Entwurf Go-App

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

19.06.2016

Contents

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 Entwickelarbeit zur Folge hat, anzubieten. Dahinterliegende Ebenen können verborgen bleiben und somit die Architektur insgesamt vereinfacht werden.

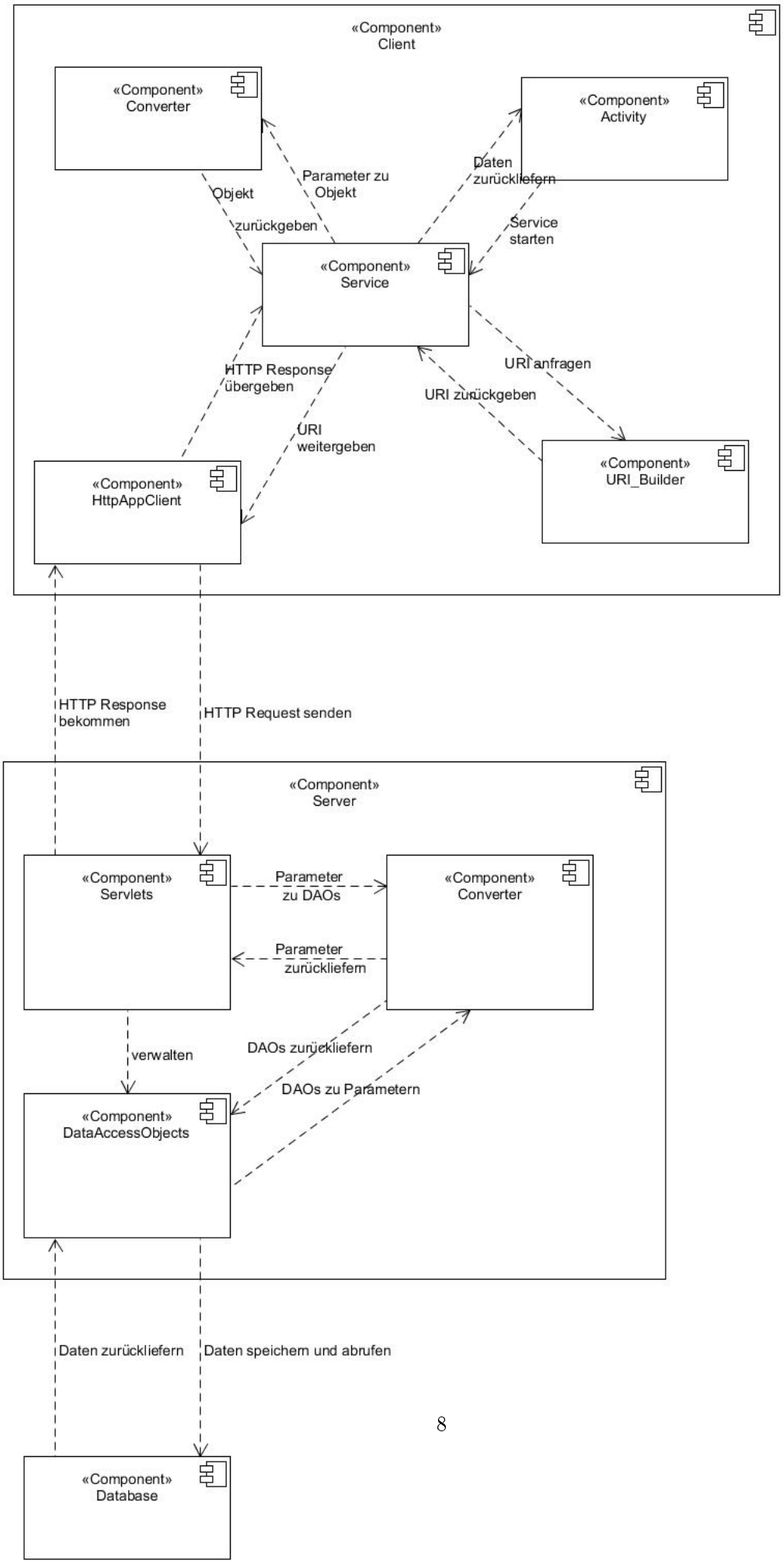
Zur Umsetzung der REST-Architektur verwedet die App das HTTP-Protokoll. Gründe dafür gibt es viele: Einerseits ist das Protokoll etabliert, dennoch vergleichsweise einfach aufgebaut und zu guter Letzt auch in so gut wie jeder Firewall offen.

Im Client wird zusätzlich die Model-View-Controller-Architektur verwendet, da die Architektur einen flexiblen Programmentwurf 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.

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 mit einer Instanz von HttpAppClient mit HTTP Request mit den Parametern der Anfrage als JSON String zu dem jeweiligen Servlet im Server. Dort wird der Request verarbeitet und zum Converter gegeben, der den JSON String zu einem Objekt parst, nämlich dem DataAccessObject (DAO). Das Servlet

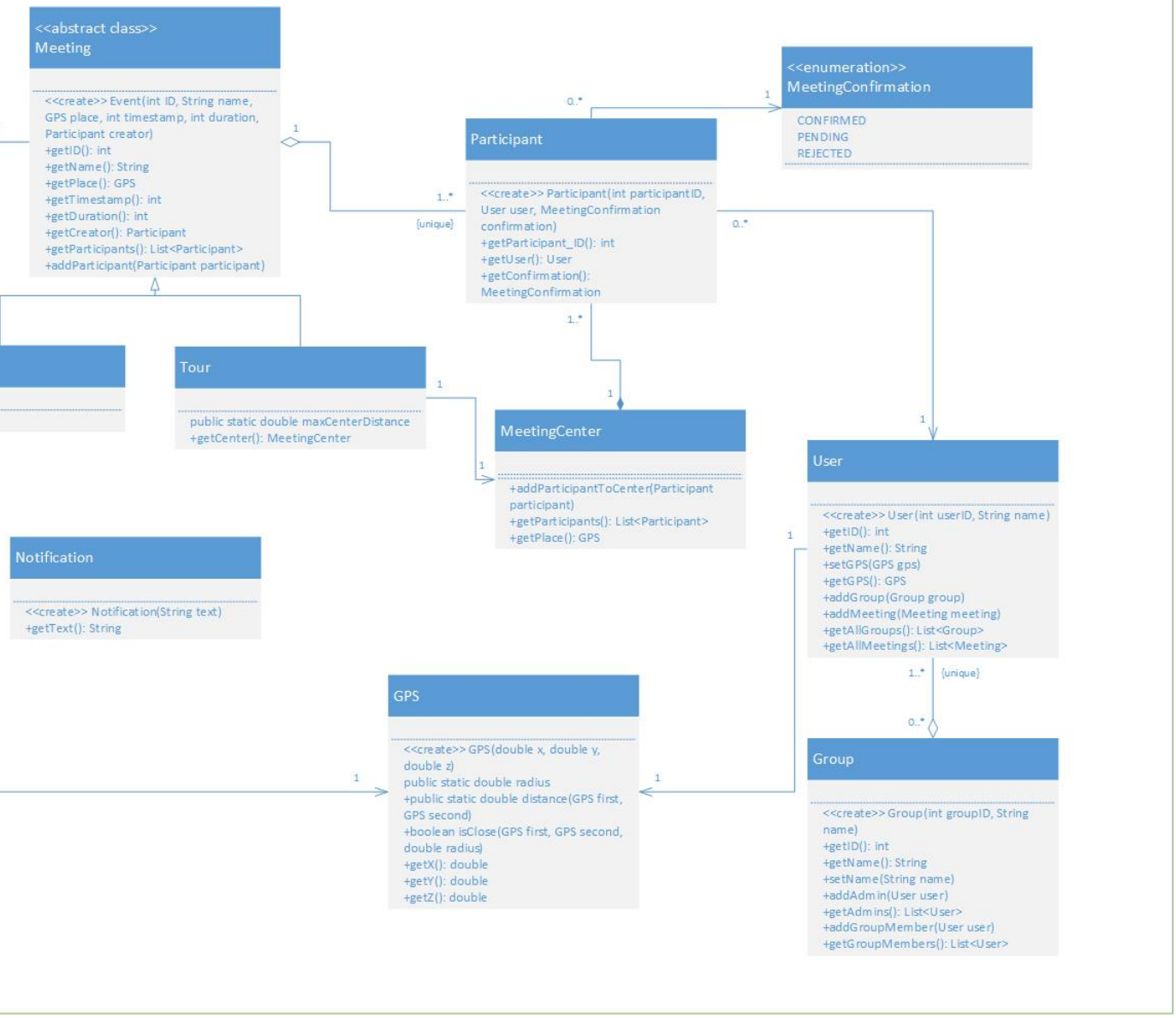
verwaltet nun das DAO. Das DAO stellt die Schnittstelle zur Datenbank dar und ruft Daten ab oder speichert sie in der Datenbank. Wenn benötigt, liefert die Datenbank Daten an das DAO zurück. Diese werden dann wiederum in dem Converter zu einem JSON String serialisiert und an das Servlet zurückgeliefert. 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 geparst und die Objekte an den Service zurückgegeben, der wiederum die Daten an die Activity übermittelt.


«Component»
Client
«Component»
Converter
«Component»
Activity
Parameter zu
Objekt
Objekt
zurückgeben
Daten
zurückliefern
Service
starten
«Component»
Service
HTTP Response
übergeben
URI anfragen
URI zurückgeben
URI
weitergeben
«Component»
HttpAppClient
«Component»
URI_Builder
HTTP Response
bekommen
HTTP Request senden
«Component»
Server
«Component»
Servlets
Parameter
zu DAOs
«Component»
Converter
Parameter
zurückliefern
verwalten
DAOs zurückliefern
DAOs zu Parametern
«Component»
DataAccessObjects
Daten zurückliefern
Daten speichern und abrufen
«Component»
Database

2 Entwurfsentscheidungen

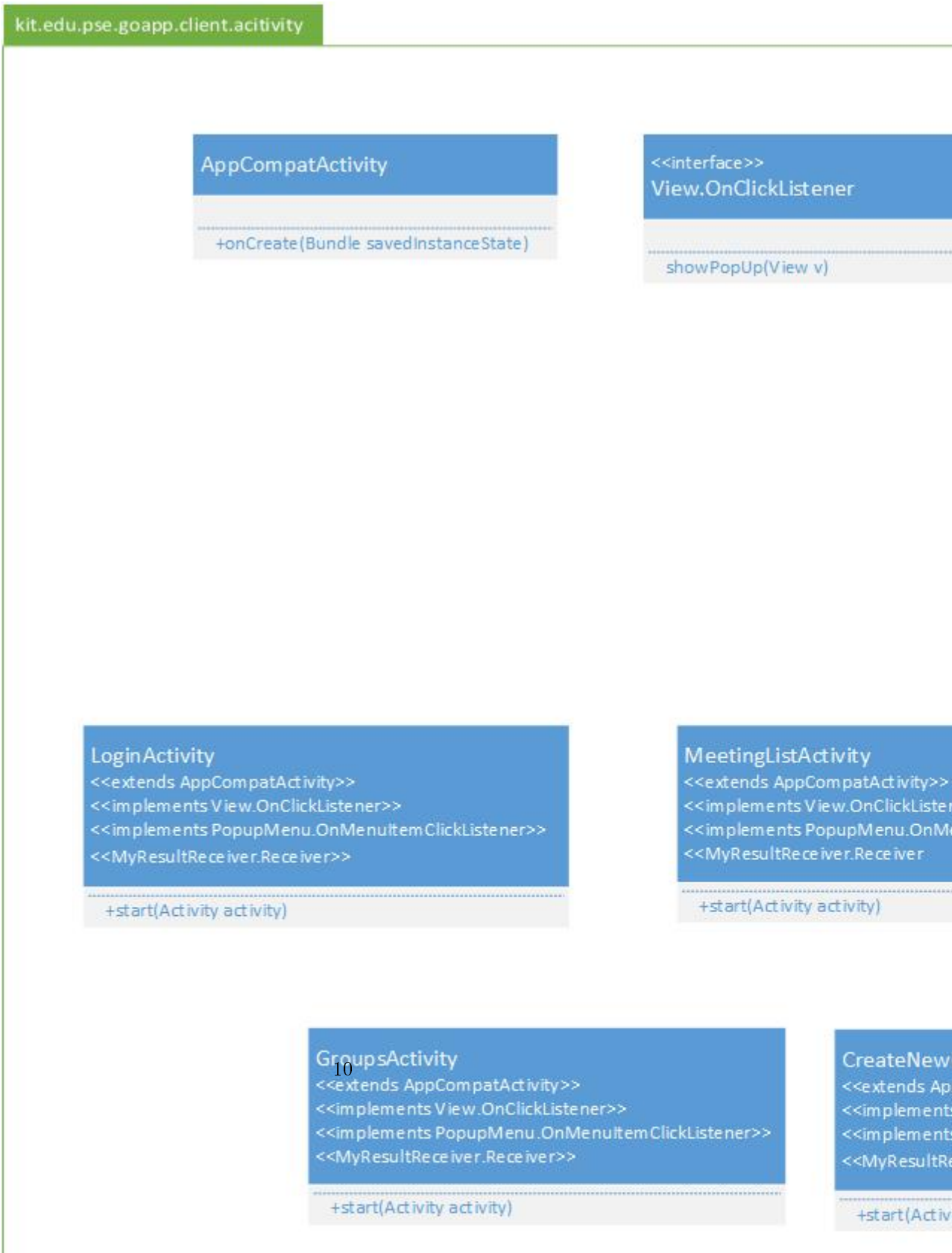
2.1 Client

lient.datamodels



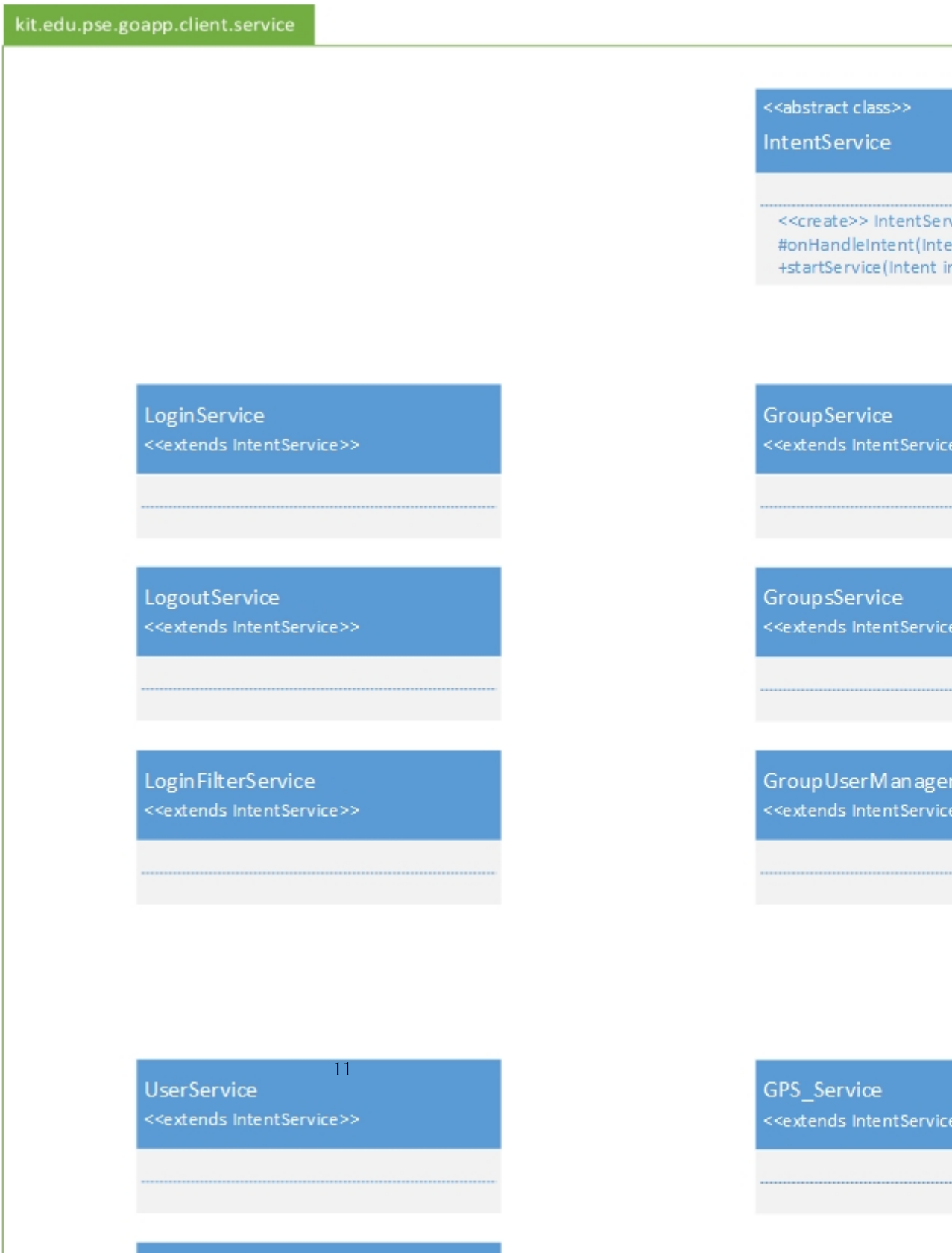
2.1.2 Activity

`kit.edu.pse.goapp.client.activity`



2.1.3 Service

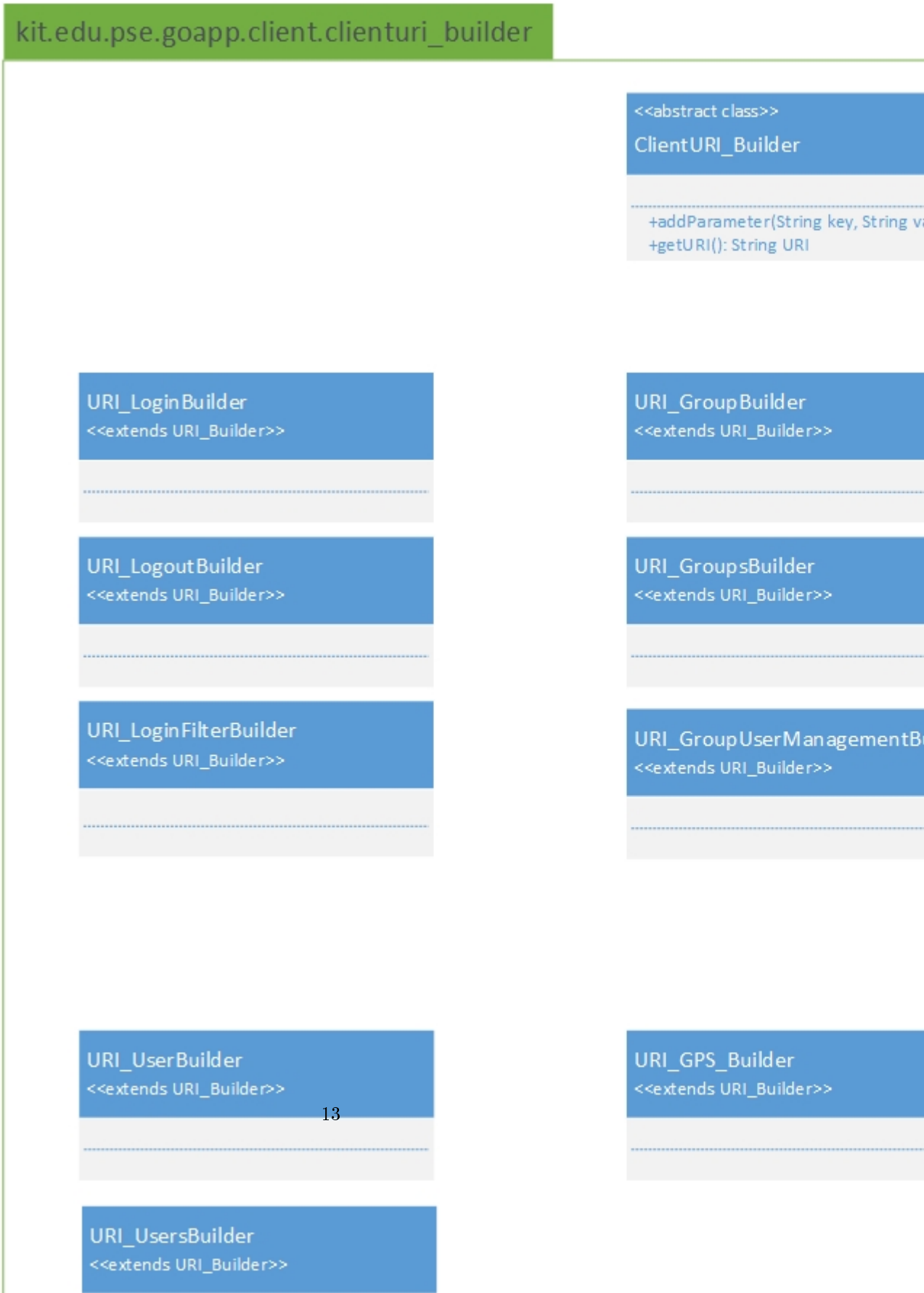
`kit.edu.pse.goapp.client.service`



2.1.4 URI_Builder

`kit.edu.pse.goapp.client.clienturi_builder`

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

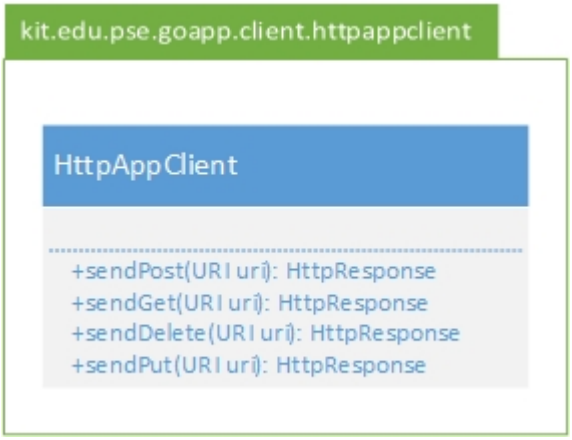

kit.edu.pse.goapp.client.clienturi_builder
<<abstract class>>
ClientURI_Builder
+addParameter(String key, String va
+getURI(): String URI
URI_LoginBuilder
<<extends URI_Builder>>
URI_GroupBuilder
<<extends URI_Builder>>
URI_LogoutBuilder
<<extends URI_Builder>>
URI_GroupsBuilder
<<extends URI_Builder>>
URI_LoginFilterBuilder
<<extends URI_Builder>>
URI_GroupUserManagementB
<<extends URI_Builder>>
URI_UserBuilder
<<extends URI_Builder>>
URI_GPS_Builder
<<extends URI_Builder>>
URI_UsersBuilder
<<extends URI_Builder>>

Grundlage hierfür ist die abstrakte Klasse UriBuilder, welche von den darunter sichtbaren Klassen erweitert wird. Die Klassen stellen eine simple möglichkeit für andere Klassen dar, um URIs(welche später einfach zu einem HTTP Request umgewandelt werden) zu erzeugen und zu füllen. 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 dem 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 HttpClient weitergibt.

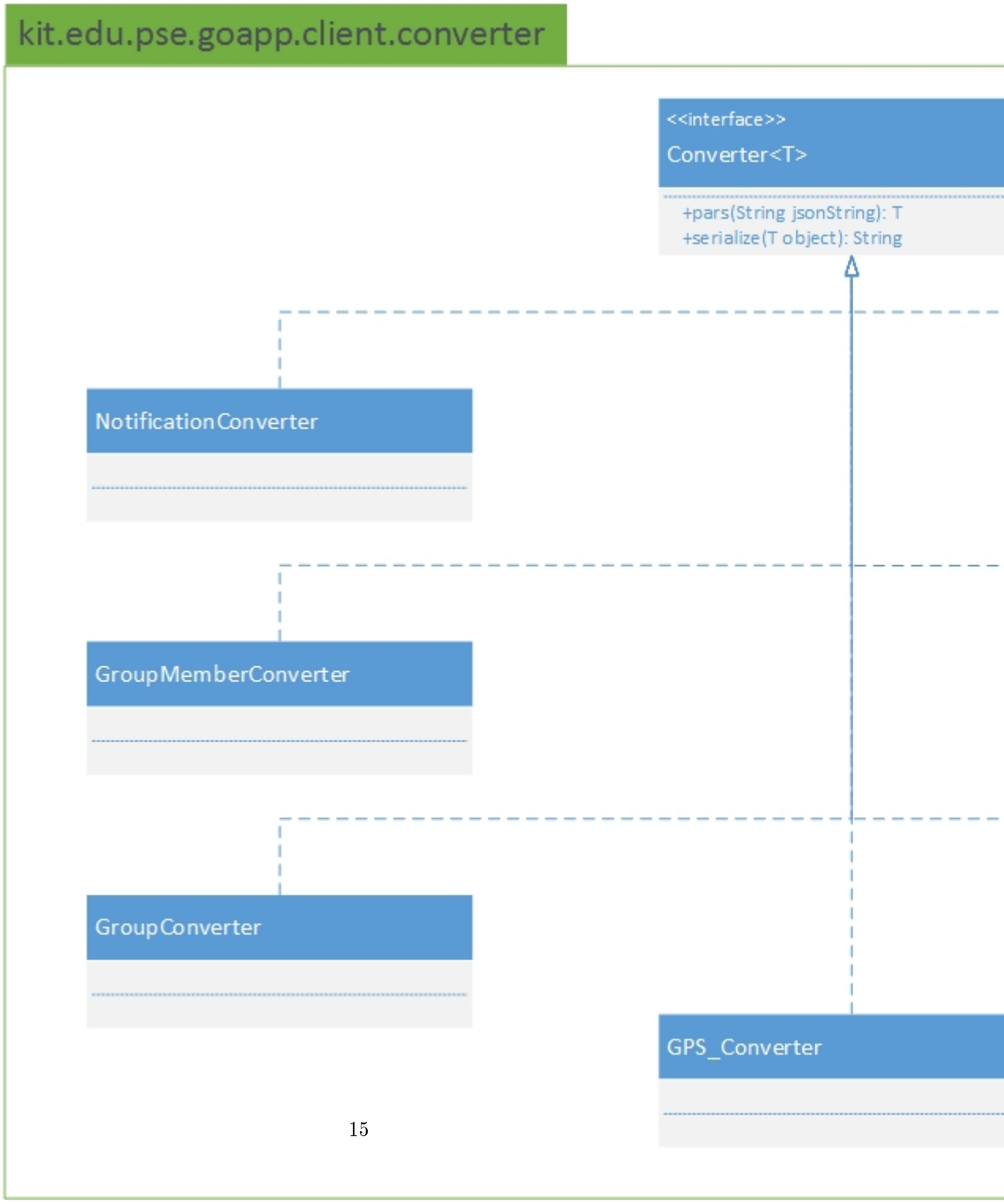
2.1.5 HttpAppClient

`kit.edu.pse.goapp.client.httpappclient`



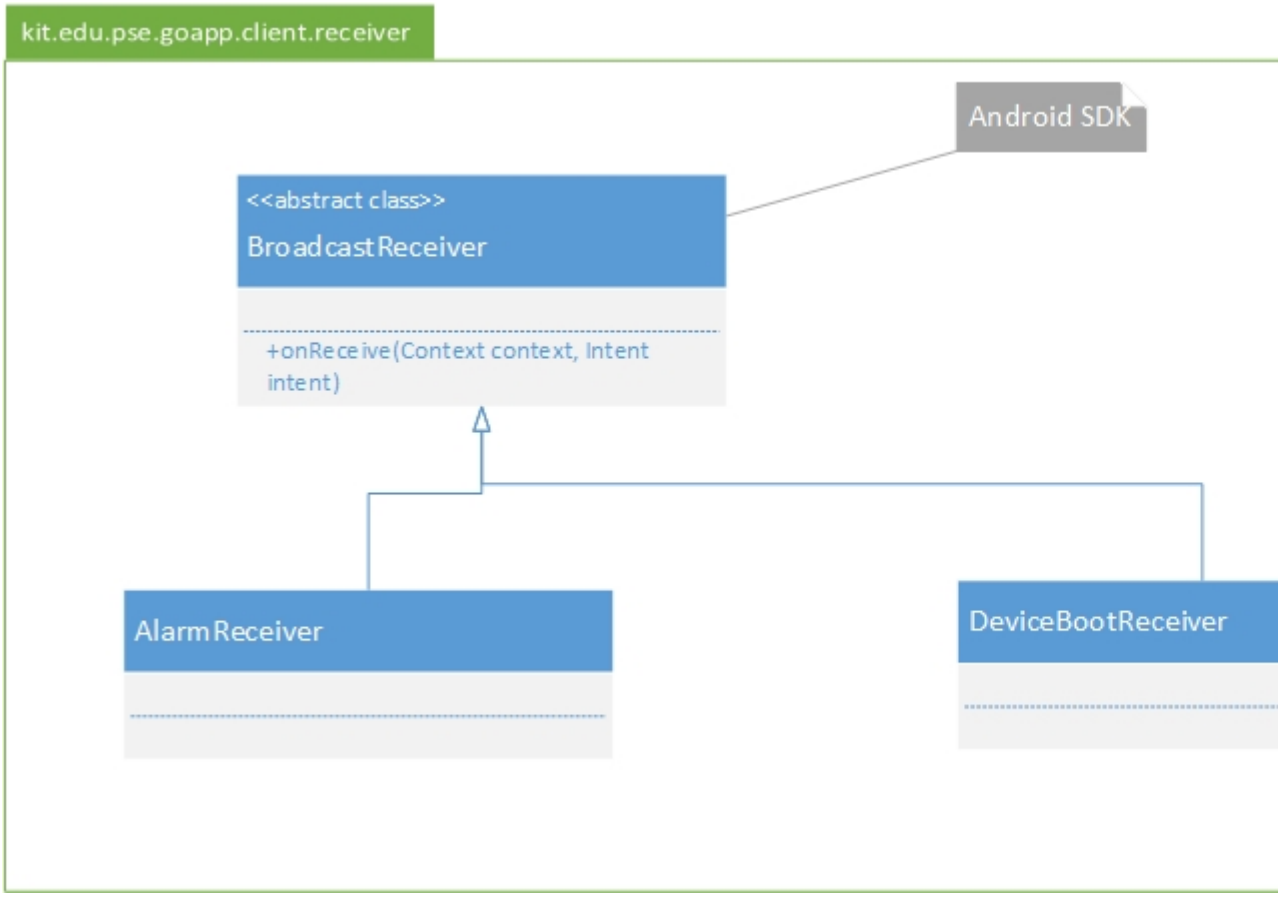
2.1.6 Converter

`kit.edu.pse.goapp.client.converter`



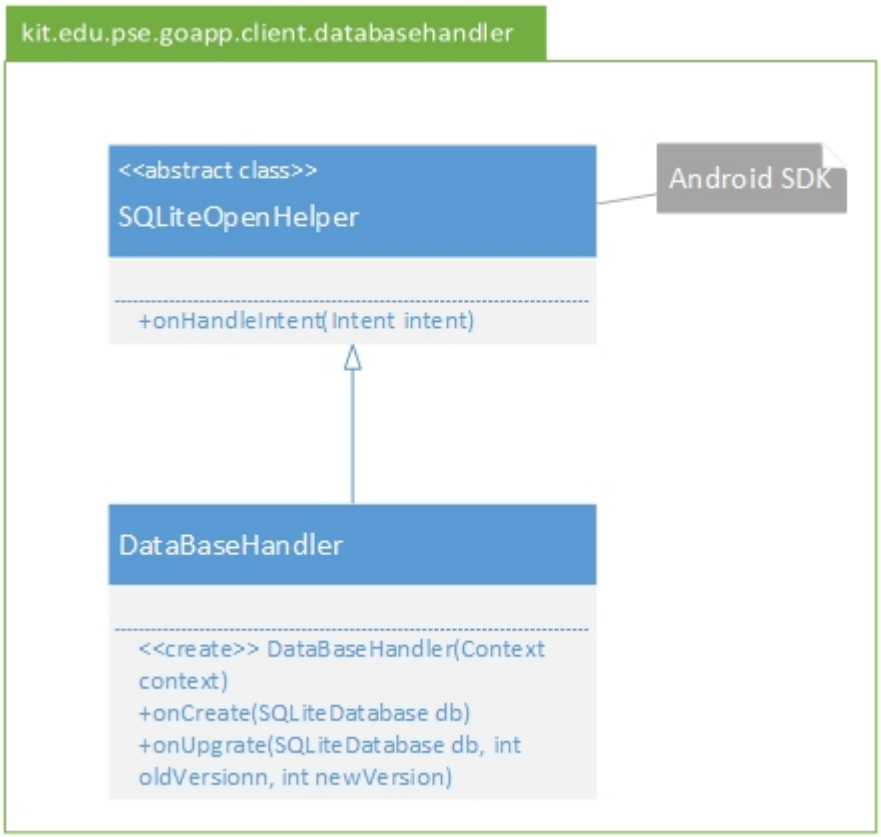
2.1.7 Receiver

`kit.edu.pse.goapp.client.receiver`



2.1.8 Database-Handler

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



2.2 Server

2.2.1 Klassendiagramm

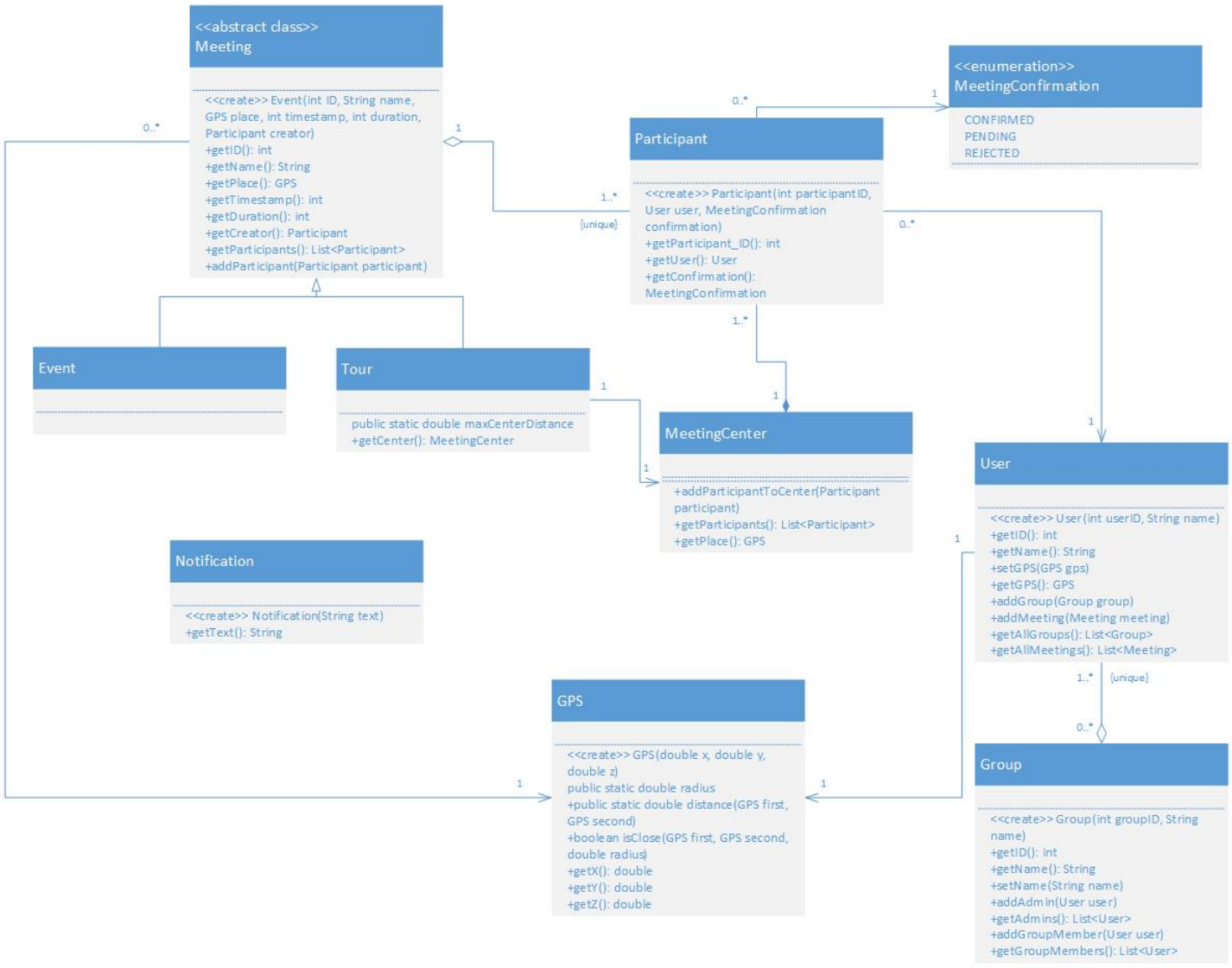
`kit.edu.pse.goapp.server.datamodels`

Die folgende Abbildung beschreibt die Klassen und ihre Beziehungen im Datenmodells 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.

Hier ist die Grafik Klassendiagramm

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. In diesen Gruppen allerdings muss immer mindestens ein Nutzer sein und niemals Nutzer doppelt, dass 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).

kit.edu.pse.goapp.server.datamodels

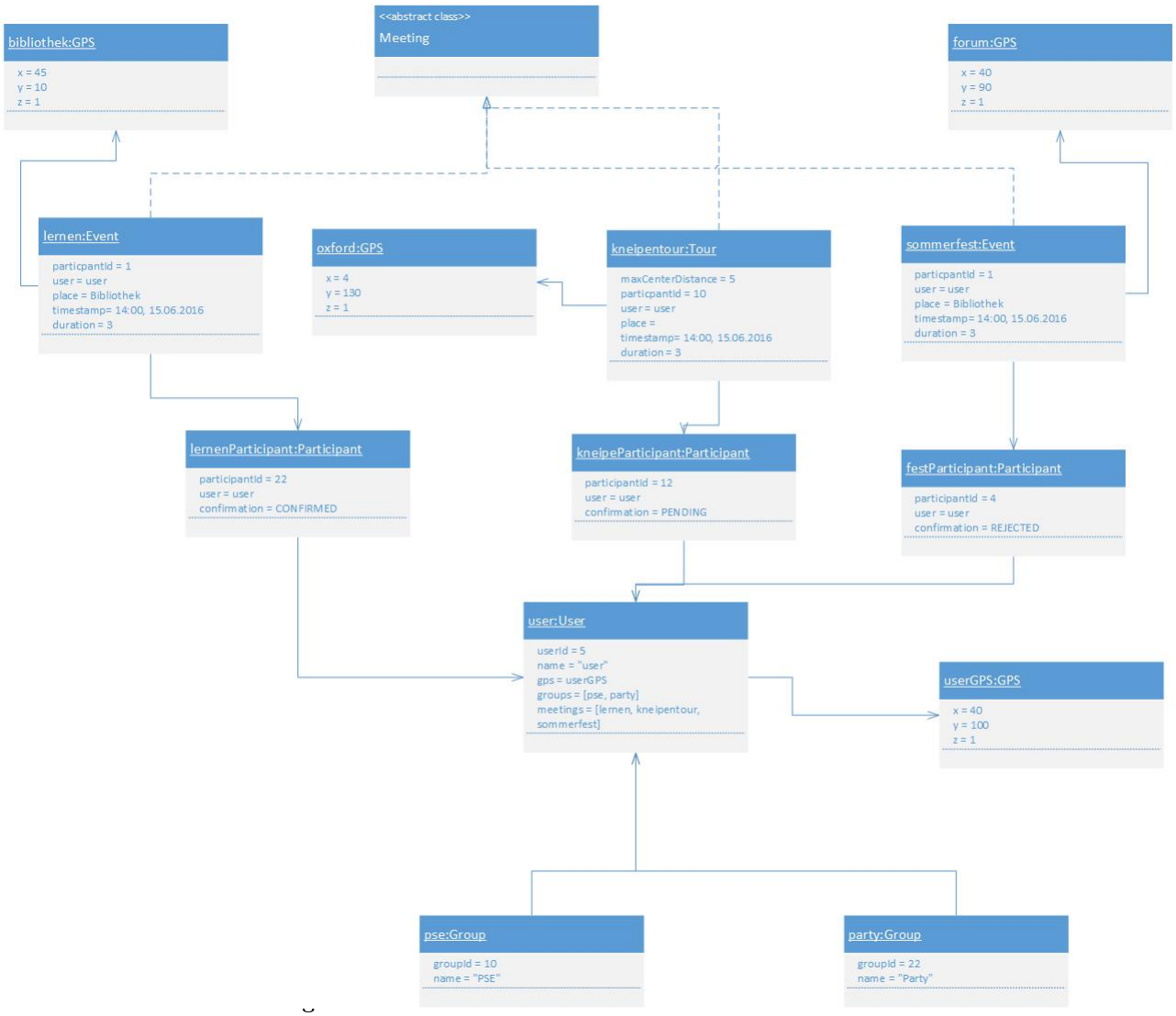


2.2.2 Objektdiagramm

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

Hier ist die Grafik Objektdiagramm

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 das kneipentour eine Tour ist, die allerdings noch nicht angefangen hat und somit noch kein MeetingCenter besitzt, und sich bei den zu user gehörenden Particicpants jeweils die


bibliothek:GPS
x = 45
y = 10
z = 1
<<abstract class>>
Meeting
forum:GPS
x = 40
y = 90
z = 1
lernen:Event
particpantId = 1
user = user
place = Bibliothek
timestamp= 14:00, 15.06.2016
duration = 3
oxford:GPS
x = 4
y = 130
z = 1
kneipentour:Tour
maxCenterDistance = 5
particpantId = 10
user = user
place =
timestamp= 14:00, 15.06.2016
duration = 3
sommerfest:Event
particpantId = 1
user = user
place = Bibliothek
timestamp= 14:00, 15.06.2016
duration = 3
lernenParticipant:Participant
participantId = 22
user = user
confirmation = CONFIRMED
kneipeParticipant:Participant
participantId = 12
user = user
confirmation = PENDING
festParticipant:Participant
participantId = 4
user = user
confirmation = REJECTED
user:User
userId = 5
name = "user"
gps = userGPS
groups = [pse, party]
meetings = [lernen, kneipentour, sommerfest]
userGPS:GPS
x = 40
y = 100
z = 1
pse:Group
groupId = 10
name = "PSE"
party:Group
groupId = 22
name = "Party"

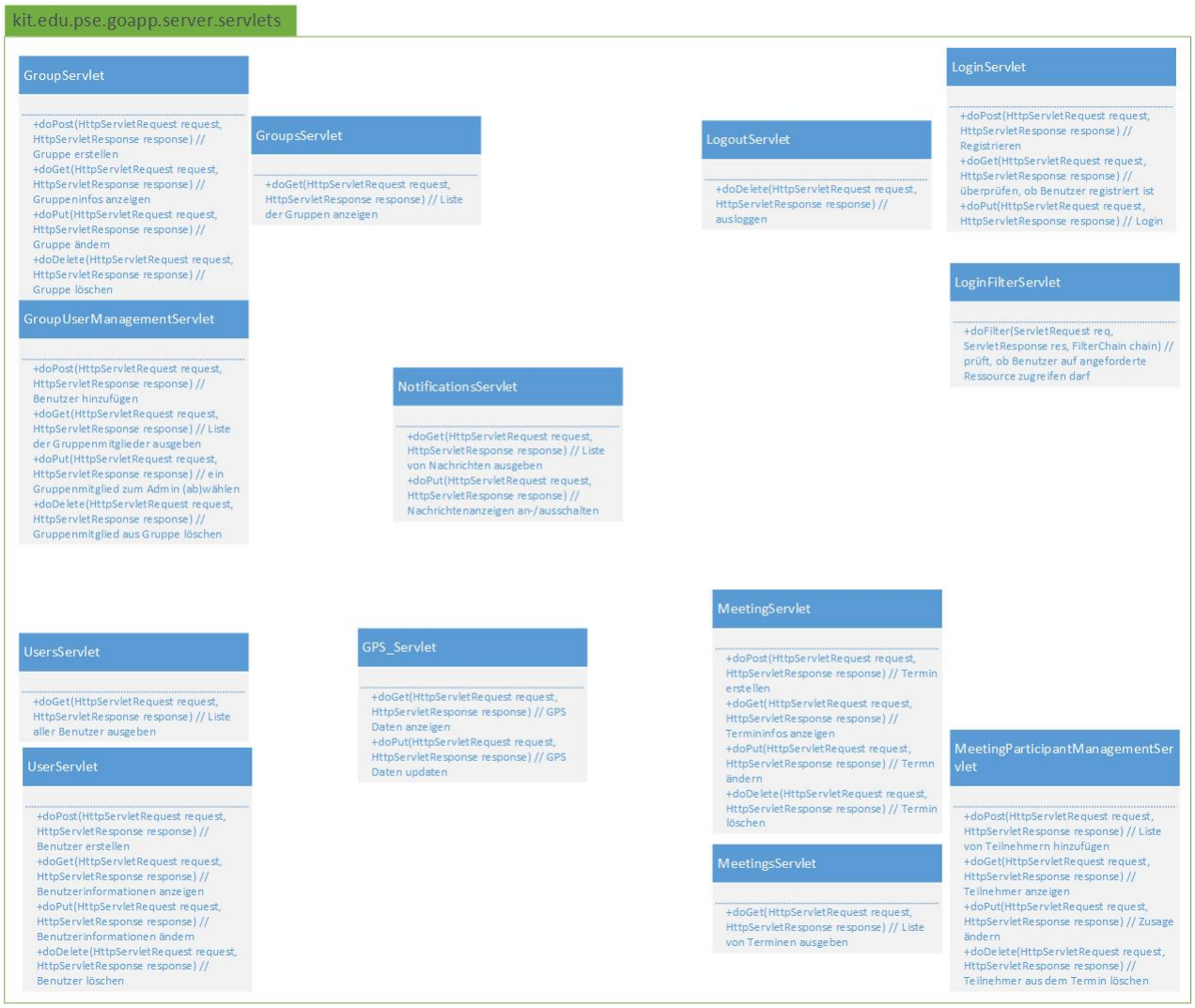
2.2.3 Servlets

`kit.edu.pse.goapp.server.servlets`

Die Servlets nehmen innerhalb des Servers Anfragen vom Client an 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 der App kümmern (LoginServlet, LogoutServlet, LoginFilterServlet), Servlets, die sich um die Verwaltung der Benutzer kümmern (UserServlet, UsersServlet), 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 das zugehörige


kit.edu.pse.goapp.server.servlets
GroupServlet
+doPost(HttpServletRequest request, HttpServletResponse response) // Gruppe erstellen
+doGet(HttpServletRequest request, HttpServletResponse response) // Gruppeninfos anzeigen
+doPut(HttpServletRequest request, HttpServletResponse response) // Gruppe ändern
+doDelete(HttpServletRequest request, HttpServletResponse response) // Gruppe löschen
GroupsServlet
+doGet(HttpServletRequest request, HttpServletResponse response) // Liste der Gruppen anzeigen
LogoutServlet
+doDelete(HttpServletRequest request, HttpServletResponse response) // ausloggen
LoginServlet
+doPost(HttpServletRequest request, HttpServletResponse response) // Registrieren
+doGet(HttpServletRequest request, HttpServletResponse response) // überprüfen, ob Benutzer registriert ist
+doPut(HttpServletRequest request, HttpServletResponse response) // Login
LoginFilterServlet
+doFilter(ServletRequest req, ServletResponse res, FilterChain chain) // prüft, ob Benutzer auf angeforderte Ressource zugreifen darf
GroupUserManagementServlet
+doPost(HttpServletRequest request, HttpServletResponse response) // Benutzer hinzufügen
+doGet(HttpServletRequest request, HttpServletResponse response) // Liste der Gruppenmitglieder ausgeben
+doPut(HttpServletRequest request, HttpServletResponse response) // ein Gruppenmitglied zum Admin (ab)wählen
+doDelete(HttpServletRequest request, HttpServletResponse response) // Gruppenmitglied aus Gruppe löschen
NotificationsServlet
+doGet(HttpServletRequest request, HttpServletResponse response) // Liste von Nachrichten ausgeben
+doPut(HttpServletRequest request, HttpServletResponse response) // Nachrichtenanzeigen an-/ausschalten
MeetingServlet
+doPost(HttpServletRequest request, HttpServletResponse response) // Termin erstellen
+doGet(HttpServletRequest request, HttpServletResponse response) // Termininfos anzeigen
+doPut(HttpServletRequest request, HttpServletResponse response) // Termn ändern
+doDelete(HttpServletRequest request, HttpServletResponse response) // Termin löschen
MeetingsServlet
+doGet(HttpServletRequest request, HttpServletResponse response) // Liste von Terminen ausgeben
MeetingParticipantManagementServlet
+doPost(HttpServletRequest request, HttpServletResponse response) // Liste von Teilnehmern hinzufügen
+doGet(HttpServletRequest request, HttpServletResponse response) // Teilnehmer anzeigen
+doPut(HttpServletRequest request, HttpServletResponse response) // Zusage ändern
+doDelete(HttpServletRequest request, HttpServletResponse response) // Teilnehmer aus dem Termin löschen
UsersServlet
+doGet(HttpServletRequest request, HttpServletResponse response) // Liste aller Benutzer ausgeben
GPS_Servlet
+doGet(HttpServletRequest request, HttpServletResponse response) // GPS Daten anzeigen
+doPut(HttpServletRequest request, HttpServletResponse response) // GPS Daten updaten
UserServlet
+doPost(HttpServletRequest request, HttpServletResponse response) // Benutzer erstellen
+doGet(HttpServletRequest request, HttpServletResponse response) // Benutzerinformationen anzeigen
+doPut(HttpServletRequest request, HttpServletResponse response) // Benutzerinformationen ändern
+doDelete(HttpServletRequest request, HttpServletResponse response) // Benutzer löschen

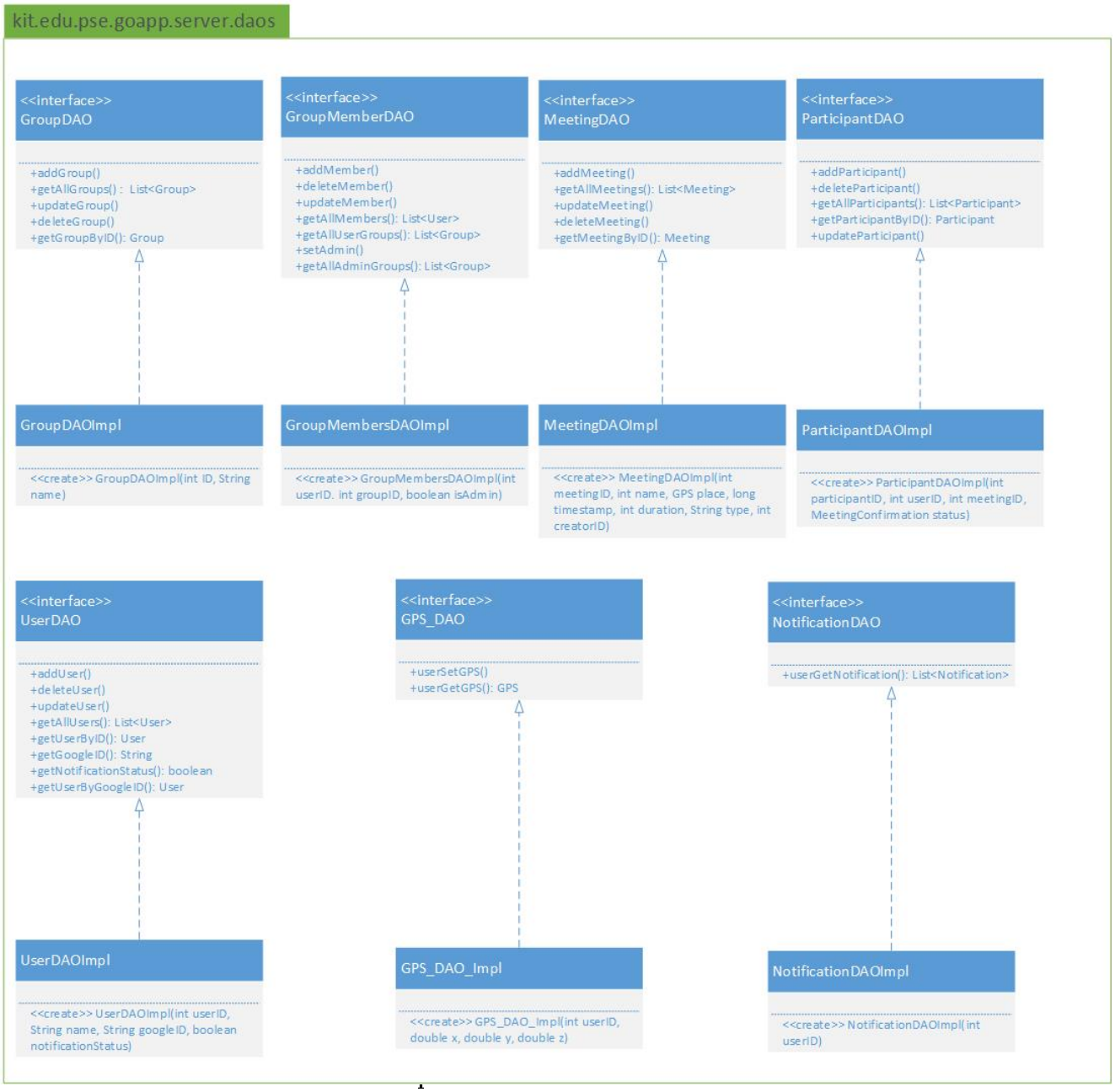
2.2.4 DataAccessObjects

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

Wir verwenden an diser 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 Model Object ausgeführt werden. Das Model Object 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 in den Converter, damit der einen JSON String daraus machen kann, um


kit.edu.pse.goapp.server.daos
<<interface>>
GroupDAO
+addGroup()
+getAllGroups() : List<Group>
+updateGroup()
+deleteGroup()
+getGroupByID(): Group
<<interface>>
GroupMemberDAO
+addMember()
+deleteMember()
+updateMember()
+getAllMembers(): List<User>
+getAllUserGroups(): List<Group>
+setAdmin()
+getAllAdminGroups(): List<Group>
<<interface>>
MeetingDAO
+addMeeting()
+getAllMeetings(): List<Meeting>
+updateMeeting()
+deleteMeeting()
+getMeetingByID(): Meeting
<<interface>>
ParticipantDAO
+addParticipant()
+deleteParticipant()
+getAllParticipants(): List<Participant>
+getParticipantByID(): Participant
+updateParticipant()
GroupDAOImpl
<<create>> GroupDAOImpl(int ID, String name)
GroupMembersDAOImpl
<<create>> GroupMembersDAOImpl(int userID. int groupID, boolean isAdmin)
MeetingDAOImpl
<<create>> MeetingDAOImpl(int meetingID, int name, GPS place, long timestamp, int duration, String type, int creatorID)
ParticipantDAOImpl
<<create>> ParticipantDAOImpl(int participantID, int userID, int meetingID, MeetingConfirmation status)
<<interface>>
UserDAO
+addUser()
+deleteUser()
+updateUser()
+getAllUsers(): List<User>
+getUserByID(): User
+getGoogleID(): String
+getNotificationStatus(): boolean
+getUserByGoogleID(): User
<<interface>>
GPS_DAO
+userSetGPS()
+userGetGPS(): GPS
<<interface>>
NotificationDAO
+userGetNotification(): List<Notification>
UserDAOImpl
<<create>> UserDAOImpl(int userID, String name, String googleID, boolean notificationStatus)
GPS_DAO_Impl
<<create>> GPS_DAO_Impl(int userID, double x, double y, double z)
NotificationDAOImpl
<<create>> NotificationDAOImpl(int userID)

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 zu den 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 beiden Methoden** `public T parse(String jsonString)` **und** `public String serialize(T object)`.

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.**

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)`.

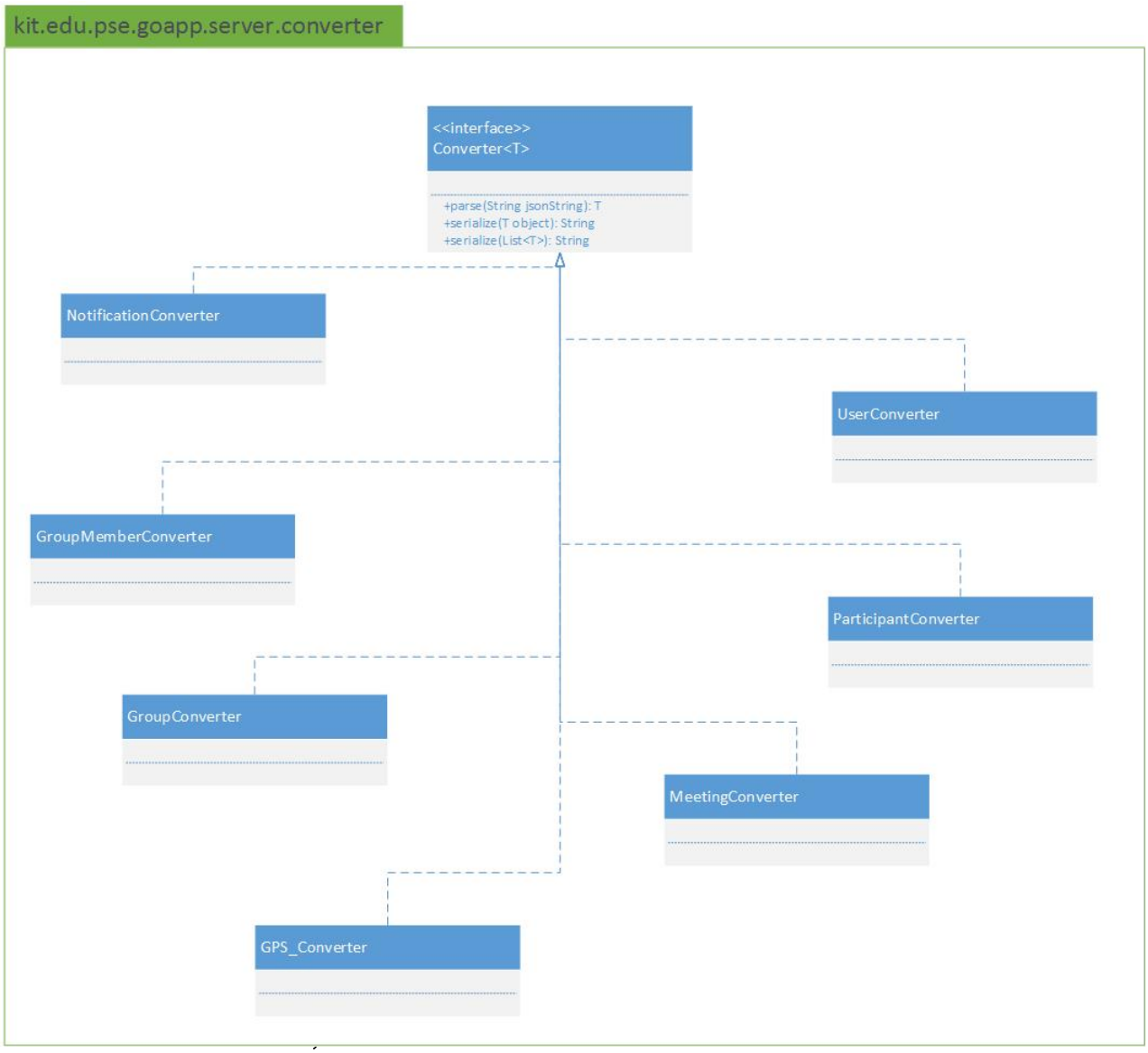
Die Klasse `GroupConverter` **konvertiert Gruppen in Strings und umgekehrt. Sie implementiert die Methoden** `public Group parse(String)`, `public String serialize(Group group)` **und** `public String serialize(List <Group> groups)`.

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`


kit.edu.pse.goapp.server.converter
<<interface>>
Converter<T>
+parse(String jsonString): T
+serialize(T object): String
+serialize(List<T>): String
NotificationConverter
UserConverter
GroupMemberConverter
ParticipantConverter
GroupConverter
MeetingConverter
GPS_Converter

2.2.6 Datenbank

`kit.edu.pse.goapp.server.database`

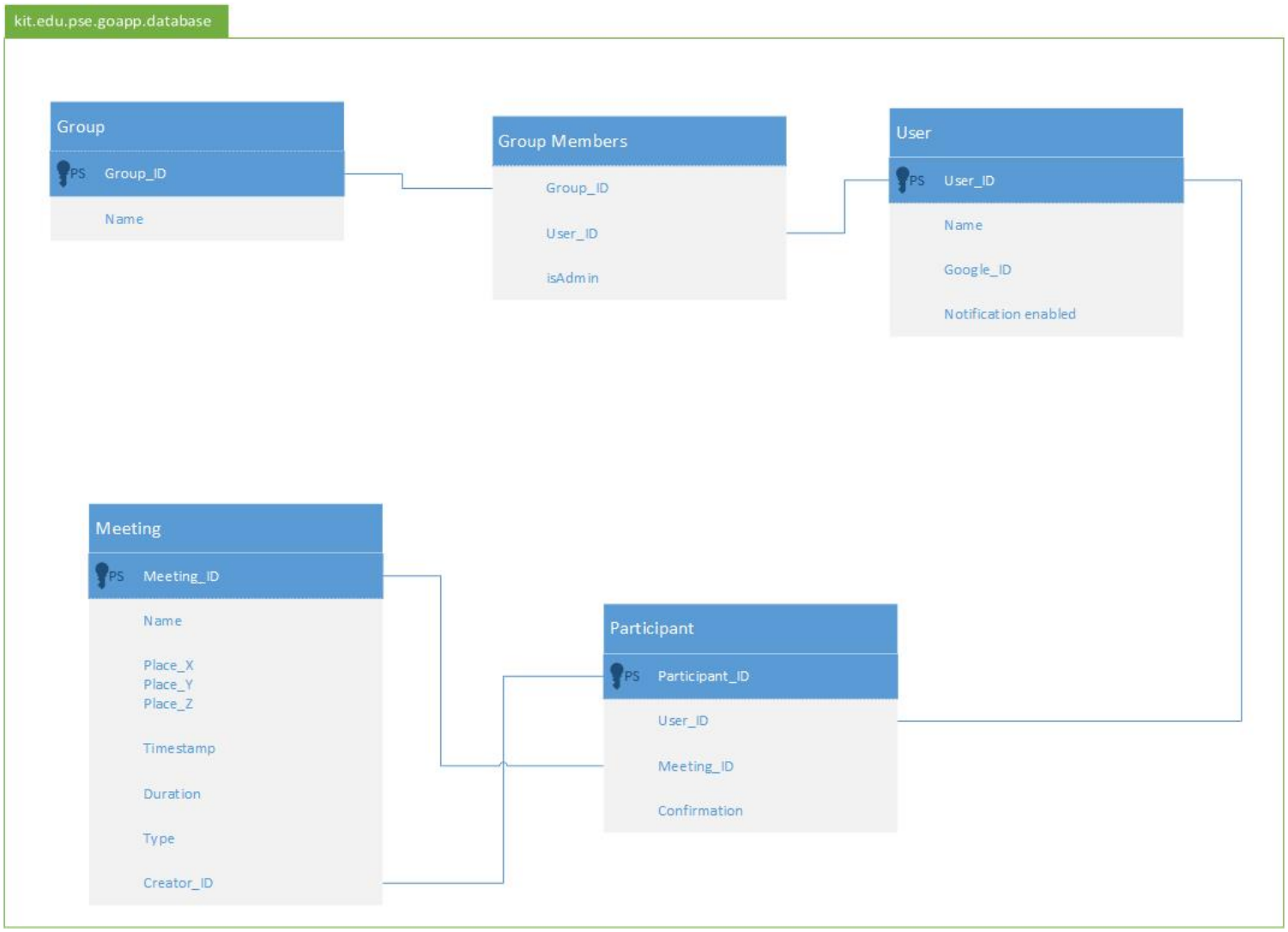
Die verschiedenen Elemente und Objekte der Go-App sowie ihre Eigenschaften werden in einer Datenbank gespeichert. Das Diagramm beschreibt die Datenbank und die Beziehungen zwischen den Objekten.

(Hier soll das Diagramm hin)

Jeder Benutzer (`User`) erhält bei der Registrierung mit seiner Google_ID eine `User_ID`, anhand der 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.


kit.edu.pse.goapp.database
Group
PS
Group_ID
Name
Group Members
Group_ID
User_ID
isAdmin
User
PS
User_ID
Name
Google_ID
Notification enabled
Meeting
PS
Meeting_ID
Name
Place_X
Place_Y
Place_Z
Timestamp
Duration
Type
Creator_ID
Participant
PS
Participant_ID
User_ID
Meeting_ID
Confirmation

3 Client ↔ Server RESTful API

package kit.edu.goapp.server.servlets

3.1 HTTP-Protokoll

Methode	Anfrage	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 neuer 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
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	Termininformarionen 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	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 können bei einer HTTP-Response auftreten:

Code	Nachricht	Bedeutung
202	accepted	Die Anfrage wurde akzeptiert, wird aber zu einem späteren Zeitpunkt aus
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 A
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 Adminstrator 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` we

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 Administrato
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 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 die Gruppe bearbeitet. Gruppeneinstellungen können geändert werden.

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 `name` hat den falschen Datentyp
404	Es konnte kein Benutzer mit dem Namen `name` gefunden werden

3.6.3 In die Go-App einloggen

`PUT /user/update`

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

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 `accessToken` hat den falschen Datentyp
401	Der Benutzer hat keinen gültigen `accessToken`

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`	Gruppenname

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`	Gruppenname

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

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
`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

3.11 `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
name	String	Name des jeweilgen Benutzers

Antwort

Bei Erfolg meldet der Server `200`, sonst einen Fehler

Fehler

Code	Beschreibung
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:

Parameter	Datentyp	Beschreibung
`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

Code	Beschreibung
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 bearbeit
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 `terminID` 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 `meetingID` 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 `terminID` hat den falschen Datentyp
404	Es konnte kein Termin mit der ID `meetingID` 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 hinzuzufüge
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 de
404	Es konnte kein Termin mit der ID `meetingID` gefunden werden
404	Es konnte kein Teilnehmer mit der ID `participantID` gefunden werden

4 Klassen des Clients

4.1 Package kit.edu.pse.goapp.client.activity

4.1.1 `Class AppCompatActivity`

Beschreibung

(Beschreibung der Klasse einfügen)

Methoden

- `public void onCreate(Bundle savedInstanceState)`
  (Beschreibung einfügen)
  Parameter
  `Bundle savedInstanceState` (Beschreibung einfügen)

4.1.2 `Interface View.OnClickListener`

Beschreibung

(Beschreibung einfügen)

Methoden

- `public void showPopUp(View v)`
  Zeigt einPopUp-Fenster an
  Parameter
  `View v` Ansicht des PopUp-Fensters

4.1.3 `Interface PopupMenu.OnMenuItemClickListener`

Beschreibung

(Beschreibung einfügen)

Methoden

- `public boolean onMenuItemClick(MenuItem item)`
  (Beschreibung einfügen)
  Parameter
  `MenuItem item` (Beschreibung einfügen)
  Rückgabewert:
  (Beschreibung einfügen)

4.1.4 `Class GroupsActivity` **extends** `AppCompatActivity` **implements** `View.OnClickListener` **implements** `PopupMenu.OnMenuItemClickListener MyResultReceiver.Receiver`

Beschreibung

Diese Klasse stellt die Client-Seite einer Gruppe dar.

Methoden

- `public void start(Activity activity)`
  Startet eine Activity für eine Gruppe
  Parameter
  `Activity activity` Activity der Gruppe

4.1.5 `Class GroupMemberActivity` **extends** `AppCompatActivity` **implements** `View.OnClickListener` **implements** `PopupMenu.OnMenuItemClickListener MyResultReceiver.Receiver`

Beschreibung

Diese Klasse stellt die Client-Seite eines Gruppenmitgliedes dar.

Methoden

- `public void start(Activity activity)`
  Startet eine Activity für ein Gruppenmitglied
  Parameter
  `Activity activity` Activity des Gruppenmitgliedes

4.1.6 `Class SettingsActivity` **extends** `AppCompatActivity` **implements** `View.OnClickListener` **implements** `PopupMenu.OnMenuItemClickListener MyResultReceiver.Receiver`

Beschreibung

Diese Klasse stellt die Client-Seite der App-Einstellungen dar.

Methoden

- `public void start(Activity activity)`
  Startet eine Activity für die Einstellungen der App
  Parameter
  `Activity activity` Activity der Einstellungen

4.1.7 Class CreateNewGroupActivity extends AppCompatActivity implements View.OnClickListener implements PopupMenu.OnMenuItemClickListener MyResultReceiver.Receiver

Beschreibung

Diese Klasse startet eineActivity, um eine neue Gruppe zu erstellen.

Methoden

- `public void start(Activity activity)`
  Startet eine Activity zur Erstellung einer Gruppe
  Parameter
  `Activity activity` Activity zur Erstellung einer Gruppe

4.1.8 Class MeetingParticipantActivity extends AppCompatActivity implements View.OnClickListener implements PopupMenu.OnMenuItemClickListener MyResultReceiver.Receiver

Beschreibung

Diese Klasse stellt die Client-Seite eines Teilnehmers eines Termins dar.

Methoden

- `public void start(Activity activity)`
  Startet eine Activity für einen Teilnehmer
  Parameter
  `Activity activity` Activity des Teilnehmers

4.1.9 Class AboutActivity extends AppCompatActivity implements View.OnClickListener implements PopupMenu.OnMenuItemClickListener MyResultReceiver.Receiver

Beschreibung

(Beschreibung hinzufügen)

Methoden

- `public void start(Activity activity)`
  (Beschreibung hinzufügen)
  Parameter
  `Activity activity` (Beschreibung hinzufügen)

4.1.10 `Class MapActivity` **extends** `AppCompatActivity` **implements** `View.OnClickListener` **implements** `PopupMenu.OnMenuItemClickListener MyResultReceiver.Receiver`

Beschreibung

Diese Klasse stellt eine Activity zur Darstellung der GPS-Positionen auf der Karte dar.

Methoden

- `public void start(Activity activity)`
  Startet eine Activity zur Darstellung der Karte
  Parameter
  `Activity activity` Activity zur Darstellung der Karte

4.1.11 `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 eine Activity zum Einloggen der Benutzer
  Parameter
  `Activity activity` Activity zum Einloggen der Benutzer

4.1.12 `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 zurückgibt.

Methoden

- `public void start(Activity activity)`
  Startet eine Activity, die eine Liste der Termine zurückgibt
  Parameter
  `Activity activity` Activity, die eine Liste der Termine zurückgibt

4.1.13 `Class NewMeetingActivity` **extends** `AppCompatActivity` **implements** `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 eine Activity zur Erstellung eines neuen Termins
  Parameter
  `Activity activity` Activity zur Erstellung eines neuen Termins

4.1.14 `AbstractClass ResultReceiver`

Beschreibung

(Beschreibung hinzufügen)

Konstruktoren

- `public ResultReceiver(Handler handler)`
  Parameter
  `handler` (Beschreibung hinzufügen)

Methoden

- `public void onReceiverResult(int resultCode, Bundle resultData)`
  (Beschreibung hinzufügen)
  Parameter
  `resultCode` (Beschreibung hinzufügen)
  `resultData` (Beschreibung hinzufügen)

4.1.15 `Interface Receiver`

Beschreibung

(Beschreibung hinzufügen)

Methoden

- `public void onReceiverResult(int resultCode, Bundle resultData)`
  (Beschreibung hinzufügen)
  Parameter
  `resultCode` (Beschreibung hinzufügen)
  `resultData` (Beschreibung hinzufügen)

4.1.16 `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)`
  (Beschreibung hinzufügen)
  Parameter
  `receiver` (Beschreibung hinzufügen)

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

`int ID`	ID des Termins
`String name`	Name des Termins
`GPS place`	Veranstaltungsort mit GPS-Koordinaten angegeben
`int timestamp`	Veranstaltungsbeginn
`int duration`	Voraussichtliche Dauer des Termins
`Participant creator`	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 überschreibt 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:

`int participantID`	ID des Teilnehmers
`User user`	Benutzer der Go-App
`MeetingConfirmation confirmation`	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

`CONFIRMED`	Die Person hat zugesagt
`PENDING`	Die Person hat bis jetzt weder zu- noch abgesagt
`REJECTED`	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 GroupMemeberDAO`

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()`
  Ernennt ein Gruppenmitglied als Gruppenadministrator.
- `public List<Group> getAllAdminGroups()`
  Gibt eine Liste mit allen Gruppen zurück, in denen der Benutzer Gruppedministrator 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

`int meetingID`	ID des Meetings
`int name`	Name des Termins
`GPS place`	GPS-Koordinaten des Veranstaltungsortes
`long timestamp`	Uhrzeit des Veranstaltungsbeginns
`int duration`	voraussichtliche Dauer des Termins
`String type`	"Veranstaltung" (bei festem Veranstaltungsort) oder "Tour" (bei sich veränderndem
`int creatorID`	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

`int participantID`	ID des Teilnehmers
`int userID`	Benutzer-ID des teilnehmenden Benutzers
`int meetingID`	ID des Termins
`MeetingConfirmation status`	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)`
  Parameter

`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 Parameterin 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	UNSIGNET 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	UNSIGNET TININT(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

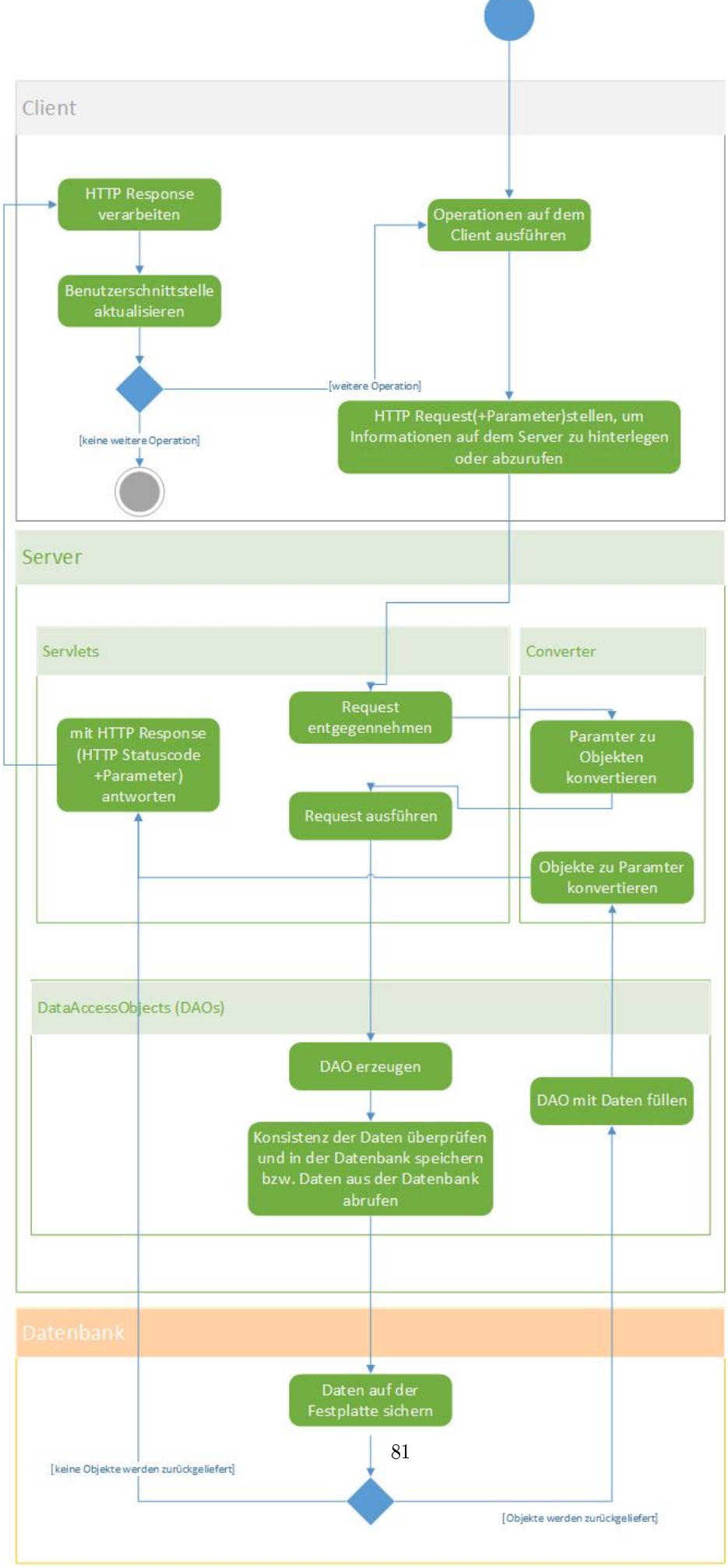
Attribute

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 Or

6 Beschreibung charakteristischer Abläufe

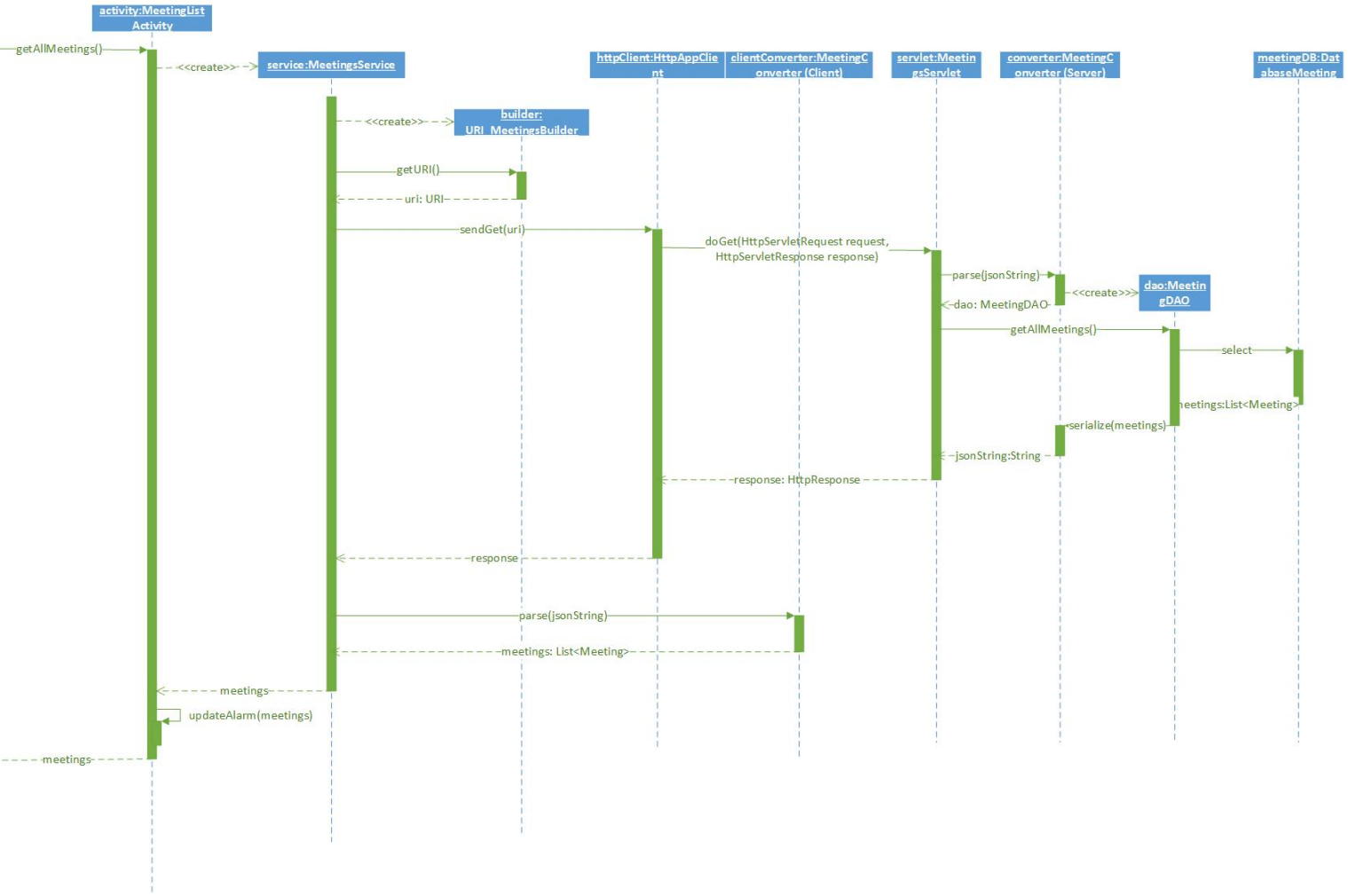
6.1 Allgemeiner Ablauf eines Requests des Clients

Im Folgenden wird der Ablauf eines Requests mithilfe eines Aktivitätsdiagramms beschrieben. Dies kann man auch auf dem Aktivitätsdiagramm nachvollziehen. Zunächst wir 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 sie gegebenen Parameter zu einem Objekt konvertiert. Nun wird der Request ausgeführt. Dies erzeugt ein DataAccesObject im Bereich der DAOs. Nun wird entweder die Konsistenz der Daten überprüft und in der Datenbank gespeichert oder die gefragten Daten dort abgerufen. Die Datenbank sicher diese auf der Festplatte. Sollten Objekte zurückgeliefert werden, wird bei den DAOs 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.


Client
HTTP Response verarbeiten
Benutzerschnittstelle aktualisieren
[weitere Operation]
[keine weitere Operation]
Operationen auf dem Client ausführen
HTTP Request(+Parameter)stellen, um Informationen auf dem Server zu hinterlegen oder abzurufen
Server
Servlets
Converter
Request entgegennehmen
mit HTTP Response (HTTP Statuscode +Parameter) antworten
Request ausführen
Paramter zu Objekten konvertieren
Objekte zu Paramter konvertieren
DataAccessObjects (DAOs)
DAO erzeugen
Konsistenz der Daten überprüfen und in der Datenbank speichern bzw. Daten aus der Datenbank abrufen
DAO mit Daten füllen
Datenbank
Daten auf der Festplatte sichern
[keine Objekte werden zurückgeliefert]
[Objekte werden zurückgeliefert]

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 den 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 Paramter 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(HttpRequest request, HttpResponse 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 geparst, die dann an die MeetingListActivity zurückgegeben wird. Die MeetingListActivity aktualisiert alle Alarme, löscht alle Alarme, die nicht mehr gültig sind, und erstellt die neuen.


activity:MeetingList Activity
getAllMeetings()
<<create>>
service:MeetingsService
<<create>>
builder: URI_MeetingsBuilder
getURI()
uri: URI
sendGet(uri)
httpClient:HttpAppClient
clientConverter:MeetingConverter (Client)
doGet(HttpServletRequest request, HttpServletResponse response)
servlet:MeetingsServlet
converter:MeetingConverter (Server)
parse(jsonString)
<<create>>
dao:MeetingDAO
dao: MeetingDAO
getAllMeetings()
meetingDB:DatabaseMeeting
select
meetings:List<Meeting>
serialize(meetings)
jsonString:String
response: HttpResponse
response
parse(jsonString)
meetings: List<Meeting>
meetings
updateAlarm(meetings)
meetings

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, dabei 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.

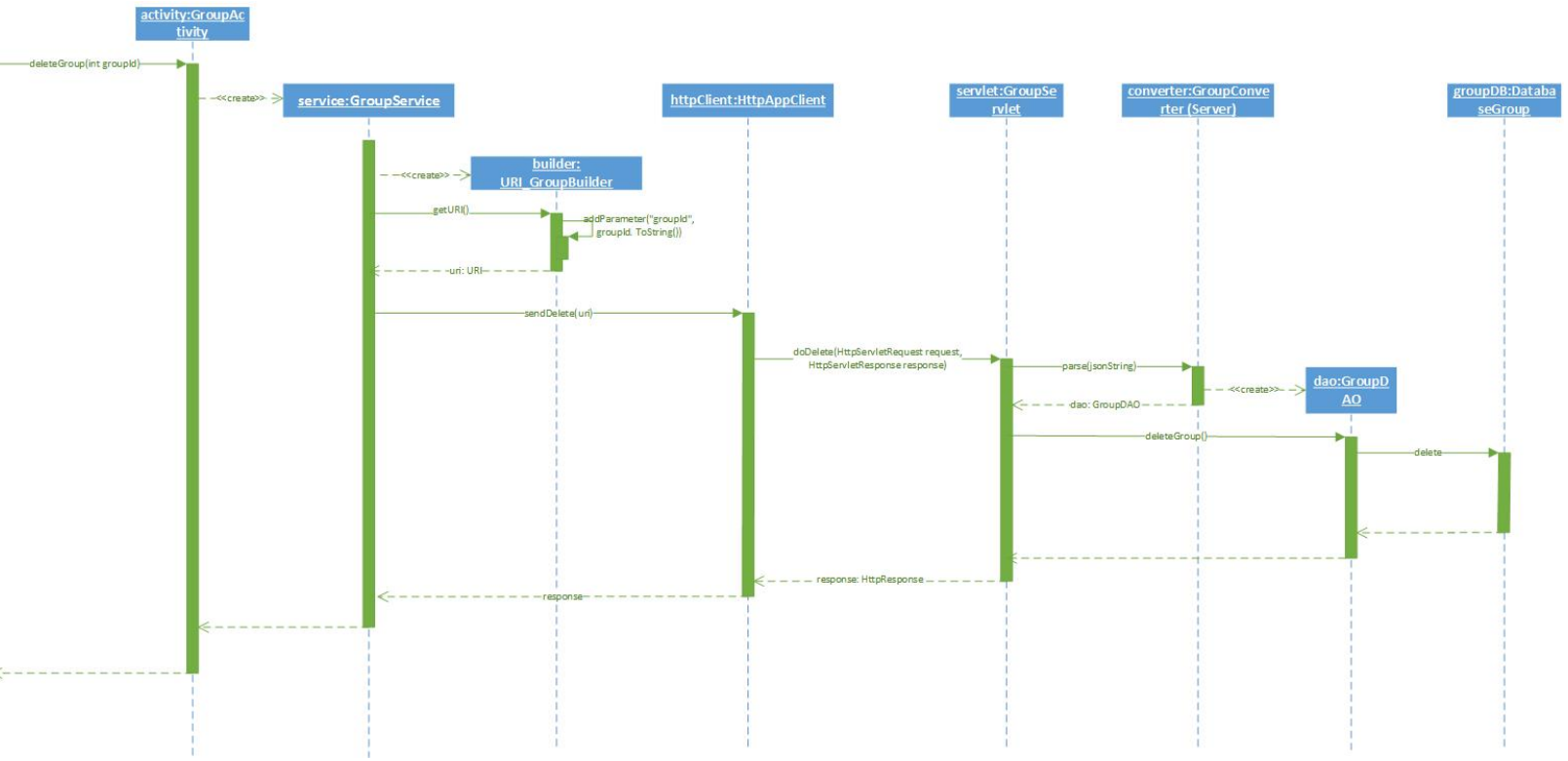
Nun schickt der Service einen HTTP DELETE Request mithilfe einer Instanz von `HttpAppClient` mit der Methode `sendDelete(uri)` an den Server, genauer gesagt an das zugehörige `GroupServlet`.

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 `DatabaseGroup` 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 Aufrtitt 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 in der ersten Version der App noch keine weiteren Administratoren hinzugefügt werden können. 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, dabei 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

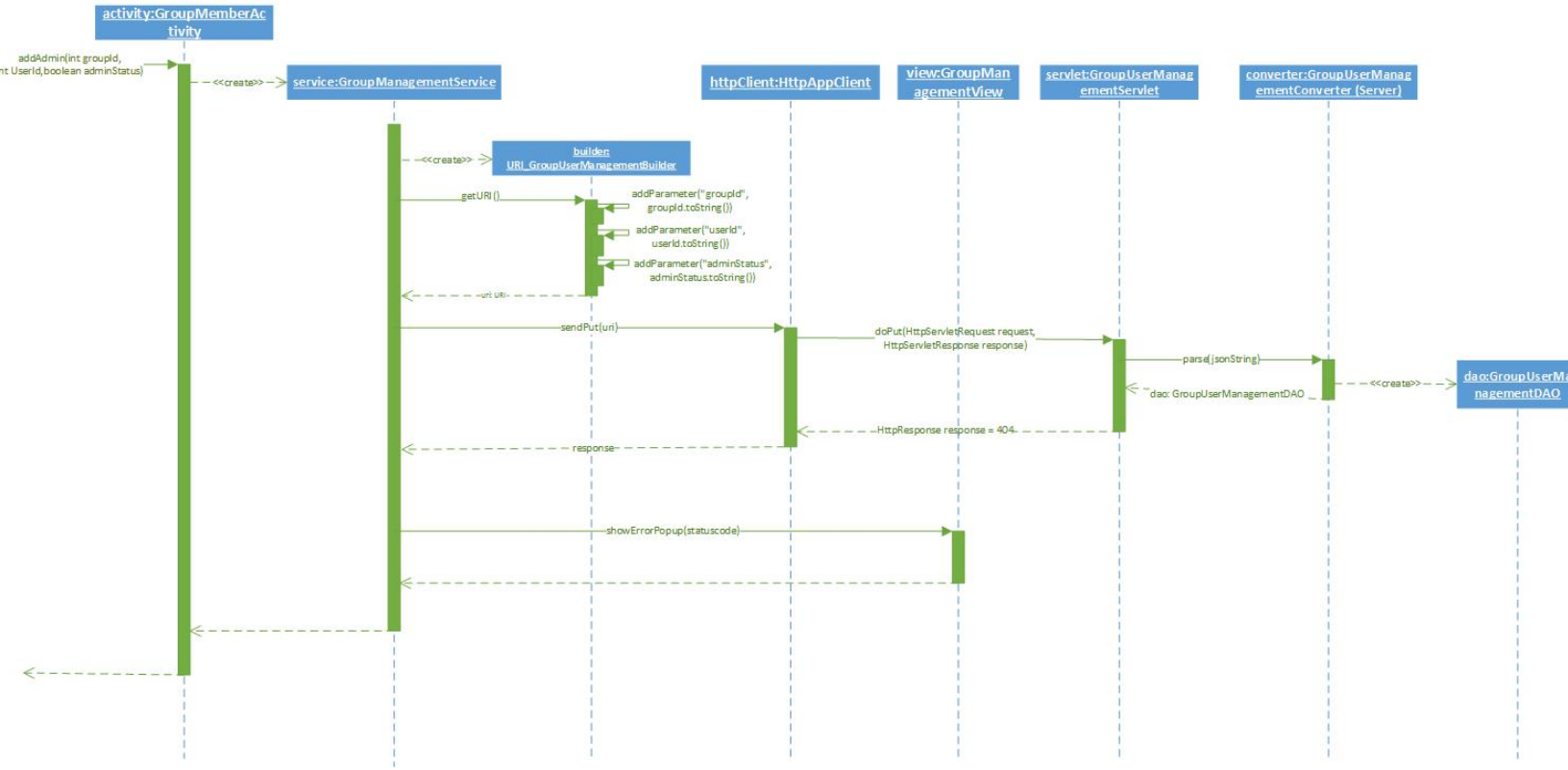
gesagt an das zugehörige `GroupUserManagementServlet`.

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 `AppHttpClient`s an den `GroupManagementService` zurück.

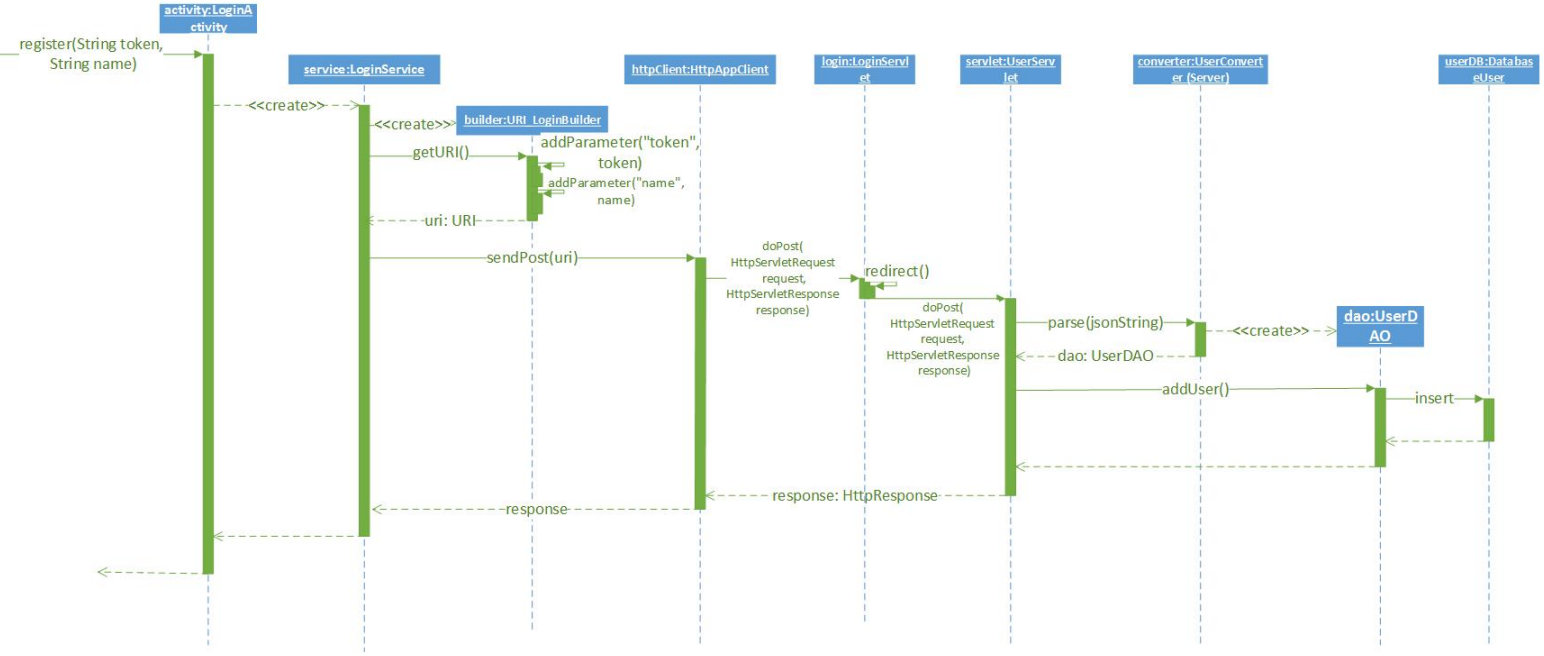
Nun muss der Client noch erfahren, ob sein Request erfolgreich war. Da zunächst keine weiteren Administratoren hinzugefügt werden können sollen, gibt die Methode `doPut(HttpServletRequest request HttpServletResponse response)`, die auf dem `GroupUserManagementServlet` ausgeführt wurde, den HTTP Response 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.


activity:GroupMemberActivity
addAdmin(int groupId,
nt UserId,boolean adminStatus)
<<create>>
service:GroupManagementService
httpClient:HttpAppClient
view:GroupManagementView
servlet:GroupUserManagementServlet
converter:GroupUserManagementConverter (Server)
<<create>>
builder:
URI_GroupUserManagementBuilder
getURI()
addParameter("groupId",
groupId.toString())
addParameter("userId",
userId.toString())
addParameter("adminStatus",
adminStatus.toString())
uri URI
sendPut(uri)
doPut(HttpServletRequest request,
HttpServletResponse response)
parse(jsonString)
<<create>>
dao: GroupUserManagementDAO
HttpResponse response = 404
response
showErrorPopup(statuscode)

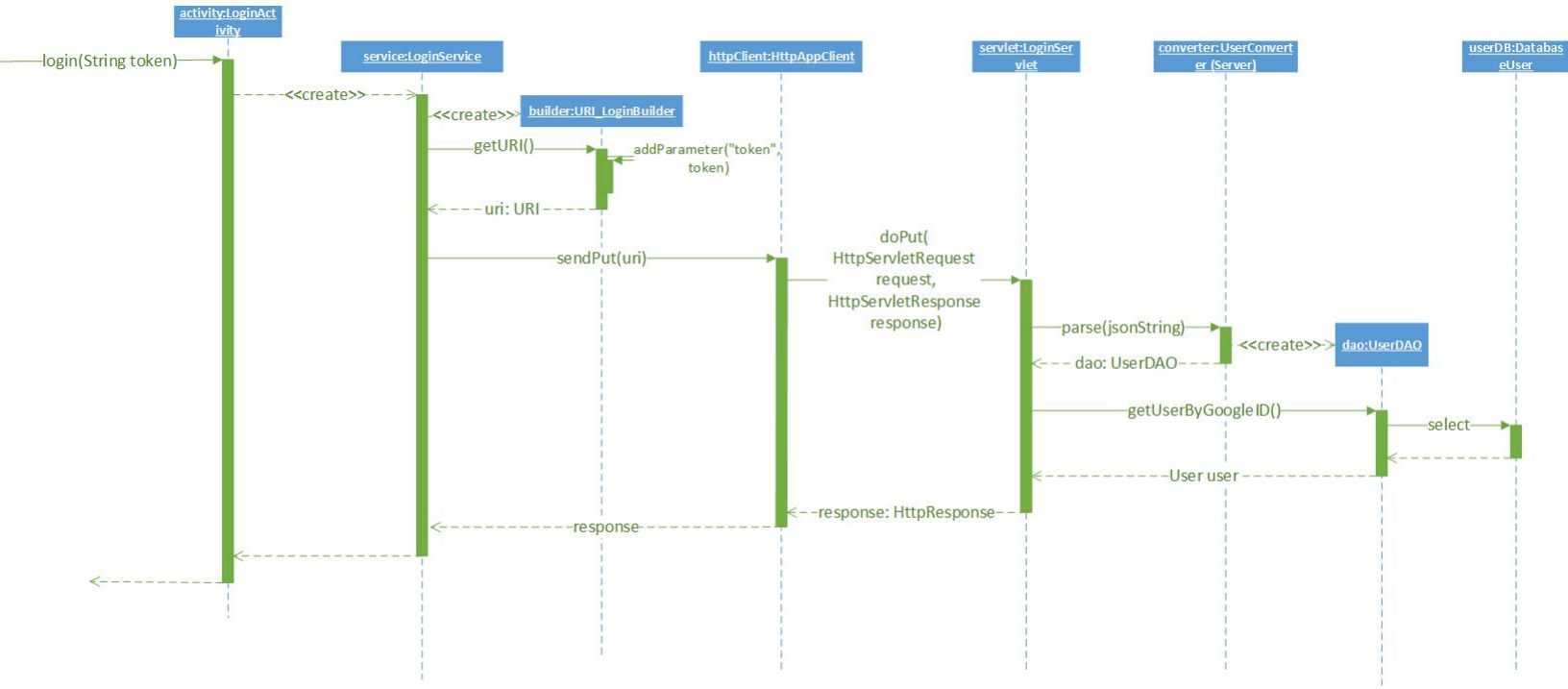
6.5 Ablauf der Registrierung in das System

Die Methode register(String token, String name) registriert einen Nutzer in System der App. Diese Methode wir in der LoginActivity des Clients ausgefürt. Als erstes erstellt die LoginActivity den zugehörigen LoginService, der im Hintergrund weiterläuft, damit die LoginAcivity nicht vollständig blockiert wird. Der LoginService erstellt einen zugehörigen URI_LoginBuilder builder, von den man eine Instanz URI uri zurückgegeben bekommt, wenn man auf ihm die Methode getURI() aufruft, dabei 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(HttpRequest request, HttpResponse response) aufgerufen, die den Request entgegen nimmt und an das UserServlet weiter reicht, dort wird noch einmal die Methode doPost(HttpRequest request, HttpResponse response) aufgerufen. Das UserServlet nimmt den Request entgegen und übergibt sie 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(HttpRequest request, HttpResponse response), die auf dem UserServlet ausgefürt 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 wir in der LoginActivity des Clients ausgeführt. Als erstes erstellt die LoginActivity den zugehörigen LoginService, der im Hintergrund weiterläuft, damit die LoginAcivity nicht vollständig blockiert wird. Der LoginService erstellt einen zugehörigen URI_LoginBuilder builder, von den man eine Instanz URI uri zurückgegeben bekommt, wenn man auf ihm die Methode getURI() aufruft, dabei 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(HttpRequest request, HttpResponse 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 muus der Client noch erfahren, ob sein Request erfolgreich war, also gibt die Methode doPost(HttpRequest request, HttpResponse response), die auf dem UserServlet ausgefürt wurde, den HTTP Response zusammen mit einem Statuscode, der angibt, ob alles fehlerfrei verlaufen ist, mithilfe des HttpAppClients an den LoginService zurück.


activity:LoginActivity
service:LoginService
builder:URI_LoginBuilder
httpClient:HttpAppClient
servlet:LoginServlet
converter:UserConverter (Server)
dao:UserDAO
userDB:DatabaseUser
login(String token)
<<create>>
<<create>>
getURI()
addParameter("token", token)
uri: URI
sendPut(uri)
doPut(HttpServletRequest request, HttpServletResponse response)
parse(jsonString)
<<create>>
dao: UserDAO
getUserByGoogleID()
select
User user
response: HttpResponse
response

7 Änderungen gegenüber dem Pflichtenheft

7.1 Gruppeneinladungen

In den funktionalen Anforderungen /F70/ und /F110/ werden beschrieben, dass der Benutzer eine Benachrichtugung 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/).

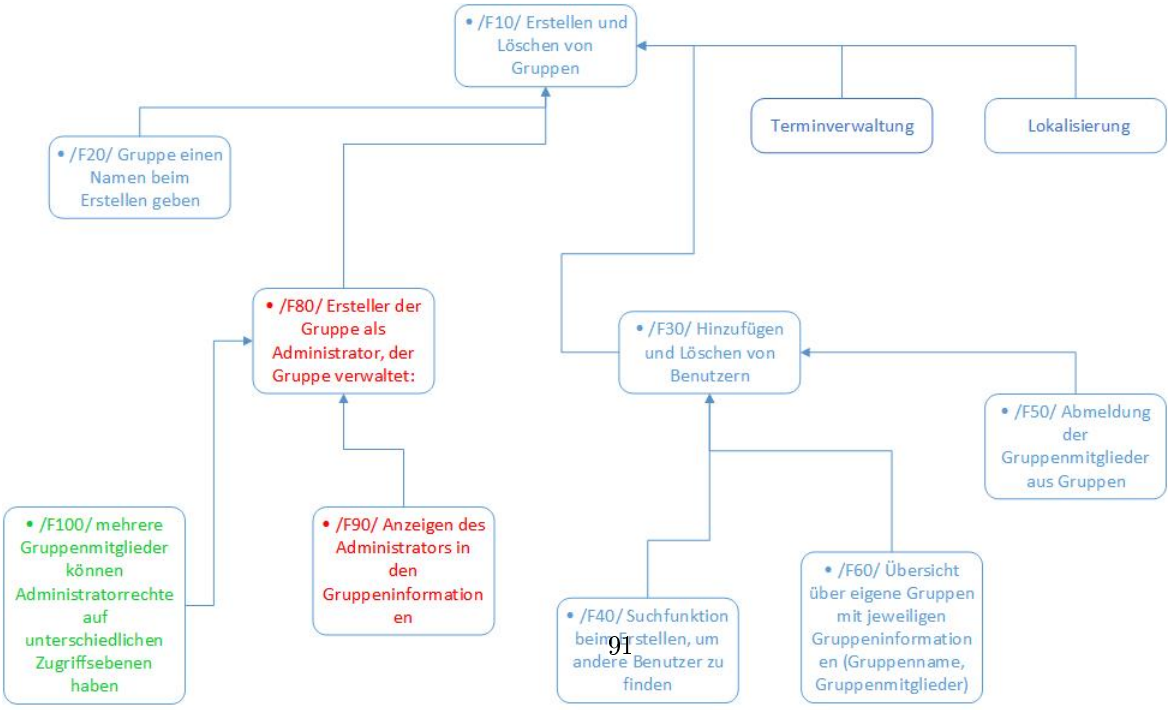
Diese funktionalen Anforderungen mit Priorität B (/F70/) und C (/F110/) werden verworfen, da es zu Problemen kommen kann, wenn eine Gruppe erstellt wird mit einem zeitnahen Termin und keiner von den eingeladenen Benutzern die Gruppeneinladung frühzeitig annimmt, sodass keiner Kenntnis von dem Termin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

7.2 Änderung der Prioritäten der funktionalen Anfoderungen

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.

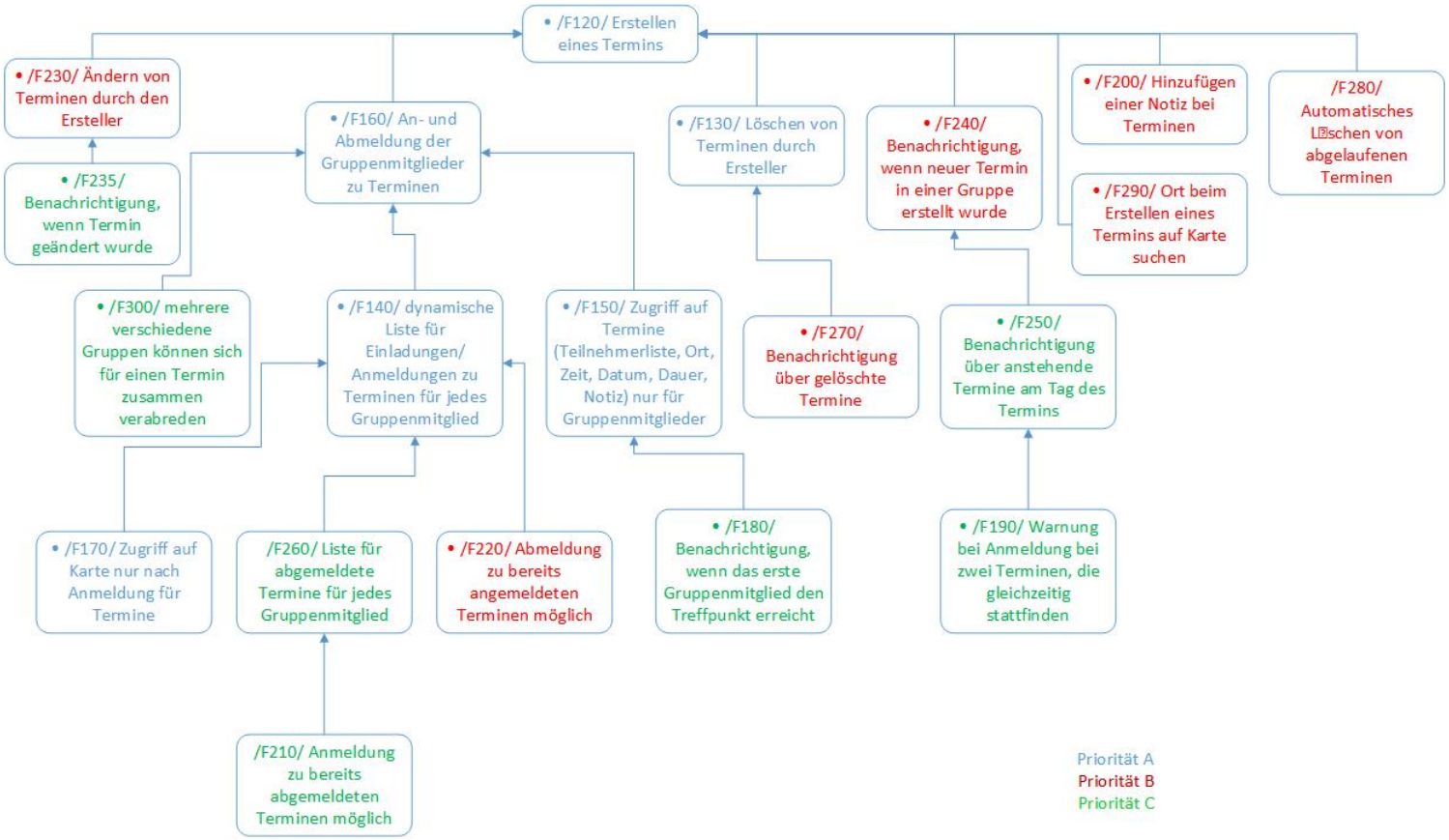
7.2.1 Gruppenverwaltung

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



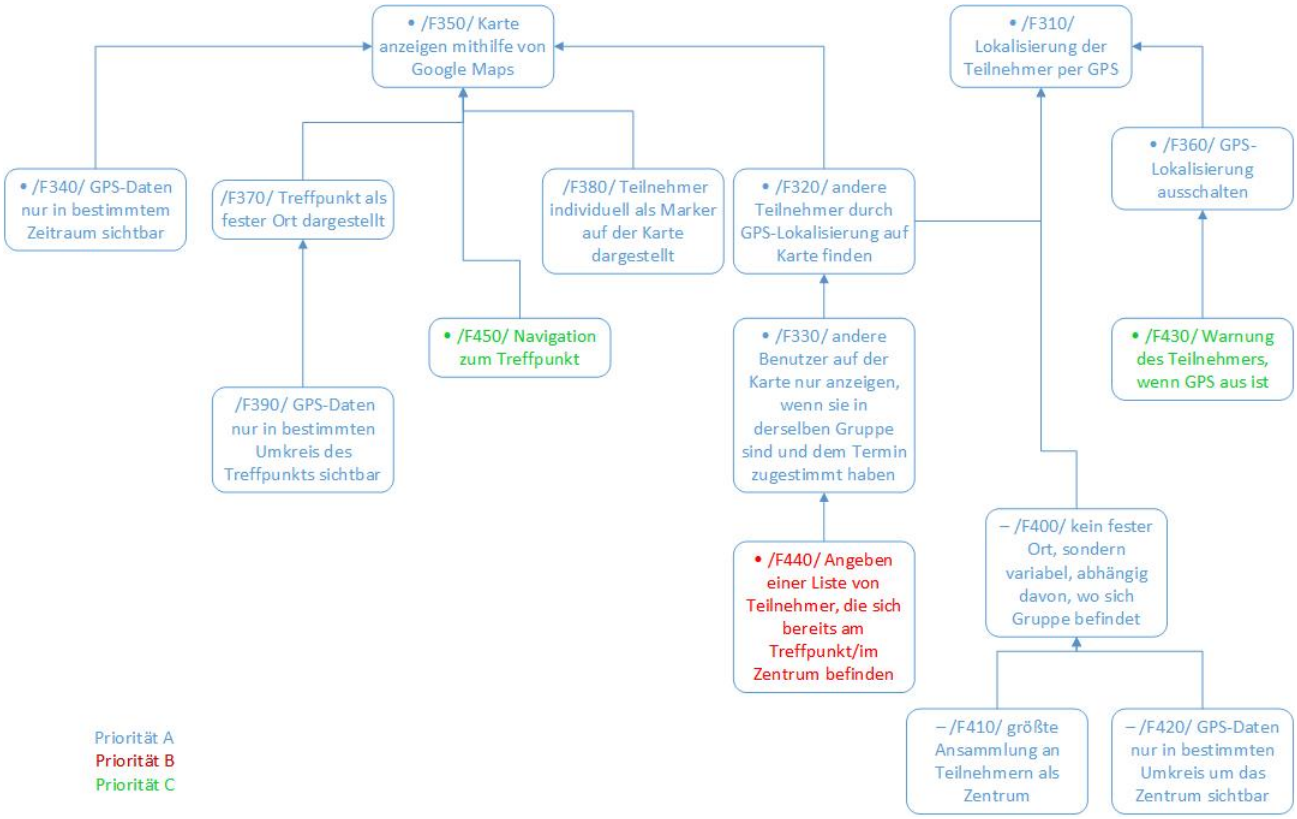
7.2.2 Terminverwaltung

- /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.



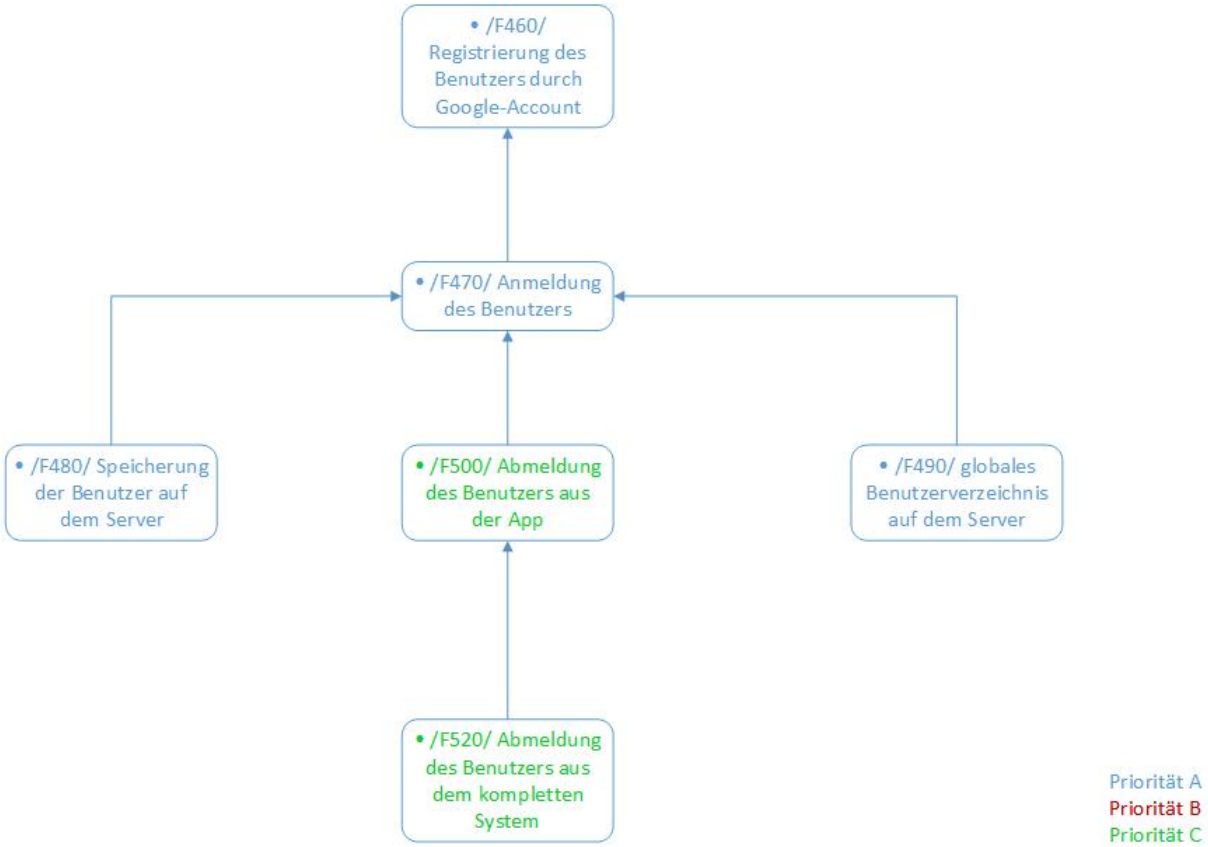
7.2.3 Lokalisierung

- /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.

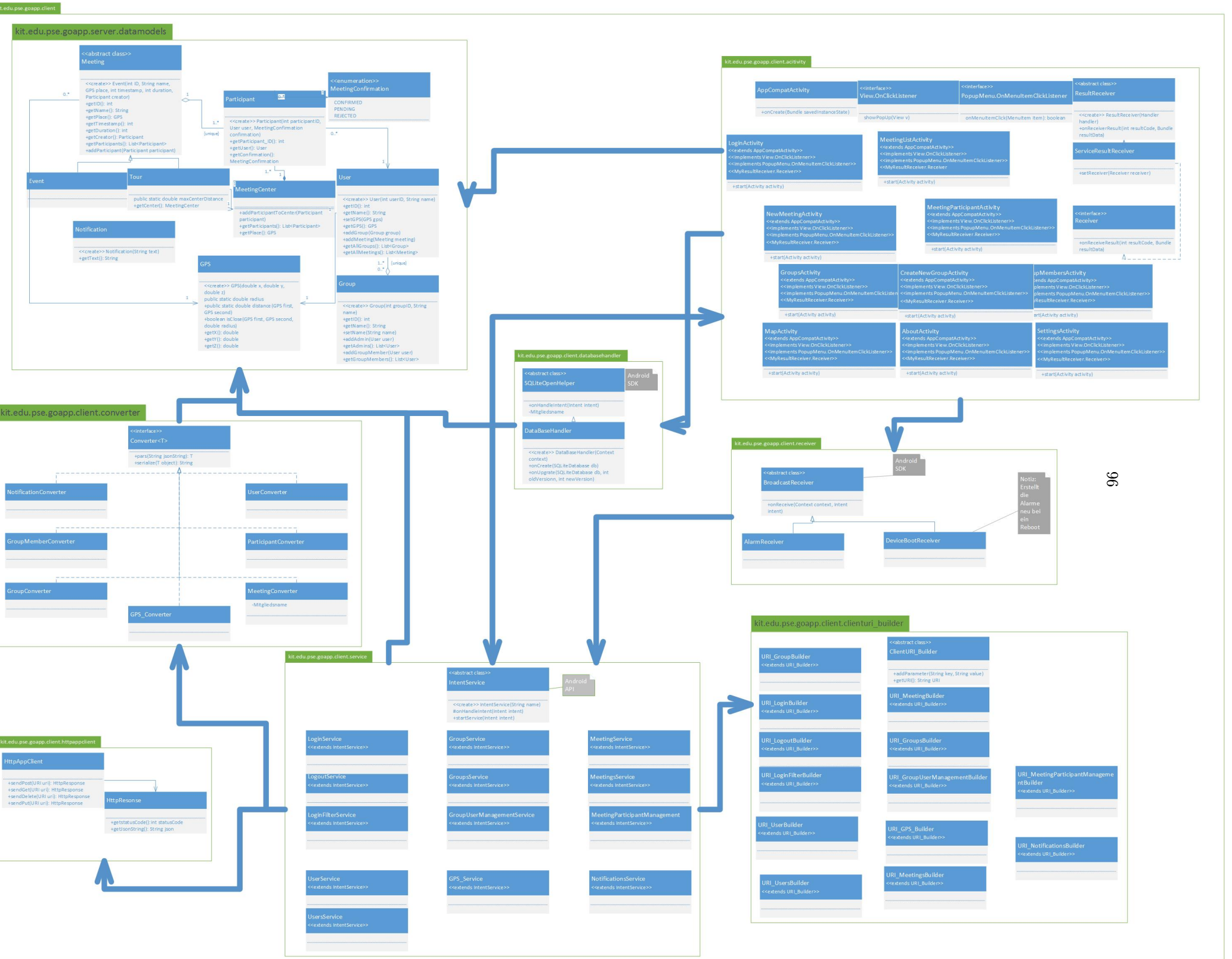


7.2.4 Benutzerverwaltung

- /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


kit.edu.pse.goapp.client
kit.edu.pse.goapp.server.datamodels
<<abstract class>>
Meeting
<<create>> Event(int ID, String name, GPS place, int timestamp, int duration, Participant creator)
+getID(): int
+getName(): String
+getPlace(): GPS
+getTimestamp(): int
+getDuration(): int
+getCreator(): Participant
+getParticipants(): List<Participant>
+addParticipant(Participant participant)
<<enumeration>>
MeetingConfirmation
CONFIRMED
PENDING
REJECTED
Participant
<<create>> Participant(int participantID, User user, MeetingConfirmation confirmation)
+getParticipant_ID(): int
+getUser(): User
+getConfirmation(): MeetingConfirmation
Event
Tour
public static double maxCenterDistance
+getCenter(): MeetingCenter
MeetingCenter
+addParticipantToCenter(Participant participant)
+getParticipants(): List<Participant>
+getPlace(): GPS
User
<<create>> User(int userID, String name)
+getID(): int
+getName(): String
+setGPS(GPS gps)
+getGPS(): GPS
+addGroup(Group group)
+addMeeting(Meeting meeting)
+getAllGroups(): List<Group>
+getAllMeetings(): List<Meeting>
Notification
<<create>> Notification(String text)
+getText(): String
GPS
<<create>> GPS(double x, double y, double z)
public static double radius
+public static double distance(GPS first, GPS second)
+boolean isClose(GPS first, GPS second, double radius)
+getX(): double
+getY(): double
+getZ(): double
Group
<<create>> Group(int groupID, String name)
+getID(): int
+getName(): String
+setName(String name)
+addAdmin(User user)
+getAdmins(): List<User>
+addGroupMember(User user)
+getGroupMembers(): List<User>
kit.edu.pse.goapp.client.acitivity
AppCompatActivity
+onCreate(Bundle savedInstanceState)
<<interface>>
View.OnClickListener
showPopUp(View v)
<<interface>>
PopupMenu.OnMenuItemClickListener
onMenuItemClick(MenuItem item): boolean
<<abstract class>>
ResultReceiver
<<create>> ResultReceiver(Handler handler)
+onReceiverResult(int resultCode, Bundle resultData)
ServiceResultReceiver
+setReceiver(Receiver receiver)
<<interface>>
Receiver
+onReceiveResult(int resultCode, Bundle resultData)
LoginActivity
<<extends AppCompatActivity>>
<<implements View.OnClickListener>>
<<implements PopupMenu.OnMenuItemClickListener>>
<<MyResultReceiver.Receiver>>
+start(Activity activity)
MeetingListActivity
<<extends AppCompatActivity>>
<<implements View.OnClickListener>>
<<implements PopupMenu.OnMenuItemClickListener>>
<<MyResultReceiver.Receiver
+start(Activity activity)
NewMeetingActivity
<<extends AppCompatActivity>>
<<implements View.OnClickListener>>
<<implements PopupMenu.OnMenuItemClickListener>>
<<MyResultReceiver.Receiver>>
+start(Activity activity)
MeetingParticipantActivity
<<extends AppCompatActivity>>
<<implements View.OnClickListener>>
<<implements PopupMenu.OnMenuItemClickListener>>
<<MyResultReceiver.Receiver>>
+start(Activity activity)
GroupsActivity
<<extends AppCompatActivity>>
<<implements View.OnClickListener>>
<<implements PopupMenu.OnMenuItemClickListener>>
<<MyResultReceiver.Receiver>>
+start(Activity activity)
CreateNewGroupActivity
<<extends AppCompatActivity>>
<<implements View.OnClickListener>>
<<implements PopupMenu.OnMenuItemClickListener>>
<<MyResultReceiver.Receiver>>
+start(Activity activity)
MapActivity
<<extends AppCompatActivity>>
<<implements View.OnClickListener>>
<<implements PopupMenu.OnMenuItemClickListener>>
<<MyResultReceiver.Receiver>>
+start(Activity activity)
AboutActivity
<<extends AppCompatActivity>>
<<implements View.OnClickListener>>
<<implements PopupMenu.OnMenuItemClickListener>>
<<MyResultReceiver.Receiver>>
+start(Activity activity)
SettingsActivity
<<extends AppCompatActivity>>
<<implements View.OnClickListener>>
<<implements PopupMenu.OnMenuItemClickListener>>
<<MyResultReceiver.Receiver>>
+start(Activity activity)
kit.edu.pse.goapp.client.converter
<<interface>>
Converter<T>
+pars(String jsonString): T
+serialize(T object): String
NotificationConverter
UserConverter
GroupMemberConverter
ParticipantConverter
GroupConverter
MeetingConverter
-Mitgliedsname
GPS_Converter
kit.edu.pse.goapp.client.databasehandler
<<abstract class>>
SQLiteOpenHelper
Android SDK
+onHandleIntent(Intent intent)
-Mitgliedsname
DataBaseHandler
<<create>> DataBaseHandler(Context context)
+onCreate(SQLiteDatabase db)
+onUpgrate(SQLiteDatabase db, int oldVersionn, int newVersion)
kit.edu.pse.goapp.client.receiver
Android SDK
<<abstract class>>
BroadcastReceiver
+onReceive(Context context, intent intent)
Notiz: Erstellt die Alarme neu bei ein Reboot
AlarmReceiver
DeviceBootReceiver
kit.edu.pse.goapp.client.httpappclient
HttpAppClient
+sendPost(URI uri): HttpResponse
+sendGet(URI uri): HttpResponse
+sendDelete(URI uri): HttpResponse
+sendPut(URI uri): HttpResponse
HttpResonse
+getstatusCode(): int statusCode
+getUsonString(): String json
kit.edu.pse.goapp.client.service
<<abstract class>>
IntentService
Android API
<<create>> IntentService(String name)
#onHandleIntent(Intent intent)
+startService(Intent intent)
LoginService
<<extends IntentService>>
LogoutService
<<extends IntentService>>
LoginFilterService
<<extends IntentService>>
UserService
<<extends IntentService>>
UsersService
<<extends IntentService>>
GroupService
<<extends IntentService>>
GroupsService
<<extends IntentService>>
GroupUserManagementService
<<extends IntentService>>
GPS_Service
<<extends IntentService>>
MeetingService
<<extends IntentService>>
MeetingsService
<<extends IntentService>>
MeetingParticipantManagement
<<extends IntentService>>
NotificationsService
<<extends IntentService>>
kit.edu.pse.goapp.client.clienturi_builder
URI_GroupBuilder
<<extends URI_Builder>>
URI_LoginBuilder
<<extends URI_Builder>>
URI_LogoutBuilder
<<extends URI_Builder>>
URI_LoginFilterBuilder
<<extends URI_Builder>>
URI_UserBuilder
<<extends URI_Builder>>
URI_UsersBuilder
<<extends URI_Builder>>
<<abstract class>>
ClientURI_Builder
+addParameter(String key, String value)
+getURI(): String URI
URI_MeetingBuilder
<<extends URI_Builder>>
URI_GroupsBuilder
<<extends URI_Builder>>
URI_GroupUserManagementBuilder
<<extends URI_Builder>>
URI_GPS_Builder
<<extends URI_Builder>>
URI_MeetingsBuilder
<<extends URI_Builder>>
URI_MeetingParticipantManagementBuilder
<<extends URI_Builder>>
URI_NotificationsBuilder
<<extends URI_Builder>>
96


Kit.edu.pse.goapp.server
kit.edu.pse.goapp.server.datamodels
<<abstract class>>
Meeting
<<create>> Event(int ID, String name, GPS place, int timestamp, int duration, Participant creator)
+getID(): int
+getName(): String
+getPlace(): GPS
+getTimestamp(): int
+getDuration(): int
+getCreator(): Participant
+getParticipants(): List<Participant>
+addParticipant(Participant participant)
Participant
<<create>> Participant(int participantID, User user, MeetingConfirmation confirmation)
+getParticipant_ID(): int
+getUser(): User
+getConfirmation(): MeetingConfirmation
<<enumeration>>
MeetingConfirmation
CONFIRMED
PENDING
REJECTED
Event
Tour
public static double maxCenterDistance
+getCenter(): MeetingCenter
MeetingCenter
+addParticipantToCenter(Participant participant)
+removeParticipantFromCenter(Participant participant)
+getParticipants(): List<Participant>
+getPlace(): GPS
User
<<create>> User(int userID, String name)
+getID(): int
+getName(): String
+setGPS(GPS gps)
+getGPS(): GPS
+addGroup(Group group)
+addMeeting(Meeting meeting)
+getAllGroups(): List<Group>
+getAllMeetings(): List<Meeting>
GPS
<<create>> GPS(double x, double y, double z)
public static double radius
+public static double distance(GPS first, GPS second)
+boolean isClose(GPS first, GPS second, double radius)
+getX(): double
+getY(): double
+getZ(): double
Group
<<create>> Group(int groupID, String name)
+getID(): int
+getName(): String
+setName(String name)
+addAdmin(User user)
+getAdmins(): List<User>
+addGroupMember(User user)
+getGroupMembers(): List<User>
Notification
<<create>> Notification(String text)
+getText(): String
Kit.edu.pse.goapp.server.servlets
GroupServlet
+doPost(HttpServletRequest request, HttpServletResponse response)
+doGet(HttpServletRequest request, HttpServletResponse response)
+doPut(HttpServletRequest request, HttpServletResponse response)
+doDelete(HttpServletRequest request, HttpServletResponse response)
LogoutServlet
+doDelete(HttpServletRequest request, HttpServletResponse response)
LoginFilterServlet
+doFilter(ServletRequest req, ServletResponse res, FilterChain chain)
GroupUserManagementServlet
+doPost(HttpServletRequest request, HttpServletResponse response)
+doGet(HttpServletRequest request, HttpServletResponse response)
+doPut(HttpServletRequest request, HttpServletResponse response)
+doDelete(HttpServletRequest request, HttpServletResponse response)
NotificationsServlet
+doGet(HttpServletRequest request, HttpServletResponse response)
+doPut(HttpServletRequest request, HttpServletResponse response)
GPS_Servlet
+doPut(HttpServletRequest request, HttpServletResponse response)
UserServlet
+doPost(HttpServletRequest request, HttpServletResponse response)
+doGet(HttpServletRequest request, HttpServletResponse response)
+doPut(HttpServletRequest request, HttpServletResponse response)
+doDelete(HttpServletRequest request, HttpServletResponse response)
MeetingsServlet
+doGet(HttpServletRequest request, HttpServletResponse response)
UsersServlet
+doGet(HttpServletRequest request, HttpServletResponse response)
LoginServlet
+doPost(HttpServletRequest request, HttpServletResponse response)
+doGet(HttpServletRequest request, HttpServletResponse response)
+doPut(HttpServletRequest request, HttpServletResponse response)
MeetingServlet
+doPost(HttpServletRequest request, HttpServletResponse response)
+doGet(HttpServletRequest request, HttpServletResponse response)
+doPut(HttpServletRequest request, HttpServletResponse response)
+doDelete(HttpServletRequest request, HttpServletResponse response)
MeetingParticipantManagementServlet
+doPost(HttpServletRequest request, HttpServletResponse response)
+doGet(HttpServletRequest request, HttpServletResponse response)
+doPut(HttpServletRequest request, HttpServletResponse response)
+doDelete(HttpServletRequest request, HttpServletResponse response)
GroupsServlet
+doGet(HttpServletRequest request, HttpServletResponse response)
kit.edu.pse.goapp.server.converter
<<interface>>
Converter<T>
+parse(String jsonString): T
+serialize(T dao): String
NotificationConverter
UserConverter
GroupMemberConverter
ParticipantConverter
GroupConverter
GPS_Converter
MeetingConverter
kit.edu.pse.goapp.server.DAOs
<<interface>>
GroupDAO
+addGroup()
+getAllGroups() : List<Group>
+updateGroup()
+deleteGroup()
+getGroupByID(): Group
GroupDAOImpl
<<create>> GroupDAOImpl(int ID, String name)
<<interface>>
GroupMemberDAO
+addMember()
+deleteMember()
+updateMember()
+getAllMembers(): List<User>
+getAllUserGroups(): List<Group>
+setAdmin()
+getAllAdminGroups(): List<Group>
GroupMembersDAOImpl
<<create>> GroupMembersDAOImpl(int userID, int groupID, boolean isAdmin)
<<interface>>
MeetingDAO
+addMeeting()
+getAllMeetings(): List<Meeting>
+updateMeeting()
+deleteMeeting()
+getMeetingByID(): Meeting
MeetingDAOImpl
<<create>> MeetingDAOImpl(int meetingID, int name, GPS place, long timestamp, int duration, String type, int creatorID)
<<interface>>
ParticipantDAO
+addParticipant()
+deleteParticipant()
+getAllParticipants(): List<Participant>
+getParticipantByID(): Participant
+updateParticipant()
ParticipantDAOImpl
<<create>> ParticipantDAOImpl(int participantID, int userID, int meetingID, MeetingConfirmation status)
<<interface>>
UserDAO
+addUser()
+deleteUser()
+updateUser()
+getAllUsers(): List<User>
+getUserByID(): User
+getGoogleID(): String
+getNotificationStatus(): boolean
+getUserByGoogleID(): User
UserDAOImpl
<<create>> UserDAOImpl(int userID, String name, String googleID, boolean notificationStatus)
<<interface>>
GPS_DAO
+userSetGPS()
+userGetGPS(): GPS
GPS_DAO_Impl
<<create>> GPS_DAO_Impl(int userID, double x, double y, double z)
<<interface>>
NotificationDAO
+userGetNotification(): List<Notification>
NotificationDAOImpl
<<create>> NotificationDAOImpl(int userID)
<<uses>>
kit.edu.pse.goapp.database
Group
Group_ID
Name
Group Members
Group_ID
User_ID
isAdmin
User
User_ID
Name
Google_ID
Notification enabled
Meeting
Meeting_ID
Name
Place_X
Place_Y
Place_Z
Timestamp
Duration
Type
Participant
Participant_ID
User_ID
Meeting_ID