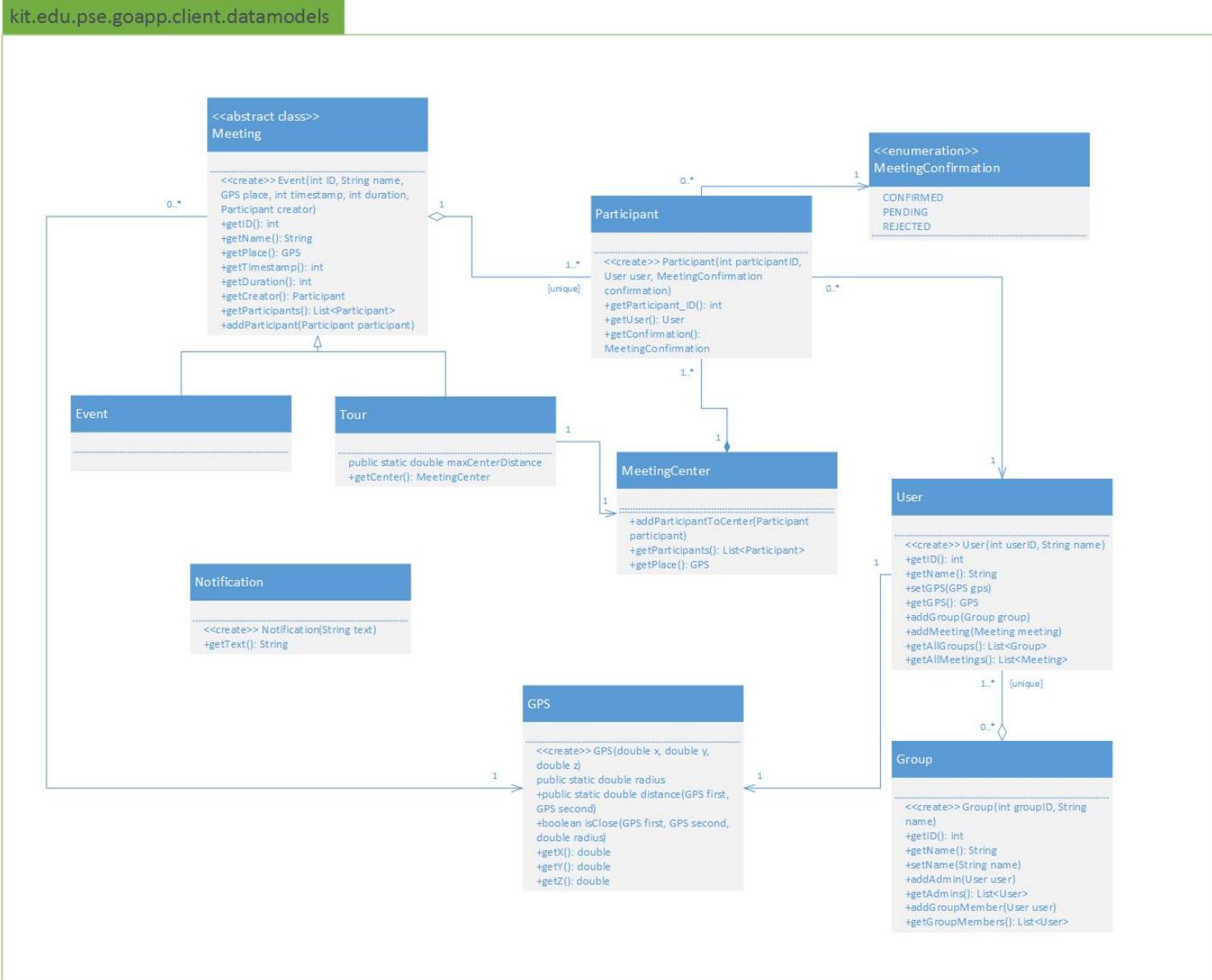
Erstellt eine NotificationService um die Benachrichtigungseinstellung an den Server weiter zu geben. (Priorität B)



## 2.1.2 Klassendiagramm

kit.edu.pse.goapp.client.datamodels

Bei dem Klassendiagramm handelt es sich um das gleiche, dass auch beim Server



### 2.1.3 Activity

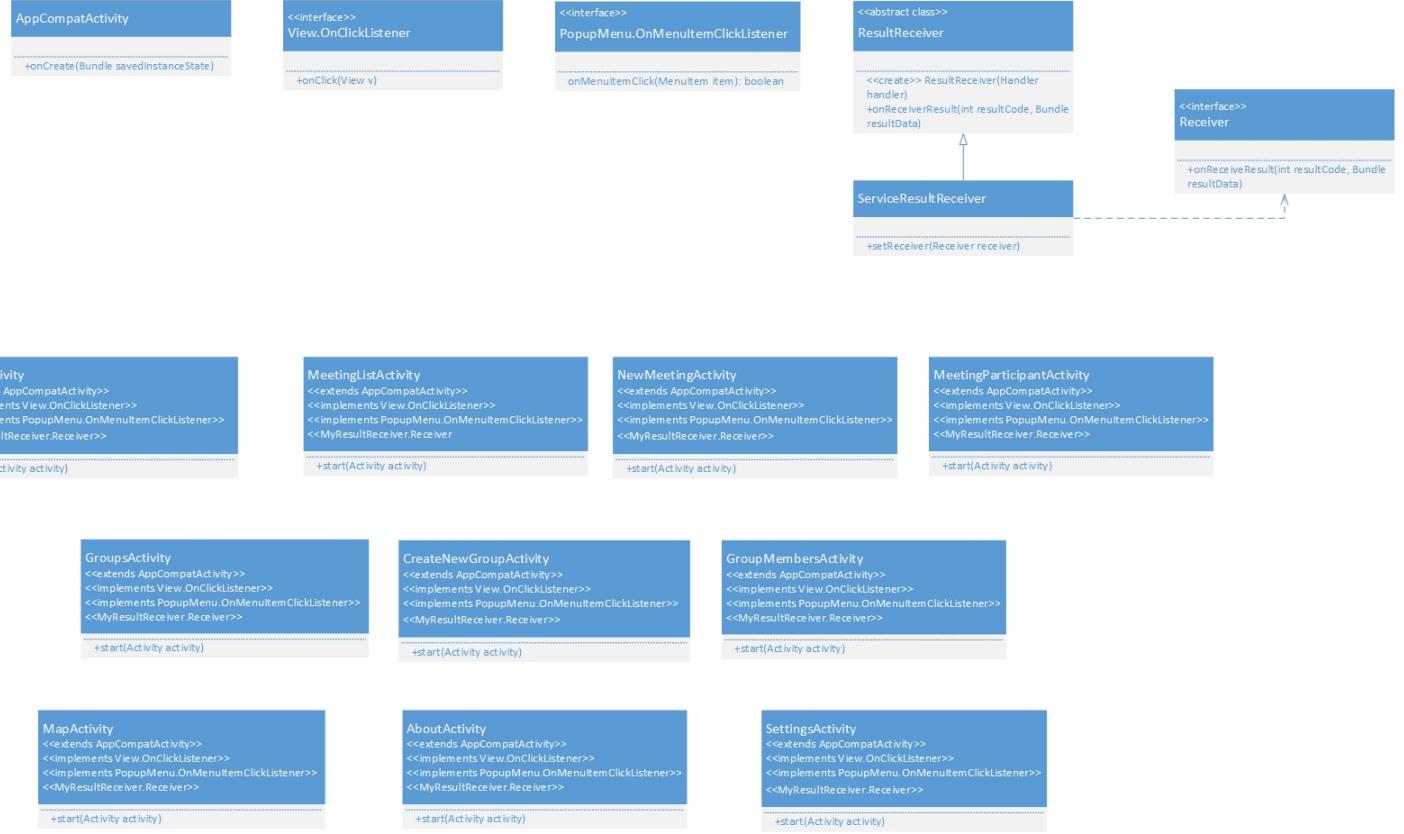
kit.edu.pse.goapp.client.activity

Die App besteht aus zehn Activities:

LoginActivity	MeetingListActivity
MeetingParticipantsActivity	NewMeetingActivity
GroupsActivit	GroupMembersActivity
CreateNewGroupActivity	SettingsActivity
AboutActivity	MapActivity

Jede Activity nimmt die Benutzerinteraktionen entgegen und stellt die darzustellenden Daten dar. Jede Klasse hat die Methoden onCreate(), onDestroy(), onPause(), onResume() und onClick(). Die ersten vier Methoden kümmern sich darum, was passiert, wenn die Activity erstellt, zerstört, pausiert oder fortgesetzt wird. Die Methode onClick() ist für die Benutzerinteraktion zuständig. Alle Methoden werden vom Android OS automatisch aufgerufen. Die Methoden onCreate() und onClick() werden von den Unterklassen überschrieben.

Alle Activities außer AboutActivity werden im Model-View-Controller-Diagramm beschrieben (siehe Kapitel 2.1.1: Model-View-Controller). Die AboutActivity zeigt nur die Informationen über die App, ihre Ersteller und Rechtliches an.



#### 2.1.4 Service

```
kit.edu.pse.goapp.client.service
```

Unsere App hat 13 Services, die die Kommunikation zwischen der View und dem Server koordinieren. Der Vorteil der Vererbung von IntentService ist, dass er auf einem eigenen Prozess läuft, der neben der MainUI läuft, ohne diese zu blockieren. Der Service wird sich am Ende des Prozesses von allein zerstören. Beispielsweise kann man in einem Benutzersuchfeld nach den Benutzern suchen, während noch der Name eingetippt wird.

Um einen Service zu starten, benötigt man einen Intent mit einer darin enthaltenen Beschreibung der Operation und dieser wird wie folgt erzeugt:

```
Intent intent = new Intent(Intent.ACTION_SYNC, null, this, Service.class);  
Intent(String action, Uri uri, Context packageContext, Class <? > cls);
```

Dann wird der Service durch startService(Intent) gestartet.

Unsere Services sind für verschiedene Anfragen verantwortlich:

##### **LoginService:**

- Benutzerregistrierung
- Login
- Überprüfen, ob ein Benutzer registriert ist

##### **LogoutService:**

- Ausloggen eines Benutzers

##### **LoginFilterService:**

- Überprüfen, ob ein Benutzer auf die angeforderten Ressourcen zugreifen darf

##### **GroupService:**

- Erstellen einer Gruppe
- Gruppeninformationsanfragen
- Gruppenänderungen
- Löschen einer Gruppen

**GroupsService:**

- Gruppenlistenanforderungen

**GroupUserManagementService:**

- Hinzufügen von Benutzern in Gruppen
- Anforderung einer Liste der Gruppenmitglieder einer Gruppe

**MeetingService:**

- Erstellen eines Termins
- Termininformationsanfragen
- Termin Änderungen
- Löschen eines Termins

**MeetingsService:**

- Terminlistenanforderungen

**MeetingParticipantManagementService:**

- Anforderung einer Liste von Teilnehmern, die dem Termin zugesagt haben
- Bestätigung bzw. das Absagen eines Termins

**UserService:**

- Erstellen eines Benutzers
- Benutzerinformationsanfragen
- Änderungen der Benutzerinformationen
- Löschen eines Benutzers

**UsersService:**

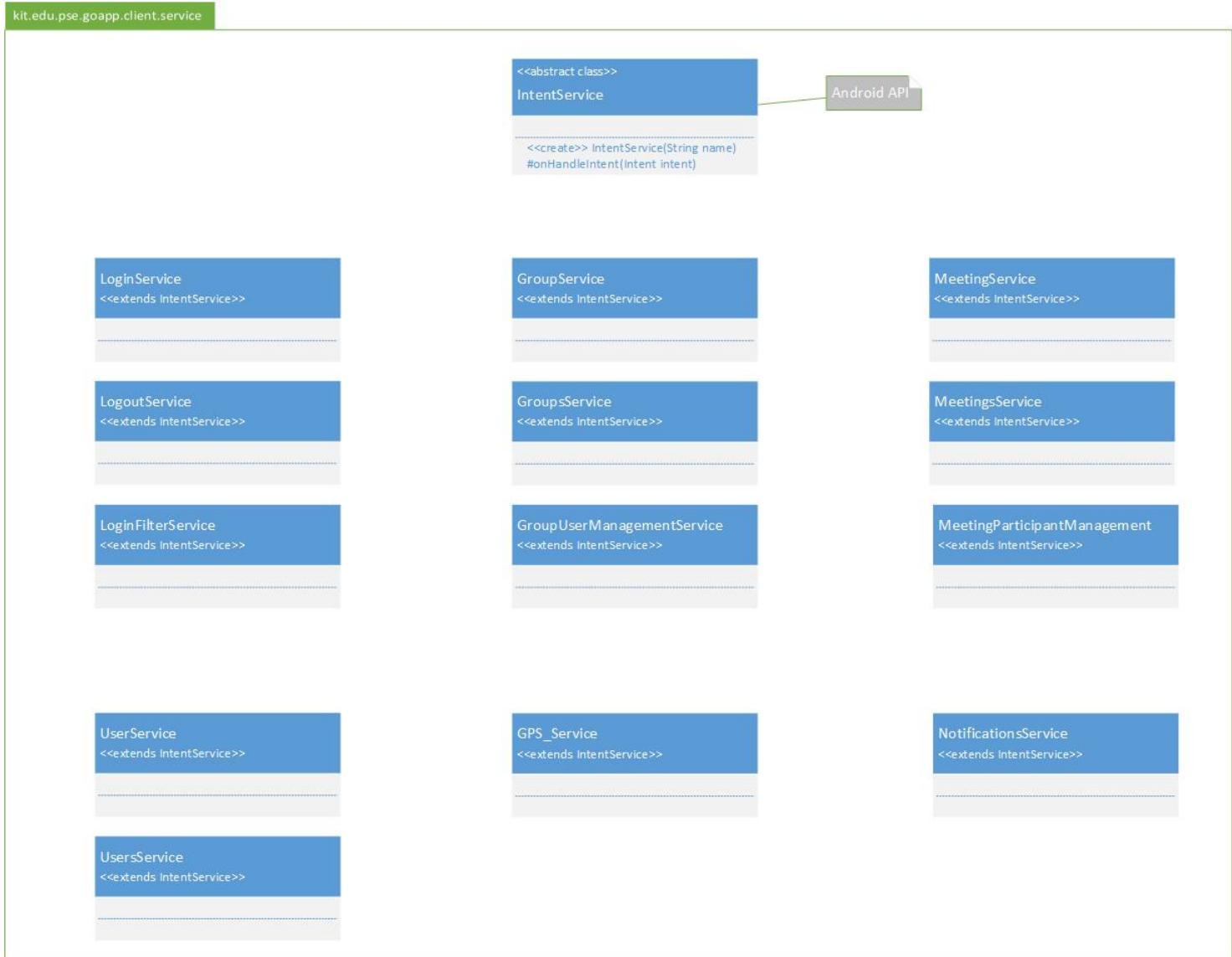
- Suchanfragen nach einem User

**GPS\_Service:**

- Schicken von GPS-Daten

## NotificationsService:

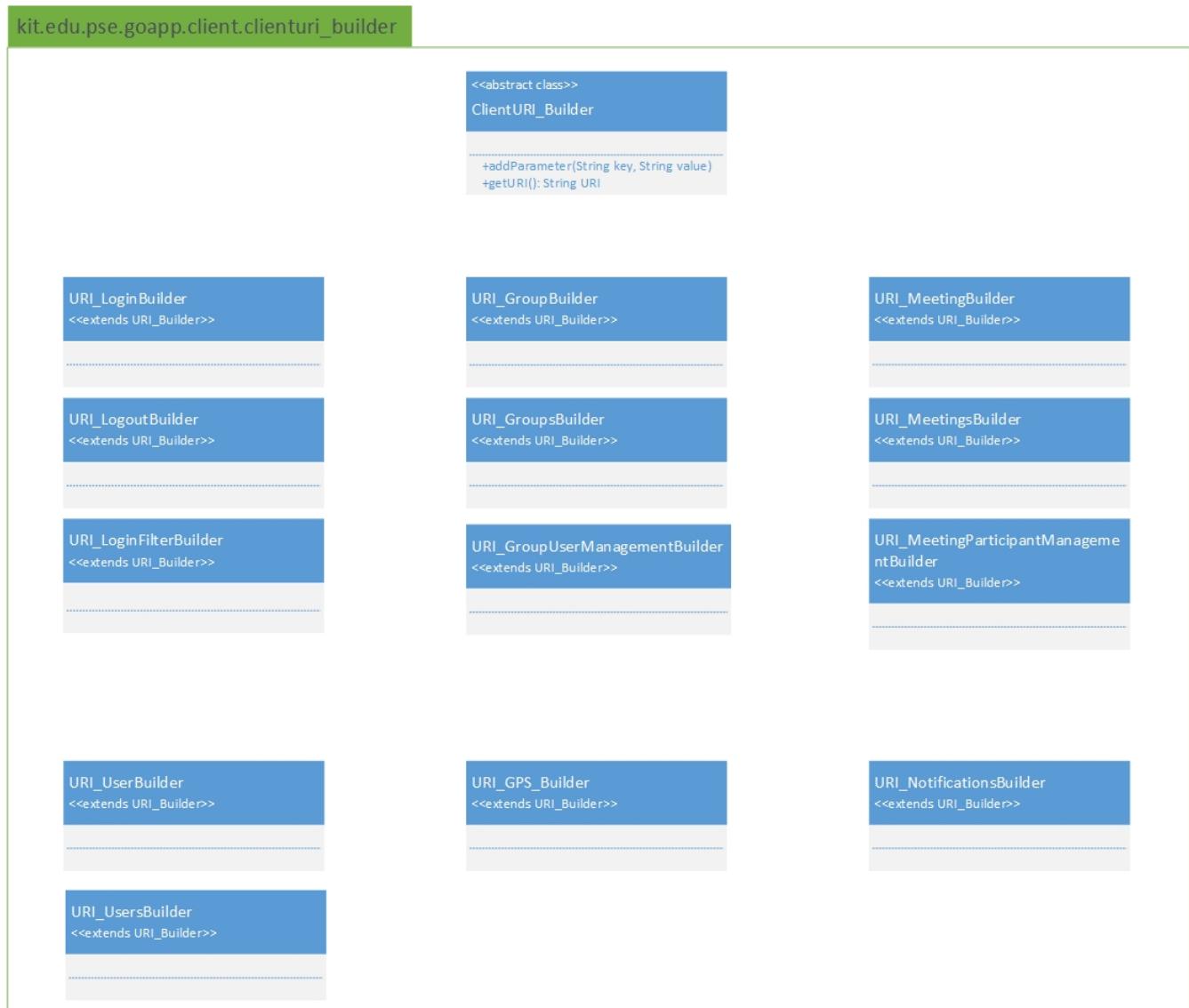
- Ausgeben einer Liste von Benachrichtigungen (Priorität B)
- Benachrichtigungen an-/ausschalten



### 2.1.5 URI\_Builder

kit.edu.pse.goapp.client.clienturi\_builder

Die folgende Abbildung beschreibt die Klassen, welche den Client dabei unterstützen, einen Request an den Server zu senden.

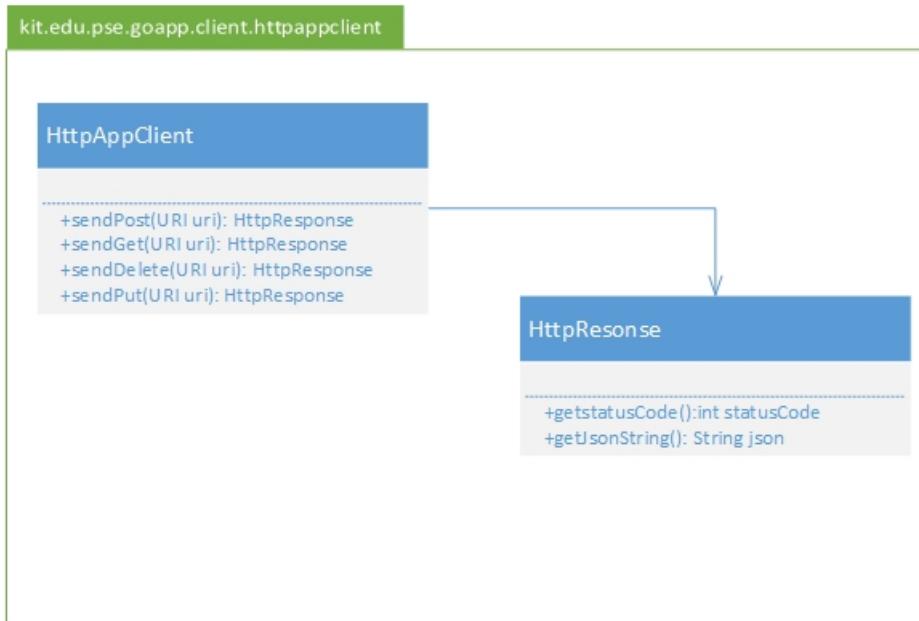


Grundlage hierfür ist die abstrakte Klasse UriBuilder, welche von den darunter sichtbaren Klassen erweitert wird. Die Klassen stellen eine einfache Möglichkeit für andere Klassen dar, um URIs zu erzeugen und zu füllen, welche später einfach zu einem HTTP Request umgewandelt werden. Dabei stellen die Klassen durch private statische Strings jeweils die Adresse für den gewünschten Request bereit. Die abstrakte Klasse beinhaltet die Serveradresse. Die Struktur ähnelt sehr der der Servlets, da für jedes Servlet eine UriBuilder-Klasse existiert. Beispiel: Der Client möchte getAllUsers() aufrufen. Dazu erzeugt er eine Instanz des GroupBuilders und, da dieser Request keine Parameter benötigt, muss die Methode addParameter() nicht aufgerufen werden. Nach dem Erzeugen der Instanz erhält der Client über getUri() die URI, welche er dann an den HttpAppClient weitergibt.

### 2.1.6 HttpAppClient

kit.edu.pse.goapp.client.httpappclient

Der HttpAppClient implementiert die HTTP-Anfragsmethoden: GET, POST, PUT und DELETE. Die HttpResponse kümmert sich darum, die Antwort von dem Server auszupacken. GetStatusCode() liest den Status Code von dem Server. GetJsonString() packt die JSON-Datei aus.

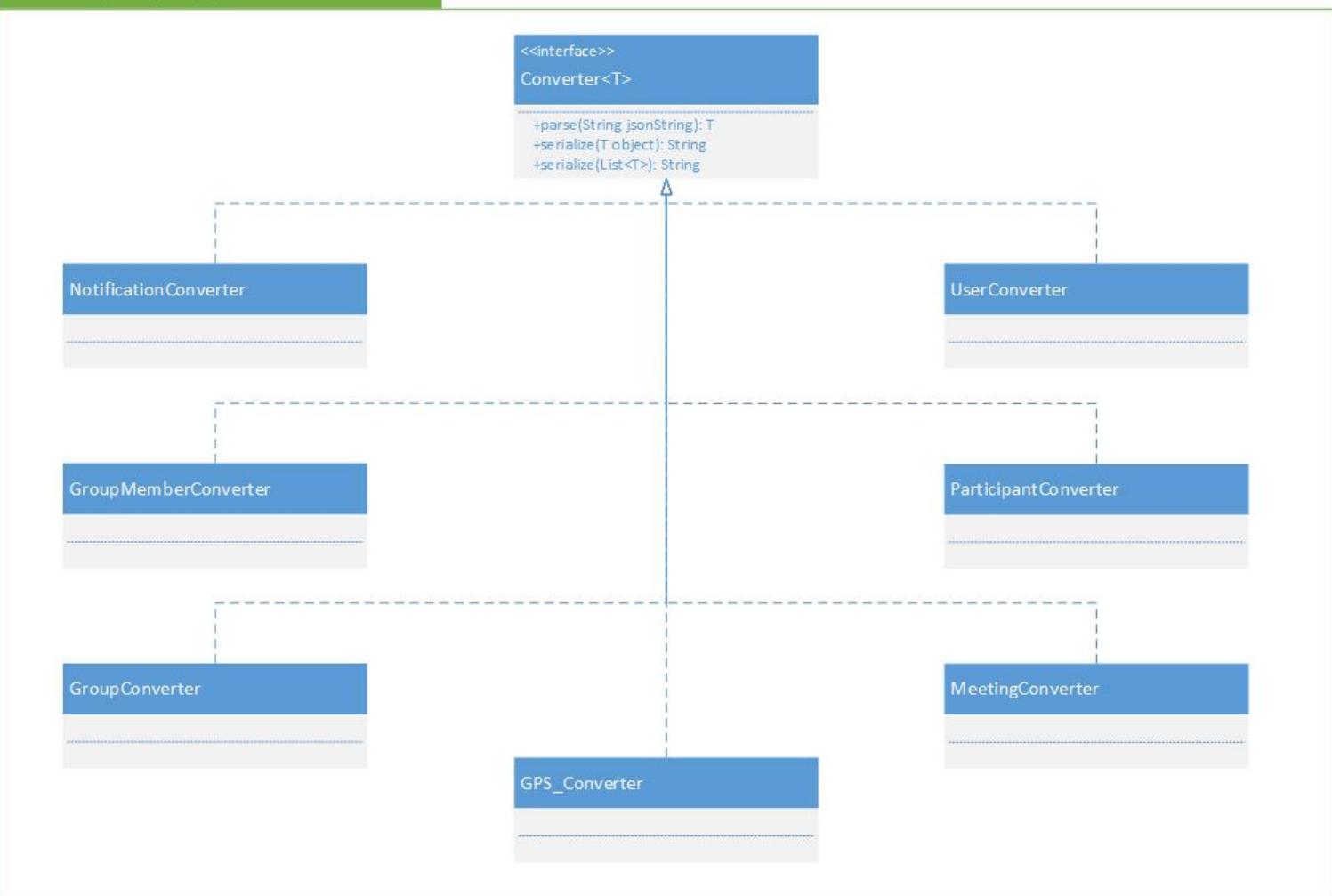


### 2.1.7 Converter

kit.edu.pse.goapp.client.converter

Klassendiagrammbeschreibung siehe Kapitel 2.2.5: kit.edu.pse.goapp.server.converter

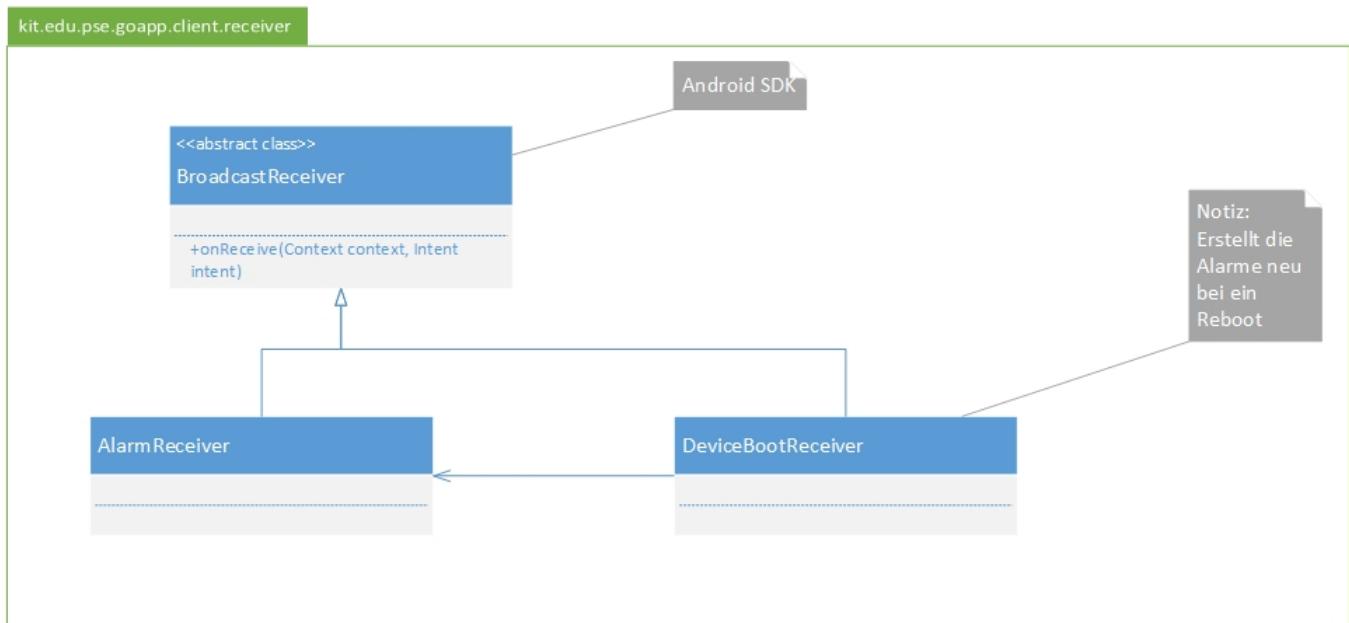
kit.edu.pse.goapp.client.converter



### 2.1.8 Receiver

kit.edu.pse.goapp.client.receiver

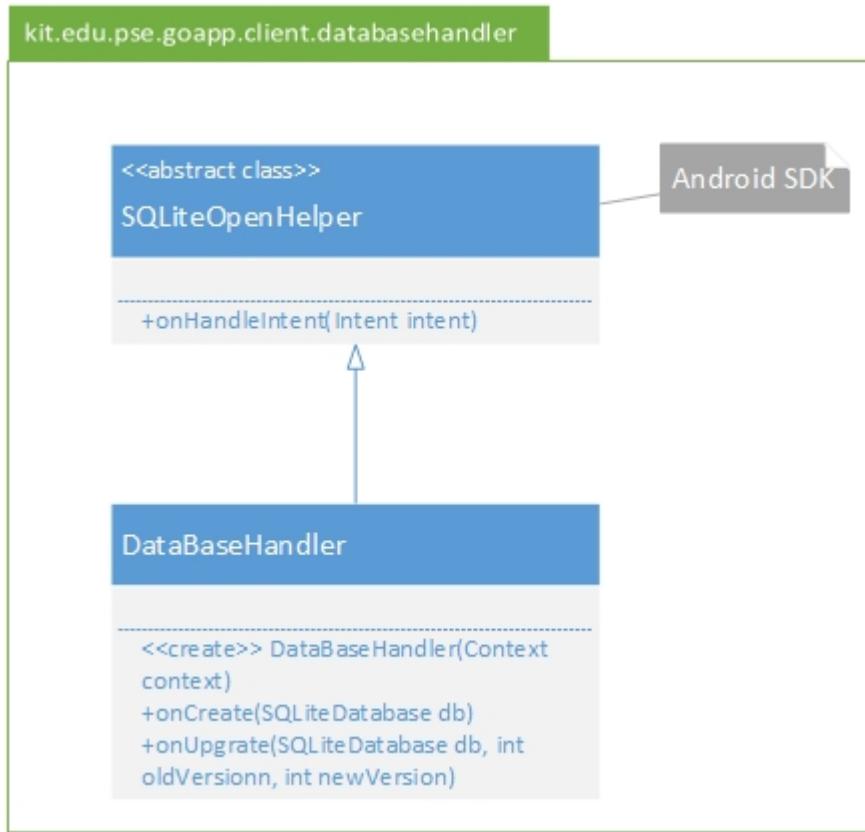
Wenn ein Alarm erstellt wird, kümmern sich die System Alarm Services darum, dass ein Broadcast geschickt wird und zu einem bestimmten Zeitpunkt ausgeführt wird. Wenn der Alarm den Broadcast bekommt, wird die Methode onReceive() aufgerufen. Beim AlarmReceiver wird der GPS\_Service gestartet. Beim DeviceBootReceiver werden alle Alarne noch einmal erstellt, weil sie nach einem Reboot verloren gehen.



### 2.1.9 Database-Handler

```
kit.edu.pse.goapp.client.databasehandler
```

Der DataBaseHandler kümmert sich um die Datenbank. Die onCreate() Methode erstellt eine Datenbank mit zwei Spalten: eine für die MeetingId und eine für die Uhrzeit. Auf diese Weise hat jeder Alarm eine ID und kann leicht ausgeschaltet werden oder beim Neustart des Gerätes wieder erstellt werden. Die onUpgrade() Methode wird benutzt, wenn die Tabelle erweitert werden muss.



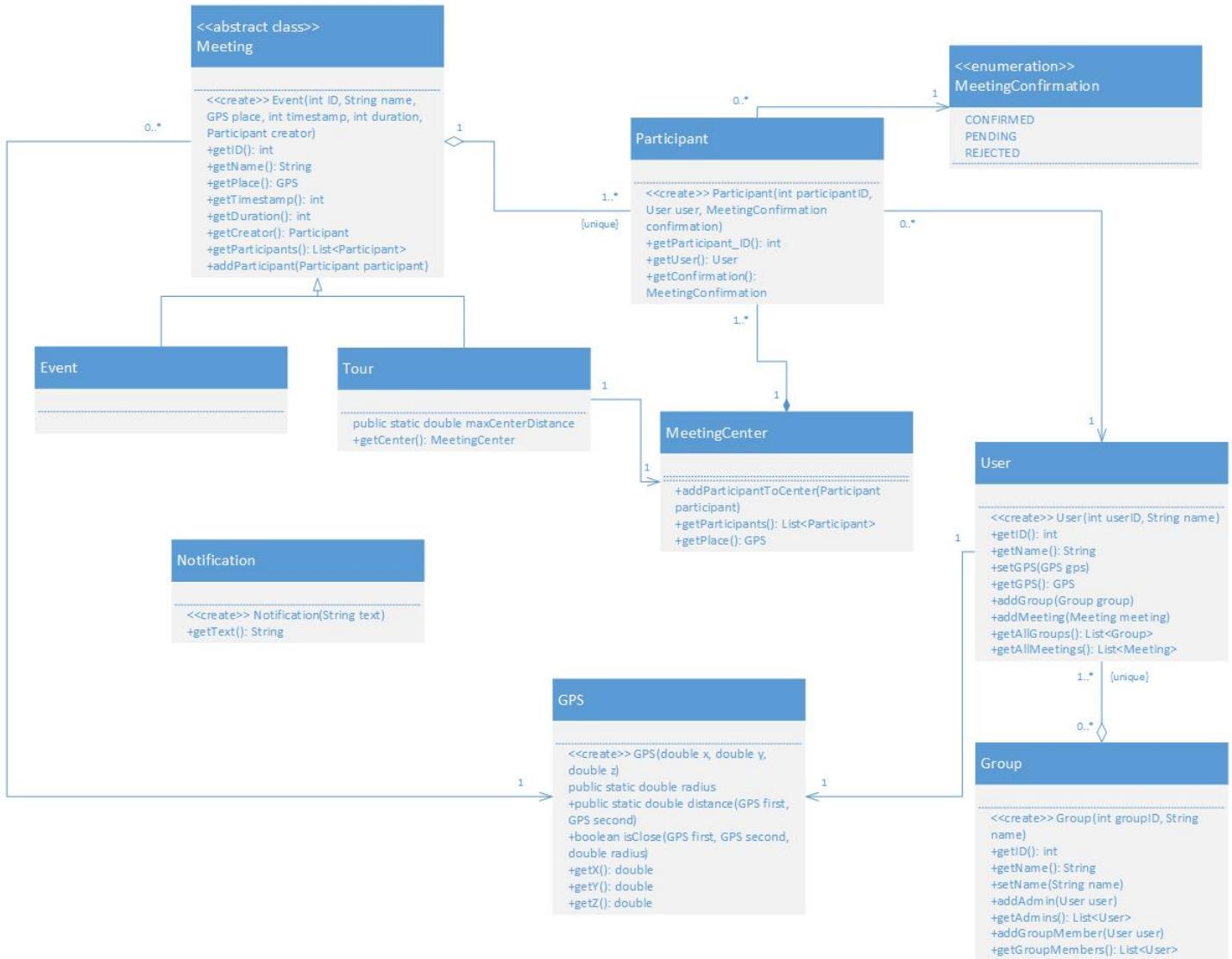
## 2.2 Server

### 2.2.1 Klassendiagramm

`kit.edu.pse.goapp.server.datamodels`

Die folgende Abbildung beschreibt die Klassen und ihre Beziehungen im Datenmodell des Servers. Hier werden die Sachverhalte aus der Realität abgebildet, genauso wie ihre Beziehungen untereinander. Genaue Beschreibungen der Klassen entnehme man dem Kapitel 5.1. Der Server ist prinzipiell auf alle geplanten Funktionalitäten vorbereitet, je nachdem welche Anfragen vom Client implementiert werden, stehen diese auch zur Verfügung.

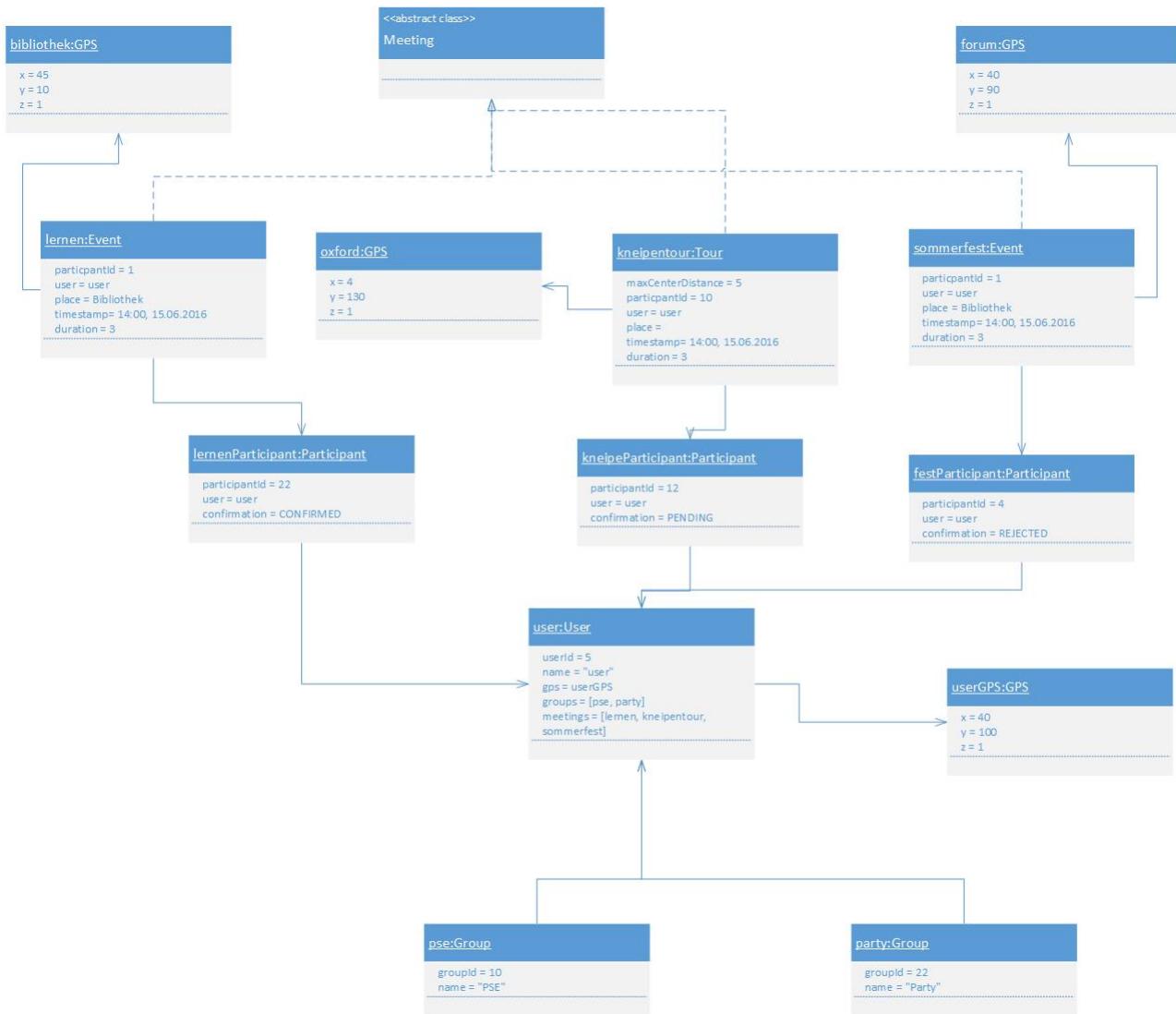
Ein Nutzer der App kann, je nach Umfeld als User oder Participant beschrieben werden. Jeder Nutzer existiert als User im Datenmodell. Sobald er an einem oder mehreren Terminen teilnimmt (im Modell Meeting) ist er dort als Participant zugehörig. Da ein Nutzer an mehreren Terminen teilnehmen kann, können zu jedem User mehrere Participants gehören. Zu einem Participant allerdings nur ein User. Nutzer der App können zu keiner, einer oder mehreren Gruppen (im Modell Group) gehören. Allerdings muss in diesen Gruppen immer mindestens ein Nutzer sein und niemals Nutzer doppelt, das heißt ein User kann zu keiner bis mehreren Groups gehören und zu einer Group einer bis mehrere einzigartige User. Zu jedem User gehört sein Standort (im Modell GPS), dies ist eine Eins-zu-eins-Beziehung. Ein Nutzer kann an keinem bis mehreren Terminen teilnehmen, das heißt er ist Teilnehmer (im Modell Participant) an diesem Termin. Dieser Termin hat auch einen Ort, dass heißt es existiert für jedes Meeting eine Eins-zu-eins-Beziehung zu einem GPS. Zu jedem Participant gehört ein Meeting; zu jedem Meeting gehört mindestens ein Participant oder eben mehrere. Jeder Participant hat einen Teilnahmestatus (im Modell MeetingConfirmation) als Enumeration. Die abstrakte Klasse Meeting kann entweder die Ausprägung Event oder Tour haben. Jede Tour hat einen Ort, an dem sich das Zentrum der Teilnehmer befindet (im Modell MeetingCenter), jedes MeetingCenter gehört zu nur einer Tour. Es gibt auch Benachrichtigung an die User (im Modell Notifiaction).



### **2.2.2 Objektdiagramm**

Im Folgenden wird noch ein Beispiel beschrieben, dass in der folgenden Objektdiagramm zu sehen ist.

Der User user ist Mitglied in zwei Gruppen: pse und party. Er hat auch ein GPS userGPS. Er ist eingeladen zu drei Terminen und ist Participant dieser Termine. lernen ist ein Event und lernenParticipants MeetingConfirmation ist CONFIRMED. Das Event lernen hat auch einen Ort: Das GPS bibliothek. Die beiden anderen Meetings sehen sehr ähnlich aus, nur dass kneipentour eine Tour ist, die allerdings noch nicht angefangen hat und somit noch kein MeetingCenter besitzt, und sich bei den zu user gehörenden Participants jeweils die MeetingConfirmations ändern.



### 2.2.3 Servlets

kit.edu.pse.goapp.server.servlets

Die Servlets nehmen innerhalb des Servers Anfragen vom Client entgegen und beantworten sie. Es gibt in unserem System folgende Arten von Servlets:

Servlets, die etwas mit Gruppenerstellen/-verwalten und -löschen zu tun haben (GroupServlet, GroupsServlet, GroupUserManagementServlet),

Servlets, die etwas mit Treffenerstellen/-verwalten und -löschen zu tun haben (MeetingServlet, MeetingsServlet, MeetingParticipantServlet),

Servlets, die sich um das Registrieren und Login/Logout in der App kümmern (LoginServlet, LogoutServlet, LoginFilterServlet),

Servlets, die sich um die Verwaltung der Benutzer kümmern (UserServlet, UserServlet),

und jeweils ein Servlet für Benachrichtigung und GPS.

Jedes Servlet implementiert zumindest eine der folgenden Methoden aus dem Http-Protokoll:

- doPost(HttpServletRequest request, HttpServletResponse response): Neue Ressourcen werden erstellt, deren URI noch nicht bekannt ist.
- doGet(HttpServletRequest request, HttpServletResponse response): Auf Ressourcen wird lesend zugegriffen. Eine GET-Anfrage darf nicht dazu führen, dass Daten auf dem Server verändert werden.
- doPut(HttpServletRequest request, HttpServletResponse response): Wird verwendet, um Ressourcen zu erstellen oder zu bearbeiten, deren URI bereits bekannt ist.
- doDelete(HttpServletRequest request, HttpServletResponse response): Ressourcen werden gelöscht.

Die Servlets bekommen einen POST-, GET-, PUT- oder DELETE-Request vom Client zugeschickt und verarbeiten diesen in der jeweiligen Methode. Außerdem schicken sie den JSON-String aus dem Request an den zugehörigen Converter, damit dieser daraus das zugehörige DAO generiert. Die Servlets verwalten die DAOs.



#### **2.2.4 DataAccess Objects**

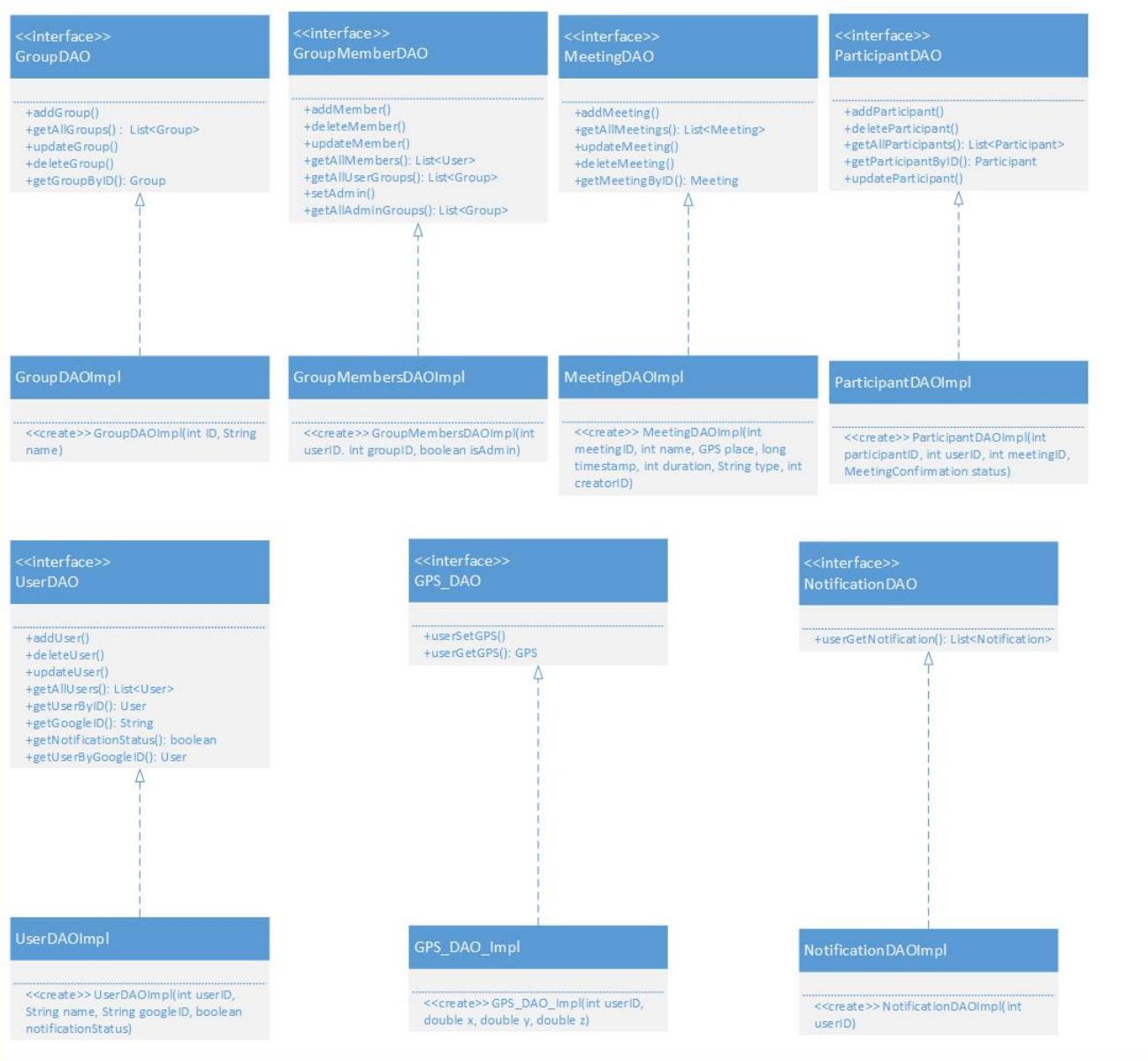
`kit.edu.pse.goapp.server.daos`

Wir verwenden an dieser Stelle das Entwurfsmuster Data Access Object (DAO), das den Zugriff auf die Datenbank so kapselt, dass die Datenbank verändert oder ausgetauscht werden kann, ohne dass der aufrufende Code geändert werden muss.

Für jede Klasse der Datenbank gibt es ein DAO-Interface und eine dazugehörige konkrete Klasse, die das Interface implementiert und dafür zuständig ist, Daten von der Datenbank zu bekommen. Die DAO-Interfaces definieren die Standardoperationen, die auf dem Modelobject ausgeführt werden. Das Modelobject ist einfach eine normale Klasse aus unserem Datenmodell (siehe Kapitel 2.2.1), welche getter- und setter-Methoden beinhaltet, um die Daten zu speichern, die man durch das DAO empfangen hat.

Der jeweilige Converter erstellt die DAOs aus dem JSON-String vom HTTP-Request und das zugehörige Servlet verwaltet dann das DAO. Es greift auf die Datenbank zu, um Daten zu speichern oder zu lesen, und gibt die Daten, die aus der Datenbank kommen, wieder zurück an den Converter, damit er daraus einen JSON-String erstellen kann, um diesen in der HTTP-Response zu verschicken.

## kit.edu.pse.goapp.server.daos



## 2.2.5 Converter

Package kit.edu.pse.goapp.server.converter

Die Converter sind im Server gespeichert. Sie wandeln die Objekte aus der Datenbank in Parameter um, bevor sie an die Servlets weitergeleitet werden. Umgekehrt wandeln sie auch die Parameter aus den Servlets in Objekte um, bevor sie der Datenbank zurückgegeben werden.

Das Interface Converter <T> hat die Methoden public T parse(String jsonString), public String serialize(T object) und public String serialize(List <T> objects).

Die Methode public T parse(String) konvertiert den String, den sie von den Servlets bekommt, in ein Objekt T und gibt es an die Datenbank zurück.

Die Methode public String serialize(T object) konvertiert ein Objekt T, das sie von der Datenbank erhält, in einen String und gibt ihn an die Servlets weiter.

Die Methode public String serialize(List <T> objects) konvertiert eine Liste von Objekten T, die sie von der Datenbank erhält, in einen String und gibt ihn an die Servlets weiter.

Es gibt jeweils einen Converter für die Klassen User, Participant, Meeting, GPS, Group, GroupMember und Notification. Diese Converter implementieren das Interface Converter<T>.

Die Klasse NotificationConverter konvertiert Benachrichtigungen in Strings und umgekehrt. Sie implementiert die Methoden public Notification parse(String), public String serialize(Notification object) und public String serialize(List <Notification> notifications).

Die Klasse GroupMemberConverter konvertiert Gruppenmitglieder in Strings und umgekehrt. Sie implementiert die Methoden public User parse(String), public String serialize(Group group), public String serialize(List <Group> groups) und public String serialize(List <User> users).

Die Klasse GroupConverter konvertiert Gruppen in Strings und umgekehrt. Sie implementiert die Methoden public Group parse(String), public String serialize(Group group) und public String serialize(List <Group> groups).

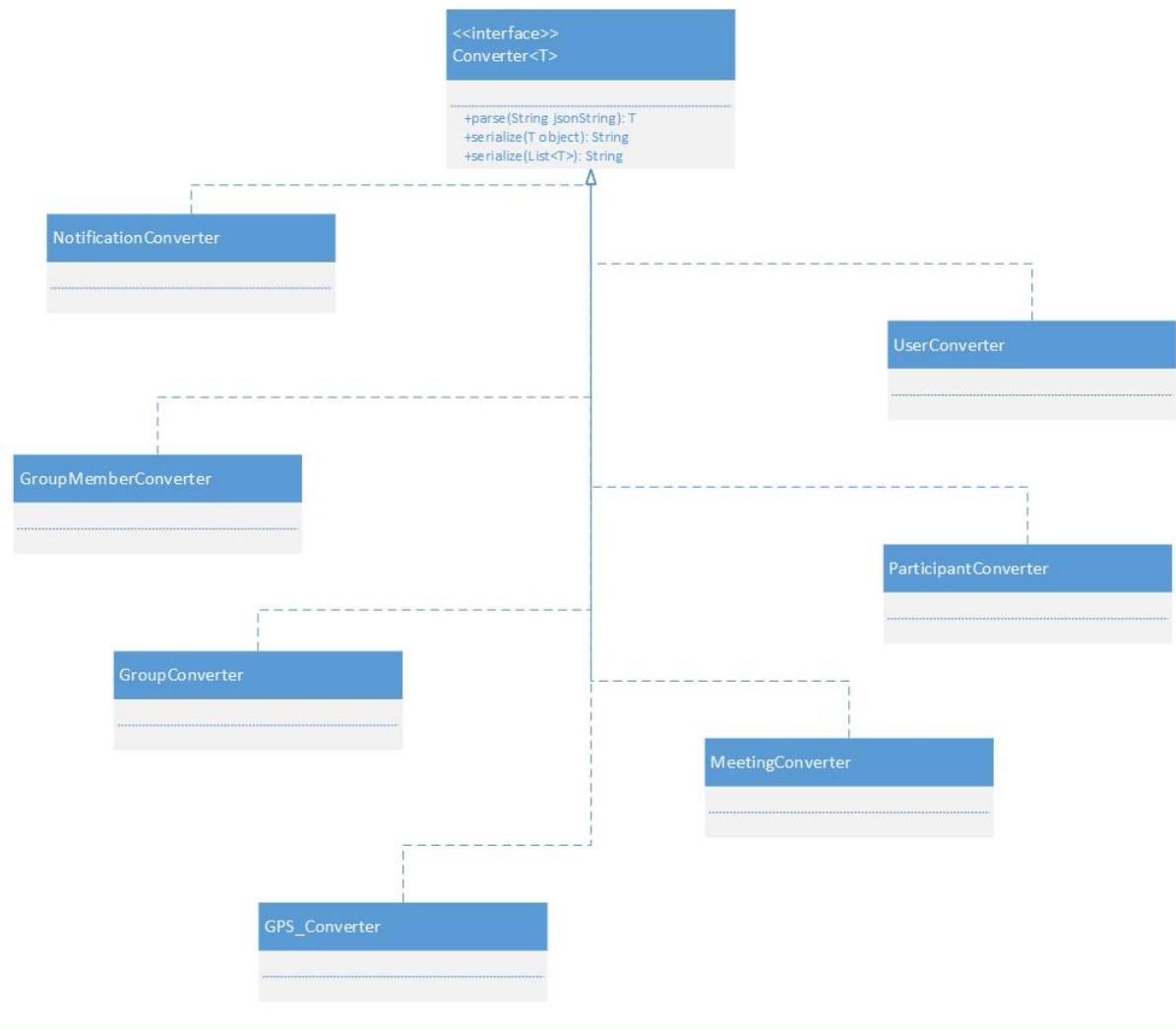
Die Klasse GPSConverter konvertiert GPS-Koordinaten in Strings und umgekehrt. Sie implementiert die Methoden public GPS parse(String), public String serialize(GPS gps) und public String serialize(List <GPS> gps).

Die Klasse MeetingConverter konvertiert Termine in Strings und umgekehrt. Sie implementiert die Methoden public Meeting parse(String), public String serialize(Meeting meeting) und public String serialize(List <Meeting> meetings).

Die Klasse ParticipantConverter konvertiert Teilnehmer eines Termins in Strings und umgekehrt. Sie implementiert die Methoden public Participant parse(String), public String serialize(Participant participant) und public String serialize(List <Participant> participants).

Die Klasse UserConverter konvertiert Benutzer der Go-App in Strings und umgekehrt. Sie implementiert die Methoden public User parse(String), public String serialize(User user) und public String serialize(List <User> users).

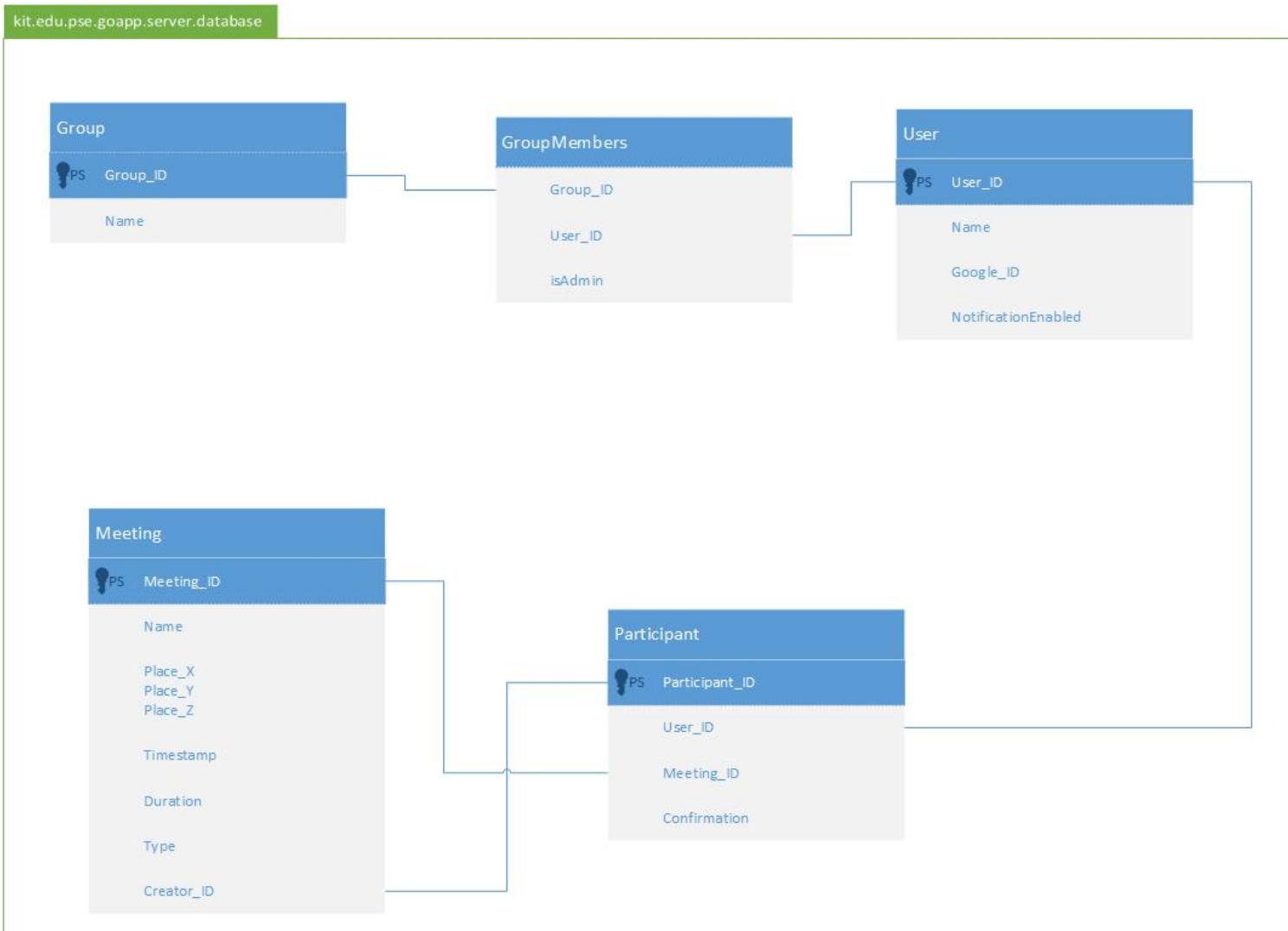
## kit.edu.pse.goapp.server.converter



## 2.2.6 Datenbank

kit.edu.pse.goapp.server.database

Die verschiedenen Elemente und Objekte der Go-App sowie ihre Eigenschaften werden in einer Datenbank gespeichert. Das Diagramm beschreibt die Datenbank und die Beziehungen zwischen den Objekten.



Jeder Benutzer (User) erhält bei der Registrierung mit seiner Google\_ID eine User\_ID, anhand derer er identifiziert werden kann.  
Ein Benutzer hat die Möglichkeit, eine Gruppe (Group) zu erstellen. Diese bekommt ebenfalls eine ID, die Group\_ID. Gruppen haben Gruppenmitglieder

(GroupMembers). Ein Mitglied einer Gruppe wird durch die Group\_ID der Gruppe und durch seine User\_ID beschrieben.

Gruppenmitglieder können für ihre Gruppe einen Termin (ein Meeting) erstellen. Jedem Meeting wird bei seiner Erstellung wiederum eine ID zugewiesen (die Meeting\_ID). Meetings enthalten Teilnehmer (Participants). Teilnehmer eines Termins sind Benutzer, die Mitglied der Gruppe sind, für die der Termin erstellt wurde. Sie werden anhand ihrer User\_ID und der Meeting\_ID des Termins identifiziert und erhalten eine Participant\_ID. Die Participant\_ID des Terminerstellers wird im Meeting als Creator\_ID hinterlegt.

### 3 Client ↔ Server RESTful API

```
package kit.edu.goapp.server.servlets
```

#### 3.1 HTTP-Protokoll

Methoden	Adresse	Beschreibung
GroupServlet		
POST	/group/create	Neue Gruppe erstellen
GET	/group/read	Gruppeninformationen anzeigen
PUT	/group/update	Gruppe bearbeiten, Gruppeneinstellungen ändern
DELETE	/group/delete	Gruppe löschen
GroupUserManagementServlet		
POST	/group/user/create	Benutzer zu einer Gruppe hinzufügen
GET	/group/user/read	Liste der Gruppenmitglieder ausgeben
PUT	/group/user/update	Ein Gruppenmitglied zum Admin (ab)wählen
DELETE	/group/user/delete	Ein Gruppenmitglied aus der Gruppe löschen
GroupsServlet		
GET	/groups/read	Liste der Gruppen anzeigen
LoginServlet		
POST	/user/create	Als neuen Benutzer in der Go-App registrieren
GET	/user/read	Überprüft, ob der Benutzer registriert ist
PUT	/user/update	In die Go-App einloggen
LogoutServlet		
DELETE	/user/delete	Aus der Go-App ausloggen
LoginFilterServlet		
GET	/user/read	Prüft, ob der Benutzer auf die angeforderte Ressource zugreifen darf
NotificationsServlet		
GET	/notifications/read	Liste von Nachrichten ausgeben
PUT	/notifications/update	Nachrichtenanzeigen an-/ausschalten
UserServlet		
POST	/user/create	Neuen Benutzer erstellen
GET	/user/read	Benutzerinformationen anzeigen
PUT	/user/update	Benutzerinformationen ändern
DELETE	/user/delete	Benutzer löschen
UsersServlet		
GET	/users/read	Liste aller Benutzer ausgeben
GPSServlet		
PUT	/user/gps/meeting/update	GPS-Daten updaten
MeetingServlet		
POST	/meeting/create	Termin erstellen
GET	/meeting/read	Termininformationen anzeigen
PUT	/meeting/put	Termin ändern
DELETE	/meeting/delete	Termin löschen
MeetingsServlet		
GET	/meetings/read	Liste mit allen Terminen ausgeben
MeetingParticipantManagementServlet		
POST	/meeting/users/create	38 Liste von Teilnehmern hinzufügen
GET	/meeting/participants/read	Teilnehmer anzeigen
PUT	/meeting/participant/update	Zusage ändern
DELETE	/meeting/participant/delete	Teilnehmer aus dem Termin löschen

## 3.2 Statuscodes

Folgende Statuscodes verwenden wir intern bei der HTTP-Response:

Code	Nachricht	Bedeutung
200	ok	Die Anfrage wurde erfolgreich bearbeitet und das Ergebnis der Anfrage wird in der Antwort übertragen.
400	bad request	Die Anfrage-Nachricht war fehlerhaft aufgebaut.
403	forbidden	Die Anfrage wurde mangels Berechtigung des Clients nicht durchgeführt.
404	not found	Die angeforderte Ressource wurde nicht gefunden.
408	request time out	Innerhalb der vom Server erlaubten Zeitspanne wurde keine vollständige Anfrage des Clients empfangen.
500	internal server error	Dies ist ein "Sammel-Statuscode" für unerwartete Serverfehler.

## 3.3 Class GroupServlet

### 3.3.1 Neue Gruppe erstellen

**POST /group/create**

Mit diesem Aufruf wird eine neue Gruppe erstellt und anhand einer ID auf dem Server gespeichert.

#### Anfrage

```
{  
    "groupID": int  
}
```

Dabei werden folgende Parameter verwendet:

Parameter	Datentyp	Beschreibung
groupID	int	ID der neuen Gruppe

#### Antwort

Bei Erfolg meldet der Server 200, sonst einen Fehler.

#### Fehler

Code	Beschreibung
400	Die Anfrage entspricht nicht dem Anfrageformat
400	Die groupID hat nicht den Datentyp int

### 3.3.2 Gruppeninformationen anzeigen

**GET /group/read**

Mit diesem Aufruf werden die Informationen zu der Gruppe mit der übergebenen groupID angezeigt.

#### Anfrage

```
{  
    "groupID": int  
}
```

Dabei werden folgende Parameter verwendet:

Parameter	Datentyp	Beschreibung
groupID	int	ID der Gruppe

#### Antwort

Bei Erfolg meldet der Server 200, sonst einen Fehler.

#### Fehler

Code	Beschreibung
400	Die Anfrage entspricht nicht dem Anfrageformat
400	Die groupID hat nicht den Datentyp int
404	Es konnte keine Gruppe mit der ID groupID gefunden werden

### 3.3.3 Gruppe bearbeiten

```
PUT /group/update
```

Mit diesem Aufruf wird die Gruppe bearbeitet. Gruppeneinstellungen können geändert werden.

#### Anfrage

```
{  
    "groupID": int  
    "name": String  
    "members": List<User>  
}
```

Dabei werden folgende Parameter verwendet:

Parameter	Datentyp	Beschreibung
groupID	int	ID der Gruppe
name	String	Gruppenname
members	List<User>	Gruppenmitglieder

#### Antwort

Bei Erfolg meldet der Server 200, sonst einen Fehler.

## Fehler

Code	Beschreibung
400	Die Anfrage entspricht nicht dem Anfrageformat
400	Die Parameter haben den falschen Datentyp
403	Die Person ist kein Gruppenadministrator und damit nicht berechtigt, die Gruppe zu ändern
404	Es konnte keine Gruppe mit der ID groupID gefunden werden

### 3.3.4 Gruppe löschen

**DELETE /group/delete**

Mit diesem Aufruf wird die Gruppe gelöscht.

#### Anfrage

```
{  
    "groupID": int  
}
```

Dabei werden folgende Parameter verwendet:

Parameter	Datentyp	Beschreibung
groupID	int	ID der Gruppe

#### Antwort

Bei Erfolg meldet der Server 200, sonst einen Fehler.

## Fehler

Code	Beschreibung
400	Die Anfrage entspricht nicht dem Anfrageformat
400	Die groupID hat nicht den Datentyp int
403	Die Person ist kein Gruppenadministrator und damit nicht berechtigt, die Gruppe zu löschen
404	Es konnte keine Gruppe mit der ID groupID gefunden werden

## 3.4 Class GroupUserManagementServlet

### 3.4.1 Benutzer zu einer Gruppe hinzufügen

**POST /group/user/create**

Mit diesem Aufruf wird zu der Gruppe mit der übergebenen groupID der Benutzer mit der userID hinzugefügt und bestimmt, ob er Gruppenadministrator werden soll.

### Anfrage

```
{  
    "groupID": int  
    "userID": int  
    "adminStatus": boolean  
}
```

Dabei werden folgende Parameter verwendet:

Parameter	Datentyp	Beschreibung
groupID	int	ID der Gruppe
userID	int	ID des Benutzers
adminStatus	boolean	true, falls das neue Gruppenmitglied Administrator sein soll, sonst false

### Antwort

Bei Erfolg meldet der Server 200, sonst einen Fehler.

### Fehler

Code	Beschreibung
400	Die Anfrage entspricht nicht dem Anfrageformat
400	Ein Parameter hat einen falschen Datentyp
404	Es konnte keine Gruppe mit der ID groupID gefunden werden
404	Es konnte kein Benutzer mit der ID userID gefunden werden

### 3.4.2 Liste der Gruppenmitglieder ausgeben

```
GET /group/user/read
```

Mit diesem Aufruf wird eine Liste mit allen Gruppenmitgliedern ausgegeben.

### Anfrage

```
{  
    "groupID": int  
}
```

Dabei werden folgende Parameter verwendet:

Parameter	Datentyp	Beschreibung
groupID	int	ID der Gruppe

### Antwort

Bei Erfolg meldet der Server 200, sonst einen Fehler.

### Fehler

Code	Beschreibung
400	Die Anfrage entspricht nicht dem Anfrageformat
400	Die groupID hat nicht den Datentyp int
404	Es konnte keine Gruppe mit der ID groupID gefunden werden

### 3.4.3 Gruppenmitglied zum Admin (ab)wählen

**PUT /group/user/update**

Mit diesem Aufruf wird ein Gruppenmitglied als Administrator gewählt oder abgewählt.

#### Anfrage

```
{  
    "groupID": int  
    "userID": int  
    "adminStatus": boolean  
}
```

Dabei werden folgende Parameter verwendet:

Parameter	Datentyp	Beschreibung
groupID	int	ID der Gruppe
userID	int	Benutzer-ID eines Gruppenmitgliedes
adminStatus	boolean	true falls das Gruppenmitglied zum Administrator gewählt wird, false wenn das Gruppenmitglied als Administrator abgewählt wird

#### Antwort

Bei Erfolg meldet der Server 200, sonst einen Fehler.

#### Fehler

Code	Beschreibung
400	Die Anfrage entspricht nicht dem Anfrageformat
400	Die Parameter haben den falschen Datentyp
403	Der Benutzer mit der ID userID ist kein Gruppenmitglied und kann deshalb nicht als Administrator gewählt werden
404	Es konnte keine Gruppe mit der ID groupID gefunden werden
404	Es konnte kein Benutzer mit der ID userID gefunden werden

### 3.4.4 Gruppenmitglied löschen

**DELETE /group/user/delete**

Mit diesem Aufruf wird ein Gruppenmitglied aus der entsprechenden Gruppe gelöscht.

### Anfrage

```
{  
    "groupID": int  
    "userID": int  
}
```

Dabei werden folgende Parameter verwendet:

Parameter	Datentyp	Beschreibung
groupID	int	ID der Gruppe
userID	int	Benutzer-ID des Gruppenmitgliedes

### Antwort

Bei Erfolg meldet der Server 200, sonst einen Fehler.

### Fehler

Code	Beschreibung
400	Die Anfrage entspricht nicht dem Anfrageformat
400	Die Parameter haben den falschen Datentyp
404	Es konnte kein Benutzer mit der ID userID gefunden werden
404	Es konnte keine Gruppe mit der ID groupID gefunden werden

## 3.5 Class GroupsServlet

### 3.5.1 Liste der Gruppen anzeigen

```
GET /groups/read
```

Mit diesem Aufruf wird eine Liste aller Gruppen angezeigt.

### Anfrage

```
{  
}
```

Dabei werden keine Parameter verwendet.

### Antwort

Bei Erfolg meldet der Server 200, sonst einen Fehler.

### Fehler

Code	Beschreibung
500	Unerwarteter Serverfehler

## 3.6 Class LoginServlet

### 3.6.1 Neuen Benutzer in der Go-App registrieren

POST /user/create

Mit diesem Aufruf wird ein neuer Benutzer in der Go-App registriert.

#### Anfrage

```
{  
    "accessToken": String  
    "name": String  
}
```

Dabei werden folgende Parameter verwendet:

Parameter	Datentyp	Beschreibung
accessToken	String	Token, das der Benutzer bei der Registrierung erhalten hat
name	String	Benutzername

#### Antwort

Bei Erfolg meldet der Server 200, sonst einen Fehler.

#### Fehler

Code	Beschreibung
400	Die Anfrage entspricht nicht dem Anfrageformat
400	Die Parameter haben den falschen Datentyp

### 3.6.2 Überprüft, ob der Benutzer registriert ist

GET /user/read

Mit diesem Aufruf wird überprüft, ob der Benutzer registriert ist.

#### Anfrage

```
{  
    "name": String  
}
```

Dabei werden folgende Parameter verwendet:

Parameter	Datentyp	Beschreibung
name	String	Benutzername

#### Antwort

Bei Erfolg meldet der Server 200, sonst einen Fehler.

## Fehler

Code	Beschreibung
400	Die Anfrage entspricht nicht dem Anfrageformat
400	Der Parameter <code>name</code> hat den falschen Datentyp
404	Es konnte kein Benutzer mit dem Namen <code>name</code> gefunden werden

### 3.6.3 In die Go-App einloggen

**PUT** /user/update

Mit diesem Aufruf wird ein Benutzer in der Go-App eingeloggt.

#### Anfrage

```
{  
    "accessToken": String  
}
```

Dabei werden folgende Parameter verwendet:

Parameter	Datentyp	Beschreibung
accessToken	String	Token, das der Benutzer bei der Registrierung erhalten hat

#### Antwort

Bei Erfolg meldet der Server 200, sonst einen Fehler.

## Fehler

Code	Beschreibung
400	Die Anfrage entspricht nicht dem Anfrageformat
400	Der Parameter <code>accessToken</code> hat den falschen Datentyp
401	Der Benutzer hat keinen gültigen <code>accessToken</code>

## 3.7 Class LogoutServlet

### 3.7.1 Aus der Go-App ausloggen

**DELETE** /user/delete

Mit diesem Aufruf wird ein Benutzer aus der Go-App ausgeloggt.

#### Anfrage

```
{  
}
```

Dabei werden keine Parameter verwendet.

#### Antwort

Bei Erfolg meldet der Server 200, sonst einen Fehler.

## Fehler

Code	Beschreibung
500	Unerwarteter Serverfehler

## 3.8 Class LoginFilterServlet

### 3.8.1 Prüft, ob der Benutzer auf die angeforderte Ressource zugreifen darf

GET /user/read

Mit diesem Aufruf wird geprüft, ob der Benutzer auf die angeforderte Ressource zugreifen darf.

#### Anfrage

```
{  
}
```

Dabei werden keine Parameter verwendet.

#### Antwort

Bei Erfolg meldet der Server 200, sonst einen Fehler.

## Fehler

Code	Beschreibung
500	Unerwarteter Serverfehler

## 3.9 Class NotificationsServlet

### 3.9.1 Liste von Nachrichten ausgeben

GET /notifications/read

Mit diesem Aufruf wird eine Liste der Benachrichtigungen ausgegeben.

#### Anfrage

```
{  
}
```

Dabei werden keine Parameter verwendet.

#### Antwort

Bei Erfolg meldet der Server 200, sonst einen Fehler.

## Fehler

Code	Beschreibung
500	Unerwarteter Serverfehler

### 3.9.2 Nachrichtenanzeigen an-/ausschalten

PUT /notifications/update

Mit diesem Aufruf wird die Funktion der Nachrichtenanzeigen ein- oder ausgeschaltet.

#### Anfrage

```
{  
    "notificationStatus": boolean  
}
```

Dabei werden folgende Parameter verwendet:

Parameter	Datentyp	Beschreibung
notificationStatus	boolean	Gibt an, ob Benachrichtigungen angezeigt werden

#### Antwort

Bei Erfolg meldet der Server 200, sonst einen Fehler.

#### Fehler

Code	Beschreibung
400	Die Anfrage entspricht nicht dem Anfrageformat
400	Der Parameter notificationStatus hat den falschen Datentyp

## 3.10 Class UserServlet

### 3.10.1 Neuen Benutzer erstellen

POST /user/create

Mit diesem Aufruf wird ein neuer Benutzer erstellt.

#### Anfrage

```
{  
    "googleID": int  
    "name": String  
}
```

Dabei werden folgende Parameter verwendet:

Parameter	Datentyp	Beschreibung
googleID	int	Google-ID des Benutzers
name	String	Benutzername

#### Antwort

Bei Erfolg meldet der Server 200, sonst einen Fehler.

## Fehler

Code	Beschreibung
400	Die Anfrage entspricht nicht dem Anfrageformat
400	Die Parameter haben den falschen Datentyp
404	Die googleID existiert nicht

### 3.10.2 Benutzerinformationen anzeigen

GET /user/read

Mit diesem Aufruf werden die Informationen zu einem bestimmten Benutzer angezeigt.

#### Anfrage

```
{  
    "userID": int  
}
```

Dabei werden folgende Parameter verwendet:

Parameter	Datentyp	Beschreibung
userID	int	ID des Benutzers

#### Antwort

Bei Erfolg meldet der Server 200, sonst einen Fehler.

## Fehler

Code	Beschreibung
400	Die Anfrage entspricht nicht dem Anfrageformat
400	Der Parameter userID hat den falschen Datentyp
404	Es konnte kein Benutzer mit der ID userID gefunden werden

### 3.10.3 Benutzerinformationen ändern

PUT /user/update

Mit diesem Aufruf werden die Benutzerinformationen des Benutzers mit der ID userID geändert.

#### Anfrage

```
{  
    "userID": int  
    "name": String  
}
```

Dabei werden folgende Parameter verwendet:

Parameter	Datentyp	Beschreibung
userID	int	ID des Benutzers
name	String	Benutzername

## Antwort

Bei Erfolg meldet der Server 200, sonst einen Fehler.

## Fehler

Code	Beschreibung
400	Die Anfrage entspricht nicht dem Anfrageformat
400	Die Parameter haben den falschen Datentyp
404	Es konnte kein Benutzer mit der ID <code>userID</code> gefunden werden

### 3.10.4 Benutzer löschen

**DELETE /user/delete**

Mit diesem Aufruf wird ein Benutzer gelöscht.

## Anfrage

```
{  
    "userID": int  
}
```

Dabei werden folgende Parameter verwendet:

Parameter	Datentyp	Beschreibung
<code>userID</code>	int	ID des Benutzers

## Antwort

Bei Erfolg meldet der Server 200, sonst einen Fehler.

## Fehler

Code	Beschreibung
400	Die Anfrage entspricht nicht dem Anfrageformat
400	Der Parameter <code>userID</code> hat den falschen Datentyp
404	Es konnte kein Benutzer mit der ID <code>userID</code> gefunden werden

## 3.11 UsersServlet

### 3.11.1 Liste aller Benutzer ausgeben

**GET /users/read**

Mit diesem Aufruf wird eine Liste aller Benutzer ausgegeben.

## Anfrage

```
{  
    "name":String  
}
```

Dabei werden folgende Parameter verwendet:

Parameter	Datentyp	Beschreibung
<code>name</code>	String	Name des jeweiligen Benutzers

## **Antwort**

Bei Erfolg meldet der Server 200, sonst einen Fehler.

## **Fehler**

<b>Code</b>	<b>Beschreibung</b>
500	Unerwarteter Serverfehler

## **3.12 Class GPSServlet**

### **3.12.1 GPS-Daten updaten**

**PUT /user/gps/meeting/update**

Mit diesem Aufruf werden die GPS-Daten eines Benutzers aktualisiert.

## **Anfrage**

```
{  
    "x": int  
    "y": int  
    "z": int  
}
```

Dabei werden folgende Parameter verwendet:

<b>Parameter</b>	<b>Datentyp</b>	<b>Beschreibung</b>
x	int	GPS x-Koordinate
y	int	GPS y-Koordinate
z	int	GPS y-Koordinate

## **Antwort**

Bei Erfolg meldet der Server 200, sonst einen Fehler.

## **Fehler**

<b>Code</b>	<b>Beschreibung</b>
400	Die Anfrage entspricht nicht dem Anfrageformat
401	Der Benutzer hat GPS in seinen Android-Einstellungen deaktiviert

## **3.13 Class MeetingServlet**

### **3.13.1 Termin erstellen**

**POST /meeting/create**

Mit diesem Aufruf wird ein neuer Termin für eine Gruppe und deren Mitglieder erstellt.

## Anfrage

```
{  
    "name": String  
    "groupID": int  
    "x": int  
    "y": int  
    "z": int  
    "time": int  
    "date": int  
    "type": String  
    "duration": int  
    "note": String  
}
```

Dabei werden folgende Parameter verwendet:

Parameter	Datentyp	Beschreibung
name	String	Name des Termins
groupID	int	ID der Gruppe
x	int	GPS x-Koordinate
y	int	GPS y-Koordinate
z	int	GPS z-Koordinate
time	int	Veranstaltungsbeginn
date	int	Veranstaltungsdatum
type	String	Art des Termins: Tour oder Veranstaltung
duration	int	Dauer der Veranstaltung
note	String	Notiz zur Veranstaltung

## Antwort

Bei Erfolg meldet der Server 200, sonst einen Fehler.

## Fehler

Code	Beschreibung
400	Die Anfrage entspricht nicht dem Anfrageformat
400	Die Parameter haben den falschen Datentyp
404	Es konnte keine Gruppe mit der ID groupID gefunden werden

### 3.13.2 Termininfos

```
GET /meeting/read
```

Mit diesem Aufruf werden alle Infos zum Termin ausgegeben.

## Anfrage

```
{  
    "name": String  
    "groupID": int  
    "x": int  
    "y": int  
    "z": int  
    "time": int  
    "date": int  
    "type": String  
    "duration": int  
    "note": String  
}
```

Dabei werden folgende Parameter verwendet:

Parameter	Datentyp	Beschreibung
name	String	Name des Termins
groupID	int	ID der Gruppe
x	int	GPS x-Koordinate
y	int	GPS y-Koordinate
z	int	GPS z-Koordinate
time	int	Veranstaltungsbeginn
date	int	Veranstaltungsdatum
type	String	Art des Termins: Tour oder Veranstaltung
duration	int	Dauer der Veranstaltung
note	String	Notiz zur Veranstaltung

## Antwort

Bei Erfolg meldet der Server 200, sonst einen Fehler.

## Fehler

Code	Beschreibung
400	Die Anfrage entspricht nicht dem Anfrageformat
400	Die Parameter haben den falschen Datentyp
404	Es konnte keine Gruppe mit der ID groupID gefunden werden

### 3.13.3 Termin ändern

```
PUT /meeting/update
```

Mit diesem Aufruf werden die Einstellungen des Termins geändert.

## Anfrage

```
{  
    "name": String  
    "groupID": int  
    "x": int  
    "y": int  
    "z": int  
    "time": int  
    "date": int  
    "type": String  
    "duration": int  
    "note": String  
}
```

Dabei werden folgende Parameter verwendet:

Parameter	Datentyp	Beschreibung
name	String	Name des Termins
groupID	int	ID der Gruppe
x	int	GPS x-Koordinate
y	int	GPS y-Koordinate
z	int	GPS z-Koordinate
time	int	Veranstaltungsbeginn
date	int	Veranstaltungsdatum
type	String	Art des Termins: Tour oder Veranstaltung
duration	int	Dauer der Veranstaltung
note	String	Notiz zur Veranstaltung

## Antwort

Bei Erfolg meldet der Server 200, sonst einen Fehler.

## Fehler

Code	Beschreibung
400	Die Anfrage entspricht nicht dem Anfrageformat
400	Die Parameter haben den falschen Datentyp
403	Der Benutzer ist nicht Ersteller des Termins und ist daher nicht berechtigt, den Termin zu bearbeiten
404	Es konnte keine Gruppe mit der ID groupID gefunden werden

### 3.13.4 Termin löschen

```
DELETE /meeting/delete
```

Mit diesem Aufruf wird der Termin gelöscht.

## Anfrage

```
{  
    "meetingID": int  
}
```

Dabei werden folgende Parameter verwendet:

Parameter	Datentyp	Beschreibung
meetingID	int	ID des Termins

## Antwort

Bei Erfolg meldet der Server 200, sonst einen Fehler.

## Fehler

Code	Beschreibung
400	Die Anfrage entspricht nicht dem Anfrageformat
400	Der Parameter <code>terminID</code> hat den falschen Datentyp
403	Der Benutzer ist nicht Ersteller des Termins und ist daher nicht berechtigt, den Termin zu löschen
404	Es konnte kein Termin mit der ID <code>meetingID</code> gefunden werden

## 3.14 Class MeetingsServlet

### 3.14.1 Liste mit allen Terminen ausgeben

GET /meetings/read

Mit diesem Aufruf wird eine Liste aller Termine ausgegeben.

## Anfrage

```
{  
    "meetingID": int  
}
```

Dabei werden folgende Parameter verwendet:

Parameter	Datentyp	Beschreibung
meetingID	int	ID des Termins

## Antwort

Bei Erfolg meldet der Server 200, sonst einen Fehler.

## Fehler

Code	Beschreibung
400	Die Anfrage entspricht nicht dem Anfrageformat
400	Der Parameter <code>terminID</code> hat den falschen Datentyp
404	Es konnte kein Termin mit der ID <code>meetingID</code> gefunden werden

## 3.15 Class MeetingParticipantManagementServlet

### 3.15.1 Liste von Teilnehmern hinzufügen

POST /meeting/users/create

Mit diesem Aufruf wird dem Termin eine Liste von Teilnehmern hinzugefügt.

#### Anfrage

```
{  
    "meetingID": int  
    "usersID": List<User>  
}
```

Dabei werden folgende Parameter verwendet:

Parameter	Datentyp	Beschreibung
meetingID	int	ID des Termins
usersID	List<User>	Liste mit Benutzern

#### Antwort

Bei Erfolg meldet der Server 200, sonst einen Fehler.

#### Fehler

Code	Beschreibung
400	Die Anfrage entspricht nicht dem Anfrageformat
400	Ein Parameter hat den falschen Datentyp
403	Der Benutzer ist nicht Ersteller des Termins und ist daher nicht berechtigt, Teilnehmer hinzuzufügen
404	Es konnte kein Termin mit der ID meetingID gefunden werden
404	Es konnte kein Benutzer mit einer bestimmten ID gefunden werden

### 3.15.2 Teilnehmer anzeigen

GET /meeting/participants/read

Mit diesem Aufruf werden die Teilnehmer eines Termins angezeigt.

#### Anfrage

```
{  
    "meetingID": int  
    "participants": List<User>  
}
```

Dabei werden folgende Parameter verwendet:

Parameter	Datentyp	Beschreibung
meetingID	int	ID des Termins
participants	List<User>	Liste der Teilnehmer

## Antwort

Bei Erfolg meldet der Server 200, sonst einen Fehler.

## Fehler

Code	Beschreibung
400	Die Anfrage entspricht nicht dem Anfrageformat
400	Ein Parameter hat den falschen Datentyp
404	Es konnte kein Termin mit der ID meetingID gefunden werden

### 3.15.3 Zusage ändern

**PUT /meeting/participant/put**

Mit diesem Aufruf wird ein Termin zu- oder abgesagt.

#### Anfrage

```
{  
    "meetingID": int  
    "meetingStatus": boolean  
}
```

Dabei werden folgende Parameter verwendet:

Parameter	Datentyp	Beschreibung
meetingID	int	ID des Termins
meetingStatus	boolean	Termin zu- oder abgesagt

## Antwort

Bei Erfolg meldet der Server 200, sonst einen Fehler.

## Fehler

Code	Beschreibung
400	Die Anfrage entspricht nicht dem Anfrageformat
400	Ein Parameter hat den falschen Datentyp
404	Es konnte kein Termin mit der ID meetingID gefunden werden

### 3.15.4 Teilnehmer aus dem Termin löschen

**DELETE /meeting/participant/delete**

Mit diesem Aufruf wird ein Teilnehmer aus dem Termin gelöscht.

#### Anfrage

```
{  
    "meetingID": int  
    "participantID": int  
}
```

Dabei werden folgende Parameter verwendet:

Parameter	Datentyp	Beschreibung
meetingID	int	ID des Termins
participantID	int	ID des Teilnehmers

## Antwort

Bei Erfolg meldet der Server 200, sonst einen Fehler.

## Fehler

Code	Beschreibung
400	Die Anfrage entspricht nicht dem Anfrageformat
400	Ein Parameter hat den falschen Datentyp
403	Der Benutzer ist nicht Ersteller des Termins und ist daher nicht berechtigt, einen Teilnehmer aus dem Termin zu löschen
404	Es konnte kein Termin mit der ID <code>meetingID</code> gefunden werden
404	Es konnte kein Teilnehmer mit der ID <code>participantID</code> gefunden werden

## 4 Klassen des Clients

### 4.1 Package kit.edu.pse.goapp.client.datamodels

Klassenbeschreibung siehe 5.1 Package kit.edu.pse.goapp.server.datamodels

### 4.2 Package kit.edu.pse.goapp.client.activity

#### 4.2.1 public class AppCompatActivity

##### Beschreibung

Basisklasse für Activities, die die Support Library Action Bar Features nutzen.

##### Methoden

- `public void onCreate(Bundle savedInstanceState)`

Hier wird das Layout der Activity erstellt.

##### Parameter

`Bundle savedInstanceState` Die zuvor gespeicherte Instanz der Activity, die wieder hergestellt wird, sonst NULL

#### 4.2.2 public interface View.OnClickListener

##### Beschreibung

Schnittstelle, wenn sich auf die Sicht (View) geklickt wird.

##### Methoden

- `public void onClick(View v)`

Aufruf bei einem Klick auf ein Element

##### Parameter

`View v` Das angeklickte Element im View-Objekt

#### 4.2.3 Interface PopupMenu.OnMenuItemClickListener

##### Beschreibung

Schnittstelle für den Menü-Button.

##### Methoden

- `public boolean onMenuItemClick(MenuItem item)`

Wählt aus, welche Interaktionen ausgeführt werden sollen, beim Klick auf das Menü.

##### Parameter

`MenuItem item` Das Menüelement, welches angeklickt wurde, im MenuItem-Objekt .

**Rückgabewert:**

True, wenn die Interaktion abgearbeitet wurde, andernfalls false.

```
4.2.4 public class GroupsActivity extends AppCompatActivity implements View.OnClickListener implements PopupMenu.OnMenuItemClickListener MyResultReceiver.Receiver
```

**Beschreibung**

Diese Klasse zeigt eine Liste von Gruppen an und nimmt die Interaktion entgegen, wenn man eine Gruppe löschen möchte.

**Methoden**

- `public void start(Activity activity)`

Startet diese Activity

**Parameter**

`Activity activity` Die Activity, die diese aufruft (wichtig z.B.: wenn man die Zurücktaste drückt.)

```
4.2.5 public class GroupMemberActivity extends AppCompatActivity implements View.OnClickListener implements PopupMenu.OnMenuItemClickListener MyResultReceiver.Receiver
```

**Beschreibung**

Diese Klasse zeigt alle Gruppenmitglieder an, man kann die Gruppe verlassen und Administratoren können nach Mitgliedern suchen und sie hinzufügen oder entfernen.

**Methoden**

- `public void start(Activity activity)`

Startet diese Activity

**Parameter**

`Activity activity` Die Activity, die diese aufruft (wichtig z.B.: wenn man die Zurücktaste drückt.)

```
4.2.6 public class SettingsActivity extends AppCompatActivity implements View.OnClickListener implements PopupMenu.OnMenuItemClickListener MyResultReceiver.Receiver
```

**Beschreibung**

Diese Klasse zeigt die Benutzereinstellung an, und man kann diese verändern.

## Methoden

- `public void start(Activity activity)`  
Startet diese Activity

### Parameter

`Activity activity` Die Activity, die diese aufruft (wichtig z.B.: wenn man die Zurücktaste drückt.)

**4.2.7** `public class CreateNewGroupActivity extends AppCompatActivity  
implements View.OnClickListener implements PopupMenu.OnMenuItemClickListener  
MyResultReceiver.Receiver`

## Beschreibung

In dieser Klasse kann man eine neue Gruppe erstellen und Benutzer zu dieser Gruppe hinzufügen.

## Methoden

- `public void start(Activity activity)`  
Startet diese Activity

### Parameter

`Activity activity` Die Activity, die diese aufruft (wichtig z.B.: wenn man die Zurücktaste drückt.)

**4.2.8** `public class MeetingParticipantActivity extends AppCompatActivity  
implements View.OnClickListener implements PopupMenu.OnMenuItemClickListener  
MyResultReceiver.Receiver`

## Beschreibung

Diese Klasse zeigt eine Liste von zugesagten Teilnehmern eines Termins an

## Methoden

- `public void start(Activity activity)`  
Startet diese Activity

### Parameter

`Activity activity` Die Activity, die diese aufruft (wichtig z.B.: wenn man die Zurücktaste drückt.)

**4.2.9** `public class AboutActivity extends AppCompatActivity implements  
View.OnClickListener implements PopupMenu.OnMenuItemClickListener  
MyResultReceiver.Receiver`

## Beschreibung

Dies ist die Activity, in der Informationen über die App, ihre Ersteller und Rechtliches angezeigt werden.

## Methoden

- `public void start(Activity activity)`  
Startet diese Activity

### Parameter

`Activity activity` Die Activity, die diese aufruft (wichtig z.B.: wenn man die Zurücktaste drückt.)

**4.2.10** `public class MapActivity extends AppCompatActivity implements View.OnClickListener implements PopupMenu.OnMenuItemClickListener MyResultReceiver.Receiver`

## Beschreibung

Diese Klasse zeigt den Treffpunkt auf der Karte an:

Bei einer Tour wird die größte Teilnehmeransammlung auf der Karte angezeigt mit den Teilnehmern, die sich darin befinden.

Bei einer Veranstaltung werden alle Teilnehmer in einem bestimmten Umkreis angezeigt.

## Methoden

- `public void start(Activity activity)`  
Startet diese Activity

### Parameter

`Activity activity` Die Activity, die diese aufruft (wichtig z.B.: wenn man die Zurücktaste drückt.)

**4.2.11** `public class LoginActivity extends AppCompatActivity implements View.OnClickListener implements PopupMenu.OnMenuItemClickListener MyResultReceiver.Receiver`

## Beschreibung

Diese Klasse stellt eine Activity zum Einloggen der Benutzer dar.

## Methoden

- `public void start(Activity activity)`  
Startet diese Activity

### Parameter

`Activity activity` Die Activity, die diese aufruft (wichtig z.B.: wenn man die Zurücktaste drückt.)

```
4.2.12 public class MeetingListActivity extends AppCompatActivity  
implements View.OnClickListener implements PopupMenu.OnMenuItemClickListener  
MyResultReceiver.Receiver
```

### Beschreibung

Diese Klasse stellt eine Activity dar, die eine Liste der Termine eines Benutzers und die jeweiligen Termininformationen anzeigt.

### Methoden

- `public void start(Activity activity)`  
Startet diese Activity

#### Parameter

`Activity activity` Die Activity, die diese aufruft (wichtig z.B.: wenn man die Zurücktaste drückt.)

```
4.2.13 public class NewMeetingActivity extends AppCompatActivity im-  
plements View.OnClickListener implements PopupMenu.OnMenuItemClickListener  
MyResultReceiver.Receiver
```

### Beschreibung

Diese Klasse stellt eine Activity zur Erstellung eines neuen Termins dar.

### Methoden

- `public void start(Activity activity)`  
Startet diese Activity

#### Parameter

`Activity activity` Die Activity, die diese aufruft (wichtig z.B.: wenn man die Zurücktaste drückt.)

```
4.2.14 public abstract class ResultReceiver
```

### Beschreibung

Enthält die Implementierung, um ein “callback result” zu erhalten.

### Konstruktoren

- `public ResultReceiver(Handler handler)`

#### Parameter

`Handler handler` Schickt und bearbeitet Nachrichten- und Runnable-Objects

## Methoden

- `public void onReceiverResult(int resultCode, Bundle resultData)`  
Ergebnisse bekommen und bearbeiten.

### Parameter

`Integer resultCode` Result-Code der vom Sender definiert ist  
`Bundle resultData` zusätzliche Daten vom Sender

### 4.2.15 `public class ServiceResultReceiver extends ResultReceiver implements Receiver`

## Beschreibung

Diese Klasse erweitert die abstrakte Klasse `ResultReceiver` und implementiert das Interface `Receiver`.

## Methoden

- `public void setReceiver(Receiver receiver)`  
Receiver setzen.

### Parameter

`Receiver receiver` BroadcastReceiver

- `public interface Receiver`

- `onReceiverResult(int resultCode, Bundle resultData)`  
Ergebniss bekommen und bearbeiten.

### Parameter

`Integer resultCode` Result-Code der vom Sender definiert ist  
`Bundle resultData` zusätzliche Daten vom Sender

## 4.3 Package `kit.edu.pse.goapp.client.service`

### 4.3.1 `public abstract class IntentService`

## Beschreibung

Ein Service wird durch den Aufruf `startService(Intent intent)` vom Android OS gestartet. Der Parameter `intent` enthält Parameter und die Information, welcher Service gestartet werden soll.

## Konstruktoren

- `public IntentService(String name)`

### Parameter

`String name` Gibt den Worker-Thread einen Namen, der nur fürs Debugging wichtig ist.

## Methoden

- `protected void onHandleIntent(Intent intent)`  
Bei `startService(intent)` wird diese Methode ausgeführt.

### Parameter

`Intent intent` Der Intent beinhaltet die Beschreibung einer Operation.

### 4.3.2 public class LoginService extends IntentService

## Beschreibung

Diese Klasse erweitert die abstrakte Klasse `IntentService` und ist für die Benutzerauthorisierungsanfragen zuständig.

## Konstruktoren

`public LoginService()` (Default-Konstruktor)

## Methoden

Die Methode wird überschrieben

- `protected void onHandleIntent(Intent intent)`  
Beim `startService(intent)` wird diese Methode ausgeführt. Je nach Intent führt es folgende Dinge durch:  
Benutzerregistrierung, Login oder das Überprüfen, ob ein Benutzer registriert ist.

### Parameter

`Intent intent` Der Intent beinhaltet die Beschreibung einer Operation.

### 4.3.3 public class LogoutService extends IntentService

## Beschreibung

Diese Klasse erweitert die abstrakte Klasse `IntentService` und ist fürs Ausloggen zuständig.

## Konstruktoren

`public LogoutService()` (Default-Konstruktor)

## Methoden

Die Methode wird überschrieben

- `protected void onHandleIntent(Intent intent)`  
Beim `startService(intent)` wird diese Methode ausgeführt. Je nach Intent führt es folgende Dinge durch:  
Das Ausloggen eines Benutzers.

#### **Parameter**

`Intent intent` Der Intent beinhaltet die Beschreibung einer Operation.

#### **4.3.4 public class LoginFilterService extends IntentService**

#### **Beschreibung**

Diese Klasse erweitert die abstrakte Klasse `IntentService` und fragt beim Server ab, ob ein Benutzer auf die angeforderten Ressourcen zugreifen darf.

#### **Konstruktoren**

`public LoginFilterService()` (Default-Konstruktor)

#### **Methoden**

Die Methode wird überschrieben

- `protected void onHandleIntent(Intent intent)`

Beim `startService(intent)` wird diese Methode ausgeführt. Je nach Intent führt es folgende Dinge durch:

Das Überprüfen, ob ein Benutzer auf die angeforderten Ressourcen zugreifen darf.

#### **Parameter**

`Intent intent` Der Intent beinhaltet die Beschreibung einer Operation.

#### **4.3.5 public class UserService extends IntentService**

#### **Beschreibung**

Diese Klasse erweitert die abstrakte Klasse `IntentService` und ist für die Benutzerverwaltung zuständig.

#### **Konstruktoren**

`public UserService()` (Default-Konstruktor)

#### **Methoden**

Die Methode wird überschrieben

- `protected void onHandleIntent(Intent intent)`

Beim `startService(intent)` wird diese Methode ausgeführt. Je nach Intent führt es folgende Dinge durch:

Das Erstellen eines Benutzers, Benutzerinformationsanfragen, Änderungen der Benutzerinformationen und Löschen eines Benutzers.

#### **Parameter**

`Intent intent` Der Intent beinhaltet die Beschreibung einer Operation.

#### **4.3.6 public class UsersService extends IntentService**

##### **Beschreibung**

Diese Klasse erweitert die abstrakte Klasse `IntentService` und ist für Suchanfragen nach einem User zuständig.

##### **Konstruktoren**

`public UsersService() (Default-Konstruktor)`

##### **Methoden**

Die Methode wird überschrieben

- `protected void onHandleIntent(Intent intent)`

Beim `startService(intent)` wird diese Methode ausgeführt. Je nach Intent führt es folgende Dinge durch:  
Suchanfragen nach einem User

##### **Parameter**

`Intent intent` Der Intent beinhaltet die Beschreibung einer Operation.

#### **4.3.7 public class GroupService extends IntentService**

##### **Beschreibung**

Diese Klasse erweitert die abstrakte Klasse `IntentService` und ist für die Gruppenverwaltung zuständig.

##### **Konstruktoren**

`public GroupService() (Default-Konstruktor)`

##### **Methoden**

Die Methode wird überschrieben

- `protected void onHandleIntent(Intent intent)`

Beim `startService(intent)` wird diese Methode ausgeführt. Je nach Intent führt es folgende Dinge durch:  
Erstellen einer Gruppe, Gruppeninformationsanfragen, Gruppenänderungen und das Löschen einer Gruppe.

##### **Parameter**

`Intent intent` Der Intent beinhaltet die Beschreibung einer Operation.

#### **4.3.8 public class GroupsService extends IntentService**

##### **Beschreibung**

Diese Klasse erweitert die abstrakte Klasse `IntentService` und ist für die Anforderungen einer Gruppenliste zuständig.

##### **Konstruktoren**

`public GroupsService()` (Default-Konstruktor)

##### **Methoden**

Die Methode wird überschrieben

- `protected void onHandleIntent(Intent intent)`

Beim `startService(intent)` wird diese Methode ausgeführt. Je nach Intent führt es folgende Dinge durch:  
Gruppenlistenanforderungen

##### **Parameter**

`Intent intent` Der Intent beinhaltet die Beschreibung einer Operation.

#### **4.3.9 public class GroupUserManagementService extends IntentService**

##### **Beschreibung**

Diese Klasse erweitert die abstrakte Klasse `IntentService` und ist für die Gruppenmitgliederverwaltung zuständig.

##### **Konstruktoren**

`public GroupUserManagementService()` (Default-Konstruktor)

##### **Methoden**

Die Methode wird überschrieben

- `protected void onHandleIntent(Intent intent)`

Beim `startService(intent)` wird diese Methode ausgeführt. Je nach Intent führt es folgende Dinge durch:  
Das Hinzufügen von Benutzern in Gruppen und die Anforderung einer Liste der Gruppenmitglieder einer Gruppe

##### **Parameter**

`Intent intent` Der Intent beinhaltet die Beschreibung einer Operation.

```
4.3.10 public class GPS_Service extends IntentService
```

### Beschreibung

Diese Klasse erweitert die abstrakte Klasse `IntentService` und ist für das Schicken der GPS-Daten zuständig.

### Konstruktoren

```
public GPSService() (Default-Konstruktor)
```

### Methoden

Die Methode wird überschrieben

- `protected void onHandleIntent(Intent intent)`

Beim `startService(intent)` wird diese Methode ausgeführt. Je nach Intent führt es folgende Dinge durch:  
GPS-Daten schicken.

#### Parameter

`Intent intent` Der Intent beinhaltet die Beschreibung einer Operation.

```
4.3.11 public class MeetingService extends IntentService
```

### Beschreibung

Diese Klasse erweitert die abstrakte Klasse `IntentService` und ist für die Terminverwaltung zuständig.

### Konstruktoren

```
public MeetingService() (Default-Konstruktor)
```

### Methoden

Die Methode wird überschrieben

- `protected void onHandleIntent(Intent intent)`

Beim `startService(intent)` wird diese Methode ausgeführt. Je nach Intent führt es folgende Dinge durch:  
Das Erstellen eines Termins, Termininformationsanfragen, Terminänderungen und das Löschen eines Termins

#### Parameter

`Intent intent` Der Intent beinhaltet die Beschreibung einer Operation.

```
4.3.12 public class MeetingsService extends IntentService
```

### Beschreibung

Diese Klasse erweitert die abstrakte Klasse `IntentService` und ist für Terminlistenanforderungen zuständig.

### Konstruktoren

```
public MeetingsService() (Default-Konstruktor)
```

### Methoden

Die Methode wird überschrieben

- `protected void onHandleIntent(Intent intent)`

Beim `startService(intent)` wird diese Methode ausgeführt. Je nach Intent führt es folgende Dinge durch:  
Terminlistenanforderungen

#### Parameter

`Intent intent` Der Intent beinhaltet die Beschreibung einer Operation.

```
4.3.13 public class MeetingParticipantManagementService extends IntentService
```

### Beschreibung

Diese Klasse erweitert die abstrakte Klasse `IntentService` und ist für die Teilnehmerverwaltung zuständig.

### Konstruktoren

```
public MeetingParticipantManagementService() (Default-Konstruktor)
```

### Methoden

Die Methode wird überschrieben

- `protected void onHandleIntent(Intent intent)`

Beim `startService(intent)` wird diese Methode ausgeführt. Je nach Intent führt es folgende Dinge durch:  
Anforderungen einer Liste von Teilnehmern, die für den Termin zugesagt haben und die Bestätigung bzw. Absagen eines Termins

#### Parameter

`Intent intent` Der Intent beinhaltet die Beschreibung einer Operation.

#### **4.3.14 public class NotificationService extends IntentService**

##### **Beschreibung**

Diese Klasse erweitert die abstrakte Klasse `IntentService` und ist für die Einstellungen der Benachrichtigungen zuständig.

##### **Konstruktoren**

`public NotificationService()` (Default-Konstruktor)

##### **Methoden**

Die Methode wird überschrieben

- `protected void onHandleIntent(Intent intent)`

Beim `startService(intent)` wird diese Methode ausgeführt. Je nach Intent führt es folgende Dinge durch:

Liste von Benachrichtigungen ausgeben (Priorität B) und Benachrichtigungen an-/ausschalten

##### **Parameter**

`Intent intent` Der Intent beinhaltet die Beschreibung einer Operation.

#### **4.4 Package kit.edu.pse.goapp.client.uri\_builder**

##### **4.4.1 public abstract class ClientURI\_Builder**

##### **Beschreibung**

Beinhaltet einen privat static final string `serverAdd`, welcher die Serveradresse beinhaltet und ein privates Feld `URI`, in dem die URI zwischengespeichert wird.

##### **Methoden**

- `public void addParameter(String key, String value)`

Fügt über die gleichnamige Methode Parameter zum privaten URI hinzu.

##### **Parameter:**

`String key` Schlüssel

`String value` Wert

- `public String getURI()`

Liefert das lokale URI-Objekt zurück

##### **Rückgabewert:**

URI als String

#### **4.4.2 public class URI\_LoginBuilder**

##### **Beschreibung**

Beinhaltet einen privat static final String servletAdd, welcher die Adresse zum LoginServlet beinhaltet.

##### **Konstruktoren**

- **public URI\_LoginBuilder()**  
Default-Konstruktor

##### **Methoden**

Die URI\_LoginBuilder hat keine eigenen Methoden, übernimmt alle Methoden der Oberklasse und überschreibt keine.

#### **4.4.3 public class URI\_LogoutBuilder**

##### **Beschreibung**

Beinhaltet einen privat static final String servletAdd, welcher die Adresse zum LogoutServlet beinhaltet.

##### **Konstruktoren**

- **public URI\_LogoutBuilder()**  
Default-Konstruktor

##### **Methoden**

Die URI\_LoginBuilder hat keine eigenen Methoden, übernimmt alle Methoden der Oberklasse und überschreibt keine.

#### **4.4.4 public class URI\_LoginFilterBuilder**

##### **Beschreibung**

Beinhaltet einen privat static final String servletAdd, welcher die Adresse zum LoginFilterServlet beinhaltet.

##### **Konstruktoren**

- **public URI\_LoginFilterBuilder()**  
Default-Konstruktor

## Methoden

Die URI\_LoginBuilder hat keine eigenen Methoden, übernimmt alle Methoden der Oberklasse und überschreibt keine.

### 4.4.5 public class URI\_GroupBuilder

#### Beschreibung

Beinhaltet einen privat static final String servletAdd, welcher die Adresse zum GroupServlet beinhaltet.

#### Konstruktoren

- public URI\_GroupBuilder()  
Default-Konstruktor

## Methoden

Die URI\_LoginBuilder hat keine eigenen Methoden, übernimmt alle Methoden der Oberklasse und überschreibt keine.

### 4.4.6 public class URI\_GroupsBuilder

#### Beschreibung

Beinhaltet einen privat static final String servletAdd, welcher die Adresse zum GroupsServlet beinhaltet.

#### Konstruktoren

- public URI\_GroupsBuilder()  
Default-Konstruktor

## Methoden

Die URI\_LoginBuilder hat keine eigenen Methoden, übernimmt alle Methoden der Oberklasse und überschreibt keine.

### 4.4.7 public class URI\_GroupUserManagementBuilder

#### Beschreibung

Beinhaltet einen privat static final String servletAdd, welcher die Adresse zum GroupUserManagementServlet beinhaltet.

## Konstruktoren

- `public URI_GroupUserManagementBuilder()`  
Default-Konstruktor

## Methoden

Die `URI_LoginBuilder` hat keine eigenen Methoden, übernimmt alle Methoden der Oberklasse und überschreibt keine.

### 4.4.8 `public class URI_MeetingBuilder`

#### Beschreibung

Beinhaltet einen privat static final String `servletAdd`, welcher die Adresse zum `MeetingServlet` beinhaltet.

## Konstruktoren

- `public URI_MeetingBuilder()`  
Default-Konstruktor

## Methoden

Die `URI_LoginBuilder` hat keine eigenen Methoden, übernimmt alle Methoden der Oberklasse und überschreibt keine.

### 4.4.9 `public class URI_MeetingsBuilder`

#### Beschreibung

Beinhaltet einen privat static final String `servletAdd`, welcher die Adresse zum `MeetingsServlet` beinhaltet.

## Konstruktoren

- `public URI_MeetingsBuilder()`  
Default-Konstruktor

## Methoden

Die `URI_LoginBuilder` hat keine eigenen Methoden, übernimmt alle Methoden der Oberklasse und überschreibt keine.

#### **4.4.10 public class URI\_MeetingParticipantManagementBuilder**

##### **Beschreibung**

Beinhaltet einen privat static final String servletAdd, welcher die Adresse zum MeetingParticipantManagementServlet beinhaltet.

##### **Konstruktoren**

- **public URI\_MeetingParticipantManagementBuilder()**  
Default-Konstruktor

##### **Methoden**

Die URI\_LoginBuilder hat keine eigenen Methoden, übernimmt alle Methoden der Oberklasse und überschreibt keine.

#### **4.4.11 public class URI\_UserBuilder**

##### **Beschreibung**

Beinhaltet einen privat static final String servletAdd, welcher die Adresse zum UserServlet beinhaltet.

##### **Konstruktoren**

- **public URI\_UserBuilder()**  
Default-Konstruktor

##### **Methoden**

Die URI\_LoginBuilder hat keine eigenen Methoden, übernimmt alle Methoden der Oberklasse und überschreibt keine.

#### **4.4.12 public class URI\_UsersBuilder**

##### **Beschreibung**

Beinhaltet einen privat static final String servletAdd, welcher die Adresse zum UsersServlet beinhaltet.

##### **Konstruktoren**

- **public URI\_UsersBuilder()**  
Default-Konstruktor

## Methoden

Die URI\_LoginBuilder hat keine eigenen Methoden, übernimmt alle Methoden der Oberklasse und überschreibt keine.

### 4.4.13 public class URI\_GPS\_Builder

#### Beschreibung

Beinhaltet einen privat static final String servletAdd, welcher die Adresse zum GPS\_Servlet beinhaltet.

#### Konstruktoren

- public URI\_GPS\_Builder()  
Default-Konstruktor

## Methoden

Die URI\_LoginBuilder hat keine eigenen Methoden, übernimmt alle Methoden der Oberklasse und überschreibt keine.

### 4.4.14 public class URI\_NotificationsBuilder

#### Beschreibung

Beinhaltet einen privat static final String servletAdd, welcher die Adresse zum NotificationsServlet beinhaltet.

#### Konstruktoren

- public URI\_NotificationsBuilder()  
Default-Konstruktor

## Methoden

Die URI\_LoginBuilder hat keine eigenen Methoden, übernimmt alle Methoden der Oberklasse und überschreibt keine.

## 4.5 Package kit.edu.pse.goapp.client.httpappclient

### 4.5.1 public class HttpResponse

#### Beschreibung

Diese Klasse ist das Rückgabeformat des HttpAppClient für alle Methoden.

## Methoden

- `public int getStatusCode()`  
Liefert den Statuscode vom Server zurück.  
**Rückgabewert:**  
`Int statusCode` StatusCode vom Server
- `public int getJsonString()`  
Liefert den JSON-String vom Server zurück.  
**Rückgabewert:**  
`String json` Rückgabeobjekt vom Server in String-Format

### 4.5.2 public class HttpAppClient

#### Beschreibung

Diese Klasse gibt ein `HttpResponse`-Objekt zurück, welches einen Statuscode enthält, der angibt, ob die Anfrage fehlerfrei bearbeitet werden konnte, und, falls gefordert, die angefragte Objekte als String beinhaltet.

## Methoden

- `public HttpResponse sendPost(URI uri)`  
Sendet einen HTTP POST Request an den Server  
**Parameter**  
`Uri uri` Gibt die URL des Servlets an und hält Parameter gespeichert.  
**Rückgabewert:**  
`HttpResponse`-Objekt mit Statuscode
- `public HttpResponse sendGet(URI uri)`  
Sendet einen HTTP GET Request an den Server  
**Parameter**  
`Uri uri` Gibt die URL des Servlets an und hält Parameter gespeichert.  
**Rückgabewert:**  
`HttpResponse`-Objekt mit Statuscode und JSON-String
- `public HttpResponse sendDelete(URI uri)`  
Sendet einen HTTP DELETE Request an den Server.  
**Parameter**  
`Uri uri` Gibt die URL des Servlets an und hält Parameter gespeichert.  
**Rückgabewert:**  
`HttpResponse`-Objekt mit Statuscode
- `public HttpResponse sendPut(URI uri)`  
Sendet einen HTTP PUT Request an den Server  
**Parameter**  
`Uri uri` Gibt die URL des Servlets an und hält Parameter gespeichert  
**Rückgabewert:**  
`HttpResponse`-Objekt mit Statuscode

## 4.6 Package kit.edu.pse.goapp.client.converter

### 4.6.1 Interface Converter<T>

#### Beschreibung

Die Konverter, die JSON-Strings zu Objekten konvertieren und umgekehrt, müssen das Interface Converter<T> implementieren.

#### Methoden

- `public T parse(String jsonString)`  
Parst den JSON-String  
**Parameter:**  
`String jsonString` Zu überprüfender String  
**Rückgabewert:**  
JSON-String als T
- `public String serialize(T object)`  
Erstellt einen String aus dem eingegebenen Objekt im Format T  
**Parameter:**  
`T object` Objekt im Format T  
**Rückgabewert:**  
object als String
- `public String serialize(List <T> objects)`  
Erstellt einen String aus List<T>  
**Parameter:**  
`List<T> objects` Liste von Objekten  
**Rückgabewert:**  
List<T> als String

### 4.6.2 Class NotificationConverter implements Converter<T>

#### Beschreibung

Die Klasse NotificationConverter implementiert das Interface Converter<T> und konvertiert Benachrichtigungen von Parametern in Objekte und umgekehrt.

#### Konstruktoren

- `public NotificationConverter()`  
Default-Konstruktor

#### Methoden

Die Klasse NotificationConverter implementiert und überschreibt die Methoden `public T parse(String jsonString)`, `public String serialize(T object)` und `public String serialize(List<T> objects)`.

- `public Notification parse(String jsonString)`  
Parst einen Json-String  
**Parameter:**  
`String jsonString` Zu überprüfender String der Benachrichtigung  
**Rückgabewert:**  
Json-String des Benachrichtigung als `Notification`
- `public String serialize(Notification notification)`  
Erstellt einen String aus der Benachrichtigung  
**Parameter:**  
`Notification notification` die Benachrichtigung  
**Rückgabewert:**  
Benachrichtigung als String
- `public String serialize(List <Notification> notifications)`  
Erstellt einen String aus List<Notification>  
**Parameter:**  
`List<Notification> notifications` Liste von Benachrichtigungen  
**Rückgabewert:**  
List<Notification> als String

#### 4.6.3 Class GroupMemberConverter implements Converter<T>

##### Beschreibung

Die Klasse `GroupMemberConverter` implementiert das Interface `Converter<T>` und konvertiert die Gruppenmitglieder von Parametern in Objekte und umgekehrt.

##### Konstruktoren

- `public GroupMemberConverter()`  
Default-Konstruktor

##### Methode

Die Klasse `GroupMemberConverter` implementiert und überschreibt die Methoden `public T parse(String jsonString)`, `public String serialize(T object)` und `public String serialize(List<T> objects)`.

- `public User parse(String jsonString)`  
Parst einen Json-String  
**Parameter:**  
`String jsonString` Zu überprüfender String der Gruppe  
**Rückgabewert:**  
Json-String des Gruppenmitglieds als User

- `public String serialize(Group group)`  
Erstellt einen String aus der Gruppe  
**Parameter:**  
Group group die Gruppe  
**Rückgabewert:**  
Gruppe als String
- `public String serialize(List <Group> groups)`  
Erstellt einen String aus List<Group>  
**Parameter:**  
List<Group> group Liste von Gruppen  
**Rückgabewert:**  
List<Group> als String
- `public String serialize(List <User> users)`  
Erstellt einen String aus List<User>  
**Parameter:**  
List<User> users Liste von Benutzern  
**Rückgabewert:**  
List<User> als String

#### 4.6.4 Class GroupConverter implements Converter<T>

##### Beschreibung

Die Klasse `GroupConverter` implementiert das Interface `Converter<T>` und konvertiert eine Gruppe von Parametern in Objekte und umgekehrt.

##### Konstruktoren

- `public GroupConverter()`  
Default-Konstruktor

##### Methoden

Die Klasse `GroupConverter` implementiert und überschreibt die Methoden `public T parse(String jsonString)`, `public String serialize(T object)` und `public String serialize(List<T> objects)`.

- `public Group parse(String jsonString)`  
Parst einen Json-String  
**Parameter:**  
String jsonString Zu überprüfender String der Gruppe  
**Rückgabewert:**  
Json-String der Gruppe als Group
- `public String serialize(Group group)`  
Erstellt einen String aus der Gruppe

**Parameter:**

Group group die Gruppe

**Rückgabewert:**

Gruppe als String

- `public String serialize(List <Group> groups)`  
Erstellt einen String aus List<Group>

**Parameter:**

List<Group> groups Liste von Gruppen

**Rückgabewert:**

List<Group> als String

#### 4.6.5 Class GPS\_Converter implements Converter<T>

##### Beschreibung

Die Klasse GPS\_Converter implementiert das Interface Converter<T> und konvertiert die GPS-Daten von Parametern in Objekte und umgekehrt.

##### Konstruktoren

- `public GPS_Converter()`  
Default-Konstruktor

##### Methoden

##### Methoden

Die Klasse GPS\_Converter implementiert und überschreibt die Methoden `public T parse(String jsonString)`, `public String serialize(T object)` und `public String serialize(List<T> objects)`.

- `public GPS parse(String jsonString)`  
Parst einen Json-String  
**Parameter:**  
String jsonString Zu überprüfender String des GPS  
**Rückgabewert:**  
Json-String des GPS als GPS
- `public String serialize(GPS gps)`  
Erstellt einen String aus dem GPS  
**Parameter:**  
GPS gps das GPS  
**Rückgabewert:**  
GPS als String
- `public String serialize(List <GPS> gps)`  
Erstellt einen String aus List<GPS>

**Parameter:**

List<GPS> gps Liste von GPS-Daten

**Rückgabewert:**

List<GPS> als String

#### 4.6.6 Class UserConverter implements Converter<T>

##### Beschreibung

Die Klasse `UserConverter` implementiert das Interface `Converter<T>` und konvertiert die Benutzer von Parametern in Objekte und umgekehrt.

##### Konstruktoren

- `public UserConverter()`  
Default-Konstruktor

##### Methoden

Die Klasse `UserConverter` implementiert und überschreibt die Methoden `public T parse(String jsonString)`, `public String serialize(T object)` und `public String serialize(List<T> objects)`.

- `public User parse(String jsonString)`  
Parst einen Json-String  
**Parameter:**  
String jsonString Zu überprüfender String des Benutzers  
**Rückgabewert:**  
Json-String des Benutzers als User
- `public String serialize(User user)`  
Erstellt einen String aus dem Benutzer  
**Parameter:**  
User user der Benutzer  
**Rückgabewert:**  
Benutzer als String
- `public String serialize(List <User> users)`  
Erstellt einen String aus List<User>  
**Parameter:**  
List<User> users Liste von Benutzern  
**Rückgabewert:**  
List<User> als String

#### 4.6.7 Class ParticipantConverter implements Converter<T>

##### Beschreibung

Die Klasse `ParticipantConverter` implementiert das Interface `Converter<T>` und konvertiert einen Teilnehmer von Parametern in Objekte und umgekehrt.

##### Konstruktoren

- `public ParticipantConverter()`  
Default-Konstruktor

##### Methoden

Die Klasse `ParticipantConverter` implementiert und überschreibt die Methoden `public T parse(String jsonString)`, `public String serialize(T object)` und `public String serialize(List<T> objects)`.

- `public Participant parse(String jsonString)`  
Parst einen Json-String  
**Parameter:**  
`String jsonString` Zu überprüfender String des Participant  
**Rückgabewert:**  
Json-String des Participant als Participant
- `public String serialize(Participant participant)`  
Erstellt einen String aus dem Participant  
**Parameter:**  
`Participant participant` der Teilnehmer  
**Rückgabewert:**  
Participant als String
- `public String serialize(List <Participant> participants)`  
Erstellt einen String aus List<Participant>  
**Parameter:**  
`List<Participant> participant` Liste von Participants  
**Rückgabewert:**  
List<Participant> als String

#### 4.6.8 Class MeetingConverter implements Converter<T>

##### Beschreibung

Die Klasse `MeetingConverter` implementiert das Interface `Converter<T>` und konvertiert einen Termin von Parametern in Objekte und umgekehrt.

## Konstruktoren

- `public MeetingConverter()`  
Default-Konstruktor

## Methoden

Die Klasse `MeetingConverter` implementiert und überschreibt die Methoden `public T parse(String jsonString)`, `public String serialize(T object)` und `public String serialize(List<T> objects)`.

- `public Meeting parse(String jsonString)`  
Parst einen Json-String  
**Parameter:**  
`String jsonString` Zu überprüfender String des Treffens  
**Rückgabewert:**  
Json-String des Treffens als Meeting
- `public String serialize(Meeting meeting)`  
Erstellt einen String aus dem Meeting  
**Parameter:**  
`Meeting meeting` der Termin  
**Rückgabewert:**  
Meeting als String
- `public String serialize(List <Meeting> meetings)`  
Erstellt einen String aus List<Meeting>  
**Parameter:**  
`List<Meeting> meetings` Liste von Terminen  
**Rückgabewert:**  
List<Meeting> als String

## 4.7 Package kit.edu.pse.goapp.client.receiver

### 4.7.1 Abstract class BroadcastReceiver

#### Beschreibung

Basisklasse zum Empfangen von Anfragen mit `sendBroadcast()`. Der BroadcastReceiver läuft im Hintergrund.

## Methoden

- `public void onReceive(Context context, Intent intent)`  
Diese Methode wird aufgerufen, wenn die Klasse ein Broadcast mit Intent empfängt.  
**Parameter**  
`Context context` Der Kontext, in dem der Receiver läuft  
`Intent intent` Die Beschreibung einer Operation, die empfangen wurde

#### **4.7.2 Class AlarmReceiver extends BroadcastReceiver**

##### **Beschreibung**

Diese Klasse erweitert die abstrakte Klasse `BroadcastReceiver`. Wenn dieser einen Broadcast erhält, wird der `GPS_Service` gestartet.

##### **Konstruktoren**

```
public AlarmReceiver() (Default-Konstruktor)
```

##### **Methoden**

- `public void onReceive(Context context, Intent intent)`

Der `GPS_Service` wird gestartet

###### **Parameter**

`Context context` Der Kontext, in dem der Receiver läuft

`Intent intent` Beinhaltet, dass man den `GPS_Service` starten soll.

#### **4.7.3 Class DeviceBootReceiver extends BroadcastReceiver**

##### **Beschreibung**

Diese Klasse erweitert die abstrakte Klasse `BroadcastReceiver`. Sie erstellt die Alarne neu bei einem Reboot.

##### **Konstruktoren**

```
public DeviceBootReceiver() (Default-Konstruktor)
```

##### **Methoden**

- `public void onReceive(Context context, Intent intent)`

Erhält einen Broadcast beim Neustart und erstellt die Alarne neu.

###### **Parameter**

`Context context` Der Kontext, in dem der Receiver läuft

`Intent intent` Beinhaltet, dass man alle Alarne neu starten soll.

### **4.8 Package kit.edu.pse.goapp.client.databasehandler**

#### **4.8.1 abstract class SQLiteOpenHelper**

##### **Beschreibung**

Eine Hilfsklasse, um Datenbanken zu erstellen und ihre Versionen zu verwalten.

## Methoden

- `public void onCreate(SQLiteDatabase db)`  
Erstellt eine SQLiteDatabase  
**Parameter**  
`SQLiteDatabase db` Die Datenbank
- `public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion)`  
Aktualisiert die SQLiteDatabase  
**Parameter**  
`SQLiteDatabase db` Die Datenbank  
`int oldVersion` Alte Versionsnummer  
`int newVersion` Neue Versionsnummer

### 4.8.2 Class DataBaseHandler extends SQLiteOpenHelper

#### Beschreibung

Hilft bei dem Erstellen und Managen der Datenbank, in der die MeetingIDs und MeetingTimes gespeichert werden.

#### Konstruktoren

- `public DataBaseHandler(Context context)`  
**Parameter:**  
`Context context` Der Kontext der Datenbank

#### Methoden

- `public void onHandleIntent(Intent intent)`  
Führt die im Intent beschriebenen Befehle aus.  
**Parameter**  
`Intent intent` Beschreibt die auszuführenden Befehle.

## 5 Klassen des Servers

### 5.1 Package kit.edu.pse.goapp.server.datamodels

#### 5.1.1 Abstract class Meeting

##### Beschreibung

Die abstrakte Klasse Meeting stellt einen Termin einer Gruppe dar.

##### Konstruktoren

- `public Event(int ID, String name, GPS place, int timestamp, int duration, Participant creator)`  
**Parameter**

<code>int ID</code>	ID des Termins
<code>String name</code>	Name des Termins
<code>GPS place</code>	Veranstaltungsort mit GPS-Koordinaten angegeben
<code>int timestamp</code>	Veranstaltungsbeginn
<code>int duration</code>	Voraussichtliche Dauer des Termins
<code>Participant creator</code>	Ersteller des Termins

##### Methoden

- `public int getID()`  
Gibt die ID des Termins zurück.  
**Rückgabewert:**  
ID des Termins als Integer
- `public String getName()`  
Gibt den Terminnamen zurück.  
**Rückgabewert:**  
String mit Terminnamen
- `public GPS getPlace()`  
Gibt den Ort des Termins zurück.  
**Rückgabewert:**  
Ort des Termins mithilfe von GPS-Koordinaten
- `public int getTimestamp()`  
Gibt die Uhrzeit zurück, wann der Termin beginnt.  
**Rückgabewert:**  
Uhrzeit als Integer
- `public int getDuration()`  
Gibt die voraussichtliche Dauer des Termins zurück.  
**Rückgabewert:**  
Dauer des Termins als Integer

- `public Participant getCreator()`  
Gibt den Ersteller des Termins zurück.  
**Rückgabewert:**  
Ersteller des Termins als Participant
- `public List<Participant> getParticipants()`  
Gibt eine Liste mit allen Teilnehmern zurück.  
**Rückgabewert:**  
Liste mit Teilnehmern (Participants)
- `public void addParticipants(Participant participant)`  
Fügt einen Teilnehmer zu dem Termin hinzu.  
**Parameter:**  
`Participant participant` Hinzuzufügender Teilnehmer

### 5.1.2 Class Event extends Meeting

#### Beschreibung

Die Klasse Event erweitert die abstrakte Klasse Meeting. Sie beschreibt ein Treffen mit festem Ort (Veranstaltung).

#### Konstruktoren

`public Event()` (Default-Konstruktor)

#### Methoden

Die Klasse Event enthält keine eigenen Methoden. Sie übernimmt alle Methoden aus der Klasse Meeting und überschreibt keine Methode. Neben den Methoden der Klasse Meeting werden keine weiteren Methoden benötigt.

### 5.1.3 Class Tour extends Meeting

#### Beschreibung

Die Klasse Tour erweitert die abstrakte Klasse Meeting. Sie beschreibt ein Treffen mit wechselndem Veranstaltungsort.

#### Parameter

- `public static double maxCenterDistance`

#### Konstruktoren

`public Tour()` (Default-Konstruktor)

## Methoden

Die Klasse `Tour` übernimmt alle Methoden aus der Klasse `Meeting` und über-schreibt keine Methode. Sie fügt folgende Methode hinzu:

- `public MeetingCenter getCenter()`  
Gibt die Koordinaten der Ansammlung der meisten Teilnehmer (Zentrum) an.

### 5.1.4 Class MeetingCenter

#### Beschreibung

Die Klasse `MeetingCenter` stellt den Ort dar, an dem sich während einer Tour die meisten Teilnehmer befinden (Zentrum).

## Methoden

- `public void addParticipantToCenter(Participant participant)`  
Fügt einen Teilnehmer zum Zentrum hinzu, sobald er sich am Zentrum befindet.

#### Parameter

`Participant participant` Teilnehmer, der zum Zentrum hinzugefügt werden soll

- `public void removeParticipantFromCenter(Participant participant)`  
Entfernt einen Teilnehmer aus dem Zentrum, sobald er sich vom Zentrum entfernt.  
`Participant participant` Teilnehmer, der aus dem Zentrum entfernt werden soll

- `public List<Participant> getParticipants()`  
Gibt Teilnehmer zurück, die sich im Zentrum befinden.

#### Rückgabewert:

Liste mit Teilnehmern (Participants) im Zentrum

- `public GPS getPlace()`  
Gibt Ort des Zentrums zurück.

#### Rückgabewert:

GPS-Koordinaten des Zentrums

### 5.1.5 Class Participant

#### Beschreibung

Die Klasse `Participant` stellt einen Teilnehmer eines Termins dar.

## Konstruktoren

- `public Participant(int participantID, User user, MeetingConfirmation confirmation)`

**Parameter:**

<code>int participantID</code>	ID des Teilnehmers
<code>User user</code>	Benutzer der Go-App
<code>MeetingConfirmation confirmation</code>	Zu- oder Absage des Termins

**Methoden**

- `public int getParticipant_ID()`  
Gibt die Teilnehmer-ID zurück.

**Rückgabewert:**

Teilnehmer-ID als Integer

- `public User getUser()`  
Gibt den teilnehmenden Benutzer zurück.

**Rückgabewert:**

Teilnehmenden Benutzer (`User`)

- `public MeetingConfirmation getConfirmation()`  
Gibt zurück, ob der Benutzer den Termin zu- oder abgesagt hat.

**Rückgabewerte:**

`CONFIRMED` Die Person hat zugesagt

`PENDING` Die Person hat bis jetzt weder zu- noch abgesagt

`REJECTED` Die Person hat abgesagt

**5.1.6 Enumeration MeetingConfirmation****Beschreibung**

Die Enumeration `MeetingConfirmation` beschreibt, ob eine Person eine Veranstaltung schon zu- oder abgesagt hat.

**Parameter**

<code>CONFIRMED</code>	Die Person hat zugesagt
<code>PENDING</code>	Die Person hat bis jetzt weder zu- noch abgesagt
<code>REJECTED</code>	Die Person hat abgesagt

**5.1.7 Class User****Beschreibung**

Die Klasse `User` stellt einen Benutzer der Go-App dar.

**Konstruktoren**

- `public User(int userID, String name)`

**Parameter:**

`int userID` ID des Benutzers

`String name` Name des Benutzers

## Methoden

- `public int getID()`  
Gibt die ID des Benutzers zurück.  
**Rückgabewerte:**  
ID des Benutzers als Integer
- `public String getName()`  
Gibt den Namen des Benutzers zurück.  
**Rückgabewerte:**  
Name des Benutzers als String
- `public void setGPS(GPS gps)`  
Setzt die GPS-Position des Benutzers  
**Parameter:**  
GPS gps GPS-Position des Benutzers
- `public GPS getGPS()`  
Gibt den Standort des Benutzers zurück.  
**Rückgabewert:**  
GPS-Koordinaten des Benutzers
- `public void addGroup(Group group)`  
Erstellt eine neue Gruppe  
**Parameter:**  
Group group Neue Gruppe
- `public void addMeeting(Meeting meeting)`  
Erstellt einen neuen Termin für eine Gruppe  
**Parameter:**  
Meeting meeting Neuer Termin
- `public List<Group> getAllGroups()`  
Gibt alle Gruppen des Benutzers zurück.  
**Rückgabewert:**  
Liste mit allen Gruppen
- `public List<Meeting> getAllMeetings()`  
Gibt alle Termine des Benutzers zurück.  
**Rückgabewert:**  
Liste mit allen Terminen

### 5.1.8 Class Group

#### Beschreibung

Die Klasse `Group` stellt eine Gruppe von Benutzern dar, die sich in der Go-App zusammenschließen.

## Konstruktoren

- `public Group(int groupID, String name)`  
**Parameter:**  
int groupID ID der Gruppe  
String name Name der Gruppe

## Methoden

- `public int getID()`  
Gibt die ID der Gruppe zurück.  
**Rückgabewert:**  
ID der Gruppe als Integer
- `public String getName()`  
Gibt den Namen der Gruppe zurück.  
**Rückgabewert:**  
Gruppenname als String
- `public void setName(String name)`  
Setzt den Namen der Gruppe.  
**Parameter:**  
String name Name der Gruppe
- `public void addAdmin(User user)`  
Fügt einen Administrator hinzu.  
**Parameter:**  
User user Benutzer, der zum Administrator ernannt werden soll
- `public List<User> getAdmins()`  
Gibt alle Administratoren der Gruppe zurück.  
**Rückgabewert:**  
Liste mit allen Gruppenadministratoren
- `public void addGroupMember(User user)`  
Fügt ein Gruppenmitglied zu der Gruppe hinzu.  
**Parameter:**  
User user Benutzer, der zur Gruppe hinzugefügt werden soll
- `public List<User> getGroupMembers()`  
Gibt alle Gruppenmitglieder zurück.  
**Rückgabewert:**  
Liste mit allen Gruppenmitgliedern

### 5.1.9 Class GPS

## Beschreibung

Die Klasse GPS stellt den Standort mithilfe von GPS-Koordinaten dar.

## Konstruktoren

- public GPS(double x, double y, double z)  
**Parameter:**  
double x GPS x-Koordinate  
double y GPS y-Koordinate  
double z GPS z-Koordinate

## Parameter

- public static double radius

## Methoden

- public static double distance(GPS first, GPS second)  
Berechnet die Distanz zwischen den beiden ermittelten Standorten.  
**Rückgabewert:**  
Distanz als Integer
- public boolean isClose(GPS first, GPS second, double radius)  
**Rückgabewert:**  
true, falls sonst false
- public double getX()  
Gibt die x-Koordinate zurück.  
**Rückgabewert:**  
GPS x-Koordinate
- public double getY()  
Gibt die y-Koordinate zurück.  
**Rückgabewert:**  
GPS y-Koordinate
- public double getZ()  
Gibt die z-Koordinate zurück.  
**Rückgabewert:**  
GPS z-Koordinate

### 5.1.10 Class Notification

## Beschreibung

Die Klasse Notification benachrichtigt den Benutzer über seine Termine.

## Konstruktoren

- public Notification(String text)  
**Parameter:**  
String text Benachrichtigungstext

## Methoden

- `public String getText()`  
Gibt den Benachrichtigungstext zurück.  
**Rückgabewert:**  
Benachrichtigungstext als String

## 5.2 Package kit.edu.pse.goapp.server.daos

### 5.2.1 Interface GroupDAO

#### Beschreibung

Dieses Interface stellt eine Gruppe von Benutzern der Go-App dar.

#### Methoden

- `public void addGroup()`  
Eine neue Gruppe wird erstellt.
- `public List<Group> getAll Groups()`  
Gibt alle Gruppen der Go-App zurück.  
**Liefert**  
Liste mit allen Gruppen
- `public void updateGroup()`  
Aktualisiert die Gruppendaten.
- `public void deleteGroup()`  
Löscht die Gruppe.
- `public Group getGroupByID()`  
Gibt Gruppe anhand Gruppen-ID zurück.  
**Rückgabewert:**  
Gruppe anhand ihrer ID

### 5.2.2 Class GroupDAOImpl implements GroupDAO

#### Beschreibung

Diese Klasse implementiert das Interface GroupDAO.

#### Konstruktoren

- `public GroupDAOImpl(int ID, String name)`  
**Parameter**  
ID ID der Gruppe

### 5.2.3 Interface GroupMemberDAO

#### Beschreibung

Dieses Interface stellt die Mitglieder einer Gruppe dar.

#### Methoden

- `public void addMember()`  
Fügt ein Mitglied zur Gruppe hinzu.
- `public void deleteMember()`  
Entfernt ein Mitglied aus der Gruppe.
- `public void updateMember()`  
Aktualisiert die Gruppenmitglieder der Gruppe.
- `public List<User> getAllMembers()`  
Gibt eine Liste aller Gruppenmitglieder zurück.  
**Rückgabewert:**  
eine Liste mit allen Gruppenmitgliedern
- `public List<Group> getAllUserGroups()`  
Gibt eine Liste mit allen Gruppen zurück, in denen eine Person Mitglied ist.  
**Rückgabewert:**  
Liste mit Gruppen
- `public void setAdmin()`  
Ernennst ein Gruppenmitglied als Gruppenadministrator.
- `public List<Group> getAllAdminGroups()`  
Gibt eine Liste mit allen Gruppen zurück, in denen der Benutzer Gruppenadministrator ist.

### 5.2.4 Class GroupMemberDAOImpl implements GroupMemberDAO

#### Beschreibung

Die Klasse implementiert das Interface GroupMemberDAO.

#### Konstruktoren

- `public GroupMemberDAOImpl(int userID, int groupID, boolean isAdmin)`  
**Parameter**  
`userID` ID des Gruppenmitgliedes  
`groupID` ID der Gruppe  
`isAdmin` `true`, wenn das Gruppenmitglied Gruppenadministrator ist, sonst `false`

### 5.2.5 Interface MeetingDAO

#### Beschreibung

Dieses Interface stellt ein Treffen einer Gruppe dar.

#### Methoden

- `public void addMeeting()`  
Erstellt einen neuen Termin für die Gruppe.
- `public List<Meeting> getAllMeetings()`  
Gibt eine Liste aller Termine aus.  
**Rückgabewert:**  
Liste mit Terminen
- `public void updateMeeting()`  
Aktualisiert die Daten eines Termins.
- `public void deleteMeeting()`  
Löscht einen Termin.
- `public Meeting getMeetingByID()`  
Gibt den Termin anhand seiner ID zurück.  
**Rückgabewert:**  
Termin anhand seiner ID

### 5.2.6 Class MeetingDAOImpl implements MeetingDAO

#### Beschreibung

Diese Klasse implementiert das Interface MeetingDAO.

#### Konstruktoren

- `public MeetingDAOImpl(int meetingID, int name, GPS place, long timestamp, int duration, String type, int creatorID)`  
**Parameter**

<code>int meetingID</code>	ID des Meetings
<code>int name</code>	Name des Termins
<code>GPS place</code>	GPS-Koordinaten des Veranstaltungsortes
<code>long timestamp</code>	Uhrzeit des Veranstaltungsbeginns
<code>int duration</code>	voraussichtliche Dauer des Termins
<code>String type</code>	“Veranstaltung” (bei festem Veranstaltungsort) oder “Tour” (bei sich veränderndem Veranstaltungsort)
<code>int creatorID</code>	ID des Gruppenmitgliedes, das den Termin erstellt hat

### 5.2.7 Interface ParticipantDAO

#### Beschreibung

Dieses Interface stellt einen Teilnehmer eines Termins dar.

#### Methoden

- `public void addParticipant()`  
Fügt ein Gruppenmitglied in die Liste der teilnehmenden Personen hinzu.
- `public void deleteParticipant()`  
Löscht Teilnehmer aus der Liste der teilnehmenden Personen.
- `public List<Participant> getAllParticipants()`  
Gibt eine Liste aller Teilnehmer zurück.  
**Rückgabewert:**  
Liste aller Teilnehmer
- `public Participant getParticipantByID()`  
Gibt einen Teilnehmer mit einer bestimmten Teilnehmer-ID zurück.  
**Rückgabewert:**  
Teilnehmer anhand seiner Teilnehmer-ID
- `public void updateParticipant()`  
Aktualisiert die Teilnehmer des Termins.

### 5.2.8 Class ParticipantDAOImpl implements ParticipantDAO

#### Beschreibung

Diese Klasse implementiert das Interface ParticipantDAO.

#### Konstruktoren

- `public ParticipantDAOImpl(int participantID, int userID, int meetingID, MeetingConfirmation status)`  
**Parameter**

<code>int participantID</code>	ID des Teilnehmers
<code>int userID</code>	Benutzer-ID des teilnehmenden Benutzers
<code>int meetingID</code>	ID des Termins
<code>MeetingConfirmation status</code>	Teilnahme oder Absage zu einem Termin

### 5.2.9 Interface UserDAO

#### Beschreibung

Dieses Interface stellt einen Benutzer der Go-App dar.

## Methoden

- `public void addUser()`  
Erstellt einen neuen Benutzer der Go-App.
- `public void deleteUser()`  
Löscht einen Benutzer aus der Go-App.
- `public void updateUser()`  
Aktualisiert einen Benutzer
- `public List<User> getAllUsers()`  
Gibt eine Liste aller Benutzer der Go-App zurück.  
**Rückgabewert:**  
Liste aller Benutzer der Go-App
- `public User getUserByID()`  
Gibt einen Benutzer anhand seiner ID zurück.  
**Rückgabewert:**  
Benutzer anhand seiner ID
- `public String GoogleID()`  
Gibt die Google-ID eines Benutzers zurück.  
**Rückgabewert:**  
String mit Google-ID
- `public boolean getNotificationStatus()`  
Gibt an, ob der Benutzer Benachrichtigungen über Termine erhalten möchte.  
**Rückgabewert:**  
`true`, wenn der Benutzer Benachrichtigungen erhalten möchte, sonst `false`
- `public User getUserByGoogleID()`  
Gibt Benutzer anhand seiner Google-ID zurück.  
**Rückgabewert:**  
Benutzer anhand seiner Google-ID

### 5.2.10 Class UserDAOImpl implements UserDAO

#### Beschreibung

Diese Klasse implementiert das Interface UserDAO.

#### Konstruktoren

- `public UserDAOImpl(int userID, String name, String googleID, boolean notificationStatus)`

<b>Parameter</b>	
int userID	ID des Benutzers
String name	Benutzername
String googleID	Google-ID des Benutzers
boolean notificationStatus	true, falls Benutzer Benachrichtigungen erhalten möchte, sonst false

### 5.2.11 Interface GPS.DAO

#### Beschreibung

Dieses Interface stellt den Standort eines Benutzers mithilfe von GPS-Koordinaten dar.

#### Methoden

- `public void userSetGPS()`  
Setzt den Standort eines Benutzers
- `public GPS userGetGPS()`  
Gibt den Standort eines Benutzers.  
**Rückgabewert:**  
GPS-Daten des Benutzers

### 5.2.12 Class GPS.DAO\_Impl implements GPS.DAO

#### Beschreibung

Diese Klasse implementiert das Interface GPS.DAO.

#### Konstruktoren

- `public GPS.DAO_Impl(int userID, double x, double y, double z)`  
**Parameter**  
int userID ID des Benutzer, dessen GPS-Daten dargestellt werden  
double x GPS x-Koordinate  
double y GPS y-Koordinate  
double z GPS z-Koordinate

### 5.2.13 Interface NotificationDAO

#### Beschreibung

Dieses Interface stellt die Benachrichtigungen für einen Benutzer dar.

#### Methoden

- `public List<Notification> userGetNotification()`  
Gibt alle Benachrichtigungen eines Benutzers zurück.

**Rückgabewert:**

Liste mit Benachrichtigungen

**5.2.14 Class NotificationDAOImpl implements NotificationDAO****Beschreibung**

Diese Klasse implementiert das Interface NotificationDAO.

**Konstruktoren**

- public NotificationDAOImpl(int userID)

**Parameter**

int userID ID des Benutzers, dessen Benachrichtigungen zurückgegeben werden sollen

**5.3 Package kit.edu.pse.goapp.server.converter****5.3.1 Interface Converter<T>****Beschreibung**

Das Interface Converter<T> stellt die Konverter von Parametern zu Objekten und umgekehrt dar.

**Methoden**

- public T parse(String jsonString)

Parst den Json-String

**Parameter:**

String jsonString Zu überprüfender String

**Rückgabewert:**

Json-String als T

- public String serialize(T object)

Erstellt einen String aus dem eingegebenen Objekt im Format T

**Parameter:**

T object Objekt im Format T

**Rückgabewert:**

object als String

- public String serialize(List <T> objects)

Erstellt einen String aus List<T>

**Parameter:**

List<T> objects Liste von Objekten

**Rückgabewert:**

List<T> als String

### 5.3.2 Class NotificationConverter implements Converter<T>

#### Beschreibung

Die Klasse `NotificationConverter` implementiert das Interface `Converter<T>` und konvertiert Benachrichtigungen von Parametern in Objekte und umgekehrt.

#### Konstruktoren

- `public NotificationConverter()`  
Default-Konstruktor

#### Methoden

Die Klasse `NotificationConverter` implementiert und überschreibt die Methoden `public T parse(String jsonString)`, `public String serialize(T object)` und `public String serialize(List<T> objects)`.

- `public Notification parse(String jsonString)`  
Parst einen Json-String  
**Parameter:**  
`String jsonString` Zu überprüfender String der Benachrichtigung  
**Rückgabewert:**  
Json-String des Benachrichtigung als `Notification`
- `public String serialize(Notification notification)`  
Erstellt einen String aus der Benachrichtigung  
**Parameter:**  
`Notification notification` die Benachrichtigung  
**Rückgabewert:**  
Benachrichtigung als String
- `public String serialize(List <Notification> notifications)`  
Erstellt einen String aus List<Notification>  
**Parameter:**  
`List<Notification> notifications` Liste von Benachrichtigungen  
**Rückgabewert:**  
List<Notification> als String

### 5.3.3 Class GroupMemberConverter implements Converter<T>

#### Beschreibung

Die Klasse `GroupMemberConverter` implementiert das Interface `Converter<T>` und konvertiert die Gruppenmitglieder von Parametern in Objekte und umgekehrt.

## Konstruktoren

- `public GroupMemberConverter()`  
Default-Konstruktor

## Methode

Die Klasse `GroupMemberConverter` implementiert und überschreibt die Methoden `public T parse(String jsonString)`, `public String serialize(T object)` und `public String serialize(List<T> objects)`.

- `public User parse(String jsonString)`  
Parst einen Json-String  
**Parameter:**  
`String jsonString` Zu überprüfender String der Gruppe  
**Rückgabewert:**  
Json-String des Gruppenmitglieds als User
- `public String serialize(Group group)`  
Erstellt einen String aus der Gruppe  
**Parameter:**  
`Group group` die Gruppe  
**Rückgabewert:**  
Gruppe als String
- `public String serialize(List <Group> groups)`  
Erstellt einen String aus List<Group>  
**Parameter:**  
`List<Group> group` Liste von Gruppen  
**Rückgabewert:**  
List<Group> als String
- `public String serialize(List <User> users)`  
Erstellt einen String aus List<User>  
**Parameter:**  
`List<User> users` Liste von Benutzern  
**Rückgabewert:**  
List<User> als String

### 5.3.4 Class `GroupConverter` implements `Converter<T>`

## Beschreibung

Die Klasse `GroupConverter` implementiert das Interface `Converter<T>` und konvertiert eine Gruppe von Parametern in Objekte und umgekehrt.

## Konstruktoren

- `public GroupConverter()`  
Default-Konstruktor

## Methoden

Die Klasse `GroupConverter` implementiert und überschreibt die Methoden `public T parse(String jsonString)`, `public String serialize(T object)` und `public String serialize(List<T> objects)`.

- `public Group parse(String jsonString)`  
Parst einen Json-String  
**Parameter:**  
`String jsonString` Zu überprüfender String der Gruppe  
**Rückgabewert:**  
Json-String der Gruppe als Group
- `public String serialize(Group group)`  
Erstellt einen String aus der Gruppe  
**Parameter:**  
`Group group` die Gruppe  
**Rückgabewert:**  
Gruppe als String
- `public String serialize(List <Group> groups)`  
Erstellt einen String aus List<Group>  
**Parameter:**  
`List<Group> groups` Liste von Gruppen  
**Rückgabewert:**  
List<Group> als String

### 5.3.5 Class GPS\_Converter implements Converter<T>

## Beschreibung

Die Klasse `GPS_Converter` implementiert das Interface `Converter<T>` und konvertiert die GPS-Daten von Parametern in Objekte und umgekehrt.

## Konstruktoren

- `public GPS_Converter()`  
Default-Konstruktor

## Methoden

### Methoden

Die Klasse `GPS_Converter` implementiert und überschreibt die Methoden `public T parse(String jsonString)`, `public String serialize(T object)` und `public String serialize(List<T> objects)`.

- `public GPS parse(String jsonString)`

Parst einen Json-String

**Parameter:**

`String jsonString` Zu überprüfender String des GPS

**Rückgabewert:**

Json-String des GPS als GPS

- `public String serialize(GPS gps)`

Erstellt einen String aus dem GPS

**Parameter:**

`GPS gps` das GPS

**Rückgabewert:**

GPS als String

- `public String serialize(List <GPS> gps)`

Erstellt einen String aus List<GPS>

**Parameter:**

`List<GPS> gps` Liste von GPS-Daten

**Rückgabewert:**

List<GPS> als String

### 5.3.6 Class UserConverter implements Converter<T>

## Beschreibung

Die Klasse `UserConverter` implementiert das Interface `Converter<T>` und konvertiert die Benutzer von Parametern in Objekte und umgekehrt.

## Konstruktoren

- `public UserConverter()`  
Default-Konstruktor

### Methoden

Die Klasse `UserConverter` implementiert und überschreibt die Methoden `public T parse(String jsonString)`, `public String serialize(T object)` und `public String serialize(List<T> objects)`.

- `public User parse(String jsonString)`  
Parst einen Json-String

**Parameter:**

String jsonString Zu überprüfender String des Benutzers

**Rückgabewert:**

Json-String des Benutzers als User

- `public String serialize(User user)`  
Erstellt einen String aus dem Benutzer

**Parameter:**

User user der Benutzer

**Rückgabewert:**

Benutzer als String

- `public String serialize(List <User> users)`  
Erstellt einen String aus List<User>

**Parameter:**

List<User> users Liste von Benutzern

**Rückgabewert:**

List<User> als String

### 5.3.7 Class ParticipantConverter implements Converter<T>

#### Beschreibung

Die Klasse ParticipantConverter implementiert das Interface Converter<T> und konvertiert einen Teilnehmer von Parametern in Objekte und umgekehrt.

#### Konstruktoren

- `public ParticipantConverter()`  
Default-Konstruktor

#### Methoden

Die Klasse ParticipantConverter implementiert und überschreibt die Methoden `public T parse(String jsonString)`, `public String serialize(T object)` und `public String serialize(List<T> objects)`.

- `public Participant parse(String jsonString)`  
Parst einen Json-String  
**Parameter:**  
String jsonString Zu überprüfender String des Participant  
**Rückgabewert:**  
Json-String des Participant als Participant
- `public String serialize(Participant participant)`  
Erstellt einen String aus demParticipant  
**Parameter:**  
Participant participant der Teilnehmer

**Rückgabewert:**

Participant als String

- `public String serialize(List <Participant> participants)`  
Erstellt einen String aus List<Participant>

**Parameter:**

`List<Participant> participant` Liste von Participants

**Rückgabewert:**

List<Participant> als String

### 5.3.8 Class MeetingConverter implements Converter<T>

#### Beschreibung

Die Klasse MeetingConverter implementiert das Interface Converter<T> und konvertiert einen Termin von Parametern in Objekte und umgekehrt.

#### Konstruktoren

- `public MeetingConverter()`  
Default-Konstruktor

#### Methoden

Die Klasse MeetingConverter implementiert und überschreibt die Methoden `public T parse(String jsonString)`, `public String serialize(T object)` und `public String serialize(List<T> objects)`.

- `public Meeting parse(String jsonString)`  
Parst einen Json-String  
**Parameter:**  
`String jsonString` Zu überprüfender String des Treffens  
**Rückgabewert:**  
Json-String des Treffens als Meeting
- `public String serialize(Meeting meeting)`  
Erstellt einen String aus dem Meeting  
**Parameter:**  
`Meeting meeting` der Termin  
**Rückgabewert:**  
Meeting als String
- `public String serialize(List <Meeting> meetings)`  
Erstellt einen String aus List<Meeting>  
**Parameter:**  
`List<Meeting> meetings` Liste von Terminen  
**Rückgabewert:**  
List<Meeting> als String

## 5.4 Package kit.edu.pse.goapp.server.database

### 5.4.1 Class Group

#### Beschreibung

Die Klasse Group stellt eine Gruppe aus Benutzern in der Go-App dar.

#### Primärschlüssel

Primärschlüssel	Datentyp	Beschreibung
Group_ID	INT	ID, unter der die Gruppe auf dem Server gespeichert ist

#### Attribute

Attribut	Datentyp	Beschreibung
Name	CHAR(50)	Gruppenname

### 5.4.2 Class GroupMembers

#### Beschreibung

Die Klasse GroupMembers stellt die Mitglieder einer Gruppe dar.

#### Attribute

Attribut	Datentyp	Beschreibung
Group_ID	INT	ID der Gruppe
User_ID	INT	ID des Benutzer
isAdmin	UNSIGNED TINYINT(1)	Gibt an, ob das Gruppenmitglied Gruppenadministrator ist

### 5.4.3 Class User

#### Beschreibung

Die Klasse User stellt einen Benutzer der Go-App dar.

#### Primärschlüssel

Primärschlüssel	Datentyp	Beschreibung
User_ID	INT	ID, unter der der User auf dem Server gespeichert ist

#### Attribute

Attribut	Datentyp	Beschreibung
Name	CHAR(50)	Benutzername
Google_ID	INT	Google-ID des Benutzers
NotificationEnabled	UNSIGNED TINYINT(1)	Gibt an, ob der Benutzer Benachrichtigungen aktiviert hat

#### 5.4.4 Class Meeting

##### Beschreibung

Die Klasse Meeting stellt einen Termin einer Gruppe dar

##### Primärschlüssel

Primärschlüssel	Datentyp	Beschreibung
Meeting_ID	INT	ID, unter der das Meeting auf dem Server gespeichert ist

##### Attribute

Attribut	Datentyp	Beschreibung
Name	CHAR(50)	Name des Termins
Place_X	INT	GPS x-Koordinate des Veranstaltungsortes
Place_Y	INT	GPS y-Koordinate des Veranstaltungsortes
Place_Z	INT	GPS z-Koordinate des Veranstaltungsortes
Timestamp	DATETIME	Datum und Uhrzeit des Termins
Duration	INT	Dauer des Termins
Type	ENUM	Art des Termins (Veranstaltung mit festem Ort oder Tour mit variablem Ort)
Creator_ID	INT	ID des Terminerstellers

#### 5.4.5 Class Participant

##### Beschreibung

Die Klasse Participant stellt einen Teilnehmer an einem Termin dar.

##### Primärschlüssel

Primärschlüssel	Datentyp	Beschreibung
Participant_ID	INT	ID, unter der der Teilnehmer auf dem Server gespeichert ist

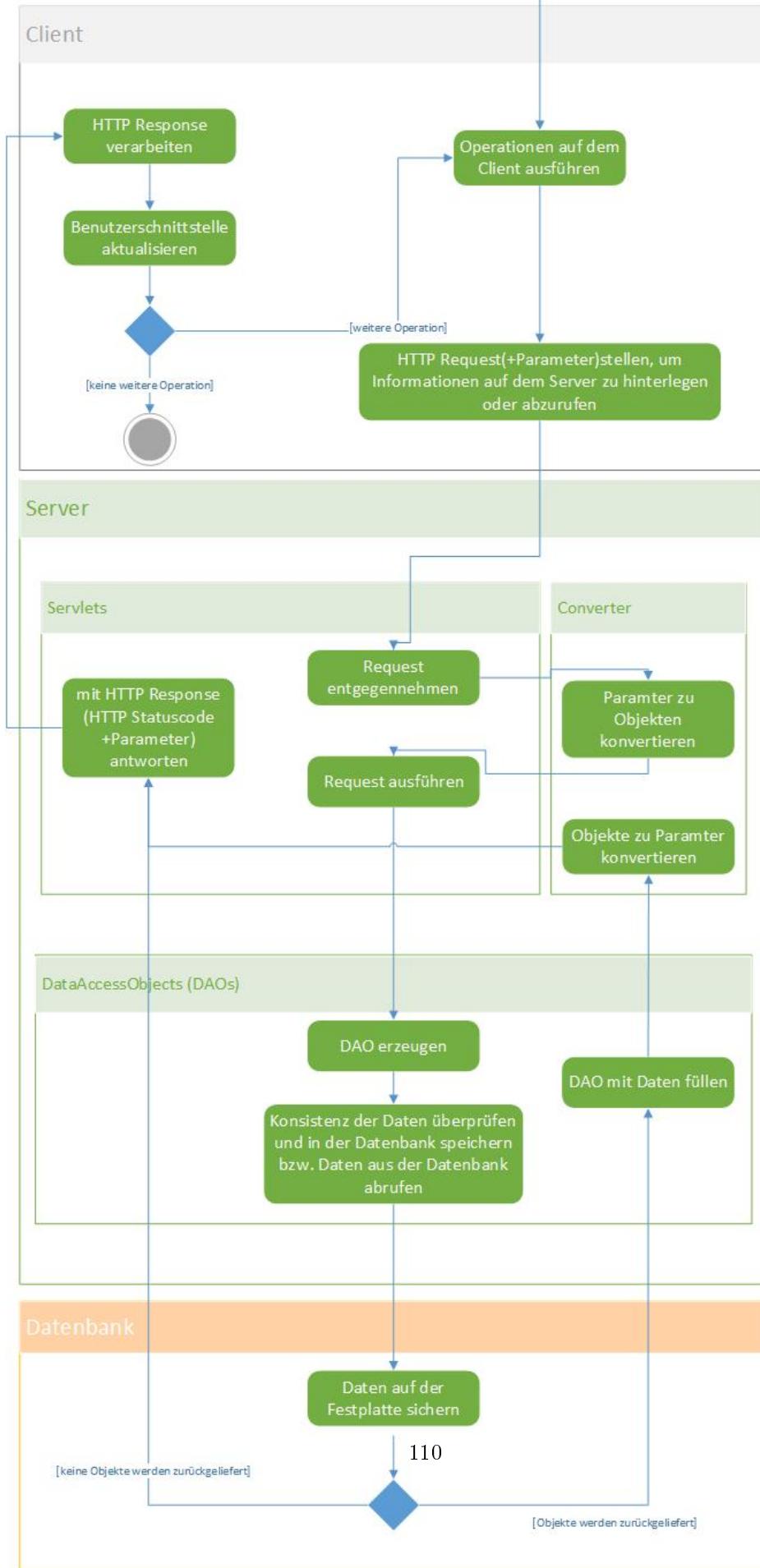
##### Attribute

Attribut	Datentyp	Beschreibung
User_ID	INT	Benutzer-ID des Teilnehmers
Meeting_ID	INT	Termin-ID des Termins, an dem der Teilnehmer teilnimmt
Confirmation	ENUM	Art des Termins (Veranstaltung mit festem Ort oder Tour mit variablem Ort)

## 6 Beschreibung charakteristischer Abläufe

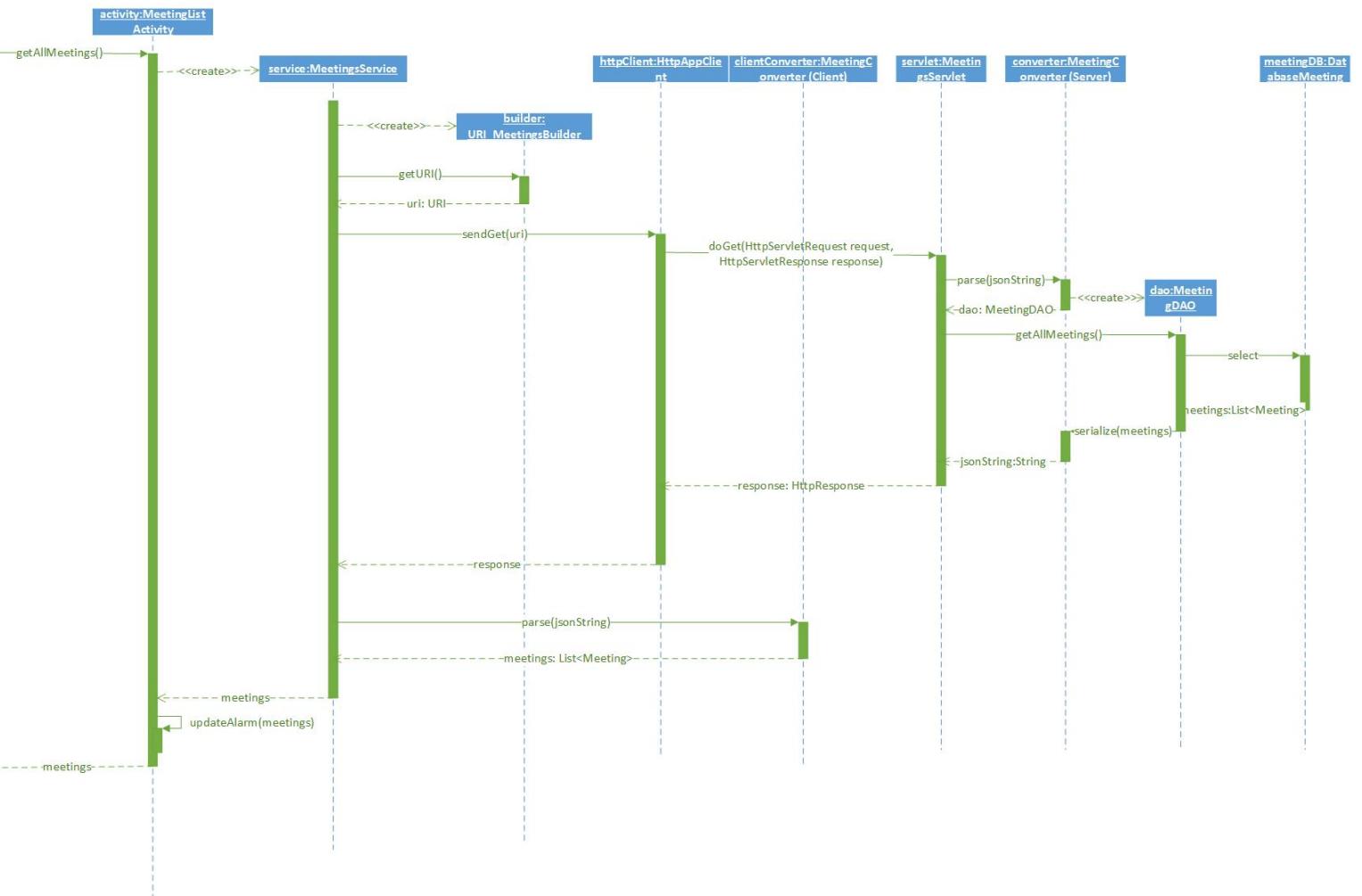
### 6.1 Allgemeiner Ablauf eines Requests des Clients

Im Folgenden wird der Ablauf eines Requests mithilfe eines Aktivitätsdiagramms beschrieben. Dies kann man auch auf dem Aktivitätsdiagramm nachvollziehen. Zunächst wird eine Operation auf dem Client ausgeführt, diese braucht Daten vom Server oder will sie dort ablegen, also wird ein HTTP Request mit den entsprechenden Parametern gestellt. Dieser wird von den Servlets im Server entgegengenommen und an den Converter übergeben, der die gegebenen Parameter zu einem Objekt konvertiert. Nun wird der Request ausgeführt. Dies erzeugt ein DataAccesObject. Nun wird entweder die Konsistenz der Daten überprüft und in der Datenbank gespeichert oder die gefragten Daten werden dort abgerufen. Die Datenbank sichert diese auf der Festplatte. Sollten Objekte zurückgeliefert werden, wird das entsprechende DAO mit Daten gefüllt und an den Converter übergeben, der dieses wieder zu Parametern konvertiert. Dann antworten die Servlets mit einem HTTP Response. Sollte kein Objekt zurückgeliefert werden, geschieht dies sofort. Der HTTP Response besitzt einen HTTP Statuscode und die entsprechenden Parameter. Dieser wird zum Client geschickt. Dort wird der HTTP Response verarbeitet und die Benutzerschnittstelle aktualisiert. Werden weitere Operationen auf dem Client ausgeführt, startet der Kreis wieder von vorne, ist dies nicht der Fall sind wir fertig.



## 6.2 Ablauf der Methode getAllMeetings()

Die Methode getAllMeetings() listet alle Termine eines Benutzers auf. Diese Methode wird in der Klasse MeetingListActivity des Clients ausgeführt. Als erstes erstellt die MeetingListActivity den zugehörigen MeetingsService, der im Hintergrund weiterläuft, damit die MeetingListActivity nicht vollständig blockiert wird. Der MeetingsService erstellt einen zugehörigen URI\_MeetingBuilder builder, von dem man eine Instanz URI uri zurückgegeben bekommt, wenn man auf ihm die Methode getURI() aufruft. uri stellt eine URI dar, welche die URL des MeetingsServlets angibt und außerdem die Parameter, die der Server bekommen soll, in sich gespeichert hat. Bei der Methode getAllMeetings() werden jedoch keine Parameter übergeben. Nun schickt der Service einen HTTP GET Request mithilfe einer Instanz von HttpAppClient mit der Methode sendGet(uri) an den Server, genauer gesagt an das zugehörige MeetingsServlet. Im MeetingsServlet wird die doGet(HttpServletRequest request, HttpServletResponse response) aufgerufen, die den Request entgegennimmt und an den zugehörigen MeetingConverter übergibt, der den JSON-String, der sich im HTTP GET Request befindet, parst und eine Instanz dao des zugehörigen MeetingDAOs zurückgibt. Das DAO agiert als Schnittstelle zwischen dem Server und der Datenbank. Auf der Instanz des MeetingDAOs wird die Methode getAllMeetings() ausgeführt und daraufhin wird durch einen select Befehl die Termine des Benutzers abgerufen und als List<Meeting> meetings zurückgegeben. Nun müssen die Objekte meetings wieder in einen JSON-String verwandelt werden, um sie dann als HTTP Response wieder an den Client zu verschicken. Dazu werden meetings an den MeetingsConverter übergeben und serialisiert. Der zurückgegebene JSON-String jsonString wird als HTTP Response zusammen mit einem Statuscode, der angibt, ob alles fehlerfrei verlaufen ist, mithilfe des HttpAppClients an den MeetingsService zurückgeschickt. Der jsonString wird wiederum mit einem clientseitigen MeetingConverter zu einer List<Meeting> meetings geparsst, die dann an die MeetingListActivity zurückgegeben wird. Die MeetingListActivity aktualisiert und löscht alle Alarne, die nicht mehr gültig sind, und erstellt die neuen.



### 6.3 Ablauf der Methode deleteGroup(int groupId)

Die Methode deleteGroup(int groupId) löscht eine Gruppe von Benutzern in der App. Diese Methode wird in der GroupActivity des Clients ausgeführt.

Als erstes erstellt die GroupActivity den zugehörigen GroupService, der im Hintergrund weiterläuft, damit die GroupActivity nicht vollständig blockiert wird. Der GroupService erstellt einen zugehörigen URI\_GroupBuilder, von dem man eine Instanz uri zurückgegeben bekommt, wenn man auf ihm die Methode getUri() aufruft, davor wird noch über addParameter("groupId", groupId.toString()) der gegebene Parameter hinzugefügt. Der URI\_GroupBuilder stellt eine URI dar, welche die URL des GroupServlets angibt und außerdem die Parameter, die der Server bekommen soll, in sich gespeichert hat, der wie oben beschrieben hinzugefügt wurde.

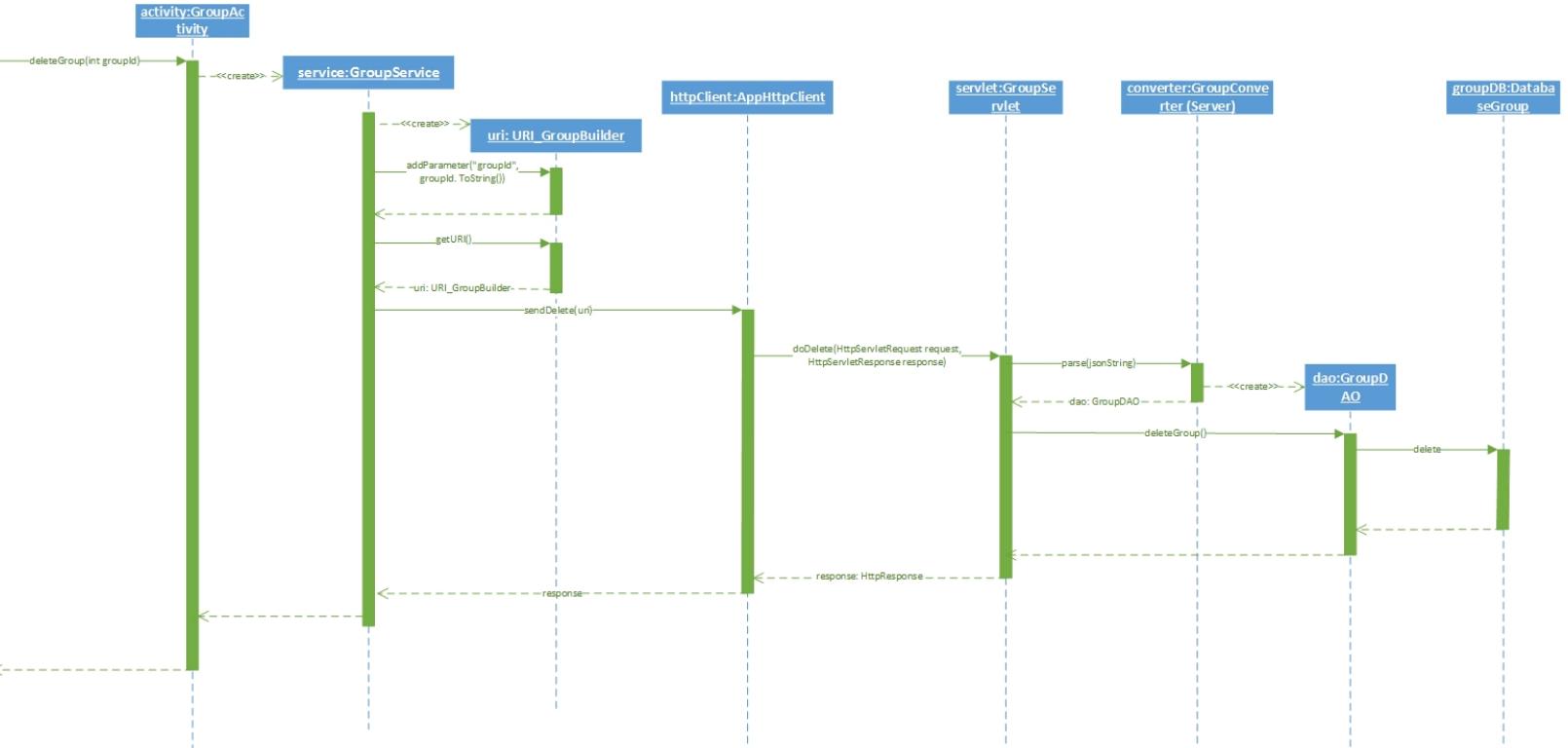
Nun schickt der Service einen HTTP DELETE Request mithilfe einer Instanz von HttpAppClient mit der Methode sendDelete(uri) an den Server, genauer gesagt an das zugehörige GroupServlet.

Im GroupServlet wird die Methode doDelete(HttpServletRequest request, HttpServletResponse response) aufgerufen, die den Request entgegen nimmt und ihn an den zugehörigen GroupConverter übergibt, die den JSON String, der sich im HTTP DELETE Request befindet, parst und eine Instanz dao des zugehörigen GroupDAO erstellt und zurückgibt. Das DAO agiert als Schnittstelle zwischen dem Server und der Datenbank.

Auf der Instanz des GroupDAOs wird die Methode deleteGroup() ausgeführt und daraufhin wird durch einen delete Befehl die Gruppe aus der Database-Group gelöscht.

Nun muss der Client noch erfahren, ob sein Request erfolgreich war, also gibt die Methode doPut(HttpServletRequest request, HttpServletResponse response), die auf dem UserServlet ausgeführt wurde, den HTTP Response zusammen mit einem Statuscode, der angibt, ob alles fehlerfrei verlaufen ist, mithilfe des AppHttpClients an den GroupManagementService zurück.

Nun muss der Client noch erfahren, ob sein Request erfolgreich war, also gibt die Methode doDelete(HttpServletRequest request, HttpServletResponse response), die auf dem GroupUserManagementServlet ausgeführt wurde, den HTTP Response zusammen mit einem Statuscode zurück, der angibt, ob alles fehlerfrei verlaufen ist, mithilfe des AppHttpClients an den GroupService zurück.



#### 6.4 Ablauf der Methode addAdmin(int groupId, int UserId, boolean adminStatus) bei Auftaft eines Errors

Die Methode addAdmin(int groupId, int UserId, boolean adminStatus) versucht einer Gruppe einen neuen Administrator hinzuzufügen und liefert einen 404-Error (not found) zurück, da dieser Benutzer nicht in der Gruppe gefunden wurde. Diese Methode wird in der GroupMemberActivity des Clients ausgeführt.

Als erstes erstellt die GroupMemberActivity den zugehörigen GroupManagementService, der im Hintergrund weiterläuft, damit die GroupMemberActivity nicht vollständig blockiert wird. Der GroupManagementService erstellt einen zugehörigen URI\_GroupUserManagementBuilder, von dem man eine Instanz uri zurückgegeben bekommt, wenn man auf ihm die Methode getUri() aufruft, davor wird noch über addParameter("groupId", groupId.toString()) der gegebene Parameter hinzugefügt. Der URI\_GroupUserManagementBuilder stellt eine URI dar, welche die URL des GroupUserManagementServlets angibt und außerdem die Parameter, die der Server bekommen soll, in sich gespeichert hat, der wie oben beschrieben hinzugefügt wurde.

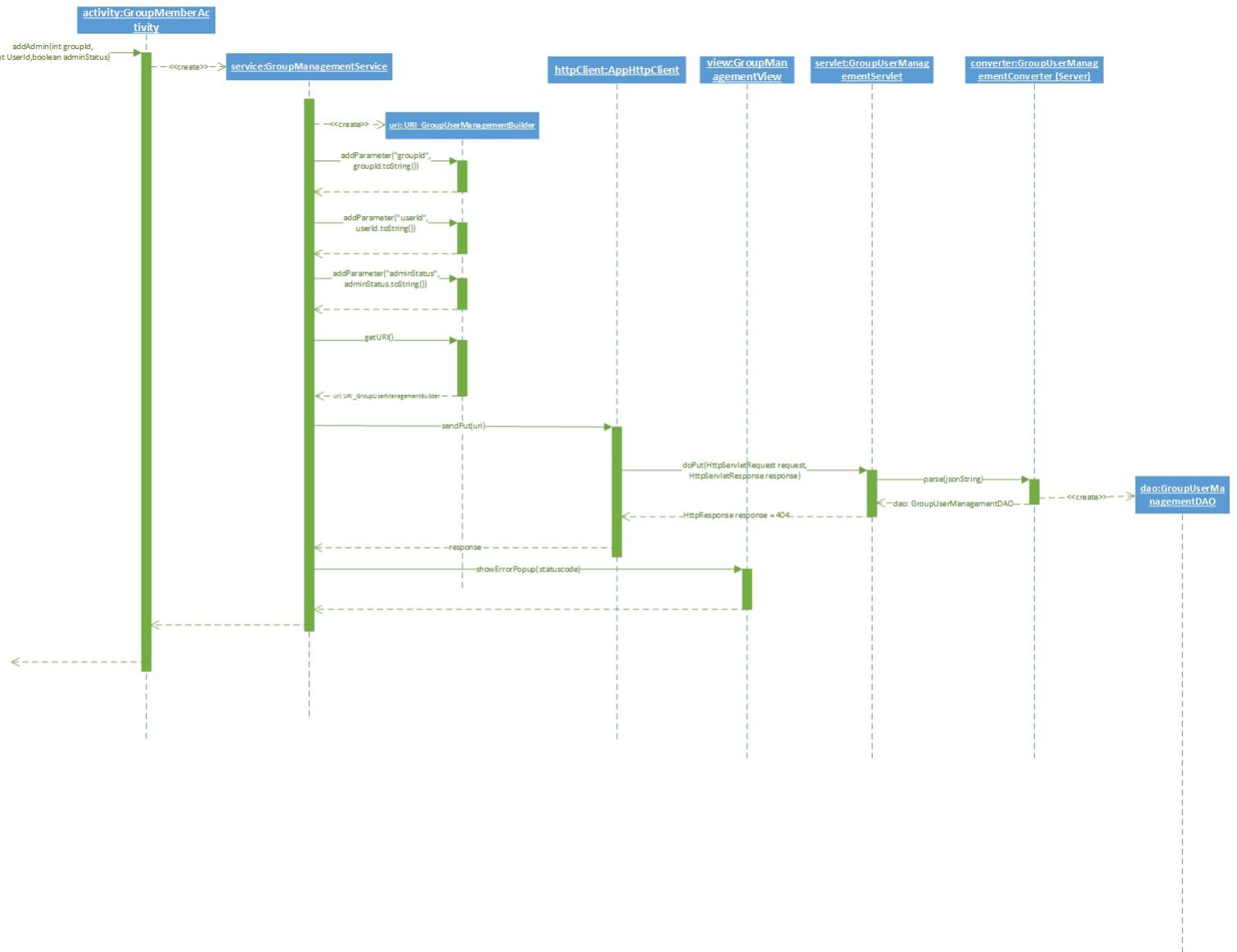
Nun schickt der Service einen HTTP PUT Request mithilfe einer Instanz von HttpAppClient mit der Methode sendPut(uri) an den Server, genauer gesagt an

das zugehörige GroupUserManagementServlet.

Im GroupUserManagementServlet wird die Methode doPut(HttpServletRequest request, HttpServletResponse response) aufgerufen, die den Request entgegen nimmt und ihn an den zugehörigen GroupUserManagementConverter übergibt, die den JSON-String, der sich im HTTP PUT Request befindet, parst und eine Instanz dao des zugehörigen GroupUserManagementDAO erstellt und zurückgibt. Das DAO agiert als Schnittstelle zwischen dem Server und der Datenbank. Nun muss der Client noch erfahren, ob sein Request erfolgreich war, also gibt die Methode doPut(HttpServletRequest request, HttpServletResponse response), die auf dem UserServlet ausgeführt wurde, den HTTP Response zusammen mit einem Statuscode, der angibt, ob alles fehlerfrei verlaufen ist, mithilfe des AppHttpClients an den GroupManagementService zurück.

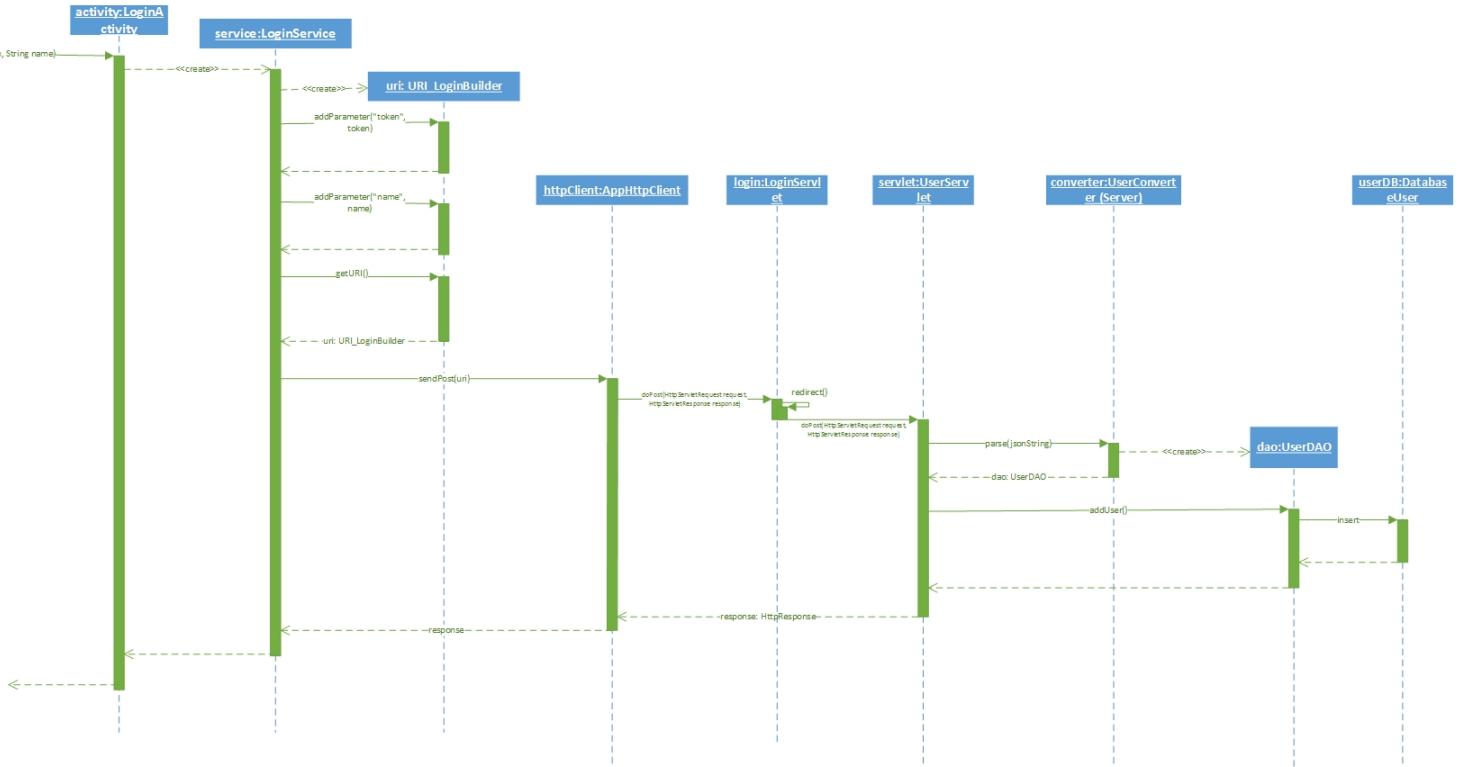
Da der Benutzer nicht in der Gruppe gefunden wurde, gibt die Methode doPut(HttpServletRequest request, HttpServletResponse response), die auf dem GroupUserManagementServlet ausgeführt wurde, den HTTP Response mit dem Statuscode 404 zurück, der angibt, dass die angegebene Ressource nicht gefunden wurde, mithilfe des AppHttpClients an den GroupManagementService zurück.

Schließlich sendet der GroupManagementService die Fehlermeldung mit showErrorPopup(statuscode) an die GroupManagementView, wo die Fehlermeldung ausgegeben wird.



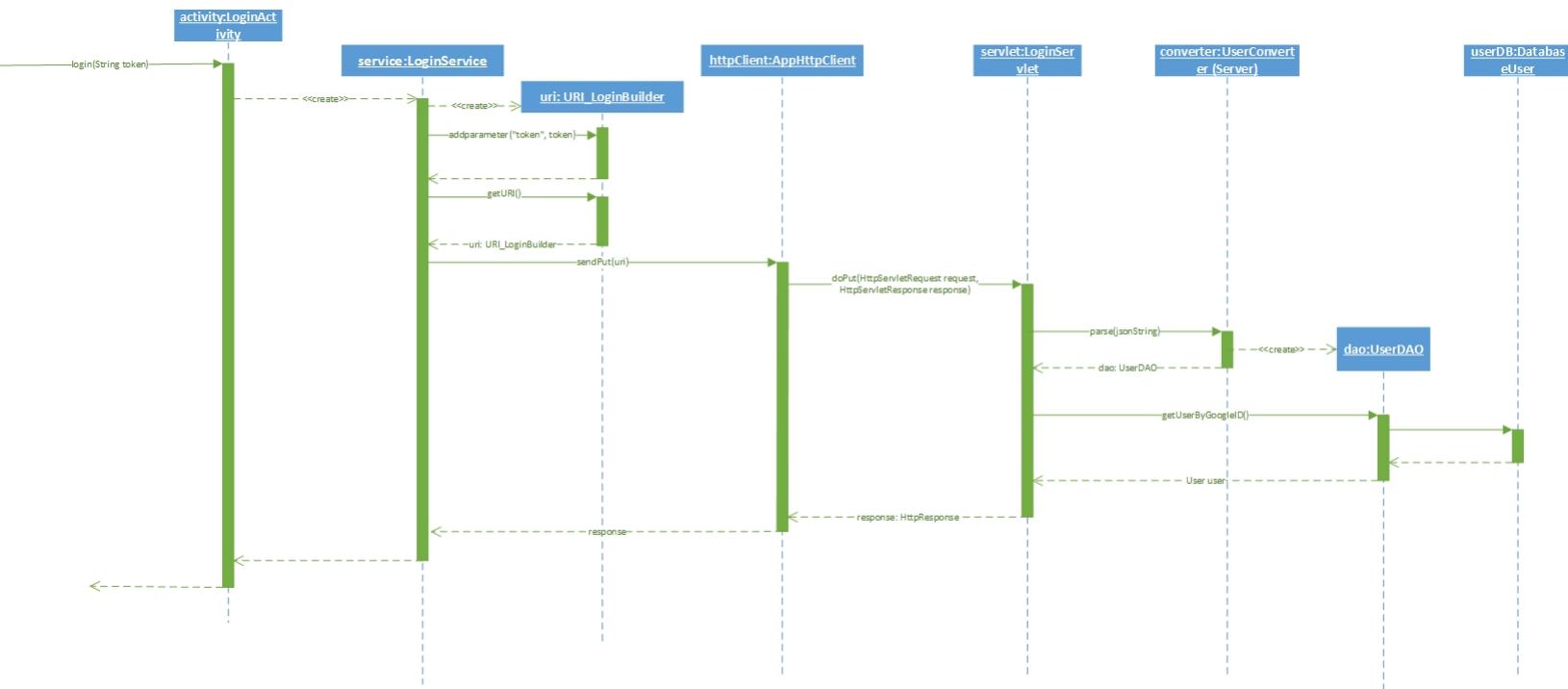
## 6.5 Ablauf der Registrierung in das System

Die Methode register(String token, String name) registriert einen Nutzer in System der App. Diese Methode wird in der LoginActivity des Clients ausgeführt. Als erstes erstellt die LoginActivity den zugehörigen LoginService, der im Hintergrund weiterläuft, damit die LoginActivity nicht vollständig blockiert wird. Der LoginService erstellt einen zugehörigen URI\_LoginBuilder builder, von dem man eine Instanz URI uri zurückgegeben bekommt, wenn man auf ihm die Methode getURI() aufruft, davor werden noch über addParameter("token", token) und addParameter("name", name) die gegebenen Parameter hinzugefügt. Uri stellt eine URI dar, welche die URL des LoginServlets angibt und außerdem die Parameter, die der Server bekommen soll, in sich gespeichert hat, die wie oben beschrieben hinzugefügt wurden. Nun schickt der Service einen HTTP POST Request mithilfe einer Instanz von HttpAppClient mit der Methode sendPost(uri) an den Server, genauer gesagt an das zugehörige LoginServlet. Im LoginServlet wird die doPost(HttpServletRequest request, HttpServletResponse response) aufgerufen, die den Request entgegen nimmt und an das UserServlet weiter reicht, dort wird noch einmal die Methode doPost(HttpServletRequest request, HttpServletResponse response) aufgerufen. Das UserServlet nimmt den Request entgegen und übergibt ihn an den zugehörigen UserConverter, der den JSON-String, der sich im HTTP POST Request befindet, parst und eine Instanz dao des zugehörigen UserDAOs erstellt und zurückgibt. Das DAO agiert als Schnittstelle zwischen dem Server und der Datenbank. Auf der Instanz des UserDAOs wird die Methode addUser() ausgeführt und daraufhin wird durch einen insert Befehl der Nutzer zu DatabaseUser hinzugefügt. Nun muss der Client noch erfahren, ob sein Request erfolgreich war, also gibt die Methode doPost(HttpServletRequest request, HttpServletResponse response), die auf dem UserServlet ausgeführt wurde, den HTTP Response zusammen mit einem Statuscode, der angibt, ob alles fehlerfrei verlaufen ist, mithilfe des HttpAppClients an den LoginService zurück.



## 6.6 Ablauf des Logins in das System

Die Methode `login(String token)` logt einen Nutzer in der App ein. Diese Methode wird in der LoginActivity des Clients ausgeführt. Als erstes erstellt die LoginActivity den zugehörigen LoginService, der im Hintergrund weiterläuft, damit die LoginActivity nicht vollständig blockiert wird. Der LoginService erstellt einen zugehörigen URI\_LoginBuilder builder, von dem man eine Instanz Uri zurückgegeben bekommt, wenn man auf ihm die Methode `getURI()` aufruft, davor wird noch über `addParameter("token", token)` der gegebene Parameter hinzugefügt. Uri stellt eine URI dar, welche die URL des LoginServlets angibt und außerdem die Parameter, die der Server bekommen soll, in sich gespeichert hat, der wie oben beschrieben hinzugefügt wurde. Nun schickt der Service einen HTTP PUT Request mithilfe einer Instanz von HttpAppClient mit der Methode `sendPut(uri)` an den Server, genauer gesagt an das zugehörige LoginServlet. Im LoginServlet wird die `doPost(HttpServletRequest request, HttpServletResponse response)` aufgerufen, die den Request entgegen nimmt und ihn an den zugehörigen UserConverter übergibt, der den JSON String, der sich im HTTP PUT Request befindet, parst und eine Instanz dao des zugehörigen UserDAOs erstellt und zurückgibt. Das DAO agiert als Schnittstelle zwischen dem Server und der Datenbank. Auf der Instanz des UserDAOs wird die Methode `getUserByGoogleID()` ausgeführt und daraufhin wird durch einen select Befehl der Nutzer aus DatabaseUser geholt. Dieser wird an das LoginServlet übergeben, also war der login erfolgreich. Nun muss der Client noch erfahren, ob sein Request erfolgreich war, also gibt die Methode `doPost(HttpServletRequest request, HttpServletResponse response)`, die auf dem UserServlet ausgeführt wurde, den HTTP Response zusammen mit einem Statuscode, der angibt, ob alles fehlerfrei verlaufen ist, mithilfe des HttpAppClients an den LoginService zurück.



## 6.7 GPS mithilfe des AlarmReceivers

Über awakeFromAndroidOS wird der AlarmReceiver aufgeweckt und zwar 30 Minuten vor Beginn des nächsten Termins, da dann damit begonnen werden soll, die GPS-Daten an den Server zu schicken.

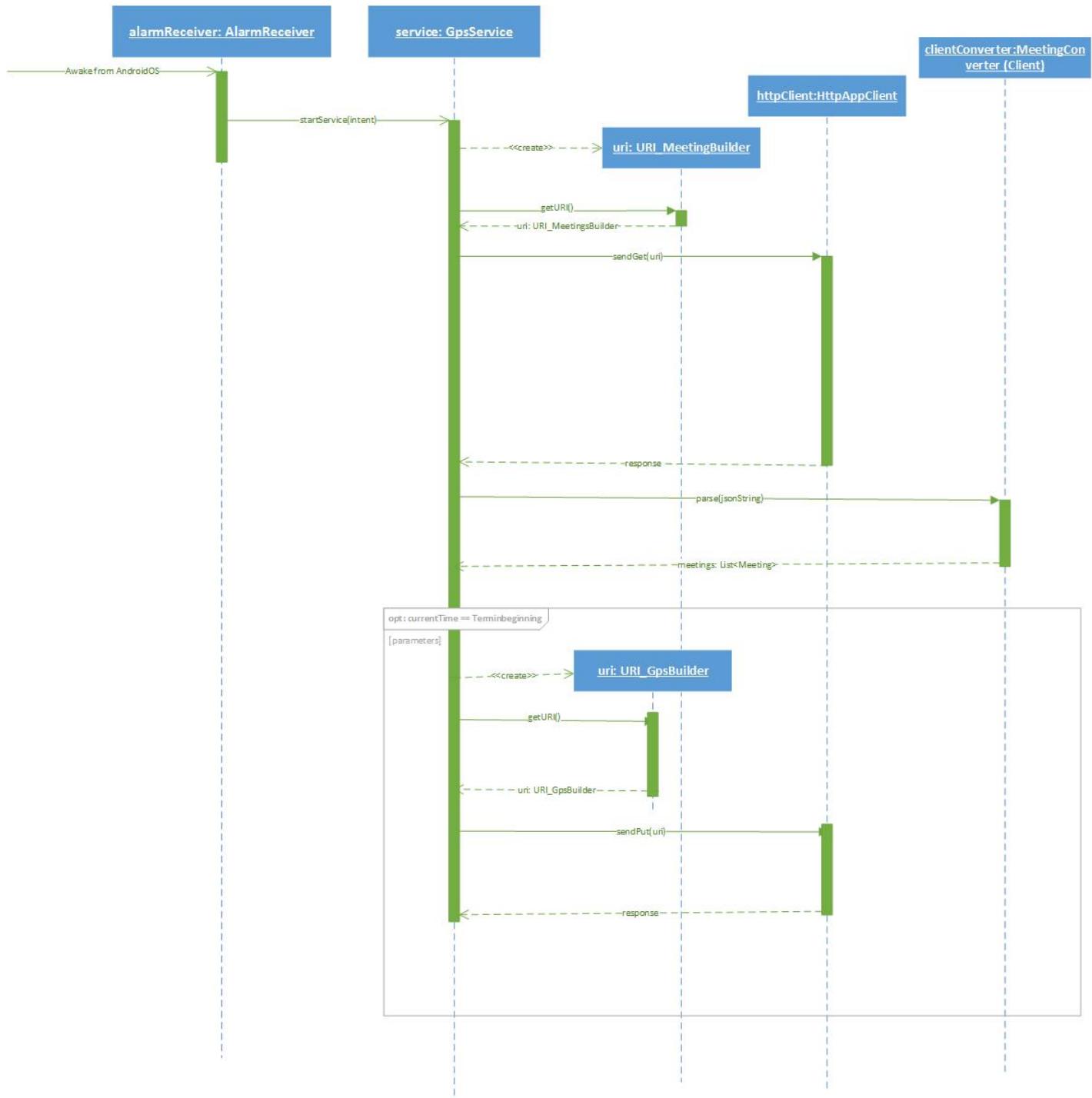
Der AlarmReceiver startet über startService(intent) einen GpsService und der Receiver wird anschließend zerstört.

Der Service erstellt eine Instanz uri von URI\_MeetingBuilder, dort wird über addParameter(meeting) das genaue Meeting spezifiziert, und erhält diese über getURI(). Dann sendet er diese an einen HttpAppClient mit der Methode sendGet(uri) weiter. Dieser macht dann eine HTTP GET Request beim Server.

Die Information, die der HttpAppClient zurückliefert, ist eine HTTP Response mit Statuscode und JSON-String, dieser wird über die Methode parse(jsonString) beim MeetingConverter in eine Liste von Meetings umgewandelt, die aber nur ein Meeting enthält.

Vom Meeting wird die Termininformationen gelesen und wenn TerminBeginnig mit CurrentTime übereinstimmt, wird eine Instanz vom URI\_GPS\_Builder erstellt, diesem der aktuelle Standort als Parameter mitgegeben und über getUri() die URI abgefragt.

Anschließend wird über einen Aufruf von sendPut(uri) beim HttpAppClient ein HTTP PUT Request an den Server geschickt. Der im Response enthaltene Statuscode, teilt uns mit, ob alles geklappt hat.



## 7 Änderungen gegenüber dem Pflichtenheft

### 7.1 Gruppeneinladungen

In den funktionalen Anforderungen /F70/ und /F110/ werden beschrieben, dass der Benutzer eine Benachrichtigung erhält, wenn er zu einer Gruppe eingeladen wurde (/F70/), und dass der Benutzer darüber entscheiden kann, ob er einer Gruppe beitreten möchte oder nicht (/F110/).

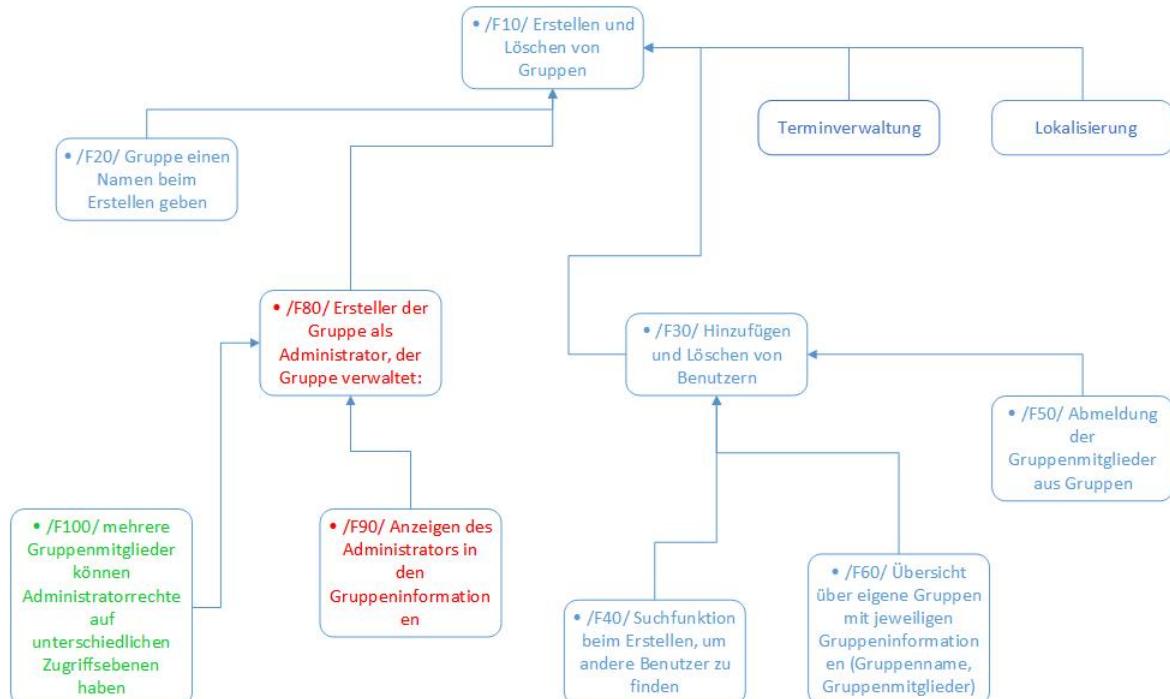
Diese funktionalen Anforderungen mit Priorität B (/F70/) und C (/F110/) werden verworfen, da es zu Problemen kommen kann, wenn eine Gruppe erstellt wird mit einem zeitnahen Termin und keiner von den eingeladenen Benutzern die Gruppeneinladung frühzeitig annimmt, sodass keiner Kenntnis von dem Termin nehmen wird und somit auch niemand zu dem Termin kommt.

### 7.2 Änderung der Prioritäten der funktionalen Anforderungen

Die funktionalen Anforderungen mit Priorität B wurden nochmals überarbeitet und dabei festgelegt, welche größere Priorität haben und damit ihre Priorität B behalten, und welche nicht so wichtig sind und damit zu Priorität C abgestuft werden.

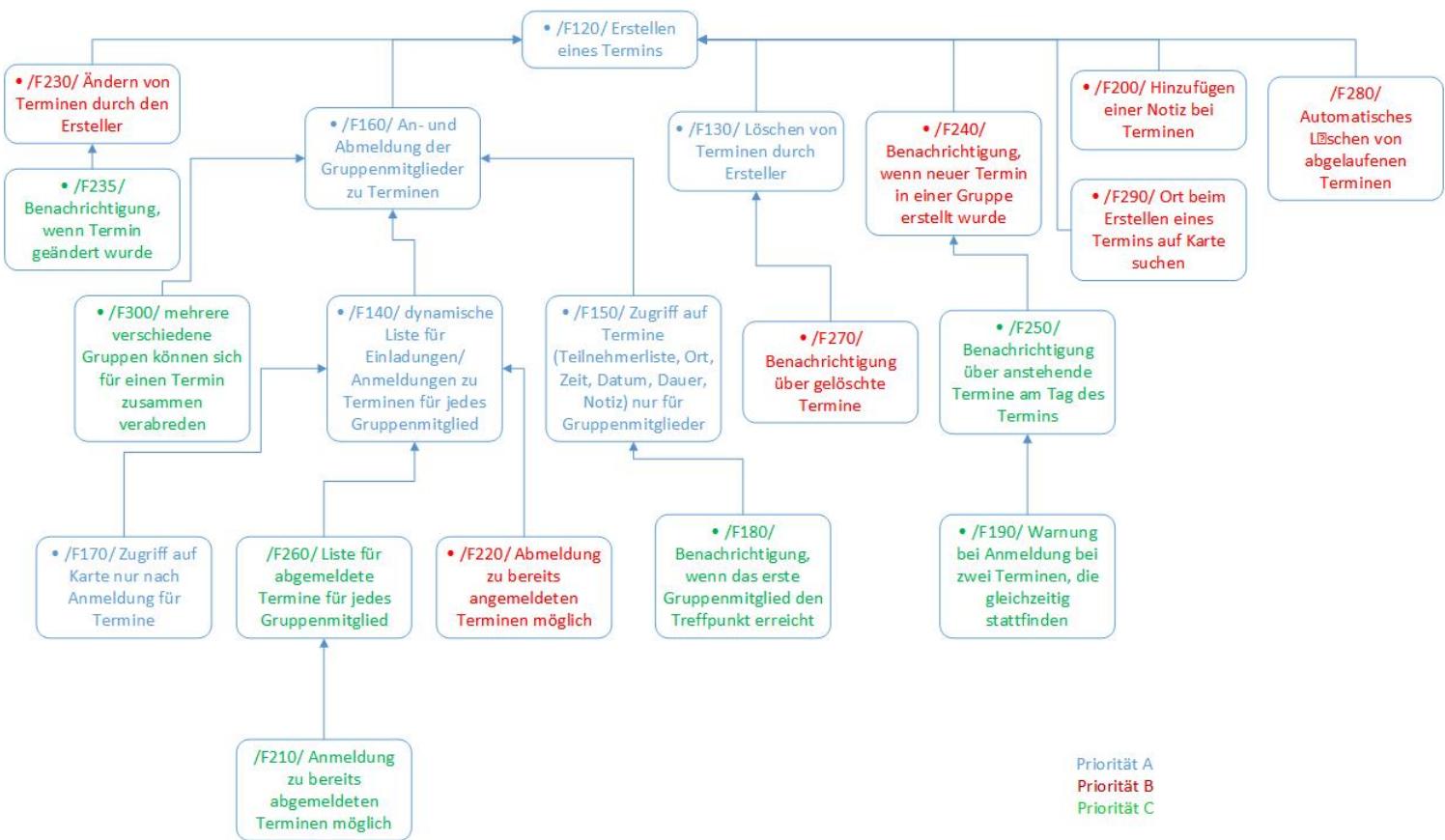
#### 7.2.1 Gruppenverwaltung

- /F70/ und /F110/ (siehe 4.1) wurden entfernt.



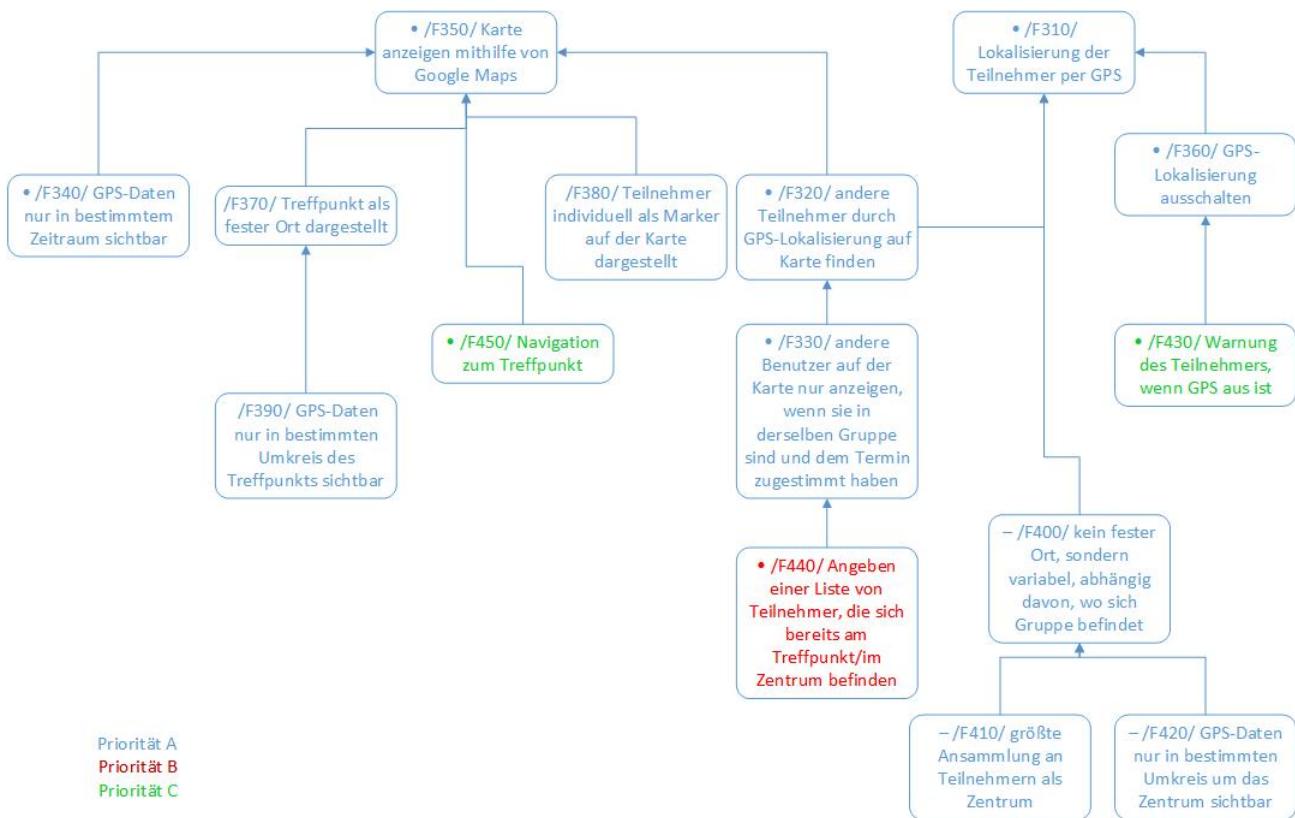
### 7.2.2 Terminverwaltung

- /F180/, /F250/ und /F235/ wurden zu Priorität C heruntergestuft, weil sie zwar nützliche Informationen für den Benutzer bereitstellen, aber für die Funktion der App eher unwichtig sind.
- /F260/ und /F210/ wurden heruntergestuft, weil der Benutzer seine Entscheidung, den Termin abzusagen, aus einem bestimmten Grund abgesagt hat, z.B. aus Zeitmangel, und dann die Wahrscheinlichkeit eher gering ist, dass sich der Benutzer wieder zu einem bereits abgemeldeten Termin anmeldet.
- /F190/ wurde heruntergestuft, weil der Benutzer selbst herausfinden kann, ob sich zwei Termine überschneiden und daher die Anforderung lediglich der Bequemlichkeit dient.



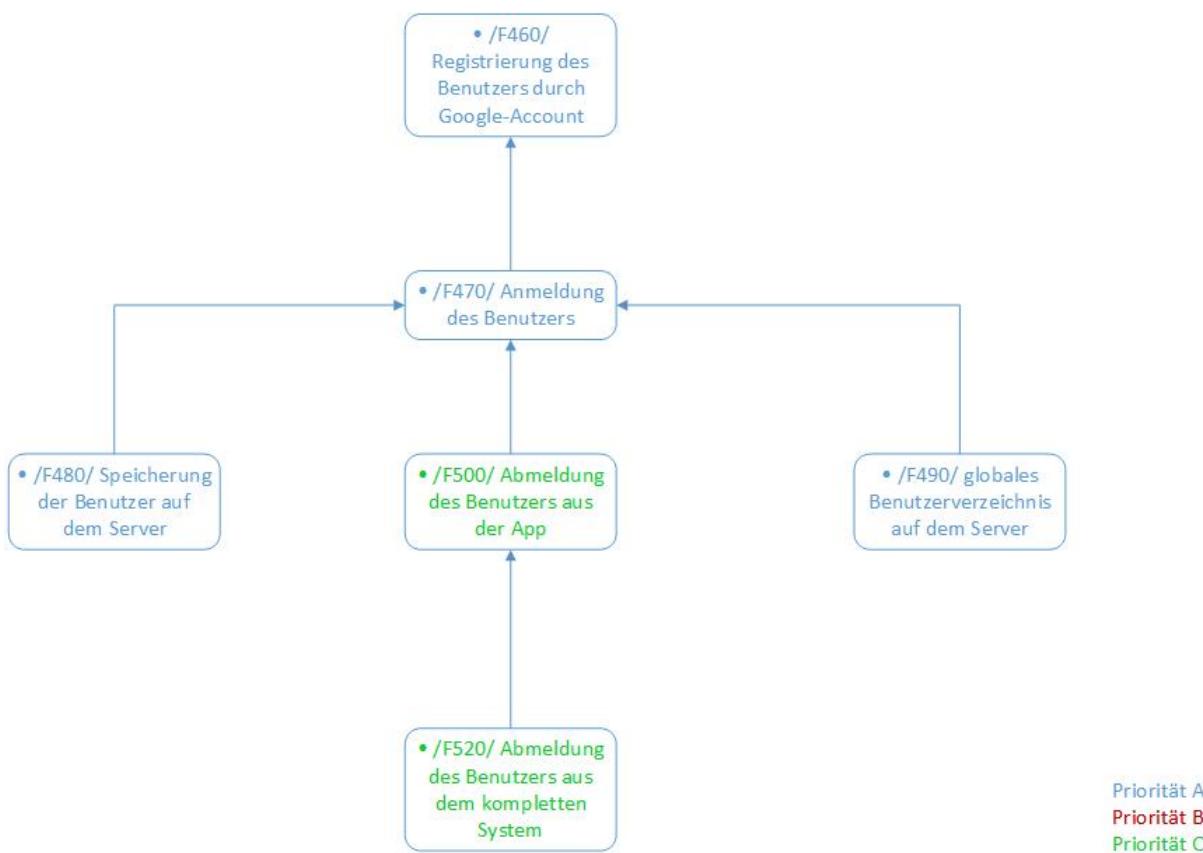
### 7.2.3 Lokalisierung

- /F430/ wurde heruntergestuft, da der Benutzer selbst erkennen kann, ob sein GPS an oder aus ist und diese Anforderung somit keinen großen Effekt auf die Funktion der App hat.



#### 7.2.4 Benutzerverwaltung

- /F500/ wurde heruntergestuft, weil der Benutzer kaum Anlass findet, sich aus der App abzumelden, und damit die Anforderung eher unwichtig ist.



## 8 Anhang: Vollständiges großformatiges Klassendiagramm



