Entwurfsdokument

Praktikum der Softwareentwicklung
Sommersemester 2017

Android GO! App

- Gruppe 3 -

erstellt von:
Arsenii Dunaev
Florian Kröger
Tina Maria Strößner
Volodymyr Shpylka

09.07.17

Zusammenfassung

Die Android App GO! ist eine mobile Applikation, die speziell zur Organisation von Treffen (z. B. gemeinsames Essen im Café oder in der Mensa) entwickelt wird. Beim erfolgreichen gemeinsamen Losgehen wird der gemittelte GPS-Standort von Mitgliedern der Gruppe angezeigt.

Dieses Dokument erläutert den Entwurf des Systems auf der Grundlage des Pflichtenhefts. Dazu wird zunächst die Modulstruktur und der Architekturstil erläutert. Danach werden die Klassen, Attribute und Methoden der Anwendung detailliert beschreiben. Das Zusammenspiel der einzelnen Klassen wird dargestellt im Klassendiagramm. Weitere Abschnitte erläutern die Client-Server Schnittstelle, das Datenbankschema er Applikation sowie typische Ausführungsabläufe.

Inhaltsverzeichnis

1 Änderungen zum Pflichtenheft **12**

2 Architekturstil und Paketstruktur **13**
2.1 Client . . . 13
2.1.1 Views . . . 13
2.1.2 Entities . . . 14
2.1.3 ViewModell . . . 14
2.1.4 ServerCommunication . . . 14
2.1.5 ServerCommands . . . 15
2.1.6 Repositories . . . 15
2.2 Server . . . 15
2.2.1 CommunicationLayer . . . 16
2.2.2 BusinessLayer . . . 17
2.2.3 PersistenceLayer . . . 17

3 verwendete Entwurfsmuster **18**
3.1 Schablonenmethode für SignInHelper . . . 18
3.2 Beobachter zum Aktualisieren des UI . . . 18
3.3 Beobachter zum Weiterleiten von Änderungen der Datenbasis . . . 18
3.4 DAO-Pattern zur Persistierung von Daten . . . 19
3.5 Vermittler zur Koordination von Datenzugriffen . . . 20
3.6 Strategiemuster zur Kapselung des Clustering-Algorithmus . . . 20
3.7 Fassade zur Vereinfachung des Server Interfaces . . . 20
3.8 Dekorierer zur Erweiterung der Aktivitäten für Sonderbenutzer . . . 21
3.9 Command Muster zur Bearbeitung der Server Messages auf der Client Seite . . . 21
3.10 Singleton in UserViewModel . . . 21

4 Klassenübersicht - Client **23**

5 Package ServerCommands **24**
5.1 Klasse AdminAddedCommand . . . 24
5.1.1 Deklaration . . . 24
5.1.2 Konstruktoren . . . 24
5.1.3 Methoden . . . 24
5.1.4 Von ServerCommand vererbte Methoden . . . 25
5.2 Klasse GoAddedCommand . . . 25
5.2.1 Deklaration . . . 25
5.2.2 Konstruktoren . . . 25
5.2.3 Methoden . . . 25
5.2.4 Von ServerCommand vererbte Methoden . . . 25
5.3 Klasse GoEditedCommand . . . 25
5.3.1 Deklaration . . . 25
5.3.2 Konstruktoren . . . 25
5.3.3 Methoden . . . 26
5.3.4 Von ServerCommand vererbte Methoden . . . 26
5.4 Klasse GoRemovedCommand . . . 26
5.4.1 Deklaration . . . 26
5.4.2 Konstruktoren . . . 26
5.4.3 Methoden . . . 26

5.4.4 Von ServerCommand vererbte Methoden . . . 26
5.5 Klasse GroupEditedCommand . . . 26
5.5.1 Deklaration . . . 27
5.5.2 Konstruktoren . . . 27
5.5.3 Methoden . . . 27
5.5.4 Von ServerCommand vererbte Methoden . . . 27
5.6 Klasse GroupRemovedCommand . . . 27
5.6.1 Deklaration . . . 27
5.6.2 Konstruktoren . . . 27
5.6.3 Methoden . . . 27
5.6.4 Von ServerCommand vererbte Methoden . . . 28
5.7 Klasse GroupRequestReceivedCommand . . . 28
5.7.1 Deklaration . . . 28
5.7.2 Konstruktoren . . . 28
5.7.3 Methoden . . . 28
5.7.4 Von ServerCommand vererbte Methoden . . . 28
5.8 klasse MemberAddedCommand . . . 28
5.8.1 Deklaration . . . 28
5.8.2 Konstruktoren . . . 28
5.8.3 Methoden . . . 29
5.8.4 Von ServerCommand vererbte Methoden . . . 29
5.9 Klasse MemberRemovedCommand . . . 29
5.9.1 Deklaration . . . 29
5.9.2 Konstruktoren . . . 29
5.9.3 Methoden . . . 29
5.9.4 Von ServerCommand vererbte Methoden . . . 29
5.10 Klasse RequestDeniedCommand . . . 30
5.10.1 Deklaration . . . 30
5.10.2 Konstruktoren . . . 30
5.10.3 Methoden . . . 30
5.10.4 Von ServerCommand vererbte Methoden . . . 30
5.11 Klasse ServerCommand . . . 30
5.11.1 Deklaration . . . 30
5.11.2 All known subclasses . . . 30
5.11.3 Konstruktoren . . . 31
5.11.4 Methoden . . . 31
5.12 Klasse StatusChangedCommand . . . 31
5.12.1 Deklaration . . . 31
5.12.2 Konstruktoren . . . 31
5.12.3 Methoden . . . 31
5.12.4 Von ServerCommand vererbte Methoden . . . 31
5.13 Klasse UserDeletedCommand . . . 32
5.13.1 Deklaration . . . 32
5.13.2 Konstruktoren . . . 32
5.13.3 Methoden . . . 32
5.13.4 Von ServerCommand vererbte Methoden . . . 32

6 Package Login **32**
6.1 Klasse FirebaseSignInHelper . . . 32
6.1.1 Deklaration . . . 32
6.1.2 Konstruktoren . . . 33
6.1.3 Methoden . . . 33
6.1.4 Von SignInHelper vererbte Methoden . . . 34
6.2 Klasse GoSignInHelper . . . 34
6.2.1 Deklaration . . . 34
6.2.2 Konstruktoren . . . 34
6.2.3 Methoden . . . 34
6.2.4 Von SignInHelper vererbte Methoden . . . 35
6.3 Klasse SignInHelper . . . 35
6.3.1 Deklaration . . . 35
6.3.2 All known subclasses . . . 35
6.3.3 statische Felder . . . 35
6.3.4 Konstruktoren . . . 35
6.3.5 Methoden . . . 35

7 Package ViewModel **36**
7.1 Klasse GoViewModel . . . 36
7.1.1 Deklaration . . . 37
7.1.2 Konstruktoren . . . 37
7.1.3 Methoden . . . 37
7.2 Klasse GroupListViewModel . . . 37
7.2.1 Deklaration . . . 37
7.2.2 statische Felder . . . 37
7.2.3 Konstruktoren . . . 38
7.2.4 Methoden . . . 38
7.3 Klasse GroupViewModel . . . 38
7.3.1 Deklaration . . . 38
7.3.2 Konstruktoren . . . 38
7.3.3 Methoden . . . 38
7.4 Class UserViewModel . . . 39
7.4.1 Deklaration . . . 39
7.4.2 Konstruktoren . . . 39
7.4.3 Methoden . . . 39

8 Package Model.entities **40**
8.1 Class Cluster . . . 40
8.1.1 Deklaration . . . 40
8.1.2 Konstruktoren . . . 41
8.1.3 Methoden . . . 41
8.2 Klasse Go . . . 41
8.2.1 Deklaration . . . 41
8.2.2 Konstruktoren . . . 41
8.2.3 Methoden . . . 42
8.3 Klasse Group . . . 44
8.3.1 Deklaration . . . 44
8.3.2 statische Felder . . . 44
8.3.3 Konstruktoren . . . 44
8.3.4 Methoden . . . 44
8.4 Class GroupMembership . . . 45

8.4.1 Deklaration . . . 45
8.4.2 Konstruktoren . . . 45
8.4.3 Methoden . . . 46
8.5 Klasse User . . . 46
8.5.1 Deklaration . . . 46
8.5.2 Konstruktoren . . . 46
8.5.3 Methoden . . . 47
8.6 Klasse UserGoStatus . . . 47
8.6.1 Deklaration . . . 48
8.6.2 Konstruktoren . . . 48
8.6.3 Methoden . . . 48

9 Package Model **48**
9.1 Enum Status . . . 49
9.1.1 Deklaration . . . 49
9.1.2 statische Felder . . . 49
9.1.3 Methoden . . . 49
9.1.4 von Enum vererbte Methoden . . . 49

10 Package ServerCommunication.downstream **49**
10.1 Klasse MessageReceiver . . . 49
10.1.1 Deklaration . . . 50
10.1.2 Konstruktoren . . . 50
10.1.3 Methoden . . . 50
10.2 Klasse TokenService . . . 50
10.2.1 Deklaration . . . 51
10.2.2 Konstruktoren . . . 51
10.2.3 Methoden . . . 51

11 Package ServerCommunication.upstream **51**
11.1 Interface TomcatRestApi . . . 51
11.1.1 Deklaration . . . 51
11.1.2 Methoden . . . 51

12 Package Repositories **53**
12.1 Klasse GoRepository . . . 53
12.1.1 Deklaration . . . 53
12.1.2 Konstruktoren . . . 53
12.1.3 Methoden . . . 54
12.1.4 von Repository geerbte Methoden . . . 54
12.2 Klasse GroupRepository . . . 54
12.2.1 Deklaration . . . 54
12.2.2 Konstruktoren . . . 54
12.2.3 Methoden . . . 55
12.2.4 von Repository geerbte Methoden . . . 55
12.3 Klasse Repository . . . 55
12.3.1 Deklaration . . . 56
12.3.2 Unterklassen . . . 56
12.3.3 Konstruktoren . . . 56
12.3.4 Methoden . . . 56
12.4 Klasse UserRepository . . . 56
12.4.1 Deklaration . . . 56

12.4.2 Konstruktoren . . . 56
12.4.3 Methoden . . . 57
12.4.4 von Repository geerbte Methoden . . . 57

13 Package View **57**
13.1 Klasse BaseActivity . . . 57
13.1.1 Deklaration . . . 57
13.1.2 Konstruktoren . . . 57
13.2 Klasse EditGoActivity . . . 57
13.2.1 Deklaration . . . 58
13.2.2 Konstruktoren . . . 58
13.3 Klasse EditGroupActivity . . . 58
13.3.1 Deklaration . . . 58
13.3.2 Konstruktoren . . . 58
13.4 Klasse GoDetailActivity . . . 58
13.4.1 Deklaration . . . 58
13.4.2 Unterklassen . . . 58
13.4.3 Konstruktoren . . . 58
13.4.4 Methoden . . . 59
13.5 Klasse GoDetailActivityOwner . . . 59
13.5.1 Deklaration . . . 59
13.5.2 Konstruktoren . . . 59
13.5.3 Methoden . . . 59
13.5.4 Members inherited from class GoDetailActivity . . . 60
13.6 Klasse GroupDetailActivity . . . 60
13.6.1 Deklaration . . . 60
13.6.2 Unterklassen . . . 60
13.6.3 Konstruktoren . . . 60
13.6.4 Methoden . . . 60
13.7 Klasse GroupDetailActivityOwner . . . 61
13.7.1 Deklaration . . . 61
13.7.2 Konstruktoren . . . 61
13.7.3 Methoden . . . 61
13.7.4 von GroupDetailActivity vererbte Methoden und Feler . . . 61
13.8 Klasse GroupListActivity . . . 61
13.8.1 Deklaration . . . 61
13.8.2 Konstruktoren . . . 61
13.8.3 Methoden . . . 62
13.9 Klasse InformationActivity . . . 62
13.9.1 Deklaration . . . 62
13.9.2 Konstruktoren . . . 62
13.10 Klasse SettingsActivity . . . 63
13.10.1 Deklaration . . . 63
13.10.2 Konstruktoren . . . 63
13.10.3 Methoden . . . 63
13.11 Klasse SignInActivity . . . 63
13.11.1 Deklaration . . . 63
13.11.2 Konstruktoren . . . 63
13.11.3 Methoden . . . 63

14 Package RecyclerView **64**
14.1 Interface OnListItemClicked . . . 64
14.1.1 Deklaration . . . 64
14.1.2 Subinterfaces . . . 64
14.1.3 Klassen, die das Interface implementieren . . . 64
14.1.4 Methoden . . . 64
14.2 Klasse ListAdapter . . . 64
14.2.1 Deklaration . . . 65
14.2.2 statische Felder . . . 65
14.2.3 Konstruktoren . . . 65
14.2.4 Methoden . . . 65
14.3 Klasse ListViewHolder . . . 66
14.3.1 Deklaration . . . 66
14.3.2 statische Felder . . . 66
14.3.3 Konstruktoren . . . 66
14.3.4 Methoden . . . 67

15 Package RecyclerView.listItems **67**
15.1 Interface ListItem . . . 67
15.1.1 Deklaration . . . 67
15.1.2 Subinterfaces . . . 67
15.1.3 Klassen, die das Interface implementieren . . . 67
15.1.4 Methoden . . . 67
15.2 Klasse GOListItem . . . 68
15.2.1 Deklaration . . . 69
15.2.2 Konstruktoren . . . 69
15.2.3 Methoden . . . 69
15.3 Klasse GroupListItem . . . 70
15.3.1 Deklaration . . . 70
15.3.2 Konstruktoren . . . 70
15.3.3 Methoden . . . 71
15.4 Klasse UserMailListItem . . . 72
15.4.1 Deklaration . . . 72
15.4.2 Konstruktoren . . . 72
15.4.3 Methoden . . . 73
15.5 Klasse UserStatusListItem . . . 74
15.5.1 Deklaration . . . 74
15.5.2 Konstruktoren . . . 74
15.5.3 Methoden . . . 74

16 Klassenübersicht - Server **76**

17 Package PersistenceLayer **76**
17.1 Klasse GoEntity . . . 77
17.1.1 Deklaration . . . 77
17.1.2 Konstruktoren Auflistung . . . 77
17.1.3 Methoden Auflistung . . . 77
17.1.4 Attribute . . . 77
17.1.5 Konstruktoren . . . 79
17.1.6 Methoden . . . 79
17.2 Klasse GroupEntity . . . 81
17.2.1 Deklaration . . . 81

17.2.2 Konstruktoren Auflistung . . . 81
17.2.3 Methoden Auflistung . . . 81
17.2.4 Attribute . . . 81
17.2.5 Konstruktoren . . . 82
17.2.6 Methoden . . . 83
17.3 Enum Status . . . 84
17.3.1 Deklaration . . . 84
17.3.2 Attribute . . . 84
17.3.3 Methoden Auflistung . . . 84
17.3.4 Felder . . . 84
17.3.5 Methoden . . . 85
17.3.6 Von der Klasse Enum vererbte Methoden . . . 85
17.4 Klasse UserEntity . . . 85
17.4.1 Deklaration . . . 85
17.4.2 Konstruktoren Auflistung . . . 85
17.4.3 Methoden Auflistung . . . 86
17.4.4 Konstruktoren . . . 86
17.4.5 Methoden . . . 86

18 Package PersistenceLayer.daos **87**
18.1 Interface AbstractDao . . . 88
18.1.1 Deklaration . . . 88
18.1.2 Subinterfaces . . . 88
18.1.3 Klassen, die das Interface implementieren . . . 88
18.1.4 Methoden Auflistung . . . 88
18.1.5 Methoden . . . 89
18.2 Interface GoDao . . . 90
18.2.1 Deklaration . . . 90
18.2.2 Subinterfaces . . . 90
18.2.3 All classes known to implement interface . . . 90
18.2.4 Methoden Auflistung . . . 90
18.2.5 Methoden . . . 90
18.3 Interface GroupDao . . . 91
18.3.1 Deklaration . . . 91
18.3.2 Methoden Auflistung . . . 91
18.3.3 Methoden . . . 91
18.4 Interface UserDao . . . 94
18.4.1 Deklaration . . . 94
18.4.2 All known subinterfaces . . . 94
18.4.3 Klassen, die das Interface implementieren . . . 94
18.4.4 Methoden Auflistung . . . 94
18.4.5 Methoden . . . 94
18.5 Klasse GoDaoImp . . . 96
18.5.1 Deklaration . . . 96
18.5.2 Konstruktoren Auflistung . . . 96
18.5.3 Methoden Auflistung . . . 96
18.5.4 Attribute . . . 97
18.5.5 Konstruktoren . . . 97
18.5.6 Methoden . . . 97
18.6 Klasse GroupDaoImp . . . 99
18.6.1 Deklaration . . . 99

18.6.2 Konstruktoren Auflistung . . . 99
18.6.3 Methoden Auflistung . . . 99
18.6.4 Attribute . . . 100
18.6.5 Konstruktoren . . . 100
18.6.6 Methoden . . . 100
18.7 Klasse UserDaoImp . . . 102
18.7.1 Deklaration . . . 102
18.7.2 Konstruktoren Auflistung . . . 102
18.7.3 Methoden Auflistung . . . 102
18.7.4 Attribute . . . 103
18.7.5 Konstruktoren . . . 103
18.7.6 Methoden . . . 103

19 Package ClientCommunication.Downstream **106**
19.1 Enum EventArg . . . 106
19.1.1 Deklaration . . . 106
19.1.2 Attribute . . . 106
19.1.3 Methoden Auflistung . . . 107
19.1.4 Felder . . . 107
19.1.5 Methoden . . . 108
19.1.6 von Enum geerbte Methoden . . . 108
19.2 Klasse FcmClient . . . 108
19.2.1 Deklaration . . . 108
19.2.2 Konstruktoren Auflistung . . . 108
19.2.3 Methoden Auflistung . . . 109
19.2.4 Attribute . . . 109
19.2.5 Konstruktoren . . . 109
19.2.6 Methoden . . . 109

20 Package ClientCommunication.Upstream **110**
20.1 Klasse GoRestController . . . 110
20.1.1 Deklaration . . . 110
20.1.2 Konstruktoren Auflistung . . . 110
20.1.3 Methoden Auflistung . . . 110
20.1.4 Attribute . . . 111
20.1.5 Konstruktoren . . . 111
20.1.6 Methoden . . . 111
20.2 Klasse GroupRestController . . . 115
20.2.1 Deklaration . . . 115
20.2.2 Konstruktoren Auflistung . . . 115
20.2.3 Methoden Auflistung . . . 115
20.2.4 Attribute . . . 116
20.2.5 Konstruktoren . . . 116
20.2.6 Methoden . . . 116
20.3 Klasse UserRestController . . . 120
20.3.1 Deklaration . . . 120
20.3.2 Konstruktoren Auflistung . . . 120
20.3.3 Methoden Auflistung . . . 120
20.3.4 Attribute . . . 120
20.3.5 Konstruktoren . . . 121
20.3.6 Methoden . . . 121

21 Package edu.kit.pse17.go_app **123**
21.1 Klasse Main . . . 123
21.1.1 Deklaration . . . 123
21.1.2 Konstruktoren Auflistung . . . 123
21.1.3 Methoden Auflistung . . . 123
21.1.4 Konstruktoren . . . 123
21.1.5 Methoden . . . 123

22 Package ServiceLayer **123**
22.1 Interface ClusterStrategy . . . 124
22.1.1 Deklaration . . . 124
22.1.2 All known subinterfaces . . . 124
22.1.3 Klassen, die das Interface implementieren . . . 124
22.1.4 Methoden Auflistung . . . 124
22.1.5 Methoden . . . 124
22.2 Interface Observable . . . 125
22.2.1 Deklaration . . . 125
22.2.2 All known subinterfaces . . . 125
22.2.3 Klassen, die das Interface implementieren . . . 125
22.2.4 Methoden Auflistung . . . 125
22.2.5 Methoden . . . 125
22.3 Interface Observer . . . 126
22.3.1 Deklaration . . . 127
22.3.2 All known subinterfaces . . . 127
22.3.3 Klassen, die das Interface implementieren . . . 127
22.3.4 Methoden Auflistung . . . 127
22.3.5 Methoden . . . 127
22.4 Klasse Cluster . . . 128
22.4.1 Deklaration . . . 128
22.4.2 Konstruktoren Auflistung . . . 128
22.4.3 Methoden Auflistung . . . 128
22.4.4 Attribute . . . 128
22.4.5 Konstruktoren . . . 128
22.4.6 Methoden . . . 128
22.5 Klasse EntitiyRemovedObserver . . . 129
22.5.1 Deklaration . . . 129
22.5.2 Field summary . . . 129
22.5.3 Konstruktoren Auflistung . . . 129
22.5.4 Methoden Auflistung . . . 129
22.5.5 statische Felder . . . 130
22.5.6 Attribute . . . 130
22.5.7 Konstruktoren . . . 130
22.5.8 Methoden . . . 130
22.6 Klasse EntityAddedObserver . . . 132
22.6.1 Deklaration . . . 132
22.6.2 Field summary . . . 132
22.6.3 Konstruktoren Auflistung . . . 132
22.6.4 Methoden Auflistung . . . 132
22.6.5 statische Felder . . . 132
22.6.6 Attribute . . . 133
22.6.7 Konstruktoren . . . 133

22.6.8 Methoden . . . 133
22.7 Klasse EntityChangedObserver . . . 135
22.7.1 Deklaration . . . 135
22.7.2 Field summary . . . 135
22.7.3 Konstruktoren Auflistung . . . 135
22.7.4 Methoden Auflistung . . . 135
22.7.5 statische Felder . . . 136
22.7.6 Attribute . . . 136
22.7.7 Konstruktoren . . . 136
22.7.8 Methoden . . . 136
22.8 Klasse GoClusterStrategy . . . 138
22.8.1 Deklaration . . . 138
22.8.2 Konstruktoren Auflistung . . . 138
22.8.3 Methoden Auflistung . . . 138
22.8.4 Attribute . . . 138
22.8.5 Konstruktoren . . . 138
22.8.6 Methoden . . . 138
22.9 Klasse LocationService . . . 139
22.9.1 Deklaration . . . 139
22.9.2 Konstruktoren Auflistung . . . 139
22.9.3 Methoden Auflistung . . . 139
22.9.4 Attribute . . . 140
22.9.5 Konstruktoren . . . 140
22.9.6 Methoden . . . 140
22.10 Klasse UserLocation . . . 142
22.10.1 Deklaration . . . 142
22.10.2 Konstruktoren Auflistung . . . 142
22.10.3 Methoden Auflistung . . . 142
22.10.4 Attribute . . . 142
22.10.5 Konstruktoren . . . 142
22.10.6 Methoden . . . 142

23 Client-Server-Schnittstelle **144**
23.1 REST API des Servers . . . 144
23.2 FCM Schnittstelle . . . 144

24 Klassendiagramme **147**
24.1 Server . . . 147

25 Client **148**

26 Sequenzdiagramme **149**
26.1 Hinzufügen eines Gruppenmitglieds . . . 149
26.2 Entfernen einer Gruppe . . . 151
26.3 Teilnahmestatus ändern . . . 152
26.4 GPS-Standort ermitteln . . . 153

1 Änderungen zum Pflichtenheft

Es wurden im Entwurf folgende Änderungen gegenüber dem Pflichtenheft vorgenommen:

1. **Produktdaten - Benutzer**
   Es werden in den Produktdaten zusätzlich eine (von Firebase automatisch generierte) InstanceID gespeichert, die es dem Server erlaubt, Daten an das Android-Gerät eines bestimmten Benutzers zu senden.

2. **Detailansicht der GOs**
   Es ist jedem Mitglied einer Gruppen (unabhängig von Teilnahmestatus) möglich, die Detailansicht eines GOs aufzurufen. Um den Karten-Tab öffnen zu können, um die Standorte der anderen Teilnehmer zu verfolgen, gilt weiterhin, dass der Teilnahmestatus 'Bestätigt' oder 'Unterwegs' lauten muss.

3. **Profil- und Gruppenbilder**
   Das Wunschkriterium der Profil- und Gruppenbilder wurde im Entwurf nicht berücksichtigt. Stattdessen werden in der App definierte Standardbilder dazu angezeigt.

2 Architekturstil und Paketstruktur

Dieses Kapitel erläutert den Architekturstil und die Paketstruktur des Systems. es wird noch nicht auf einzelne Klassen eingegangen sondern lediglich die Zusammenarbeit der Module beschrieben und die Abhängigkeiten untereinander.

Aufgebaut ist das System mit einer Client-Server-Architektur, d.h. es gibt eine Client-Applikation, die auf den Android-Geräten der Benutzer läuft und Services des Servers in Anspruch nimmt und eine Server-Anwendung, die auf dem Tomcat-Server läuft, der diese Services bereitstellt und auf Anfrage eines Clients seine Dienste erfüllt.

2.1 Client

Die Architektur der Client-Applikation orientiert sich am Model-View-View-Model (MVVM) Muster. Dies wird auf Android-Systemen durch die, auch hier eingesetzten, Architecture Components unterstützt. Das untenstehende Paketdiagramm zeigt den groben Aufbau der Client-Applikation, welche Abhängigkeiten bestehen und welche Aufgaben jedes Paket übernimmt. Genauere Erläuterungen hierzu finden sich in den darauffolgenden Abschnitten.

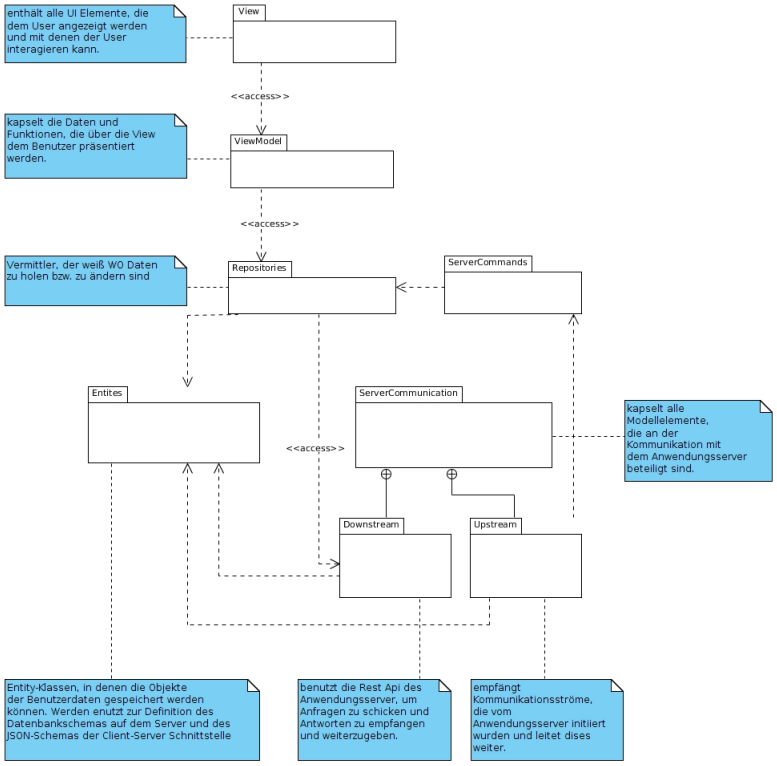

Abbildung 1: Paketdiagramm der Clientanwendung

2.1.1 Views

Das Paket Views enthält alle Klassen, die am User Interface des Benutzers beteiligt sind. Das sind sämtliche Activities, Fragments und dazugehörige .xml-Layouts. Das Modul ist für die Präsentation der Appdaten sowie die Implementierung der Präsentationslogik (Umsetzung der Eigenschaften der Daten und Weiterleitung von Benutzereingaben) zuständig.

Abhängigkeiten zu anderen Paketen:
Das Paket Views kann die Informationen, die dem Benutzer angezeigt werden, nicht selbst generieren, sondern bekommt diese bereitgestellt von den entsprechenden ViewModels. Es kennt

ebenfalls nicht die Funktionen, die ein User in einer bestimmten Ansicht ausführen kann. Es besteht also eine Abhängigkeit zu dem Paket ViewModels. Von anderen Paketen weiß die VIew nichts und wird auch nicht von ihnen angesprochen.

Unterpakete:
das Paket enthält das Unterpaket 'RecyclerView'. Da in der Applikation viele (verschiedene) RecyclerViews verwendet werden, gibt es für die Erstellung derselben ein eigenes Paket, dessen Aufgabe es ist, von den Datenobjekten die das Model liefert die gewünschten Informationen zu extrahieren und diese mit dem richtigen Layout zusammenzuführen. Innerhalb des Pakets besteht eine Abhängigkeit derjenigen View-Klassen, die einen RecyclerView verwenden zu dem Unterpaket RecyclerViews. Das Unterpaket RecyclerViews selbst ist nicht von anderen Klassen und Paketen abhängig.

2.1.2 Entities

Das Modell enthält die Entity-Klassen der App, die die reale Objekte und Konzepte als Java-Objekte abbilden. Sie bilden das Modell im Modell-View-ViewModell-Konzept. Die Entities dienen hauptsächlich zur Kapselung der Daten in der App. Darüber hinaus gibt die Struktur der Entites (die mit der Struktur der Entity-Klassen auf dem Server übereinstimmt) den Aufbau der Datenbank, sowie der JSON-Objekte, die zur Kommunikation zwischen Client und Server verwendet werden vor.

Abhängigkeiten zu anderen Paketen
Das Paket Entities hat keine Abhängigkeiten zu anderen Paketen. Da die Objekte allerdings an allen Use Cases beteiligt sind, gibt es viele Pakete die Abhängigkeiten zu den Entities haben.

2.1.3 ViewModell

Das ViewModell ist das Bindeglied zwischen View und Entites. Es tauscht Informationen mit dem Modell aus und stellt so der View öffentliche Eigenschaften und Befehle zur Verfügung, die an die Steuerungselemente der UI angebunden werden können. Dabei hat das ViewModell keine Referenz auf die View. Durch diese lose Kopplung kann die View jederzeit ausgetauscht werden, ohne dass das ViewModell verändert werden muss.

Abhängigkeiten zu anderen Paketen
Das ViewModell benötigt eine Referenz zum Modell, um die von der View empfangenen Befehle weiterleiten und die richtigen Daten von Modell anfordern zu können. Um diese Abhängigkeit zu entkoppeln und das Ansprechen der richtigen Modellkomponente zu erleichtern, wird diese Abhängigkeit über einen Vermittler (Repository") geleitet. Dies ermöglicht das einfache Austauschen des Modells, ohne dass das ViewModell verändert werden muss.

2.1.4 ServerCommunication

Das Paket ServerCommunication übernimmt die Kommunikation der App mit dem Server, also das Speichern von Daten auf dem Server bzw. das Holen von Daten von dem Server. Darüber hinaus werden in diesem Paket auch Nachrichten, die vom Server gesendet werden empfangen und an das Modell zur Verarbeitung weitergeleitet.

Abhängigkeiten zu anderen Paketen
Das Paket hat keine Abhängigkeiten zu anderen Paketen der Applikation. Die Implementierung des REST-Clients erfolgt über das Framework *Retrofit 2*. Hier besteht also eine Abhängigkeit zu einem externen Framework. Außerdem benötigt das Modul zur fehlerfreien Ausführung seiner Aufgaben ein funktionierendes Backend (REST-Api, das die entsprechenden Ressourcen bereitstellt) des Systems.

Unterpakete:
Das Modul ServerCommunication setzt sich aus zwei Untermodulen zusammen. Das Modul *Downstream* implementiert die Kommunikation, die über die REST-Api des Tomcat-Servers läuft, also jegliche Kommunikation, die von einem Client initiiert wird. Das Modul *Upstream* hingegen ist dafür zuständig Kommunikationsströme zu empfangen, die vom Server initiiert werden und diese den Vermittlern zur Verbreitung weiterzuleiten.

2.1.5 ServerCommands

Das Paket ServerCommands kapselt alle Server-Nachrichten, die ein CLient bekommen kann in einem Befehlsmuster. Die Aufgabe dieses Moduls ist es die ankommenden Nachrichten zu analysieren und die Änderungen an die Repositories zu geben und anschließend die ViewModells zu benachrichtigen, sich die aktuellen Daten bei den ViewModells abzuholen.

Anhängigkeiten zu anderen Paketen
Das Paket ServerCommands ist abhängig von dem Paket Repositories. Ohne dieses Paket können die einzelnen Commands ihre Aufgabe, aktualisierte Daten dort abzulegen, wo die ViewModels sie finden, nicht erfüllen. Die Nachrichten, die an die Commands von dem ServerCommunication Package kommen, können leicht simuliert werden, weshalb für die Implementierung keine Abhängigkeit zu diesem Modul besteht.

2.1.6 Repositories

Wie in den vorherigen Abschnitten erläutert, ist die Geschäftslogik der App aufgeteilt auf den lokalen Teil (Entities und ServerCommands) und einen Remote-Teil (ServerCommunication), die miteinander synchronisiert werden müssen. Zusätzlich müssen die ViewModells nach sämtlichen Änderungen mit den aktuellsten Daten versorgt werden. Diese Abhängigkeiten der einzelnen Komponenten werden in den Repository-Klassen zusammengefasst. Genauere Erläuterungen zur Funktionsweise finden sich in Abschnitt 3.5.

Abhängigkeiten zu anderen Paketen:
Damit die Repositories ihren Zweck erfüllen können, müssen alle Kollegen des Vermittler-Musters implementiert sein. Für die Implementierung selbst, ist es wichtig, dass das ServerCommunication Modul und die Entities implementiert werden, bevor die Repositories implementiert werden.

2.2 Server

Die Architektur des Servers orientiert sich an einer Schichten-Architektur. Dabei handelt es sich nicht um eine klassische Schichtenarchitektur, bei der nur obere Schichten Dienste der unteren Schichten aufrufen darf. Stattdessen sind die oberen Schichten in zwei Unterpakete unterteilt. In einer Hälfte jeder Schicht gehen die Methodenaufrufe von oben nach unten, in der anderen von unten nach oben. Durch diese zwei Säulen in der Anwendung ist die Implementierung, trotz

Verletzung des Schichtenmodells, leicht realisierbar, da die Unterpakete immer noch topologisch sortierbar sind, sodass eine Implementierungsreihenfolge gefunden werden kann, bei der zu jeder Zeit die Abhängigkeiten des gerade entwickelten Moduls bereits implementiert sind.

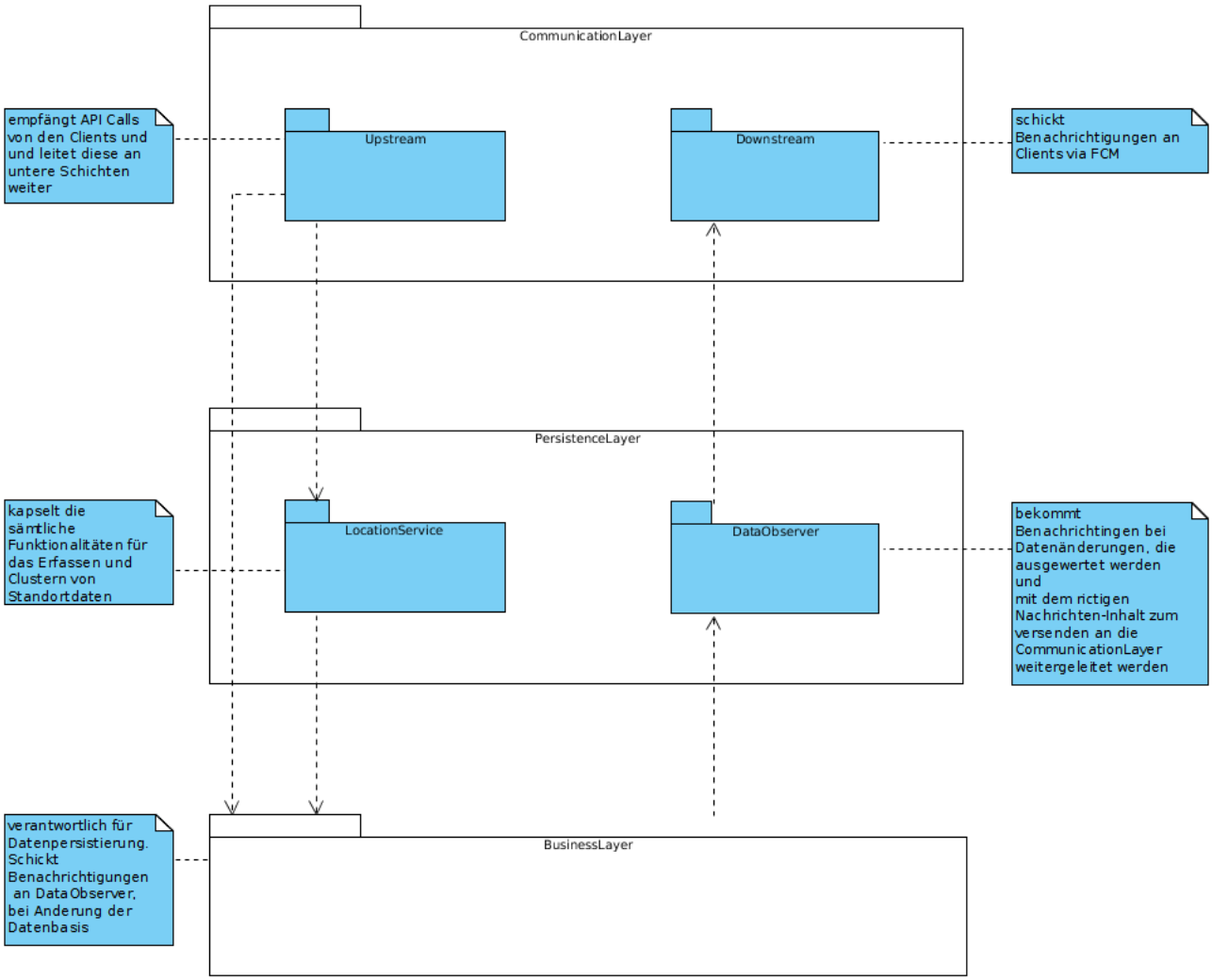

Abbildung 2: Paketdiagramm der Clientanwendung

Das Programm des Servers ist in folgende Pakete aufgeteilt:

- CommunicationLayer
- BusinessLayer
- PersistenceLayer

2.2.1 CommunicationLayer

Dieses Paket bildet die oberste Schicht der Serveranwendung und ist die einzige Schicht, die für die Clients sichtbar ist. Die CommunicationLayer vereint alle Klassen, die an der Kommunikation mit den Clients beteiligt sind. Es besteht aus den Unterpaketen Upstream und Downstream. Die Klassen des Pakets Upstream sind dafür zuständig, ein REST-API zur Verfügung zu stellen und auf Anfragen der Clients zu antworten, d.h. die Kommunikation wird von den Clients

initiiert. Das Downstream-Paket hingegen schickt Nachrichten an Clients, ohne vorher von diesen angesprochen worden zu sein. Hierfür wird der Firebase Cloud Messaging Service benutzt.

Das Untermodul Upstream nimmt die Dienste der unteren Schichten in Anspruch. Dabei handelt es sich um eine transparente Schichtenarchitektur, das bedeutet das Modul nimmt nicht nur die Dienste der mittleren, sondern auch die Dienste der unteren Schicht in Anspruch.

Das Untermodul Downstream hingegen bietet Dienste an, nämlich das versenden von Nachrichten an Clients, die von den unteren Schichten der Serveranwendung verwendet wird.

2.2.2 BusinessLayer

Die BusinessLayer ist die mittlere Schicht und beinhaltet den Großteil der Anwendungslogik der Serveranwendung. Wie die anderen Schichten des Server, besteht auch die BusinessLayer aus zwei Untermodulen: LocationService und DataObserver.

Das Modul LocationService kümmert sich um alle Angelegenheiten, die mit der Erfassung, der kurzzeitigen Persistierung und dem Clustering von Standortdaten zusammenhängen. Dieser Dienst wird von der oberen Schicht in Anspruch genommen. Zur unteren Schicht bestehen keine Abhängigkeiten, da Standortdaten nicht in der Datenbank gespeichert werden. Dieses Untermodul hat selbst also keine Abhängigkeiten zu anderen Paketen.

Das zweite Untermodul der BusinessLayer ist das DataObserver-Modul. Dieses Modul wird bei Änderungen in der Datenbasis von der PersistenceLayer angesprochen und kümmert sich folgend darum, die Änderungen zu analysieren, herauszufinden welche Clients von der Änderung erfahren müssen und diese Informationen an die obere Schicht weiterzuleiten, um eine Nachricht an genau diese Clients zu schicken. Dementsprechend bietet dieses Untermodul einen Dienst, der von der PersistenceLayer in Anspruch genommen wird und nimmt einen Dienst in Anspruch der von der CommunicationLayer angeboten wird.

2.2.3 PersistenceLayer

Die PersistenceLayer kapselt das Datenmodell der Anwendung in sich. Sie ist für die Speicherung der Daten zuständig, sowie für die Weiterleitung der Daten an die CommunicationLayer und somit an die Clients. Die PersistenceLayer setzt sich zusammen aus einer MySQL-Datenbank, die mit dem ORM-Framework Hibernate verwaltet wird und DAO-Klassen, in denen die Datenbankzugriffe gekapselt werden.

Im Gegensatz zu den zwei anderen Schichten gibt es in dieser Schicht keine Untermodule. Der Dienst des Moduls wird von dem Upstream-Modul der CommunicationLayer in Anspruch genommen. Geschieht dies, wird aus der PersistenceLayer der Dienst der DataObserver angestoßen, um die Änderungen in dem Datenbestand wieder durch die Schichten nach oben und schließlich zu den Clients zu leiten.

3 verwendete Entwurfsmuster

3.1 Schablonenmethode für SignInHelper

Die verschiedenen Anmelde-Aktivitäten aller Loginhelper-Klassen können über die signIn()-Methode angestoßenen werden. Der spezifische Ablauf der Anmelde-Aktivität wird in den Unterklassen durch die primitiven Methoden definiert.

beteiligte Klassen:

- *SignInHelper*
  besitzt die Methode signIn(), die als Schablonenmethode dient und bei der Ausführung die primitiven Methoden configureSignIn() und startSignInProcess() aufruft
- *FirebaseSignInHelper*
  Unterklasse von SignInHelper, die die primitiven Methoden configureSignIn() und start-SignInProcess() implementiert
- *GoSignInHelper*
  Unterklasse von SignInHelper, die die primitiven Methoden configureSignIn() und start-SignInProcess() implementiert

3.2 Beobachter zum Aktualisieren des UI

Durch das Ausführen von Befehlen von einem Benutzer, kann es zu Änderungen in den Daten kommen, die eine Änderung des aktuellen Views anderer Benutzer erfordern. Diese 1-zu-n Abhängigkeit wird durch ein Beobachter-muster behandelt. Die dafür benötigte Funktionalität wird von der Architecture-Components Framework Klasse LiveData<> bereitgestellt. Ein Objekt dieser Klasse kann von einem LifeCycleOwner (z.B. eine Lifecycle-Activity oder ein LifecycleFragment) beobachtet werden und löst bei Änderung den Methodenaufruf *onChanged()* aus. Die LiveData-Objekte sind Lifecycle-Aware, das bedeutet eine Benachrichtigung über eine Änderung wird nur dann an einen Beobachter weitergeleitet, wenn er sich in einem aktiven Stadium seines Lifecycles befindet.

beteiligte Klassen:

- *LiveData<>:*
  das beobachtete Subjekt
- *BaseActivity (die von LifecycleActivity erbt):*
  Der Beobachter, der bei Änderung der Daten benachrichtigt wird und daraufhin das dem Benutzer präsentierte UI aktualisiert.

3.3 Beobachter zum Weiterleiten von Änderungen der Datenbasis

Werden von einem Client Daten der App geändert (z.B. eine GO erstellt, ein Benutzer zu einer Gruppe hinzugefügt) betrifft dies in vielen auch die Daten, die ein anderer Client bei sich gespeichert hat und benutzt. Dementsprechend müssen Clients vom Server über Änderungen in der Datenbasis informiert werden können. Es ergibt sich eine 1-zu-n-Abhängigkeit: n Clients sind von 1 Datenbasis abhängig.

Zur Auflösung dieser Abhängigkeit wird in der Anwendung eine Beobachter-Muster verwendet. Die Datenbasis wird beobachtet und bei Änderungen leiten die Beobachter die Benachrichtigung der Clients ein. Da ein Beobachter nicht ein einzelnes Objekt, sondern eine gesamte

Datenbanktabelle beobachtet, ist es schwierig, den aktuellen Zustand des Subjekts im Beobachter zu speichern. Aus diesem Grund verwenden die Beobachter eine push-Methodik, bei der die Änderungen bei der Benachrichtigung der Beobachter mit übergeben werden.

beteiligte Klassen:

- *Observable<T>*
  Das Interface ist das abstrakte Subjekt. Jedes konkrete Subjekt muss dieses Interface und damit die Methode notify() implementieren. Diese Methode benachrichtigt die Beobachter über ein Update in der Datenbasis.

- *DAO-Klassen*
  Diese Klassen implementieren das Interface Observable und sind die konkreten Subjekte, die von den Beobachtern beobachtet werden.

- *Observer*
  Das Interface Observer ist der abstrakte Beobachter. Jeder konkrete Beobachter muss dieses Interface und damit die Methode update() implementieren.

- *EntityObserver*
  Konkrete Beobachter, die das Interface Observer implementieren. Jede dieser Klassen übernimmt die Verantwortung für bestimmte Änderungen in der Datenbasis, analysiert, welche Änderung geschehen ist und übergibt diese Daten zum versenden an den FcmClient.

3.4 DAO-Pattern zur Persistierung von Daten

Die Daten der App werden in eine MySQL-Datenbank persistiert. Für den Zugriff auf diese Datenbank wird ein Data Access Object Entwurfsmuster verwendet. Dabei werden zum Einen Java Beans verwendet, die die Struktur der Datenbankrelationen festlegen. Zum Anderen gibt es spezielle DAO-Klassen die alle Datenbankzugriffe in sich kapseln. So können zusätzliche Zugriffsmethoden für die Datenbank hinzugefügt werden, ohne die Entity-Klassen verändern zu müssen. Andersherum kann der Aufbau der Datenbank verändert werden, ohne dass die DAO-Klassen ihre Schnittstelle nach außen verändern. Die Umsetzung der Datenbank und der SQL-Queries wird mithilfe des Frameworks Hibernate realisiert.

beteiligte Klassen:

- *UserEntity, GoEntity, GroupEntity*
  Entity-Beans, die die Struktur der Datenbankrelationen darstellen

- *AbstractDAO*
  Interface für eine DAO-Klasse, welches Zugriffsmethoden definiert, die auf jeder Datenbanktabelle durchgeführt werden müssen (CRUD)

- *UserDao, GoDao, GroupDao*
  Data-Access-Object Interfaces, die spezielle Zugriffsmethoden enthalten, die nur für bestimmte Datenbanktabellen gebraucht werden.

- *UserDaoImp, GoDapImp, GroupDaoImp*
  konkrete Implementierungen der DAO Interfaces. Hier werden die Zugriffsmethoden anhand des von den Entity-Klassen definierten Datenbankschemas implementiert.

3.5 Vermittler zur Koordination von Datenzugriffen

Viele Apps Nutzen eine lokale Datenbank, um eine Benutzung der App, zumindest ansatzweise, auch ohne Internetverbindung zu ermöglichen. Auch wenn in diesem Entwurf keine lokale Datenbank auf den Clients vorgesehen ist, soll dieses Feature leicht erweiterbar sein. Zu diesem Zweck ist bei der Aktualisierung/Beschaffung der Daten für die ViewModells eine zusätzliche Indirektionsstufe eingebaut: Die Repositories. Diese übernehmen im Zusammenspiel der App-Komponenten eine Vermittler-Funktion: Während die ViewModells wissen WAS gemacht werden muss (da diese die aktuellen Daten und Funktionen für den User speichern), wissen die Repositories WO dies getan werden muss - in der lokalen Datenbank, in den SharedPreferences des Android-Geräts oder auf dem Remote Server.

beteiligte Klassen:

- *GoRepositor, GroupRepository, UserRepository:*
  Die Vermittler die die Koordination der Kollegen übernehmen
- *ViewModell, TomcatRestApi, lokale Datenbank, SharedPreferences:*
  Kollegen, die nichts voneinander wissen, sondern bei allen Aufrufen vom Vermittler angesprochen werden, bzw. den Vermittler ansprechen.

3.6 Strategiemuster zur Kapselung des Clustering-Algorithmus

Das Clustern der Standorte der Teilnehmer eines GOs wird von der Klasse GoClusterStrategy übernommen. Diese Klasse ist mittels eines Strategy-Patterns in das Programm eingebunden. Dies entkoppelt den Algorithmus von seinem Kontext und er kann dynamisch durch andere Clustering-Algorithmen ersetzt oder ergänzt werden.

beteiligte Klassen:

- *LocationService*
  Die Klasse ist der Kontext der Clustering-Strategie. Von hier aus wird die Ausführung des Algorithmus angestoßen.
- *ClusterStrategy*
  ClusterStrategy ist ein Interface, das von jedem Cluster-Algorithmus implementiert werden muss. Es definiert eine *calculateCluster()* Methode, die eine Liste an einzelnen User-Standorten entgegen nimmt und eine Liste an User-Clustern zurückgibt.
- *GoClusterStrategy*
  In dieser Klasse wird der Clustering-Algorithmus implementiert, der angewendet werden soll. Die Klasse erweitert das Interface ClusterStrategy.

3.7 Fassade zur Vereinfachung des Server Interfaces

Der verwendete Tomcat-Server bietet seinem Clients zur Kommunikation ein REST Interface an. Das Ansprechen der verschiedenen REST Resourcen ist in der App hinter dem Interface *TomcatRestApi*. Das Interface bietet den aufrufenden Klassen Methoden zum aufrufen der REST Resourcen an, ohne das ein Aufrufer etwas von der eigentlichen Kommunikation mit dem Server wissen muss.

beteiligte Klassen

- *TomcatRestApi*
  Das Interface ist die Fassade, die die Schnittstelle zum Tomcat-Server hinter sich versteckt.

Nach außen werden Methoden bereitgestellt, die von anderen Klassen aufgerufen werden können, um Server-Dienste in Anspruch nehmen zu können, ohne sich um die Details der Kommunikation zu kümmern.

3.8 Dekorierer zur Erweiterung der Aktivitäten für Sonderbenutzer

Für GO-Verantwortliche und Gruppenadmins werden GroupDetailActivity oder GoDetailActivity entsprechend erweitert. Zusätzliche Buttons und Methoden werden hinzugefügt.(z.B. Name der Gruppe ändern, Mitglieder hinzufügen usw.) Dazu haben wir eine Android-Version von Dekorierer benutzt. Die spezielle Unterklassen (GroupDetailActivityOwner bzw. GoDetailActivityOwner) von oben genannten Activities benutzen fertige Oberklassen Activities und fügen darauf die benötigte Funktionalität hinzu.

beteiligte Klassen

- *GroupDetailActivity*
  Activity, wo die Gruppendetails angezeigt werden. Ist eine Oberklasse von GroupDetailActivityOwner.
- *GroupDetailActivityOwner*
  Erweitert die Oberklasse um Funktionen für die Admins.
- *GoDetailActivity*
  Activity, wo Go-Details angezeigt werden. Oberklasse von GoDetailActivityOwner.
- *GoDetailActivityOwner*
  Erweitert die Oberklasse um Funktionen für den GO-Verantwortlichen

3.9 Command Muster zur Bearbeitung der Server Messages auf der Client Seite

Mit dem Command Muster haben wir die Bearbeitung der Messages von den Messages selbst getrennt. Messages kommen vom Server zu Client mit Hilfe von Firbase. Auf der Client Seite wird das Message vom MessageReceiver empfangen und zu einer der ServerCommand Unterklassen gemappt. Das erlaubt uns auch den Message Flow zu loggen z.B. In dem Command selbst wird schon entschieden, was der Client mit dem Message macht und was ausgeführt wird.

beteiligte Klassen

- *FcmClient*
  Verschickt die Messages an die Clients.
- *MessageReceiver*
  Empfängt die Messages und mappt diese zu einem ServerCommand.
- *ServerCommand*
  Kümmert sich um die Ausführung des Befehls, ändert die Daten auf dem Client entsprechend.

3.10 Singleton in UserViewModel

Benutzer Daten werden im UserViewModel gespeichert. Also ist es sinnvoll ein Singleton Muster da zu implementieren, da es nur ein User per Session gibt. Andere Klassen können dann die UserId kriegen und man ist sicher, dass UserId eindeutig ist.

beteiligte Klasse

- *UserViewModel*
  Singleton.

4 Klassenübersicht - Client

Klassen

- java.lang.Object
  -
    - edu.kit.pse17.go_app.view.recyclerView.ListAdapter (in 14.2, page 64)
  - AppCompatActivity
    - edu.kit.pse17.go_app.login.SignInHelper (in 6.3, page 35)
      - edu.kit.pse17.go_app.login.FirebaseSignInHelper (in 6.1, page 32)
      - edu.kit.pse17.go_app.login.GoSignInHelper (in 6.2, page 34)
  - FirebaseInstanceIdService
    - edu.kit.pse17.go_app.serverCommunication.downstream.TokenService (in 10.2, page 50)
  - FirebaseMessagingService
    - edu.kit.pse17.go_app.serverCommunication.downstream.MessageReceiver (in 10.1, page 49)
  - LifecycleActivity
    - edu.kit.pse17.go_app.view.BaseActivity (in 13.1, page 57)
      - edu.kit.pse17.go_app.view.GoDetailActivity (in 13.4, page 58)
        - edu.kit.pse17.go_app.view.GoDetailActivityOwner (in 13.5, page 59)
      - edu.kit.pse17.go_app.view.GroupDetailActivity (in 13.6, page 60)
        - edu.kit.pse17.go_app.view.GroupDetailActivityOwner (in 13.7, page 61)
      - edu.kit.pse17.go_app.view.GroupListActivity (in 13.8, page 61)
      - edu.kit.pse17.go_app.view.InformationActivity (in 13.9, page 62)
      - edu.kit.pse17.go_app.view.SettingsActivity (in 13.10, page 63)
      - edu.kit.pse17.go_app.view.SignInActivity (in 13.11, page 63)
  - ViewHolder
    - edu.kit.pse17.go_app.view.recyclerView.ListViewHolder (in 14.3, page 66)
  - ViewModel
    - edu.kit.pse17.go_app.viewModel.GoViewModel (in 7.1, page 36)
    - edu.kit.pse17.go_app.viewModel.GroupListViewModel (in 7.2, page 37)
    - edu.kit.pse17.go_app.viewModel.GroupViewModel (in 7.3, page 38)
  - edu.kit.pse17.go_app.ServerCommand (in 5.11, page 30)
    - edu.kit.pse17.go_app.AdminAddedCommand (in 5.1, page 24)
    - edu.kit.pse17.go_app.GoAddedCommand (in 5.2, page 25)
    - edu.kit.pse17.go_app.GoEditedCommand (in 5.3, page 25)
    - edu.kit.pse17.go_app.GoRemovedCommand (in 5.4, page 26)
    - edu.kit.pse17.go_app.GroupEditedCommand (in 5.5, page 26)
    - edu.kit.pse17.go_app.GroupRemovedCommand (in 5.6, page 27)
    - edu.kit.pse17.go_app.GroupRequestReceivedCommand (in 5.7, page 28)
    - edu.kit.pse17.go_app.MemberAddedCommand (in 5.8, page 28)
    - edu.kit.pse17.go_app.MemberRemovedCommand (in 5.9, page 29)
    - edu.kit.pse17.go_app.RequestDeniedCommand (in 5.10, page 30)
    - edu.kit.pse17.go_app.StatusChangedCommand (in 5.12, page 31)
    - edu.kit.pse17.go_app.UserDeletedCommand (in 5.13, page 32)
  - edu.kit.pse17.go_app.model.entities.Cluster (in 8.1, page 40)
  - edu.kit.pse17.go_app.model.entities.Go (in 8.2, page 41)
  - edu.kit.pse17.go_app.model.entities.Group (in 8.3, page 44)
  - edu.kit.pse17.go_app.model.entities.GroupMembership (in 8.4, page 45)
  - edu.kit.pse17.go_app.model.entities.User (in 8.5, page 46)
  - edu.kit.pse17.go_app.model.entities.UserGoStatus (in 8.6, page 47)

- edu.kit.pse17.go_app.repositories.Repository (in 12.3, page 55)
  - edu.kit.pse17.go_app.repositories.GoRepository (in 12.1, page 53)
  - edu.kit.pse17.go_app.repositories.GroupRepository (in 12.2, page 54)
  - edu.kit.pse17.go_app.repositories.UserRepository (in 12.4, page 56)
- edu.kit.pse17.go_app.view.EditGoActivity (in 13.2, page 57)
- edu.kit.pse17.go_app.view.EditGroupActivity (in 13.3, page 58)
- edu.kit.pse17.go_app.view.recyclerView.listItems.GOListItem (in 15.2, page 68)
- edu.kit.pse17.go_app.view.recyclerView.listItems.GroupListItem (in 15.3, page 70)
- edu.kit.pse17.go_app.view.recyclerView.listItems.UserMailListItem (in 15.4, page 72)
- edu.kit.pse17.go_app.view.recyclerView.listItems.UserStatusListItem (in 15.5, page 74)
- edu.kit.pse17.go_app.viewModel.UserViewModel (in 7.4, page 39)
- java.lang.Enum
  - edu.kit.pse17.go_app.model.Status (in 9.1, page 49)

Interfaces

- edu.kit.pse17.go_app.serverCommunication.upstream.TomcatRestApi (in 11.1, page 51)
- edu.kit.pse17.go_app.view.recyclerView.OnListItemClicked (in 14.1, page 64)
- edu.kit.pse17.go_app.view.recyclerView.listItems.ListItem (in 15.1, page 67)

5 Package ServerCommands

5.1 Klasse AdminAddedCommand

Diese Klasse implementiert einen Befehl, der bei der Erteilung von Admin-Rechten zu einem Mitglied der Gruppe vom Admin dieser Gruppe ausgefuehrt wird.

5.1.1 Deklaration

```
public class AdminAddedCommand
 extends edu.kit.pse17.go_app.ServerCommand
```

5.1.2 Konstruktoren

- **AdminAddedCommand**

```
public AdminAddedCommand()
```

5.1.3 Methoden

- **onCommandReceived**

```
public void onCommandReceived()
```

  - **Description**
    Diese Methode aendert folgende Daten in Repositorien der App: -Alle Mitglieder der Gruppe werden ueber den neuen Admin benachrichtigt.

5.1.4 Von ServerCommand vererbte Methoden

`edu.kit.pse17.go_app.ServerCommand` (in 5.11, page 30)

- `public abstract void` **`onCommandReceived()`**

5.2 Klasse GoAddedCommand

Diese Klasse implementiert einen Befehl, der bei dem Erstellen eines neuen GOs der Gruppe ausgefuehrt wird.

5.2.1 Deklaration

```
public class GoAddedCommand
 extends edu.kit.pse17.go_app.ServerCommand
```

5.2.2 Konstruktoren

- **GoAddedCommand**

  ```
  public GoAddedCommand()
  ```

5.2.3 Methoden

- **onCommandReceived**

  ```
  public void onCommandReceived()
  ```

  - **Description**
    Diese Methode aendert folgende Daten in Repositorien der App: -Alle Mitglieder der Gruppe werden ueber das neue GO benachrichtigt.

5.2.4 Von ServerCommand vererbte Methoden

`edu.kit.pse17.go_app.ServerCommand` (in 5.11, page 30)

- `public abstract void` **`onCommandReceived()`**

5.3 Klasse GoEditedCommand

Diese Klasse implementiert einen Befehl, der bei der Veraenderung der Daten eines GOs ausgefuehrt wird.

5.3.1 Deklaration

```
public class GoEditedCommand
 extends edu.kit.pse17.go_app.ServerCommand
```

5.3.2 Konstruktoren

- **GoEditedCommand**

  ```
  public GoEditedCommand()
  ```

5.3.3 Methoden

- **onCommandReceived**

```
public void onCommandReceived()
```

  - **Description**
    Diese Methode aendert folgende Daten in Repositorien der App: -Alle Mitglieder der Gruppe werden ueber die neuen Daten des GOs benachrichtigt.

5.3.4 Von ServerCommand vererbte Methoden

`edu.kit.pse17.go_app.ServerCommand` (in 5.11, page 30)

- `public abstract void` **`onCommandReceived()`**

5.4 Klasse GoRemovedCommand

Diese Klasse implementiert einen Befehl, der bei dem Loeschen eines GOs der Gruppe ausgefuehrt wird.

5.4.1 Deklaration

```
public class GoRemovedCommand
 extends edu.kit.pse17.go_app.ServerCommand
```

5.4.2 Konstruktoren

- **GoRemovedCommand**

```
public GoRemovedCommand()
```

5.4.3 Methoden

- **onCommandReceived**

```
public void onCommandReceived()
```

  - **Description**
    Diese Methode aendert folgende Daten in Repositorien der App: -Alle Mitglieder der Gruppe werden benachrichtigt, dass das GO nicht mehr existiert.

5.4.4 Von ServerCommand vererbte Methoden

`edu.kit.pse17.go_app.ServerCommand` (in 5.11, page 30)

- `public abstract void` **`onCommandReceived()`**

5.5 Klasse GroupEditedCommand

Diese Klasse implementiert einen Befehl, der bei der Veraenderung der Daten einer Gruppe ausgefuehrt wird.

5.5.1 Deklaration

```
public class GroupEditedCommand
 extends edu.kit.pse17.go_app.ServerCommand
```

5.5.2 Konstruktoren

- **GroupEditedCommand**

  ```
  public GroupEditedCommand()
  ```

5.5.3 Methoden

- **onCommandReceived**

  ```
  public void onCommandReceived()
  ```

  - **Description**
    Diese Methode aendert folgende Daten in Repositorien der App: -Alle Mitglieder der Gruppe werden ueber die neuen Daten benachrichtigt.

5.5.4 Von ServerCommand vererbte Methoden

`edu.kit.pse17.go_app.ServerCommand` (in 5.11, page 30)

- `public abstract void` **`onCommandReceived()`**

5.6 Klasse GroupRemovedCommand

Diese Klasse implementiert einen Befehl, der bei dem Loeschen einer Gruppe ausgefuehrt wird.

5.6.1 Deklaration

```
public class GroupRemovedCommand
 extends edu.kit.pse17.go_app.ServerCommand
```

5.6.2 Konstruktoren

- **GroupRemovedCommand**

  ```
  public GroupRemovedCommand()
  ```

5.6.3 Methoden

- **onCommandReceived**

  ```
  public void onCommandReceived()
  ```

  - **Description**
    Diese Methode aendert folgende Daten in Repositorien der App: -Alle Mitglieder der Gruppe werden benachrichtigt, dass die Gruppe nicht mehr existiert; -Alle GOs der Gruppe werden geloescht; -Die Gruppe selbst (und alle Daten) werden geloescht.

5.6.4 Von ServerCommand vererbte Methoden

`edu.kit.pse17.go_app.ServerCommand` (in 5.11, page 30)

- `public abstract void` **`onCommandReceived()`**

5.7 Klasse GroupRequestReceivedCommand

Diese Klasse implementiert einen Befehl, der bei dem Senden einer Gruppenanfrage vom Admin der Gruppe zu einem Benutzer ausgefuehrt wird.

5.7.1 Deklaration

```
public class GroupRequestReceivedCommand
 extends edu.kit.pse17.go_app.ServerCommand
```

5.7.2 Konstruktoren

- **GroupRequestReceivedCommand**

  ```
  public GroupRequestReceivedCommand()
  ```

5.7.3 Methoden

- **onCommandReceived**

  ```
  public void onCommandReceived()
  ```

  - **Description**
    Diese Methode aendert folgende Daten in Repositorien der App: -Dem ausgewaehlten Benutzer wird eine Gruppenanfrage gesendet, die in der App des Benutzers gezeigt wird.

5.7.4 Von ServerCommand vererbte Methoden

`edu.kit.pse17.go_app.ServerCommand` (in 5.11, page 30)

- `public abstract void` **`onCommandReceived()`**

5.8 klasse MemberAddedCommand

Diese Klasse implementiert einen Befehl, der bei dem Beitreten einer Gruppe vom Benutzer ausgefuehrt wird. D.h. der Benutzer hat die Gruppenanfrage bestaetigt.

5.8.1 Deklaration

```
public class MemberAddedCommand
 extends edu.kit.pse17.go_app.ServerCommand
```

5.8.2 Konstruktoren

- **MemberAddedCommand**

  ```
  public MemberAddedCommand()
  ```

5.8.3 Methoden

- **onCommandReceived**

  ```
  public void onCommandReceived()
  ```

  - **Description**
    Diese Methode aendert folgende Daten in Repositorien der App: -Dem ausgewaehlten Benutzer werden alle Daten der Gruppe gesendet; -Alle Mitglieder der Gruppe werden ueber den neuen Benutzer benachrichtigt.

5.8.4 Von ServerCommand vererbte Methoden

`edu.kit.pse17.go_app.ServerCommand` (in 5.11, page 30)

- `public abstract void` **`onCommandReceived()`**

5.9 Klasse MemberRemovedCommand

Diese Klasse implementiert einen Befehl, der bei dem Austreten (oder Loeschen) eines Mitglieds aus der Gruppe ausgefuehrt wird.

5.9.1 Deklaration

```
public class MemberRemovedCommand
 extends edu.kit.pse17.go_app.ServerCommand
```

5.9.2 Konstruktoren

- **MemberRemovedCommand**

  ```
  public MemberRemovedCommand()
  ```

5.9.3 Methoden

- **onCommandReceived**

  ```
  public void onCommandReceived()
  ```

  - **Description**
    Diese Methode aendert folgende Daten in Repositorien der App: -Alle Mitglieder der Gruppe werden benachrichtigt, dass ein Mitglied aus der Gruppe ausgetreten (oder geloescht) ist; -Alle GOs, bei denen der Benutzer GO-Verantwortlicher war, werden geloescht; -Wenn die Gruppe nur einen einzigen Mitglied hatte, wird diese Gruppe geloescht.

5.9.4 Von ServerCommand vererbte Methoden

`edu.kit.pse17.go_app.ServerCommand` (in 5.11, page 30)

- `public abstract void` **`onCommandReceived()`**

5.10 Klasse RequestDeniedCommand

Diese Klasse implementiert einen Befehl, der bei dem Ablehnen einer Gruppenanfrage vom Benutzer ausgefuehrt wird. D.h. der Benutzer hat die Gruppenanfrage abgelehnt.

5.10.1 Deklaration

```
public class RequestDeniedCommand
 extends edu.kit.pse17.go_app.ServerCommand
```

5.10.2 Konstruktoren

- **RequestDeniedCommand**

  ```
  public RequestDeniedCommand()
  ```

5.10.3 Methoden

- **onCommandReceived**

  ```
  public void onCommandReceived()
  ```

  - **Description**
    Diese Methode aendert folgende Daten in Repositorien der App: -Alle Administratoren der Gruppe werden benachrichtigt (d.h., diese eroeffnete Gruppenanfrage wird geloescht).

5.10.4 Von ServerCommand vererbte Methoden

`edu.kit.pse17.go_app.ServerCommand` (in 5.11, page 30)

- `public abstract void` **`onCommandReceived()`**

5.11 Klasse ServerCommand

Die abstrakte Klasse ist eine allgemeine Klasse fuer Befehle, die bei Ankunft einer Nachricht vom Server ausgefuehrt werden. Diese Befehle aendern die Daten in den Repositorien, sodass die App diese spaeter holen kann.

5.11.1 Deklaration

```
public abstract class ServerCommand
 extends java.lang.Object
```

5.11.2 All known subclasses

AdminAddedCommand (in 5.1, page 24), GoAddedCommand (in 5.2, page 25), GoRemovedCommand (in 5.4, page 26), StatusChangedCommand (in 5.12, page 31), RequestDeniedCommand (in 5.10, page 30), MemberRemovedCommand (in 5.9, page 29), GroupRemovedCommand (in 5.6, page 27), GoEditedCommand (in 5.3, page 25), GroupRequestReceivedCommand (in 5.7, page 28), UserDeletedCommand (in 5.13, page 32), MemberAddedCommand (in 5.8, page 28), GroupEditedCommand (in 5.5, page 26)

5.11.3 Konstruktoren

- **ServerCommand**

```
public ServerCommand()
```

5.11.4 Methoden

- **onCommandReceived**

```
public abstract void onCommandReceived()
```

  - **Description**
    Diese Methode implementiert Befehle, die die neuen Daten im Repositorium ablegen. Wird von der Methode onMessageReceived() der Klasse MessageReceiver aufgerufen, sobald die App eine Nachricht des Tomcat-Servers erhaelt und in einen Befehl dekodiert.

5.12 Klasse StatusChangedCommand

Diese Klasse implementiert einen Befehl, der bei der Veraenderung eines Status des Benutzers innerhalb eines GOs ausgefuehrt wird.

5.12.1 Deklaration

```
public class StatusChangedCommand
 extends edu.kit.pse17.go_app.ServerCommand
```

5.12.2 Konstruktoren

- **StatusChangedCommand**

```
public StatusChangedCommand()
```

5.12.3 Methoden

- **onCommandReceived**

```
public void onCommandReceived()
```

  - **Description**
    Diese Methode aendert folgende Daten in Repositorien der App: -Alle Teilnehmer des GOs werden ueber dan neuen Status des Benutzers benachrichtigt.

5.12.4 Von ServerCommand vererbte Methoden

`edu.kit.pse17.go_app.ServerCommand` (in 5.11, page 30)

- `public abstract void` **`onCommandReceived()`**

5.13 Klasse UserDeletedCommand

Diese Klasse implementiert einen Befehl, der bei dem Loeschen vom Account eines Benutzers ausgefuehrt wird.

5.13.1 Deklaration

```
public class UserDeletedCommand
 extends edu.kit.pse17.go_app.ServerCommand
```

5.13.2 Konstruktoren

- **UserDeletedCommand**

  ```
  public UserDeletedCommand()
  ```

5.13.3 Methoden

- **onCommandReceived**

  ```
  public void onCommandReceived()
  ```

  - **Description**
    Diese Methode aendert folgende Daten in Repositorien der App: -Alle Mitglieder der Gruppen, bei denen der Benutzer Mitglied war, werden benachrichtigt, dass dieser Mitglied geloescht ist; -Alle Teilnehmer der GOs, bei denen der Benutzer Teilnehmer war, werden benachrichtigt, dass dieser Benutzer geloescht ist; -Alle GOs, bei denen der Benutzer GO-Verantwortlicher war, werden geloescht; -Wenn die Gruppe nur einen einzigen Mitglied hatte, wird diese Gruppe geloescht.

5.13.4 Von ServerCommand vererbte Methoden

`edu.kit.pse17.go_app.ServerCommand` (in 5.11, page 30)

- `public abstract void` **`onCommandReceived()`**

6 Package Login

Package Contents *Page*

6.1 Klasse FirebaseSignInHelper

Diese Klasse ist fuer die Kommunikation mit Firebase und Google API waehrend des Login-Prozesses zustaendig. Sie implementiert die Methoden configureSignIn() und startSignInProcess() zur Schablonenmethode signIn() der Oberklasse SignInHelper.

6.1.1 Deklaration

```
public class FirebaseSignInHelper
 extends edu.kit.pse17.go_app.login.SignInHelper
```

6.1.2 Konstruktoren

- **FirebaseSignInHelper**

```
public FirebaseSignInHelper()
```

6.1.3 Methoden

- **configureSignIn**

```
protected void configureSignIn()
```

  - **Description**
    Implementierung gehoert zur Schablonenmethode signIn()

- **onActivityResult**

```
protected void onActivityResult(int requestCode,int resultCode,
    Intent data)
```

  - **Description**
    Methode erwartet das Resultat der signInAktivitaet der GoogleSignInApi. War die Aktivitaet erfolgreich, wird die Authentifizierung mit Firebase gestartet.
  - **Parameters**
    * `requestCode` – Request Code, mit dem Aktivitaet gestartet wurde
    * `resultCode` – Result Code der Aktivitaet
    * `data` – Intent, den die Aktivitaet uebergibt

- **onConnectionFailed**

```
public void onConnectionFailed(ConnectionResult connectionResult)
```

  - **Description**
    Diese Methode wird aufgerufen, falls Verbindung zu Google Play Services fehlschlaegt
  - **Parameters**
    * `connectionResult` – Ergebnis der fehlgeschlagenen Verbindung

- **signOut**

```
public void signOut()
```

- **startSignInProcess**

```
protected void startSignInProcess()
```

  - **Description**
    Implementierung gehoert zur Schablonenmethode signIn() Die Methode startet die signIn Aktivitaet der GoogleSignInApi

6.1.4 Von SignInHelper vererbte Methoden

`edu.kit.pse17.go_app.login.SignInHelper` (in 6.3, page 35)

- `public static final` **ACCOUNT_DATA_CODE**
- `protected abstract void` **configureSignIn**`()`
- `protected void` **onCreate**`(Bundle` **savedInstanceState**`)`
- `protected void` **onStart**`()`
- `protected void` **returnActivityResult**`(java.io.Serializable` **accountData**`)`
- `public static final` **SIGN_IN_DATA_CODE**
- `public static void` **signIn**`(Activity` **activity**`, int` **requestCode**`, java.io.Serializable` **signInData**`, java.lang.Class` **signinHelper**`)`
- `protected abstract void` **startSignInProcess**`()`

6.2 Klasse GoSignInHelper

Die Klasse ist fuer die Anmeldung eines Beutzers am Go-Server zustaendig. Sie implemetiert die Methoden configureSignIn() und startSignInProcess() zur Schablonenmethode signIn() der Oberklasse SignInHelper.

6.2.1 Deklaration

```
public class GoSignInHelper
 extends edu.kit.pse17.go_app.login.SignInHelper
```

6.2.2 Konstruktoren

- **GoSignInHelper**

  ```
  public GoSignInHelper()
  ```

6.2.3 Methoden

- **configureSignIn**

  ```
  protected abstract void configureSignIn()
  ```

  - **Description copied from SignInHelper (in 6.3, page 35)**
    Diese Methode wird von Unterklassen implementiert und in Schablonenmethode aufgerufen

- **startSignInProcess**

  ```
  protected abstract void startSignInProcess()
  ```

  - **Description copied from SignInHelper (in 6.3, page 35)**
    Diese Methode wird von Unterklassen implementiert und in Schablonenmethode aufgerufen

6.2.4 Von SignInHelper vererbte Methoden

`edu.kit.pse17.go_app.login.SignInHelper` (in 6.3, page 35)

- `public static final` **ACCOUNT_DATA_CODE**
- `protected abstract void` **configureSignIn**`()`
- `protected void` **onCreate**`(Bundle` **savedInstanceState**`)`
- `protected void` **onStart**`()`
- `protected void` **returnActivityResult**`(java.io.Serializable` **accountData**`)`
- `public static final` **SIGN_IN_DATA_CODE**
- `public static void` **signIn**`(Activity` **activity**`, int` **requestCode**`, java.io.Serializable` **signInData**`, java.lang.Class` **signinHelper**`)`
- `protected abstract void` **startSignInProcess**`()`

6.3 Klasse SignInHelper

Abstrakte Klasse, die als Schablone fuer den Anmelde-Prozess ihrer Unterklassen dient.

6.3.1 Deklaration

```
public abstract class SignInHelper
 extends AppCompatActivity
```

6.3.2 All known subclasses

GoSignInHelper (in 6.2, page 34), FirebaseSignInHelper (in 6.1, page 32)

6.3.3 statische Felder

- `public static final java.lang.String` **SIGN_IN_DATA_CODE**
  - Name des Intent-Extra, das Anmeldedaten an die SignIn-Aktivitaet uebergibt
- `public static final java.lang.String` **ACCOUNT_DATA_CODE**
  - Name des Intent-Extra, das als Ergebnis der Anmelde-Aktivitaet zurueckgegeben wird

6.3.4 Konstruktoren

- **SignInHelper**

  ```
  public SignInHelper()
  ```

6.3.5 Methoden

- **configureSignIn**

  ```
  protected abstract void configureSignIn()
  ```

  - **Description**
    Diese Methode wird von Unterklassen implementiert und in Schablonenmethode aufgerufen

- **onCreate**

```
protected void onCreate(Bundle savedInstanceState)
```

- **onStart**

```
protected void onStart()
```

- **returnActivityResult**

```
protected void returnActivityResult(java.io.Serializable
    accountData)
```

  - **Description**
    Diese Methode gibt das Ergebnis der Anmelde-Aktivitaet an das aufrufende Objekt zurueck
  - **Parameters**
    - `accountData` – Ergebnis der Anmelde-Aktivitaet

- **signIn**

```
public static void signIn(Activity activity,int requestCode,java.
    io.Serializable signInData,java.lang.Class signinHelper)
```

  - **Description**
    Schablonenmethode fuer den Anmelde-Prozess der konkreten SignInHelper
  - **Parameters**
    - `activity` – Aktivity, die die anmeldung aufruft
    - `requestCode` – Request-Code des Aktivitaets-Aufrufs
    - `signInData` – AnmeldeDaten die ggfs an Anmelde-Aktivitaet uebergeben werden muessen
    - `signinHelper` – Referenz auf die Unterklasse, die Methode ausfuehrt

- **startSignInProcess**

```
protected abstract void startSignInProcess()
```

  - **Description**
    Diese Methode wird von Unterklassen implementiert und in Schablonenmethode aufgerufen

7 Package ViewModel

Package Contents *Page*

7.1 Klasse GoViewModel

Die ViewModel Klasse, die alle Daten fuer ein GO beinhaltet, und die Bearbeitungsaufrufe von Activities auf den Daten ausfuehrt.

7.1.1 Deklaration

```
public class GoViewModel
 extends ViewModel
```

7.1.2 Konstruktoren

- **GoViewModel**

```
public GoViewModel()
```

7.1.3 Methoden

- **changeStatus**

```
public void changeStatus(java.lang.String userId,java.lang.String
    goId,edu.kit.pse17.go_app.model.Status status)
```

- **editGo**

```
public void editGo(java.lang.String goid,edu.kit.pse17.go_app.
    model.entities.Go go)
```

- **getCluster**

```
public <any> getCluster(java.lang.String userId,java.lang.String
    groupId,Location location)
```

- **init**

```
public void init()
```

7.2 Klasse GroupListViewModel

Stellt die Daten fuer die GroupDetailActivity View-Komponente zur Verfuegung und uebernimmt die Kommunikation mit der Programmlogik, um die richtigen Daten an die View weiterzugeben. Das ViewModel hat keine Abhaengigkeit zu der View und wird, anders als die Views, bei Konfigurationsaenderungen nicht zerstoert, sondern bleibt erhalten.

7.2.1 Deklaration

```
public class GroupListViewModel
 extends ViewModel
```

7.2.2 statische Felder

- `public static GroupListViewModel` **currentViewModel**

7.2.3 Konstruktoren

- **GroupListViewModel**

```
public GroupListViewModel(edu.kit.pse17.go_app.repositories.
    GroupRepository groupRepo)
```

7.2.4 Methoden

- **createGroup**

```
public void createGroup(edu.kit.pse17.go_app.model.entities.Group
    group)
```

- **getCurrentGroupListViewModel**

```
public static GroupListViewModel getCurrentGroupListViewModel()
```

- **getGroups**

```
public <any> getGroups(java.lang.String userId)
```

- **init**

```
public void init(java.lang.String uId)
```

7.3 Klasse GroupViewModel

ViewModel, die alle Daten fuer eine Gruppe beinhaltet.

7.3.1 Deklaration

```
public class GroupViewModel
 extends ViewModel
```

7.3.2 Konstruktoren

- **GroupViewModel**

```
public GroupViewModel()
```

7.3.3 Methoden

- **addMember**

```
public void addMember(java.lang.String EMail)
```

- **answerGroupRequest**

```
public void answerGroupRequest(boolean answer)
```

- **createGo**

```
public void createGo(edu.kit.pse17.go_app.model.entities.Go go)
```

- **editGroup**

```
public void editGroup(Group group)
```

- **getGos**

```
public <any> getGos(java.lang.String groupId)
```

- **init**

```
public void init()
```

7.4 Class UserViewModel

ViewModel, die alle Benutzerdaten beinhaltet. Ist als Singleton implementiert.

7.4.1 Deklaration

```
public class UserViewModel
 extends java.lang.Object
```

7.4.2 Konstruktoren

- **UserViewModel**

```
public UserViewModel()
```

7.4.3 Methoden

- **deleteUser**

```
public void deleteUser()
```

- **deleteUserCredentials**

```
public void deleteUserCredentials()
```

- **getUserData**

```
public <any> getUserData(java.lang.String uId)
```

- **getUserId**

```
public java.lang.String getUserId()
```

- **getUserInstance**

```
public static edu.kit.pse17.go_app.model.entities.User
    getUserInstance()
```

- **init**

```
public void init(java.lang.String uId)
```

- **setUserCredentials**

```
public void setUserCredentials(edu.kit.pse17.go_app.model.
    entities.User user)
```

8 Package Model.entities

Package Contents *Page*

Classes

Cluster 40
Die Objekte dieser Klasse repraesentieren die Cluster, die dem Benutzer waehrend eines GOs auf der Karte angezeigt werden.

Go 41
Entity-Klasse.

Group 44
Entity-Klasse.

GroupMembership 45
Entity-Klasse.

User 46
Entity-Klasse.

UserGoStatus 47
Entity-Klasse.

8.1 Class Cluster

Die Objekte dieser Klasse repraesentieren die Cluster, die dem Benutzer waehrend eines GOs auf der Karte angezeigt werden. Im Gegensatz zu den anderen Entity-Klassen, wird diese Klasse nicht von Room in der lokalen SQLite Datenbank gespeichert, da eine langfristige Verfuegbarkeit der Daten nicht benoetigt wird. Die Klasse dient Gson als Vorlage zum Parsen der Gso-objekte die via Retrofit gesendet und empfangen werden.

8.1.1 Deklaration

```
public class Cluster
 extends java.lang.Object
```

8.1.2 Konstruktoren

- **Cluster**

```
public Cluster(long lat,long lon,int size)
```

8.1.3 Methoden

- **getLat**

```
public long getLat()
```

- **getLon**

```
public long getLon()
```

- **getSize**

```
public int getSize()
```

- **setLat**

```
public void setLat(long lat)
```

- **setLon**

```
public void setLon(long lon)
```

- **setSize**

```
public void setSize(int size)
```

8.2 Klasse Go

Entity-Klasse. Anhand dieser Klasse wird eine Tabelle in der lokalen SQLite Datenbank generiert, die Go-Objekte persistiert. Der Zugriff auf die Daten laeuft ausschliesslich ueber die GoEntityDAO-Klasse

8.2.1 Deklaration

```
public class Go
 extends java.lang.Object
```

8.2.2 Konstruktoren

- **Go**

```
public Go()
```

8.2.3 Methoden

- **getDescription**

```
public java.lang.String getDescription()
```

- **getDesLat**

```
public long getDesLat()
```

- **getDesLon**

```
public long getDesLon()
```

- **getEnd**

```
public java.util.Date getEnd()
```

- **getId**

```
public long getId()
```

- **getName**

```
public java.lang.String getName()
```

- **getOwner**

```
public java.lang.String getOwner()
```

- **getOwnerName**

```
public java.lang.String getOwnerName()
```

- **getParticipantsList**

```
public java.util.List getParticipantsList()
```

- **getStart**

```
public java.util.Date getStart()
```

- **getUserStatus**

```
public edu.kit.pse17.go_app.model.Status getUserStatus()
```

- **setDescription**

  ```
  public void setDescription(java.lang.String description)
  ```

- **setDesLat**

  ```
  public void setDesLat(long lat)
  ```

- **setDesLon**

  ```
  public void setDesLon(long lon)
  ```

- **setEnd**

  ```
  public void setEnd(java.util.Date end)
  ```

- **setId**

  ```
  public void setId(long id)
  ```

- **setName**

  ```
  public void setName(java.lang.String name)
  ```

- **setOwner**

  ```
  public void setOwner(java.lang.String owner)
  ```

- **setOwnerName**

  ```
  public void setOwnerName(java.lang.String ownerName)
  ```

- **setParticipantsList**

  ```
  public void setParticipantsList(java.util.List participantsList)
  ```

- **setStart**

  ```
  public void setStart(java.util.Date start)
  ```

- **setUserStatus**

  ```
  public void setUserStatus(edu.kit.pse17.go_app.model.Status
      userStatus)
  ```

8.3 Klasse Group

Entity-Klasse. Anhand dieser Klasse wird eine Tabelle in der lokalen SQLite Datenbank generiert, die Gruppen-Objekte persistiert. Der Zugriff auf die Daten laeuft ausschliesslich ueber die GroupEntityDAO-Klasse

8.3.1 Deklaration

```
public class Group
 extends java.lang.Object
```

8.3.2 statische Felder

- `public Icon` **icon**
  - Das Bild der Gruppe

8.3.3 Konstruktoren

- **Group**

  ```
  public Group()
  ```

8.3.4 Methoden

- **getCurrentGos**

  ```
  public java.util.List getCurrentGos()
  ```

- **getDescription**

  ```
  public java.lang.String getDescription()
  ```

- **getIcon**

  ```
  public Icon getIcon()
  ```

- **getId**

  ```
  public long getId()
  ```

- **getMemberCount**

  ```
  public int getMemberCount()
  ```

- **getMembershipList**

  ```
  public java.util.List getMembershipList()
  ```

- **getName**

```
public java.lang.String getName()
```

- **setCurrentGos**

```
public void setCurrentGos(java.util.List currentGos)
```

- **setDescription**

```
public void setDescription(java.lang.String description)
```

- **setIcon**

```
public void setIcon(Icon icon)
```

- **setId**

```
public void setId(long id)
```

- **setMemberCount**

```
public void setMemberCount(int memberCount)
```

- **setMembershipList**

```
public void setMembershipList(java.util.List membershipList)
```

- **setName**

```
public void setName(java.lang.String name)
```

8.4 Class GroupMembership

Entity-Klasse. Anhand dieser Klasse wird eine Tabelle in der lokalen SQLite Datenbank generiert, die GroupMembership-Objekte persistiert. Diese Klasse stellt alle Mitglieder der Gruppe + Information ob das Mitglied ein Administrator ist oder ob es sich bei der Mitgliedschaft lediglich um eine offene Grupppenanfrage handelt dar.

8.4.1 Deklaration

```
public class GroupMembership
 extends java.lang.Object
```

8.4.2 Konstruktoren

- **GroupMembership**

```
public GroupMembership(User user,Group group,boolean isAdmin,
    boolean isRequest)
```

8.4.3 Methoden

- **getGroup**

```
public Group getGroup()
```

- **getUser**

```
public User getUser()
```

- **isAdmin**

```
public boolean isAdmin()
```

- **isRequest**

```
public boolean isRequest()
```

- **setAdmin**

```
public void setAdmin(boolean admin)
```

- **setGroup**

```
public void setGroup(Group group)
```

- **setRequest**

```
public void setRequest(boolean request)
```

- **setUser**

```
public void setUser(User user)
```

8.5 Klasse User

Entity-Klasse. Anhand dieser Klasse wird eine Tabelle in der lokalen SQLite Datenbank generiert, die User-Objekte persistiert. Der Zugriff auf die Daten laeuft ausschliesslich ueber die UserEntityDAO-Klasse

8.5.1 Deklaration

```
public class User
 extends java.lang.Object implements java.io.Serializable
```

8.5.2 Konstruktoren

- **User**

```
public User(java.lang.String uid,java.lang.String instanceId,java
    .lang.String name,java.lang.String email,Icon icon)
```

8.5.3 Methoden

- **getEmail**

```
public java.lang.String getEmail()
```

- **getIcon**

```
public Icon getIcon()
```

- **getInstanceId**

```
public java.lang.String getInstanceId()
```

- **getName**

```
public java.lang.String getName()
```

- **getUid**

```
public java.lang.String getUid()
```

- **setEmail**

```
public void setEmail(java.lang.String email)
```

- **setIcon**

```
public void setIcon(Icon icon)
```

- **setInstanceId**

```
public void setInstanceId(java.lang.String instanceId)
```

- **setName**

```
public void setName(java.lang.String name)
```

- **setUid**

```
public void setUid(java.lang.String uid)
```

8.6 Klasse UserGoStatus

Entity-Klasse. Anhand dieser Klasse wird eine Tabelle in der lokalen SQLite Datenbank generiert, die UserGoStatus-Objekte persistiert. Diese Klasse stellt alle Teilnehmer eines GOs + Status des Teilnehmers bei diesem GO dar. Jeder Benutzer darf nur Status innerhalb eines GOs haben.

8.6.1 Deklaration

```
public class UserGoStatus
 extends java.lang.Object
```

8.6.2 Konstruktoren

- **UserGoStatus**

  ```
  public UserGoStatus(User user,Go go,edu.kit.pse17.go_app.model.
      Status status)
  ```

8.6.3 Methoden

- **getGo**

  ```
  public Go getGo()
  ```

- **getStatus**

  ```
  public edu.kit.pse17.go_app.model.Status getStatus()
  ```

- **getUser**

  ```
  public User getUser()
  ```

- **setGo**

  ```
  public void setGo(Go go)
  ```

- **setStatus**

  ```
  public void setStatus(edu.kit.pse17.go_app.model.Status status)
  ```

- **setUser**

  ```
  public void setUser(User user)
  ```

9 Package Model

Package Contents *Page*

Classes

Status 49
Moeglicher Teilnahmestatus fuer GOs: NOT_GOING = Abgelehnt GOING = Bestaetigt GONE = Unterwegs

9.1 Enum Status

Moeglicher Teilnahmestatus fuer GOs: NOT_GOING = Abgelehnt GOING = Bestaetigt GONE = Unterwegs

9.1.1 Deklaration

```
public final class Status
 extends java.lang.Enum
```

9.1.2 statische Felder

- `public static final Status` **NOT_GOING**
- `public static final Status` **GOING**
- `public static final Status` **GONE**

9.1.3 Methoden

- **valueOf**

  ```
  public static Status valueOf(java.lang.String name)
  ```

- **values**

  ```
  public static Status[] values()
  ```

9.1.4 von Enum vererbte Methoden

`java.lang.Enum`

- `protected final Object` **clone**`() throws CloneNotSupportedException`
- `public final int` **compareTo**`(Enum` **arg0**`)`
- `public final boolean` **equals**`(Object` **arg0**`)`
- `protected final void` **finalize**`()`
- `public final Class` **getDeclaringClass**`()`
- `public final int` **hashCode**`()`
- `public final String` **name**`()`
- `public final int` **ordinal**`()`
- `public String` **toString**`()`
- `public static Enum` **valueOf**`(Class` **arg0**`, String` **arg1**`)`

10 Package ServerCommunication.downstream

10.1 Klasse MessageReceiver

Dies Klasse ist eine Unterklasse von FirebaseMessagingService, die auf den Go Tomcat-Server hoert. DAs heisst, schickt der Server via FCM eine Nachricht an den LCient, wird sie in dieser Klasse empfangen. Wird eine Nachricht an den CLient gesendet waehrend die App im Hintergrund laeuft, wird der Inhalt der Nachricht uaf dem System Tray gespeichert. Bei erneutem Aufrufen der App muss ueberprueft werden, ob in der Zwischenzeit NAchrichten angekommen sind. Sind es zu viele Nachrichten die auf dem System Tray gespeichert worden sind, so kann nicht garantiert werden, dass alle noch vorhanden sind. In diesem Fall wird die Methode onDeletedMessages() aufgerufen. Hier sollten die gesamten Nutzerdaten von Server abgefragt werden,

um sicherzugehen, dass der Client alle aktuellen bei sich hat. Bei Ankunft einer Nachricht des Servers, wird die onMessageReceived-Methode aufgerufen und dort weiter behandelt. Der Service muss beim Start der App gesstartet werden und beim Schliessen der App wieder beendet. Dabei muss der Service auf einem Backgroun Thread und nicht auf dem main UI Thread laufen.

10.1.1 Deklaration

```
public class MessageReceiver
 extends FirebaseMessagingService
```

10.1.2 Konstruktoren

- **MessageReceiver**

  ```
  public MessageReceiver()
  ```

10.1.3 Methoden

- **onDeletedMessages**

  ```
  public void onDeletedMessages()
  ```

  - **Description**
    Diese Methode wird aufgerufen, falls zu viele Benachrichtigungen an den CLient gesendet wurden, waehrend die App im Hintergrund lief. Dann sit es nicht mehr garantiert, dass alle diese Nachrichten noch in der System tray zu finden sind. Es sollten also die gesamten Nuterdaten nochmal vom Server angefragt werden. Das passiert in dieser Methode.

- **onMessageReceived**

  ```
  public void onMessageReceived(RemoteMessage remoteMessage)
  ```

  - **Description**
    Diese Methode wird aufgerufen, wenn die App eine Nachricht vom FCM Server erhaelt waehrend sie im Vordergrund laeuft. Die Nachricht sollte spaetestens 10s nach ihrer Ankunft behandelt werden, vie Spezifikation der Fcm API.
  - **Parameters**
    * `remoteMessage` – die erhaltene Nachricht, verpackt in ein RemoteMessage-Objekt. Dieses Objekt wird vom FCM Server erzeugt und enthaelt die Attribute from und data, mit denen die fuer diese Anwendung relevanten Daten ermittelt werden koennen.

10.2 Klasse TokenService

Die Klasse erzeugt ein InstanceID Token, welches an den Server uebergeben wird, um von Server-Seite aus Nachrichten an ein einzelnes Geraet schicken zu koennen. Bei jeder Anmeldung in der App muss eine solche InstanceID erzeugt werden, um sicher zu gehen, dass diese auch aktuell und gueltig ist. Danach wird diese an den Server weitergeleitet. Es kann waehrend der Ausfuehrung der App auch zu einer Erneuerung der InstanceID kommen. In diesem Fall wird die onTokenReferesh() Methode dieser Klasse aufgerufen.

10.2.1 Deklaration

```
public class TokenService
 extends FirebaseInstanceIdService
```

10.2.2 Konstruktoren

- **TokenService**

  ```
  public TokenService()
  ```

10.2.3 Methoden

- **onTokenRefresh**

  ```
  public void onTokenRefresh()
  ```

  - **Description**
    Wird ein neues Token fuer ein Geraet erzeugt, wird automatisch diese Methode aufgerufen. In der Methode wird das neue Token an den Tomcat-Server gesendet, damit weiterhin Server-Nachrichten an diesen Client gesendet werden koennen.

11 Package ServerCommunication.upstream

11.1 Interface TomcatRestApi

das Interface ist die Schnittstelle des Clients zur REST-API des Tomcat-Servers. Die Kommunikation mit der Rest-API des Servers wird von dem Framework Retrofit uebernommen. In diesem Interface werden deshalb alle Methoden definiert, die von der REST API des Servers angeboten werden. Die Implementierung ist nicht noetig, dies wird von Retrofit uebernommen. Aufgerufen werden diese Methoden in der Klassen des Repository Moduls. Genauere Beschreibungen zur Funktion, den Argumenten und Rueckgabetypen der Methoden sind in den Implementierungen der REST-API zu finden. dabei stimmen die Rueckgabetypen der Methoden dieses Interface mit den Rueckgabetypen der Rest-Methoden des Servers ueberein, sind jedoch in einem Call_Objket gewrappt, wie es bei der Benutzung des Retrofit Frameworks ueblich ist.

11.1.1 Deklaration

```
public interface TomcatRestApi
```

11.1.2 Methoden

- **acceptRequest**

  ```
  <any> acceptRequest(long groupId,java.lang.String userId)
  ```

- **addAdmin**

  ```
  <any> addAdmin(java.lang.String groupId,java.lang.String userId)
  ```

- **changeStatus**

  <any> changeStatus(**long** goId,java.lang.String userId,edu.kit.pse17.go_app.model.Status status)

- **createGo**

  <any> createGo(java.lang.String name,java.lang.String description,java.util.Date start,java.util.Date end,**double** lat,**double** lon,**int** threshold,**long** groupId,java.lang.String userId)

- **createGroup**

  <any> createGroup(java.lang.String name,java.lang.String description,java.lang.String userId)

- **createUser**

  <any> createUser(java.lang.String userId)

- **deleteGo**

  <any> deleteGo(java.lang.String goId)

- **deleteGroup**

  <any> deleteGroup(java.lang.Long groupId)

- **deleteUser**

  <any> deleteUser(java.lang.String userId)

- **denyRequest**

  <any> denyRequest(java.lang.String userId,java.lang.String groupId)

- **editGo**

  <any> editGo(java.lang.String goId,java.lang.String name,java.lang.String description,java.util.Date start,java.util.Date end,**long** lat,**long** lon,**int** threshold)

- **editGroup**

  <any> editGroup(**long** groupId,java.lang.String name,java.lang.String description)

- **getData**

  ```
  <any> getData(java.lang.String userId)
  ```

- **getLocation**

  ```
  <any> getLocation(java.lang.String goId)
  ```

- **inviteMember**

  ```
  <any> inviteMember(long groupId,java.lang.String userId)
  ```

- **registerDevice**

  ```
  <any> registerDevice(java.lang.String instanceId)
  ```

- **removeMember**

  ```
  <any> removeMember(java.lang.String userId,long groupId)
  ```

- **setLocation**

  ```
  <any> setLocation(java.lang.String userId,long lat,long lon,java.
      lang.String goId)
  ```

12 Package Repositories

12.1 Klasse GoRepository

Das Go-Repository ist verantwortlich fuer saemtliche Operationen auf den Go-Daten und stellt eine einfache Schnittstelle zum Holen, aendern und Loeschen von Daten zur Verfuegung. Bei einer Anfrage weiss das Repository, wo es die Daten holen muss (lokal oder vom remote Server). Das Repository agiert als Vermittler zwischen der lokalen Datanbank und den Daten die die App vom Server erhaelt.

12.1.1 Deklaration

```
public class GoRepository
 extends edu.kit.pse17.go_app.repositories.Repository
```

12.1.2 Konstruktoren

- **GoRepository**

  ```
  public GoRepository(edu.kit.pse17.go_app.serverCommunication.
      upstream.TomcatRestApi webService,GoDao goDao,java.util.
      concurrent.Executor executor)
  ```

12.1.3 Methoden

- **changeStatus**

```
public void changeStatus(Status status,java.lang.String goId,java
    .lang.String userId)
```

- **editGo**

```
public void editGo(edu.kit.pse17.go_app.model.entities.Go go)
```

- **fetchData**

```
public abstract <any> fetchData()
```

- **getLocations**

```
public <any> getLocations(java.lang.String goId,Location location
    )
```

- **getUpdatedData**

```
public abstract <any> getUpdatedData()
```

12.1.4 von Repository geerbte Methoden

`edu.kit.pse17.go_app.repositories.Repository` (in 12.3, page 55)

- `public abstract` **`fetchData()`**
- `public abstract` **`getUpdatedData()`**
- `public static void` **`receiveUpdatedData()`**

12.2 Klasse GroupRepository

Das Go-Repository ist verantwortlich fuer saemtliche Operationen auf den Go-Daten und stellt eine einfache Schnittstelle zum Holen, aendern und Loeschen von Daten zur Verfuegung. Bei einer Anfrage weiss das Repository, wo es die Daten holen muss (lokal oder vom remote Server). Das Repository agiert als Vermittler zwischen der lokalen Datanbank und den Daten die die App vom Server erhaelt.

12.2.1 Deklaration

```
public class GroupRepository
 extends edu.kit.pse17.go_app.repositories.Repository
```

12.2.2 Konstruktoren

- **GroupRepository**

```
public GroupRepository(edu.kit.pse17.go_app.serverCommunication.
    upstream.TomcatRestApi webservice,GroupDao groupDao,java.util.
    concurrent.Executor executor)
```

12.2.3 Methoden

- **addMember**

```
public void addMember(java.lang.String Email,java.lang.String
    groupid)
```

- **answerGroupRequest**

```
public void answerGroupRequest(java.lang.String groupId,boolean
    answer)
```

- **createGo**

```
public void createGo(edu.kit.pse17.go_app.model.entities.Go go,
    java.lang.String groupId)
```

- **createGroup**

```
public void createGroup(edu.kit.pse17.go_app.model.entities.Group
     group)
```

- **editGroup**

```
public void editGroup(edu.kit.pse17.go_app.model.entities.Group
    group)
```

- **fetchData**

```
public abstract <any> fetchData()
```

- **getUpdatedData**

```
public abstract <any> getUpdatedData()
```

12.2.4 von Repository geerbte Methoden

`edu.kit.pse17.go_app.repositories.Repository` (in 12.3, page 55)

- `public abstract` **`fetchData()`**
- `public abstract` **`getUpdatedData()`**
- `public static void` **`receiveUpdatedData()`**

12.3 Klasse Repository

Die Klasse dient als Vermittler zwischen ViewModel und Laden von Daten Hier wird entschieden, wen man anspricht um bestimmte Daten zu laden

12.3.1 Deklaration

```
public abstract class Repository
 extends java.lang.Object
```

12.3.2 Unterklassen

GroupRepository (in 12.2, page 54), GoRepository (in 12.1, page 53), UserRepository (in 12.4, page 56)

12.3.3 Konstruktoren

- **Repository**

```
public Repository()
```

12.3.4 Methoden

- **fetchData**

```
public abstract <any> fetchData()
```

- **getUpdatedData**

```
public abstract <any> getUpdatedData()
```

- **receiveUpdatedData**

```
public static void receiveUpdatedData()
```

12.4 Klasse UserRepository

Dieses Repository verwaltet und vermittelt Datenanfragen und Datenaenderungen, die mit dem Benutzerkonto selbst verknuepfte Informationen betreffen. Im Gegensatz zu anderen Repositories spricht diese Klasse auch die SharedPreferences des Systems an.

12.4.1 Deklaration

```
public class UserRepository
 extends edu.kit.pse17.go_app.repositories.Repository
```

12.4.2 Konstruktoren

- **UserRepository**

```
public UserRepository(edu.kit.pse17.go_app.serverCommunication.
    upstream.TomcatRestApi webService, SharedPreferences
    sharedPrefManager, java.util.concurrent.Executor executor)
```

12.4.3 Methoden

- **deleteUser**

```
public void deleteUser(java.lang.String uid)
```

- **fetchData**

```
public abstract <any> fetchData()
```

- **getUpdatedData**

```
public abstract <any> getUpdatedData()
```

12.4.4 von Repository geerbte Methoden

`edu.kit.pse17.go_app.repositories.Repository` (in 12.3, page 55)

- `public abstract` **`fetchData()`**
- `public abstract` **`getUpdatedData()`**
- `public static void` **`receiveUpdatedData()`**

13 Package View

13.1 Klasse BaseActivity

Die Base-Activity ist Oberklasse fuer alle weiteren Activities und kuemmert sich um Funktionalitaet, die alle Activities gemeinsam haben. BaseActivity erbt von LifecycleActivty, was eine Lifecycle-Owner Klasse ist. Dies erlaubt es Objekte, die Lifecycle-Aware sind (z.B. LiveData-Objekten) den Lifecycle der Activity zu beobachten und je nach Zustand der Activity ein entsprechendes UI-Update zu triggern.

13.1.1 Deklaration

```
public class BaseActivity
 extends LifecycleActivity
```

13.1.2 Konstruktoren

- **BaseActivity**

```
public BaseActivity()
```

13.2 Klasse EditGoActivity

Die activity, die fuer GO-Verantwortlichen zugaenglich ist, wo man die Informationen eines GOs aendern kann.

13.2.1 Deklaration

```
public class EditGoActivity
 extends java.lang.Object
```

13.2.2 Konstruktoren

- **EditGoActivity**

```
public EditGoActivity()
```

13.3 Klasse EditGroupActivity

Activity die fuer den Gruppenadmins zugaenglich ist, wo man die Gruppendaten veraendern kann.

13.3.1 Deklaration

```
public class EditGroupActivity
 extends java.lang.Object
```

13.3.2 Konstruktoren

- **EditGroupActivity**

```
public EditGroupActivity()
```

13.4 Klasse GoDetailActivity

die Activity ist zusammen mit der Layout File go_detail.xml Teil des Views, der dem user die Details eines Gos anzeigt. Die Activity ist hauptsaechlich fuer die Darstellung von Informationen zustaendig. Die einzige Datenmanipulation, die hier vorgenommen werden kann, ist die aenderung des Teilnahmestatus des Users.

13.4.1 Deklaration

```
public class GoDetailActivity
 extends edu.kit.pse17.go_app.view.BaseActivity
```

13.4.2 Unterklassen

GoDetailActivityOwner (in 13.5, page 59)

13.4.3 Konstruktoren

- **GoDetailActivity**

```
public GoDetailActivity()
```

13.4.4 Methoden

- **onCreate**

```
protected void onCreate(Bundle savedInstanceState)
```

  - **Description**
    Lifecycle-Methode der Activity, die beim Erzeugen aufgreufen wird. Dem Content-View der App wird das richtige XML Layout zugewiesen und die Informationen die das ViewModel bereitstellt den Layout_komponenten zur Darstellung uebergeben. Es die Livedata des ViewModels auf zum Beobachten registriert, um bei aenderungen die View updaten zu koennen.
  - **Parameters**
    * `savedInstanceState` –

13.5 Klasse GoDetailActivityOwner

Die Klasse dekoriert dir Aktivity-Klasse "GoDetailActivity". Die Go-Detailansicht eines Go-Verantwortlichen unterscheidet dich von der Detailansicht eines normalenTeilnahemers nur in einer zusaetzlichen Schaltflaeche (ëdit"), die geklickt werden kann, um die aenderungsansicht des GOs aufzurufen.

13.5.1 Deklaration

```
public class GoDetailActivityOwner
 extends edu.kit.pse17.go_app.view.GoDetailActivity
```

13.5.2 Konstruktoren

- **GoDetailActivityOwner**

```
public GoDetailActivityOwner()
```

13.5.3 Methoden

- **onCreate**

```
protected void onCreate(Bundle savedInstanceState)
```

  - **Description copied from GoDetailActivity (in 13.4, page 58)**
    Lifecycle-Methode der Activity, die beim Erzeugen aufgreufen wird. Dem Content-View der App wird das richtige XML Layout zugewiesen und die Informationen die das ViewModel bereitstellt den Layout_komponenten zur Darstellung uebergeben. Es die Livedata des ViewModels auf zum Beobachten registriert, um bei aenderungen die View updaten zu koennen.
  - **Parameters**
    * `savedInstanceState` –

13.5.4 Members inherited from class GoDetailActivity

`edu.kit.pse17.go_app.view.GoDetailActivity` (in 13.4, page 58)

- `protected void` **onCreate**`(Bundle` **savedInstanceState**`)`

13.6 Klasse GroupDetailActivity

die Activity ist zusammen mit der Layout File group_details.xml Teil des Views, der dem user die Details eines Gruppe anzeigt. Die Activity ist hauptsaechlich fuer die Darstellung von Informationen zustaendig. Die einzige Datenmanipulation, die hier vorgenommen werden kann, ist die aenderung das Austreten des Users aus der Gruppe.

13.6.1 Deklaration

```
public class GroupDetailActivity
 extends edu.kit.pse17.go_app.view.BaseActivity implements edu.kit.
    pse17.go_app.view.recyclerView.OnListItemClicked
```

13.6.2 Unterklassen

GroupDetailActivityOwner (in 13.7, page 61)

13.6.3 Konstruktoren

- **GroupDetailActivity**

  ```
  public GroupDetailActivity()
  ```

13.6.4 Methoden

- **onCreate**

  ```
  public void onCreate(Bundle savedInstanceState)
  ```

  - **Description**
    Lifecycle-Methode der Activity, die beim Erzeugen aufgreufen wird. Dem Content-View der App wird das richtige XML Layout zugewiesen und die Informationen die das ViewModel bereitstellt den Layout-Komponenten zur Darstellung uebergeben. Es die Livedata des ViewModels auf zum Beobachten registriert, um bei aenderungen die View updaten zu koennen.
  - **Parameters**
    * `savedInstanceState` –

- **onItemClicked**

  ```
  void onItemClicked(int position)
  ```

  - **Description copied from recyclerView.OnListItemClicked** (**in 14.1, page 64**)
    fuehrt gewuenschte Aktion der implemetierenden Klasse aus, falls auf das ListItem an Position position geklickt wird
  - **Parameters**
    * `position` – Position des ListItems, auf das geklickt wurde

13.7 Klasse GroupDetailActivityOwner

Klasse dekoriert die GroupDetailActivity und fuegt ihr die Admin-Funktionalitaeten hinzu. Diese bestehen aus zwei zusaetzlichen Schaltflaechen, die einerseits die aenderungsansicht der Gruppe aufrufen (ëdit"), anderersetis gibt es eine zusaetzliche Schaltflaeche zum Hinzuguegen eiens neuen Gruppenmitglieds (äddMember")

13.7.1 Deklaration

```
public class GroupDetailActivityOwner
 extends edu.kit.pse17.go_app.view.GroupDetailActivity
```

13.7.2 Konstruktoren

- **GroupDetailActivityOwner**

  ```
  public GroupDetailActivityOwner()
  ```

13.7.3 Methoden

- **onCreate**

  ```
  public void onCreate(Bundle savedInstanceState)
  ```

  - **Description copied from GroupDetailActivity (in 13.6, page 60)**
    Lifecycle-Methode der Activity, die beim Erzeugen aufgreufen wird. Dem Content-View der App wird das richtige XML Layout zugewiesen und die Informationen die das ViewModel bereitstellt den Layout-Komponenten zur Darstellung uebergeben. Es die Livedata des ViewModels auf zum Beobachten registriert, um bei aenderungen die View updaten zu koennen.
  - **Parameters**
    * `savedInstanceState` –

13.7.4 von GroupDetailActivity vererbte Methoden und Feler

`edu.kit.pse17.go_app.view.GroupDetailActivity` (in 13.6, page 60)

- `public void` **`onCreate(Bundle savedInstanceState)`**
- `public void` **`onItemClicked(int position)`**

13.8 Klasse GroupListActivity

Hauptansicht der App. Zeigt alle Gruppen eines Benutzers in einer RecyclerView

13.8.1 Deklaration

```
public class GroupListActivity
 extends edu.kit.pse17.go_app.view.BaseActivity implements edu.kit.
    pse17.go_app.view.recyclerView.OnListItemClicked
```

13.8.2 Konstruktoren

- **GroupListActivity**

```
public GroupListActivity()
```

13.8.3 Methoden

- **onClick**

  ```
  public void onClick(View v)
  ```

  - **Description**
    ClickListener fuer addGroupButton
  - **Parameters**
    * `v` –

- **onCreate**

  ```
  protected void onCreate(Bundle savedInstanceState)
  ```

  - **Description**
    RecyclerView und passender Listadapter werden erzeugt
  - **Parameters**
    * `savedInstanceState` –

- **onItemClicked**

  ```
  public void onItemClicked(int position)
  ```

  - **Description**
    ClickListener fuer RecyclerView-Elemente
  - **Parameters**
    * `position` – Position des ListItems, auf das geklickt wurde

- **start**

  ```
  public static void start(Activity activity,edu.kit.pse17.go_app.
      model.entities.User user)
  ```

13.9 Klasse InformationActivity

Diese Activity ist zustaendig fuer die Darstellung eines Informationstextes.

13.9.1 Deklaration

```
public class InformationActivity
 extends edu.kit.pse17.go_app.view.BaseActivity
```

13.9.2 Konstruktoren

- **InformationActivity**

  ```
  public InformationActivity()
  ```

13.10 Klasse SettingsActivity

Die Aktivitaet stellt dem User ein Menue zur Verfuegung, in dem er verschiedende Einstellungsaenderungen vornehmen kann. Die Aufgabe der Aktivitaet ist dabei, dem Benutzer die View zur Verfuegung zu stellen, den User-input entgegenzunehmen und an die Eingabe an die entsprechenden Klassen weiterzuleiten.

13.10.1 Deklaration

```
public class SettingsActivity
 extends edu.kit.pse17.go_app.view.BaseActivity
```

13.10.2 Konstruktoren

- **SettingsActivity**

```
public SettingsActivity()
```

13.10.3 Methoden

- **onCreate**

```
protected void onCreate(Bundle savedInstanceState)
```

13.11 Klasse SignInActivity

Die Klasse stellt die View fuer den LogIn- und SignIn-Prozess bereit.

13.11.1 Deklaration

```
public class SignInActivity
 extends edu.kit.pse17.go_app.view.BaseActivity
```

13.11.2 Konstruktoren

- **SignInActivity**

```
public SignInActivity()
```

13.11.3 Methoden

- **onActivityResult**

```
protected void onActivityResult(int requestCode,int resultCode,
    Intent data)
```

- **onClick**

```
public void onClick(View v)
```

- **Description**
  Click-Listener, der auf Klicken des Signin Buttons wartet –>SignIn wird gestartet
- **Parameters**
  * `v` – geklickter View

- **onCreate**

```
protected void onCreate(Bundle savedInstanceState)
```

- **onResume**

```
protected void onResume()
```

14 Package RecyclerView

14.1 Interface OnListItemClicked

ClickListener fuer die ListItems eines RecyclerViews Created by tina on 17.06.17.

14.1.1 Deklaration

```
public interface OnListItemClicked
```

14.1.2 Subinterfaces

GroupDetailActivityOwner (in 13.7, page 61), GroupDetailActivity (in 13.6, page 60), GroupListActivity (in 13.8, page 61)

14.1.3 Klassen, die das Interface implementieren

GroupDetailActivity (in 13.6, page 60), GroupListActivity (in 13.8, page 61)

14.1.4 Methoden

- **onItemClicked**

```
void onItemClicked(int position)
```

  - **Description**
    fuehrt gewuenschte Aktion der implemetierenden Klasse aus, falls auf das ListItem an Position position geklickt wird
  - **Parameters**
    * `position` – Position des ListItems, auf das geklickt wurde

14.2 Klasse ListAdapter

Abstrakte Klasse, die Schablone fuer konkrete Adapter-Klassen bietet. Unterklassen muessen die Methode setLayout() implementieren, um dem Adapter ein passendes XML-Layout zuzuweisen Created by tina on 17.06.17.

14.2.1 Deklaration

```
public class ListAdapter
 extends <any>
```

14.2.2 statische Felder

- `protected java.util.List` **data**
  - ListItems, die in dem RecyclerView angezeigt werden sollen
- `protected final OnListItemClicked` **onListItemClicked**
  - ClickListener fuer die Listenelemente

14.2.3 Konstruktoren

- **ListAdapter**

  ```
  public ListAdapter(java.util.List data,OnListItemClicked
      onListItemClicked)
  ```

  - **Description**
    Konstruktor
  - **Parameters**
    - `data` – ListItems, die in dem RecyclerView angezeigt werden sollen
    - `onListItemClicked` – ClickListener fuer die Listenelemente

14.2.4 Methoden

- **addItem**

  ```
  public void addItem(listItems.ListItem item)
  ```

- **getItem**

  ```
  public listItems.ListItem getItem(int position)
  ```

  - **Description**
    gibt das ListItem an der angegebenen Position zurueck
  - **Parameters**
    - `position` – Listenposition des gewuenschten ListItems
  - **Returns** – ListItem, an der angegebenen Position aus der Liste data

- **getItemCount**

  ```
  public int getItemCount()
  ```

- **onBindViewHolder**

```
public void onBindViewHolder(ListViewHolder holder,int position)
```

- **onCreateViewHolder**

```
public ListViewHolder onCreateViewHolder(ViewGroup parent,int
    viewType)
```

  - **Description**
    Schablonenmethode: erzeugt ListViewHolder, dem das passende XML layout zugewiesen wird wird aufgerufen, wenn ein RecyclerView einen neuen ViewHolder braucht, um ein ListItem zu repraesentieren
  - **Parameters**
    - `parent` – Viewgroup, zu der der neue View hinzugefuegt werden soll
    - `viewType` – viewType des neuen Views
  - **Returns** – neuer ViewHolder des gewuenschten Typs

14.3 Klasse ListViewHolder

Die Klasse erzeugt ViewHolder-Objekte, die die Datenobjekt fuer die RecyclerView enthalten Created by tina on 17.06.17.

14.3.1 Deklaration

```
public class ListViewHolder
 extends ViewHolder
```

14.3.2 statische Felder

- `public TextView` **title**
  - Titel des Items
- `public TextView` **subtitle**
  - Untertitel des Items
- `public ImageView` **icon**
  - Icon, das zum Item angezeigt werden soll

14.3.3 Konstruktoren

- **ListViewHolder**

```
public ListViewHolder(View itemView)
```

- **ListViewHolder**

```
public ListViewHolder(View itemView,OnListItemClicked
    onListItemClicked)
```

– **Description**
Konstruktor
– **Parameters**
* `itemView` – View, in der die Items angezeigt werden sollen
* `onListItemClicked` – ClickListener fuer ListItems

14.3.4 Methoden

- **onClick**

```
public void onClick(View v)
```

15 Package RecyclerView.listItems

15.1 Interface ListItem

Interface fuer ListItems, die die Datenobjekt in den verschiedenen RecyclerViews der App sind Created by tina on 18.06.17.

15.1.1 Deklaration

```
public interface ListItem
```

15.1.2 Subinterfaces

GOListItem (in 15.2, page 68), UserMailListItem (in 15.4, page 72), UserStatusListItem (in 15.5, page 74), GroupListItem (in 15.3, page 70)

15.1.3 Klassen, die das Interface implementieren

GOListItem (in 15.2, page 68), UserMailListItem (in 15.4, page 72), UserStatusListItem (in 15.5, page 74), GroupListItem (in 15.3, page 70)

15.1.4 Methoden

- **getIcon**

```
Icon getIcon()
```

– **Description**
getter-Methode fuer Icon des ListItems
– **Returns** – Icon des Datenobjekts

- **getSubtitle**

```
java.lang.String getSubtitle()
```

– **Description**
getter-Methode fuer Untertitel des ListItems. Muss ggfs. erst generiert werden, die Information wird als Datentyp T im Objekt gespeichert

- **Returns** – Untertitel des Datenobjekts

- **getTitle**

```
java.lang.String getTitle()
```

  - **Description**
    getter-Methode fuer ueberschrift des ListItems
  - **Returns** – Titel des Datenobjekts

- **setIcon**

```
void setIcon(Icon icon)
```

  - **Description**
    setter-Methode fuer icon des ListItems
  - **Parameters**
    * `icon` – das neue Icon

- **setSubtitle**

```
void setSubtitle(java.lang.Object t)
```

  - **Description**
    setter-Methode fuer Untertitel. Methode erwartet Datentyp T, der Untertitel wird dann innerhalb der Klasse als String-Objekt erzeugt
  - **Parameters**
    * `t` – Objekt/Datentyp, aus dem Untertitel erzeugt wird

- **setTitle**

```
void setTitle(java.lang.String title)
```

  - **Description**
    setter-Methode fuer ueberschrift des ListItems
  - **Parameters**
    * `title` – der neue Titel

15.2 Klasse GOListItem

Diese Klasse repraesentiert ListItems, die Informationen ueber ein Go in einem RecyclerView darstellen sollen Created by tina on 17.06.17.

15.2.1 Deklaration

```
public class GOListItem
 extends java.lang.Object implements ListItem
```

15.2.2 Konstruktoren

- **GOListItem**

  ```
  public GOListItem(edu.kit.pse17.go_app.model.entities.Go go)
  ```

  - **Description**
    Konstruktor
  - **Parameters**
    - `go` – Go-Objekt, das von dem ListItem repraesentiert werden soll

- **GOListItem**

  ```
  public GOListItem(java.lang.String name,java.util.Date start,Icon
      icon)
  ```

  - **Description**
    Konstruktor
  - **Parameters**
    - `name` – Go-Bezeichnung
    - `start` – Startzeitpunkt des GOs
    - `icon` – Go-Icon

15.2.3 Methoden

- **getIcon**

  ```
  Icon getIcon()
  ```

  - **Description copied from ListItem** (**in 15.1, page 67**)
    getter-Methode fuer Icon des ListItems
  - **Returns** – Icon des Datenobjekts

- **getSubtitle**

  ```
  java.lang.String getSubtitle()
  ```

  - **Description copied from ListItem** (**in 15.1, page 67**)
    getter-Methode fuer Untertitel des ListItems. Muss ggfs. erst generiert werden, die Information wird als Datentyp T im Objekt gespeichert
  - **Returns** – Untertitel des Datenobjekts

- **getTitle**

  ```
  java.lang.String getTitle()
  ```

  - **Description copied from ListItem (in 15.1, page 67)**
    getter-Methode fuer ueberschrift des ListItems
  - **Returns** – Titel des Datenobjekts

- **setIcon**

  ```
  void setIcon(Icon icon)
  ```

  - **Description copied from ListItem (in 15.1, page 67)**
    setter-Methode fuer icon des ListItems
  - **Parameters**
    * `icon` – das neue Icon

- **setSubtitle**

  ```
  public void setSubtitle(java.util.Date date)
  ```

- **setTitle**

  ```
  void setTitle(java.lang.String title)
  ```

  - **Description copied from ListItem (in 15.1, page 67)**
    setter-Methode fuer ueberschrift des ListItems
  - **Parameters**
    * `title` – der neue Titel

15.3 Klasse GroupListItem

Diese Klasse repraesentiert ListItems, die Informationen ueber eine Gruppe in einem Recycler-View darstellen sollen Created by tina on 17.06.17.

15.3.1 Deklaration

```
public class GroupListItem
 extends java.lang.Object implements ListItem
```

15.3.2 Konstruktoren

- **GroupListItem**

  ```
  public GroupListItem(edu.kit.pse17.go_app.model.entities.Group
      group)
  ```

– **Description**
Konstruktor
– **Parameters**
* `group` – gruppen-Objekt, das von dem ListItem repraesentiert werden soll

- **GroupListItem**

```
public GroupListItem(java.lang.String title, int memberCount, Icon
    icon)
```

– **Description**
Konstruktor
– **Parameters**
* `title` – Gruppenname
* `memberCount` – Anzahl der Gruppenmitglieder
* `icon` – Gruppenbild

15.3.3 Methoden

- **getIcon**

```
Icon getIcon()
```

– **Description copied from ListItem (in 15.1, page 67)**
getter-Methode fuer Icon des ListItems
– **Returns** – Icon des Datenobjekts

- **getSubtitle**

```
java.lang.String getSubtitle()
```

– **Description copied from ListItem (in 15.1, page 67)**
getter-Methode fuer Untertitel des ListItems. Muss ggfs. erst generiert werden, die Information wird als Datentyp T im Objekt gespeichert
– **Returns** – Untertitel des Datenobjekts

- **getTitle**

```
java.lang.String getTitle()
```

– **Description copied from ListItem (in 15.1, page 67)**
getter-Methode fuer ueberschrift des ListItems
– **Returns** – Titel des Datenobjekts

- **setIcon**

```
void setIcon(Icon icon)
```

- **Description copied from ListItem (in 15.1, page 67)**
  setter-Methode fuer icon des ListItems
- **Parameters**
  * `icon` – das neue Icon

- **setSubtitle**

```
public void setSubtitle(java.lang.Integer memberCount)
```

- **setTitle**

```
void setTitle(java.lang.String title)
```

- **Description copied from ListItem (in 15.1, page 67)**
  setter-Methode fuer ueberschrift des ListItems
- **Parameters**
  * `title` – der neue Titel

15.4 Klasse UserMailListItem

Diese Klasse repraesentiert ListItems, die Informationen ueber einen User in einem RecyclerView darstellen sollen Created by tina on 19.06.17.

15.4.1 Deklaration

```
public class UserMailListItem
 extends java.lang.Object implements ListItem
```

15.4.2 Konstruktoren

- **UserMailListItem**

```
public UserMailListItem(java.lang.String title,java.lang.String
    email,Icon icon)
```

- **Description**
  Konstruktor
- **Parameters**
  * `title` – Benutzername
  * `email` – EMail-Adresse, die zur Anmeldung verwendet wurde
  * `icon` – Profilbild

- **UserMailListItem**

```
public UserMailListItem(edu.kit.pse17.go_app.model.entities.User
    user)
```

– **Description**
Konstruktor
– **Parameters**
* `user` – Das User-Objekt, das von dem ListItem repraesentiert werden soll

15.4.3 Methoden

- **getIcon**

```
Icon getIcon()
```

– **Description copied from ListItem (in 15.1, page 67)**
getter-Methode fuer Icon des ListItems
– **Returns** – Icon des Datenobjekts

- **getSubtitle**

```
java.lang.String getSubtitle()
```

– **Description copied from ListItem (in 15.1, page 67)**
getter-Methode fuer Untertitel des ListItems. Muss ggfs. erst generiert werden, die Information wird als Datentyp T im Objekt gespeichert
– **Returns** – Untertitel des Datenobjekts

- **getTitle**

```
java.lang.String getTitle()
```

– **Description copied from ListItem (in 15.1, page 67)**
getter-Methode fuer ueberschrift des ListItems
– **Returns** – Titel des Datenobjekts

- **setIcon**

```
void setIcon(Icon icon)
```

– **Description copied from ListItem (in 15.1, page 67)**
setter-Methode fuer icon des ListItems
– **Parameters**
* `icon` – das neue Icon

- **setSubtitle**

```
public void setSubtitle(java.lang.String s)
```

- **setTitle**

```
void setTitle(java.lang.String title)
```

  - **Description copied from ListItem** (**in 15.1, page 67**)
    setter-Methode fuer ueberschrift des ListItems
  - **Parameters**
    * `title` – der neue Titel

15.5 Klasse UserStatusListItem

Diese Klasse repraesentiert ListItems, die Informationen ueber einen User in einem RecyclerView darstellen sollen Created by tina on 19.06.17.

15.5.1 Deklaration

```
public class UserStatusListItem
 extends java.lang.Object implements ListItem
```

15.5.2 Konstruktoren

- **UserStatusListItem**

```
public UserStatusListItem(java.lang.String title,edu.kit.pse17.
    go_app.model.Status status,Icon icon)
```

  - **Description**
    Konstruktor
  - **Parameters**
    * `title` – Benutzername
    * `status` – Status des Users
    * `icon` – Profilbild

- **UserStatusListItem**

```
public UserStatusListItem(edu.kit.pse17.go_app.model.entities.
    User user,edu.kit.pse17.go_app.model.entities.Go go)
```

15.5.3 Methoden

- **getIcon**

```
Icon getIcon()
```

  - **Description copied from ListItem** (**in 15.1, page 67**)
    getter-Methode fuer Icon des ListItems
  - **Returns** – Icon des Datenobjekts

- **getSubtitle**

  ```
  java.lang.String getSubtitle()
  ```

  - **Description copied from ListItem (in 15.1, page 67)**
    getter-Methode fuer Untertitel des ListItems. Muss ggfs. erst generiert werden, die Information wird als Datentyp T im Objekt gespeichert
  - **Returns** – Untertitel des Datenobjekts

- **getTitle**

  ```
  java.lang.String getTitle()
  ```

  - **Description copied from ListItem (in 15.1, page 67)**
    getter-Methode fuer ueberschrift des ListItems
  - **Returns** – Titel des Datenobjekts

- **setIcon**

  ```
  void setIcon(Icon icon)
  ```

  - **Description copied from ListItem (in 15.1, page 67)**
    setter-Methode fuer icon des ListItems
  - **Parameters**
    * `icon` – das neue Icon

- **setSubtitle**

  ```
  public void setSubtitle(edu.kit.pse17.go_app.model.Status s)
  ```

- **setTitle**

  ```
  void setTitle(java.lang.String title)
  ```

  - **Description copied from ListItem (in 15.1, page 67)**
    setter-Methode fuer ueberschrift des ListItems
  - **Parameters**
    * `title` – der neue Titel

16 Klassenübersicht - Server

Classes

- java.lang.Object
  - ClientCommunication.Downstream.FcmClient (in 19.2, page 108)
  - ClientCommunication.Upstream.GoRestController (in 20.1, page 110)
  - ClientCommunication.Upstream.GroupRestController (in 20.2, page 115)
  - ClientCommunication.Upstream.UserRestController (in 20.3, page 120)
  - Main (in 21.1, page 123)
  - PersistenceLayer.GoEntity (in 17.1, page 77)
  - PersistenceLayer.GroupEntity (in 17.2, page 81)
  - PersistenceLayer.UserEntity (in 17.4, page 85)
  - PersistenceLayer.daos.GoDaoImp (in 18.5, page 96)
  - PersistenceLayer.daos.GroupDaoImp (in 18.6, page 99)
  - PersistenceLayer.daos.UserDaoImp (in 18.7, page 102)
  - ServiceLayer.Cluster (in 22.4, page 128)
  - ServiceLayer.EntitiyRemovedObserver (in 22.5, page 129)
  - ServiceLayer.EntityAddedObserver (in 22.6, page 132)
  - ServiceLayer.EntityChangedObserver (in 22.7, page 135)
  - ServiceLayer.GoClusterStrategy (in 22.8, page 138)
  - ServiceLayer.LocationService (in 22.9, page 139)
  - ServiceLayer.UserLocation (in 22.10, page 142)
  - java.lang.Enum
    - ClientCommunication.Downstream.EventArg (in 19.1, page 106)
    - PersistenceLayer.Status (in 17.3, page 84)

Interfaces

- PersistenceLayer.daos.AbstractDao (in 18.1, page 88)
- PersistenceLayer.daos.GoDao (in 18.2, page 90)
- PersistenceLayer.daos.GroupDao (in 18.3, page 91)
- PersistenceLayer.daos.UserDao (in 18.4, page 94)
- ServiceLayer.ClusterStrategy (in 22.1, page 124)
- ServiceLayer.Observable (in 22.2, page 125)
- ServiceLayer.Observer (in 22.3, page 126)

17 Package PersistenceLayer

Package Inhalt *Page*

Klassen

GoEntity 77
Dies ist eine Entity Klasse.
GroupEntity 81
Dies ist eine Entity Klasse.
Status 84
Dieses Enum definiert die verschiedenen Teilnahmestatus, die ein Benutzer in einem GO innehaben kann.
UserEntity 85
Dies ist eine Entity Klasse.

17.1 Klasse GoEntity

Dies ist eine Entity Klasse. Sie wird von dem Framework Hibernate dazu verwendet, POJOS auf Tupel in einer Datenbank zu mappen. Wie das Mapping geschieht ist in dieser Klasse durch Annotations festgelegt. Der Rest der Anwendung kann somit überall mit Java-Objekten hantieren und muss sich nicht um die konkrete Implementierung der Datenbank kümmern. Der Zugriff auf die Datenbank erfolgt nicht in dieser Klasse, sondern nur über eine DAO Klasse, die das Interface GoDao implementiert. Zusätzlich dient diese Klasse als Vorlage des Frameworks Gson zum Parsen von JSON-Objekten, die von der REST API empfangen und gesendet werden. Die Attribute der Klasse bestimmen dabei die Struktur des JSON-Objekts.

17.1.1 Deklaration

```
public class GoEntity
 extends java.lang.Object
```

17.1.2 Konstruktoren Auflistung

GoEntity()

17.1.3 Methoden Auflistung

equals(Object)
getDescription()
getEnd()
getGoingUsers()
getGoneUsers()
getID()
getLat()
getLon()
getName()
getNotGoingUsers()
getStart()
hashCode()
setDescription(String)
setEnd(Date)
setGoingUsers(List)
setGoneUsers(List)
setID(long)
setLat(long)
setLon(long)
setName(String)
setNotGoingUsers(List)
setStart(Date)

17.1.4 Attribute

private long id Eine global eindeutige Nummer, anhand derer ein Go-Objekt eindeutig identifiziert werden kann. Die ID ist eine positive ganze Zahl im Wertebereich des Datentyps long. Nach Erzeugung des Objekts kann sie bis zu seiner Zerstörung nicht verändert werden.

private long groupId Die ID der Gruppe, in der dieses GO angelegt wurde. Es muss sich dabei um eine gültige Gruppen-ID handeln. Nach Erzeugung der Entity ist der Wert dieser Variable nicht mehr veränderbar.

private String owner Die userId des Benutzer der das GO erstellt hat und somit der Go-Verantwortliche ist. Es muss sich dabei um eine gültige UserId handeln. Nach Erzeugung der Entity ist der Wert dieser Variable nicht mehr veränderbar.

private String name Der Name des GOs. Dieser muss nicht eindeutig sein. Es handelt sich dabei um einen String, der weniger als 50 Zeichen enthält. Der Name eines GOs kann nachträglich (nach Erzeugung des Objekts) geändert werden, es sind entsprechende Methoden zu implementieren. Generiert wird die Id automatisch bei der Persistierung des Entity-Objekts in der Datenbank. Dadurch ist die Eindeutigkeit der ID garantiert.

private String description Eine textuelle Ebschreibung des GOs. Diese muss nicht eindeutig sein. Es handelt sich dabei um einen String, der weniger als 140 Zeichen enthält. Die Beschreibung eines GOs kann nachträglich (nach Erzeugung des Objekts) geändert werden, es sind entsprechende Methoden zu implementieren.

private Date start Der Startzeitpunkt des GOs. Er bestimmt ab wann die Standortverfolgung bei einem GO gestartet wird. Dabei darf der Zeitpunkt bei der Zuweisung der Variable nicht in der Vergangenheit befinden. Der Startzeitpunkt eines GOs kann nachträglich (nach Erzeugung des Objekts) geändert werden, es sind entsprechende Methoden zu implementieren.

private Date end Der Endzeitpunkt des GOs. Er bestimmt ab wann die Standortverfolgung bei einem GO gestoppt wird. Dabei darf der Zeitpunkt nie vor dem Startzeitpunkt befinden.

Der Endzeitpunkt eines GOs kann nachträglich (nach Erzeugung des Objekts) geändert werden, es sind entsprechende Methoden zu implementieren.

private long lat Falls es einen Zielort für das GO gibt, wird in diesem Feld der geografische Breitengrad des Zielorts gespeichert. Der Wert muss als Breitengrad interpretierbar sein, muss also zwischen +90 und -90 liegen. Wurde kein Zielort für das GO bestimmt, kann der Wert dieses Feldes auch null sein. Der Wert kann nachträglich (nach Erzeugung des Objekts) geändert werden, es sind entsprechende Methoden zu implementieren.

private long lon Falls es einen Zielort für das GO gibt, wird in diesem Feld der geografische Längengrad des Zielorts gespeichert. Der Wert muss als Längengrad interpretierbar sein, muss also zwischen +180 und -180 liegen.

Wurde kein Zielort für das GO bestimmt, kann der Wert dieses Feldes auch null sein.

Der Wert kann nachträglich (nach Erzeugung des Objekts) geändert werden, es sind entsprechende Methoden zu implementieren.

private Map<UserEntity, Status> userStatus Eine Map mit allen Teilnehmern des GOs, um ihnen ihren Teilnahmestatus zuzuweisen.

Bei Erzeugung eines Objekts dieser Klasse wird dem GO-Verantwortlichen automatisch der Status BESTÄTIGT zugewiesen, allen anderen Gruppenmitgliedern der Status ABGELEHNT. Es besteht keine Abhängigkeit dieser Liste zur LocationService-Klasse oder anderen Klassen, die von LocationService benutzt werden.

Es dürfen nur Benutzer Teil dieser Liste sein, die auch Teil der Gruppe sind, in der das GO erstellt wurde. Jedes Mitglied der Gruppe des GOs muss in der Liste

enthalten sein.

Diese Liste ist veränderlich, es müssen also entsprechende Methoden implementiert werden, um Objekte zu der Liste hinzufügen und entfernen zu können.

17.1.5 Konstruktoren

- **GoEntity**

```
public GoEntity()
```

17.1.6 Methoden

- **equals**

```
public boolean equals(java.lang.Object arg0)
```

- **getDescription**

```
public java.lang.String getDescription()
```

- **getEnd**

```
public java.util.Date getEnd()
```

- **getGoingUsers**

```
public java.util.List getGoingUsers()
```

- **getGoneUsers**

```
public java.util.List getGoneUsers()
```

- **getID**

```
public long getID()
```

- **getLat**

```
public long getLat()
```

- **getLon**

```
public long getLon()
```

- **getName**

```
public java.lang.String getName()
```

- **getNotGoingUsers**

```
public java.util.List getNotGoingUsers()
```

- **getStart**

```
public java.util.Date getStart()
```

- **hashCode**

```
public native int hashCode()
```

- **setDescription**

```
public void setDescription(java.lang.String description)
```

- **setEnd**

```
public void setEnd(java.util.Date end)
```

- **setGoingUsers**

```
public void setGoingUsers(java.util.List goingUsers)
```

- **setGoneUsers**

```
public void setGoneUsers(java.util.List goneUsers)
```

- **setLat**

```
public void setLat(long lat)
```

- **setLon**

```
public void setLon(long lon)
```

- **setName**

```
public void setName(java.lang.String name)
```

- **setNotGoingUsers**

```
public void setNotGoingUsers(java.util.List notGoingUsers)
```

- **setStart**

```
public void setStart(java.util.Date start)
```

17.2 Klasse GroupEntity

Dies ist eine Entity Klasse. Sie wird von dem Framework Hibernate dazu verwendet, POJOS auf Tupel in einer Datenbank zu mappen. Wie das Mapping geschieht ist in dieser Klasse durch Annotations festgelegt. Der Rest der Anwendung kann somit überall mit Java-Objekten hantieren und muss sich nicht um die konkrete Implementierung der Datenbank kümmern. Der Zugriff auf die Datenbank erfolgt nicht in dieser Klasse, sondern nur über eine DAO Klasse, die das Interface GroupDao implementiert. Zusätzlich dient diese Klasse als Vorlage des Frameworks Gson zum Parsen von JSON-Objekten, die von der REST API empfangen und gesendet werden. Die Attribute der Klasse bestimmen dabei die Struktur des JSON-Objekts.

17.2.1 Deklaration

```
public class GroupEntity
 extends java.lang.Object
```

17.2.2 Konstruktoren Auflistung

GroupEntity()

17.2.3 Methoden Auflistung

equals(Object)
getAdmins()
getDescription()
getID()
getMembers()
getName()
getRequests()
hashCode()
setAdmins(List)
setDescription(String)
setID(int)
setMembers(List)
setName(String)
setRequests(List)

17.2.4 Attribute

private long id Eine im gesamten System eindeutige ID-Nummer, anhand derer eine Gruppe eindeutig identifiziert werden kann. Die ID ist eine positive ganze Zahl im Wertebereich des Datentyps long. Nach Erzeugung des Objekts kann sie bis zu seiner Zerstörung nicht verändert werden. Generiert wird die Id automatisch bei der Persistierung des Entity-Objekts in der Datenbank. Dadurch ist die Eindeutigkeit der ID garantiert.

private String name Der Name der Gruppe. Dieser muss nicht eindeutig sein. Es handelt sich dabei um einen String, der weniger als 50 Zeichen enthält. Der Name einer Gruppe kann nachträglich (nach Erzeugung des Objekts) geändert werden, es sind entsprechende Methoden zu implementieren.

private String description Eine textuelle Beschreibung der Gruppe. Diese muss nicht eindeutig sein. Es handelt sich dabei um einen String, der weniger als 140

Zeichen enthält. Die Beschreibung einer Gruppe kann nachträglich (nach Erzeugung des Objekts) geändert werden, es sind entsprechende Methoden zu implementieren.

private List<UserEntity> members Eine Liste, die alle Benutzer enthält, die vollwertige Mitglieder der Gruppe sind, also Benutzer die eine Mitgliedsanfrage erhalten und diese bestätigt haben, plus der Ersteller der Gruppe, sollte er noch nicht ausgetreten sein.

Bei der Erzeugung eines Objekts dieser Klasse, wird der Ersteller automatisch diser Liste hinzugefügt.

Die Länge der Liste liegt zwischen 1 und 50. Sie kann niemals komplett leer sein. Gruppen mit 0 Mitgliedern werden automatisch gelöscht.

Die Liste muss auch nach der Erzeugung des Objekts veränderbar sein, entsprechende Methoden sind zu implementieren.

private List<UserEntity> requests Eine Liste, die alle Benutzer enthält, die eine Mitgliedschaftsanfrage zu dieser Gruppe erhalten haben, diese aber noch nicht beantwortet haben.

Bei Erzeugung dieses Objekts ist die Liste leer. Die Länge der Liste kann zwischen 0 und 50 liegen. Dabei hängt die maximale Länge auch von der aktuellen Mitgliederanzahl ab. Die Summe von beidem darf 50 nicht übersteigen.

Die Liste muss auch nach der Erzeugung des Objekts veränderbar sein, entsprechende Methoden sind zu implementieren.

private List<UserEntity> admins Eine Liste mit allen Admins der Gruppe. Jeder Benutzer, der Teil dieser Liste ist, muss auch teil der members-Liste sein, da nur ein Gruppenmitglied Administrator einer Gruppe sein kann.

Bei der Erzeugung eines Objekts dieser Klasse, wird der Ersteller automatisch diser Liste hinzugefügt.

Die Länge der Liste liegt zwischen 1 und 50. Sie kann niemals komplett leer sein. Gruppen mit 0 Mitgliedern werden automatisch gelöscht.

Die Liste muss auch nach der Erzeugung des Objekts veränderbar sein, entsprechende Methoden sind zu implementieren.

private List<GoEntity> gos Eine Liste mit allen GOs, die in dieser Gruppe erstellt wurden. DAbei handelt es sich nur um GOs, die gerade aktiv sind, oder in Zukunft noch aktiv sein werden. GOs, die schon vüruber sind werden automatisch aus dieser Liste gelöscht.

Bei Erstellung eines Objekts dieser Klasse ist diese Liste leer. Die Länge der Liste liegt zwischen 0 und 10 GOs.

Die Liste muss auch nach der Erzeugung des Objekts veränderbar sein, entsprechende Methoden sind zu implementieren.

17.2.5 Konstruktoren

- **GroupEntity**

```
public GroupEntity()
```

17.2.6 Methoden

- **equals**

```
public boolean equals(java.lang.Object arg0)
```

- **getAdmins**

```
public java.util.List getAdmins()
```

- **getDescription**

```
public java.lang.String getDescription()
```

- **getID**

```
public int getID()
```

- **getMembers**

```
public java.util.List getMembers()
```

- **getName**

```
public java.lang.String getName()
```

- **getRequests**

```
public java.util.List getRequests()
```

- **hashCode**

```
public native int hashCode()
```

- **setAdmins**

```
public void setAdmins(java.util.List admins)
```

- **setDescription**

```
public void setDescription(java.lang.String description)
```

- **setID**

```
public void setID(int ID)
```

- **setMembers**

```
public void setMembers(java.util.List members)
```

- **setName**

```
public void setName(java.lang.String name)
```

- **setRequests**

```
public void setRequests(java.util.List requests)
```

17.3 Enum Status

Dieses Enum definiert die verschiedenen Teilnahmestatus, die ein Benutzer in einem GO innehaben kann.

17.3.1 Deklaration

```
public final class Status
 extends java.lang.Enum
```

17.3.2 Attribute

ABGELEHNT bedeutet, dass der Benutzer nicht an dem GO teilnehmen wird.
BESTÄTIGT bedeutet, dass der Benutzer an dem GO teilnehmen wird.
UNTERWEGS bedeutet, dass der Benutzer an dem GO teilnimmt und bereits aktiv ist, d.h. er teilt gerade seinen Standort mit anderen GO-Teilnehmern.

17.3.3 Methoden Auflistung

valueOf(String)
values()

17.3.4 Felder

- `public static final Status` **ABGELEHNT**
  - bedeutet, dass der Benutzer nicht an dem GO teilnehmen wird. Er teilt seinen Standort nicht mit anderen und kann auch die Standorte der anderen Go-Teilnehmer nicht sehen. Dieser Teilnahmestatus ist der default-Status bei der Erstellung eines GOs für alle Teilnehmer, außer dem Ersteller selbst. Der Ersteller kann niemals den Status ÄBGELEHNTännehmen.
- `public static final Status` **BESTÄTIGT**
  - bedeutet, dass der Benutzer an dem GO teilnehmen wird. Er ist in dem GO allerdings noch nicht aktiv, d.h. er teilt seinen Standort nicht mit anderen. Gibt es andere Teilnehmer in dem GO, die bereits ÜNTERWEGSßind, kann der Benutzer deren Standorte sehen. Dieser Teilnahmestatus ist der default-Status für den Ersteller eines GOs.
- `public static final Status` **UNTERWEGS**

– bedeutet, dass der Benutzer an dem GO teilnimmt und bereits aktiv ist, d.h. er teilt gerade seinen Standort mit anderen GO-Teilnehmern. Dieser kann von Benutzern mit dem Status ÜNTERWEGSöder "BESTÄTIGTängesehen werden. Er selbst kann den Standort von anderen aktiven Go-Teilnehmern sehen.

17.3.5 Methoden

- **valueOf**

```
public static Status valueOf(java.lang.String name)
```

- **values**

```
public static Status[] values()
```

17.3.6 Von der Klasse Enum vererbte Methoden

`java.lang.Enum`

- `protected final Object` **clone**`() throws CloneNotSupportedException`
- `public final int` **compareTo**`(Enum` **arg0**`)`
- `public final boolean` **equals**`(Object` **arg0**`)`
- `protected final void` **finalize**`()`
- `public final Class` **getDeclaringClass**`()`
- `public final int` **hashCode**`()`
- `public final String` **name**`()`
- `public final int` **ordinal**`()`
- `public String` **toString**`()`
- `public static Enum` **valueOf**`(Class` **arg0**`, String` **arg1**`)`

17.4 Klasse UserEntity

Dies ist eine Entity Klasse. Sie wird von dem Framework Hibernate dazu verwendet, POJOS auf Tupel in einer Datenbank zu mappen. Wie das Mapping geschieht ist in dieser Klasse durch Annotations festgelegt. Der Rest der Anwendung kann somit überall mit Java-Objekten hantieren und muss sich nicht um die konkrete Implementierung der Datenbank kümmern. Der Zugriff auf die Datenbank erfolgt nicht in dieser Klasse, sondern nur über eine DAO Klasse, die das Interface UserDao implementiert. Zusätzlich dient diese Klasse als Vorlage des Frameworks Gson zum Parsen von JSON-Objekten, die von der REST API empfangen und gesendet werden. Die Attribute der Klasse bestimmen dabei die Struktur des JSON-Objekts.

17.4.1 Deklaration

```
public class UserEntity
 extends java.lang.Object
```

17.4.2 Konstruktoren Auflistung

UserEntity()

17.4.3 Methoden Auflistung

equals(Object)
getEmail()
getInstanceId()
getName()
getUid()
hashCode()
setEmail(String)
setInstanceId(String)
setName(String)
setUid(String)

private String uid Eine im System eindeutige UserID. Diese ID wird generiert von dem Firebase Authentication Service und von dieser Andwendung unverändert übernommen. Die ID wird nach der Registrierung nicht mehr verändert, bis der User seinen Account löscht.

private String instanceID Ein String, mit dem das Gerät, das der Benutzer im Augenblick benutzt identifiziert werden kann. Diese ID wird vom Firebease Cloud Messaging Service benötigt und erlaubt es dem Server eine Nachricht an einen Client zu schicken, ohne dass dieser Client vorher den Server angesprochen haben muss.

Da sich durch Gerätewechsel oder Konfigurationsänderungen am Gerät die InstanceId ändern kann, muss die DAO-Klasse eine Methode zur Änderung der InstanceId besitzen.

private String name Der Benutzername des Users. Dieser muss nicht eindeutig sein. Er kann vom User nicht selbst bestimmt werden, sondern wird übernommen, von dem Google-Account mit dem sich der User in der App angemeldet hat. Da der Benutzer nach der Regsitrierun diesen Account nicht wechseln kann, bleibt auch der Benutzername die ganze Zeit unverändert.

private String email Die Email-Adresse, die mit dem Google-Account asoziiert ist, mit dem sich der Benutzer registriert hat. Da der Benutzer nach der Registrierung diesen Account nicht wechseln kann, bleibt auch der Benutzername die ganze Zeit unverändert.

private List<GroupEntity> groups Eine Liste mit allen Gruppen, in denen der Benutzer Mitglied ist. Dieses Feld enthält einen ForeignKeyConstraint: Die IDs der Gruppenobjekte der Liste sind die Primärschlüssel in der Gruppenrelation. Für dieses Feld müssen Methoden zum Ändern der Liste vorhanden sein, da sich die Gruppen, in den der Benutzer Mitglied ist verändern können.

17.4.4 Konstruktoren

- **UserEntity**

```
public UserEntity()
```

17.4.5 Methoden

- **equals**

```
public boolean equals(java.lang.Object arg0)
```

- **getEmail**

```
public java.lang.String getEmail()
```

- **getInstanceId**

```
public java.lang.String getInstanceId()
```

- **getName**

```
public java.lang.String getName()
```

- **getUid**

```
public java.lang.String getUid()
```

- **hashCode**

```
public native int hashCode()
```

- **setEmail**

```
public void setEmail(java.lang.String email)
```

- **setInstanceId**

```
public void setInstanceId(java.lang.String instanceId)
```

- **setName**

```
public void setName(java.lang.String name)
```

- **setUid**

```
public void setUid(java.lang.String uid)
```

18 Package PersistenceLayer.daos

Package Contents *Page*

Interfaces

AbstractDao 88
Bei diesem Interface handelt es sich um ein Interface für eine Data Access Object Klasse, die die Datenbankzugriffe in sich kapselt.
GoDao 90

Bei diesem Interface handelt es sich um ein Interface für eine Data Access Object Klasse, die die Datenbankzugriffe in sich kapselt.
GroupDao 91
Bei diesem Interface handelt es sich um ein Interface für eine Data Access Object Klasse, die die Datenbankzugriffe in sich kapselt.
UserDao 94
Bei diesem Interface handelt es sich um ein Interface für eine Data Access Object Klasse, die die Datenbankzugriffe in sich kapselt.

Klassen
GoDaoImp 96
Created by tina on 30.06.17.
GroupDaoImp 99
Created by tina on 30.06.17.
UserDaoImp 102
Diese Klasse implementiert die Interfaces UserDao, AbstractDao und Observable.

18.1 Interface AbstractDao

Bei diesem Interface handelt es sich um ein Interface für eine Data Access Object Klasse, die die Datenbankzugriffe in sich kapselt. Dieses Interface definiert Methoden auf einer Datenbank, die standardmäßig für jede Tabelle zur Verfügung stehen sollten (CRUD). daher wird dieses Interface von jeder DAO-Klasse implementiert. Das Interface beinhaltet zwei Generics. Das Generic T definiert den entity-Typen, der in der jeweiligen Tabelle verwaltet wird. Das Generic PK ist der Datentyps des Primärschlüssels der Datenbanktabelle. Die Methoden dieses Interfaces werden von dieser DAO Klasse implementiert und sind nach außen sichtbar. Sie werden aufgerufen, von den RestController-Klassen, denn von dort werden die Server-Anfragen, die von Clients gestellt werden, an die Persistence-Klassen weitergeleitet.

18.1.1 Deklaration

```
public interface AbstractDao
```

18.1.2 Subinterfaces

GoDaoImp (in 18.5, page 96), UserDaoImp (in 18.7, page 102), GroupDaoImp (in 18.6, page 99)

18.1.3 Klassen, die das Interface implementieren

GoDaoImp (in 18.5, page 96), UserDaoImp (in 18.7, page 102), GroupDaoImp (in 18.6, page 99)

18.1.4 Methoden Auflistung

delete(PK) Diese Methode löscht ein Entity-Objekt aus der Datenbank.
get(PK) Diese Methode gibt ein Entity-Objekt zurück, das anhand seines Primärschlüssels aus der Datenbank geholt wurde.
persist(T) Diese Methode speichert eine neue Entity vom Typ T in der Datenbank ab. dabei wird das Entity-Objekt vor dem Methodenaufruf erzeugt und der Methode fertig"übergeben.
update(T) Diese Methode ändert Attributwerte eines bereits bestehenden Entity-Objekts.

18.1.5 Methoden

- **delete**

  ```
  void delete(java.lang.Object key) throws EntityNotFoundException
  ```

  - **Description**
    Diese Methode löscht ein Entity-Objekt aus der Datenbank. Es werden außerdem automatisch alle Entities gelöscht, die mit der gelöschten Entity assoziiert sind (Auch in anderen Datenbanktabellen, sodass der Datenbestand nach der Ausführung der Methode konsistent ist)
  - **Parameters**
    * `key` – Der Primärschlüssel der Entity, die aus der Datenbanktabelle gelöscht werden soll. Der Datentyp wird durch das Generic PK bei der Implementierung der Klasse spezifiziert.
  - **Throws**
    * `EntityNotFoundException` – existiert keine Entity mit dem spezifizierten Schlüssel, wird eine EntityNotFoundException geworfen, die von der aufrufenden Klasse behandelt werden muss.

- **get**

  ```
  java.lang.Object get(java.lang.Object key)
  ```

  - **Description**
    Diese Methode gibt ein Entity-Objekt zurück, das anhand seines Primärschlüssels aus der Datenbank geholt wurde.
  - **Parameters**
    * `key` – Der Primärschlüssel der Entity, die aus der Datenbank geholt werden soll. Der Datentyp wird von dem Generic PK bestimmt, mit dem das Interface implementiert wird.
  - **Returns** – Ein Entity-Objekt, das durch den Schlüssel identifiziert wurde. Konnte keine passende Entity in der Datenbank gefunden werden, gibt die Methode null zurück.

- **persist**

  ```
  void persist(java.lang.Object entity)
  ```

  - **Description**
    Diese Methode speichert eine neue Entity vom Typ T in der Datenbank ab. dabei wird das Entity-Objekt vor dem Methodenaufruf erzeugt und der Methode fertig"übergeben.
  - **Parameters**
    * `entity` – Das Entity-Objekt, das in der Datenbank gespeichert werden soll. Es wird garantiert, dass das Objekt, welches der Methode übergeben wird gültig ist (alle Konsistenzbedingungen der Datenbank werden erfüllt).

- **update**

  ```
  void update(java.lang.Object t) throws EntityNotFoundException
  ```

  - **Description**
    Diese Methode ändert Attributwerte eines bereits bestehenden Entity-Objekts. Dabei können nicht in jeder Entity-Klasse alle Attribute geändert werden. Welche Attribute geändert werden können ist in den Entity-Klassen und in den implementierenden Dao-Klassen spezifiziert.
  - **Parameters**
    * `t` – Ein Entity-Objekt, dass die geänderten Daten enthält. Das Objekt enthält die ID der Entity, die geändert werden soll und die Werte der Attribute die neu zugewiesen werden sollen. Alle anderen Attribute sind null, was der Methode signalisiert, dass diese Werte nicht geändert werden müssen.
  - **Throws**
    * `EntityNotFoundException` – existiert keine Entity mit dem spezifizierten Schlüssel, wird eine EntityNotFoundException geworfen, die von der aufrufenden Klasse behandelt werden muss.

18.2 Interface GoDao

Bei diesem Interface handelt es sich um ein Interface für eine Data Access Object Klasse, die die Datenbankzugriffe in sich kapselt. Die Methoden dieses Interfaces werden von dieser DAO Klasse implementiert und sind nach außen sichtbar. Sie werden aufgerufen, von den RestController-Klassen, denn von dort werden die Server-Anfragen, die von Clients gestellt werden, an die Persistence-Klassen weitergeleitet.

18.2.1 Deklaration

```
public interface GoDao
```

18.2.2 Subinterfaces

GoDaoImp (in 18.5, page 96), GroupDaoImp (in 18.6, page 99)

18.2.3 All classes known to implement interface

GoDaoImp (in 18.5, page 96), GroupDaoImp (in 18.6, page 99)

18.2.4 Methoden Auflistung

changeStatus(String, long, Status) Mit dieser Methode wird der Teilnahmestatus eines GO-Teilnehmers geändert.

18.2.5 Methoden

- **changeStatus**

  ```
  void changeStatus(java.lang.String userId,long goId,edu.kit.pse17
      .go_app.PersistenceLayer.Status status)
  ```

– **Description**

Mit dieser Methode wird der Teilnahmestatus eines GO-Teilnehnmers geändert. Der Status kann entweder ÄBGELEHNT", "BESTÄTIGT", oder ÜNTERWEGSlauten. Vor dem Aufruf der Methode muss sichergestellt werden, dass es sich bei dem Benutzer um ein Mitglied der Gruppe handelt und die Statusänderung die vorgenommen werden soll legal ist.

– **Parameters**

* `userId` – Die ID des Benutzers, dessen Teilnahmestatus geändert werden soll. Dabei handelt es sich um eine gültige Id, ansonsten kann die Methode nicht erfolgreich ausgeführt werden.
* `goId` – Die des GOs, für den der Teilnahmestatus geändert werden soll. Dabei handelt es sich um eine gültige Id, ansonsten kann die Methode nicht erfolgreich ausgeführt werden.
* `status` – Der neue Status des Benutzers.

18.3 Interface GroupDao

Bei diesem Interface handelt es sich um ein Interface für eine Data Access Object Klasse, die die Datenbankzugriffe in sich kapselt. Die Methoden dieses Interfaces werden von dieser DAO Klasse implementiert und sind nach außen sichtbar. Sie werden aufgerufen, von den RestController-Klassen, denn von dort werden die Server-Anfragen, die von Clients gestellt werden, an die Persistence-Klassen weitergeleitet.

18.3.1 Deklaration

```
public interface GroupDao
```

18.3.2 Methoden Auflistung

addAdmin(String, String) Diese Methode fügt einen Administrator bei einer Gruppe hinzu.

addGroupMember(String, long) Fügt der Gruppe mit der Id groupId den Benutzer mit der Id userId hinzu.

addGroupRequest(String, long) Mit dieser Methode lässt sich eine neue Gruppenanfrage in der Datenbank speichern.

removeGroupMember(String, long) Diese Methode entfernt ein Gruppenmitglied aus einer Gruppe.

removeGroupRequest(String, long) Diese Methode entfernt eine Gruppenmitgliedschaftsanfrage aus der Datenbank.

18.3.3 Methoden

- **addAdmin**

```
void addAdmin(java.lang.String userId,java.lang.String groupId)
    throws EntityNotFoundException
```

– **Description**

Diese Methode fügt einen Administrator bei einer Gruppe hinzu. Anfrage zu dieser Methode sollte nur von anderen Administratoren dieser Gruppe kommen. Es muss vor dem Aufruf der Methode sichergestellt werden, dass es sich bei dem Benutzer um

ein vollwertiges Gruppenmitglied handelt und dieser nicht bereits ein Administrator ist.

- **Parameters**
  - `userId` – Die ID des Benutzers, der als Administrator hinzugefügt werden soll. Dabei handelt es sich um eine gültige Id, ansonsten kann die Methode nicht erfolgreich ausgeführt werden.
  - `groupId` – die ID der Gruppe, zu der der Administrator hinzugefügt werden soll. Dabei handelt es sich um eine gültige Id, ansonsten kann die Methode nicht erfolgreich ausgeführt werden.
- **Throws**
  - `EntityNotFoundException` – existiert keine Entity mit dem spezifizierten Schlüssel, wird eine EntityNotFoundException geworfen, die von der aufrufenden Klasse behandelt werden muss.

- **addGroupMember**

```
void addGroupMember(java.lang.String userId,long groupId) throws
    EntityNotFoundException
```

- **Description**

  Fügt der Gruppe mit der Id groupId den Benutzer mit der Id userId hinzu. Vor dem Aufruf dieser Methode, muss der Aufrufer sicherstellen, dass der Benutzer nicht bereits ein Mitglied in der Gruppe ist und, dass der Benutzer zuvor eine Mitgliedschaftsanfrage zu der Gruppe bekommen hat bzw. es sich um dem Ersteller einer neuen Gruppe handelt.
- **Parameters**
  - `groupId` – Die ID der Gruppe, zu der der Benutzer hinzugefügt werden soll. Dabei handelt es sich um eine gültige Id, ansonsten kann die Methode nicht erfolgreich ausgeführt werden.
  - `userId` – Die ID des Benutzers, der der Gruppe hinzugefügt werden soll. Dabei handelt es sich um eine gültige Id, ansonsten kann die Methode nicht erfolgreich ausgeführt werden.
- **Throws**
  - `EntityNotFoundException` – existiert keine Entity mit dem spezifizierten Schlüssel, wird eine EntityNotFoundException geworfen, die von der aufrufenden Klasse behandelt werden muss.

- **addGroupRequest**

```
void addGroupRequest(java.lang.String userId,long groupId) throws
    EntityNotFoundException
```

- **Description**

  Mit dieser Methode lässt sich eine neue Gruppenanfrage in der Datenbank speichern. Sie muss also aufgerufen werden, wenn ein Administrator einen Benutzer zur Gruppe einlädt. Vor dem Aufruf der Methode muss sichergestellt werden, dass der Empfänger der Anfrage kein Gruppenmitglied ist und auch noch keine Anfrage erhalten hat.
- **Parameters**

    - `userId` – Die ID des Benutzers, der zu der Gruppe eingeladen wird. Dabei handelt es sich um eine gültige Id, ansonsten kann die Methode nicht erfolgreich ausgeführt werden.
    - `groupId` – die ID der Gruppe, zu der der Benutzer eingeladen wird. Dabei handelt es sich um eine gültige Id, ansonsten kann die Methode nicht erfolgreich ausgeführt werden.
  - **Throws**
    - `EntityNotFoundException` – existiert keine Entity mit dem spezifizierten Schlüssel, wird eine EntityNotFoundException geworfen, die von der aufrufenden Klasse behandelt werden muss.

- **removeGroupMember**

```
void removeGroupMember(java.lang.String userId,long groupId)
    throws EntityNotFoundException
```

  - **Description**
    Diese Methode entfernt ein Gruppenmitglied aus einer Gruppe. Sie wird aufgerufen, wenn entweder ein Gruppenmitglied aus einer Gruppe austritt oder ein Administrator ein gruppenmitglied entfernt. Vor dem Aufruf der Methode muss sichergestellt werden, dass es sich bei dem Benutzer um ein Mitglied der Gruppe handelt. Diese Methode kann nicht dazu verwendet werden, eine Gruppenanfrage zu löschen. Sämtliche GOs, die dem entfernten Gruppenmitglied gehören werden automatisch gelöscht bei Aufruf der Methode, um die Konsistenz des Datenbestands zu erhalten.
  - **Parameters**
    - `groupId` – die ID der Gruppe, aus der der Benutzer entfernt werden soll. Dabei handelt es sich um eine gültige Id, ansonsten kann die Methode nicht erfolgreich ausgeführt werden.
    - `userId` – Die ID des Benutzers, der aus der Gruppe entfernt werden soll. Dabei handelt es sich um eine gültige Id, ansonsten kann die Methode nicht erfolgreich ausgeführt werden.
  - **Throws**
    - `EntityNotFoundException` – existiert keine Entity mit dem spezifizierten Schlüssel, wird eine EntityNotFoundException geworfen, die von der aufrufenden Klasse behandelt werden muss.

- **removeGroupRequest**

```
void removeGroupRequest(java.lang.String userId,long groupId)
    throws EntityNotFoundException
```

  - **Description**
    Diese Methode entfernt eine Gruppenmitgliedschaftsanfrage aus der Datenbank. Sie wird aufgerufen, wenn ein Benutzer eine Gruppenmitgliedschaftsanfrage beantwortet hat.
  - **Parameters**
    - `groupId` – die ID der Gruppe, zu der der Benutzer eingeladen war. Dabei handelt es sich um eine gültige Id, ansonsten kann die Methode nicht erfolgreich ausgeführt werden.

* `userId` – Die ID des Benutzers, der zu der Gruppe eingeladen war. Dabei handelt es sich um eine gültige Id, ansonsten kann die Methode nicht erfolgreich ausgeführt werden.

– **Throws**

* `EntityNotFoundException` – existiert keine Entity mit dem spezifizierten Schlüssel, wird eine EntityNotFoundException geworfen, die von der aufrufenden Klasse behandelt werden muss.

18.4 Interface UserDao

Bei diesem Interface handelt es sich um ein Interface für eine Data Access Object Klasse, die die Datenbankzugriffe in sich kapselt. Die Methoden dieses Interfaces werden von dieser DAO Klasse implementiert und sind nach außen sichtbar. Sie werden aufgerufen, von den RestController-Klassen, denn von dort werden die Server-Anfragen, die von Clients gestellt werden, an die Persistence-Klassen weitergeleitet.

18.4.1 Deklaration

```
public interface UserDao
```

18.4.2 All known subinterfaces

UserDaoImp (in 18.7, page 102)

18.4.3 Klassen, die das Interface implementieren

UserDaoImp (in 18.7, page 102)

18.4.4 Methoden Auflistung

addUser(UserEntity) Die Methode fügt eine neue UserEntity in die Datenbank ein.
deleteUser(String) Diese Methode entfernt eine Entity aus der Datenbank.
getGroups(String) Diese Methode gibt eine Liste mit allen Gruppen zurück, in denen der Benutzer Mitglied ist.
getRequests(String) Diese Methode gibt eine Liste von Gruppen zurück, zu denen der Benutzer eine Gruppenanfrage erhalten hat, die er noch nicht beantwortet hat.
getUserByEmail(String) Diese Methode sucht ein User-Objekt anhand einer E-Mailadresse und gibt, falls die Suche erfolgreich ist, dieses Objekt zurück.

18.4.5 Methoden

- **addUser**

```
void addUser(edu.kit.pse17.go_app.PersistenceLayer.UserEntity
    user)
```

– **Description**
Die Methode fügt eine neue UserEntity in die Datenbank ein.

– **Parameters**

* `user` – Die Entity, die in die Datenbank eingefügt werden soll. Dieses Objekt muss eine in der Datenbank noch nicht vorhandene ID enthalten, sonst schlägt die Ausführung fehl.

- **deleteUser**

```
void deleteUser(java.lang.String userId)
```

  - **Description**
    Diese Methode entfernt eine Entity aus der Datenbank. Zusätzlich werden alle mit diesem Benutzer assoziierten Objekte ebenfalls entfernt. Dazu gehören: - GOs, bei denen der Benutzer der GO-Verantwortliche war - Gruppen, bei denen der Benutzer der einzige Administrator war - Gruppenmitgliedschaften des Benutzers - unbeantwortete Gruppenanfragen, die an den Benutzer gestellt wurden
  - **Parameters**
    * `userId` – Die userId des Benutzers, dessen Account gelöscht werden soll. Es wird garantiert, dass es sich beim Aufruf der Methode, um eine gültige ID handelt.

- **getGroups**

```
java.util.List getGroups(java.lang.String userId)
```

  - **Description**
    Diese Methode gibt eine Liste mit allen Gruppen zurück, in denen der Benutzer Mitglied ist. Dies schließt Gruppen nicht mit ein, zu denen der Benutzer eingeladen wurde, er die Gruppenanfrage aber noch nicht beantwortet hat.
  - **Parameters**
    * `userId` – Die ID des Benutzers, dessen Gruppen zurückgegeben werden sollen. Es wird garantiert, dass es sich beim Aufruf der Methode um eine gültige userid handelt.
  - **Returns** – Eine Liste mit GroupEntities. Die Länge der Liste liegt zwischen 0 und 300. Bei allen Listenelementen handelt es sich um vollständige, gültige GroupEntity Objekte.

- **getRequests**

```
java.util.List getRequests(java.lang.String userId)
```

  - **Description**
    Diese Methode gibt eine Liste von Gruppen zurück, zu denen der Benutzer eine Gruppenanfrage erhalten hat, die er noch nicht beantwortet hat.
  - **Parameters**
    * `userId` – Die ID des Benutzers, dessen Gruppenanfragen zurückgegeben werden sollen. Es wird garantiert, dass es sich beim Aufruf der Methode um eine gültige userId handelt.
  - **Returns** – Eine Liste mit GroupEntities. Die Länge der Liste liegt zwischen 0 und 300. Bei allen Listenelementen handelt es sich um vollständige, gültige GroupEntity Objekte.

- **getUserByEmail**

```
edu.kit.pse17.go_app.PersistenceLayer.UserEntity getUserByEmail(
    java.lang.String mail)
```

  - **Description**
    Diese Methode sucht ein User-Objekt anhand einer E-Mailadresse und gibt, falls die Suche erfolgreich ist, dieses Objekt zurück.
  - **Parameters**
    - `mail` – Die E-Mailadresse, anhand derer der Benutzer gesucht werden soll. Der String muss keinem besonderen Muster entsprechen, damit diese Methode fehlerfrei ausgeführt werden kann.
  - **Returns** – Die Methode gibt das gefundene UserEntity Objekt zurück. Gibt es keinen Benutzer mit der übergebenen E-mailadresse, gibt die Methode null zurück.

18.5 Klasse GoDaoImp

Diese Klasse implementiert die Interfaces GoDao, AbstractDao und Observable. Sie übernimmt die konkreten Datenbankzugriffe auf die Tabelle "gos". Dazu werden alle Methoden aus den DAO Interfaces entsprechend implementiert. Aufgerufen werden die Methoden dieser Klasse von den RestController-Klassen, wenn ein Client dem Server eine Anfrage zur Manipulation seiner Daten geschickt hat. Die Klasse gehört außerdem zu einer Implementierung des Beobachter-Entwurfsmusters und übernimmt dabei die Rolle des konkreten Subjekts. Die Klasse hat eine Liste von Beobachtern, die benachrichtigt werden, wenn sich in der Datenbank eine Änderung ergibt. Es ist die Verantwortung der Beobachter zu entscheiden, ob die Änderung eine Folgeaktion auslöst oder nicht.

18.5.1 Deklaration

```
public class GoDaoImp
 extends java.lang.Object implements AbstractDao, GoDao, edu.kit.
    pse17.go_app.ServiceLayer.Observable
```

18.5.2 Konstruktoren Auflistung

GoDaoImp()

18.5.3 Methoden Auflistung

changeStatus(String, long, Status)
delete(Long)
get(Long)
notify(String, Observable, GoEntity)
persist(GoEntity)
register(Observer)
unregister(Observer)
update(GoEntity)

18.5.4 Attribute

private SessionFactory sessionFactory Eine Sessionfactory, die Sessions bereitstellt. Die Sessions werden benötigt, damit die Klasse direkt mit der Datenbank kommunizieren kann und dort die Änderungen vornehmen. Das Attribut ist mit '@Autowired' annotiert, damit es automatisch mit einem gültigen Objekt instanziiert wird.

Auf dieses Feld darf nur innerhalb dieser Klasse zugegriffen werden. Nach der Instanzizerung ist diese Objekt unveränderbar und bleibt bestehen, bis die Instanz dieser Klasse wieder zerstört wird.

private List<Observer> observer Eine Liste mit Observern, die benachrichtigt werden, sobald eine Änderung an der Datenbank vorgenommen wird, die auch die Daten anderer Benutzer betrifft.

18.5.5 Konstruktoren

- **GoDaoImp**

```
public GoDaoImp()
```

18.5.6 Methoden

- **changeStatus**

```
public void changeStatus(java.lang.String userId,long goId,edu.
    kit.pse17.go_app.PersistenceLayer.Status status)
```

  - **Parameters**
    - `userId` – Die ID des Benutzers, dessen Teilnahmestatus geändert werden soll. Dabei handelt es sich um eine gültige Id, ansonsten kann die Methode nicht erfolgreich ausgeführt werden.
    - `goId` – Die des GOs, für den der Teilnahmestatus geändert werden soll. Dabei handelt es sich um eine gültige Id, ansonsten kann die Methode nicht erfolgreich ausgeführt werden.
    - `status` – Der neue Status des Benutzers.

- **delete**

```
public void delete(java.lang.Long key) throws
    EntityNotFoundException
```

  - **Parameters**
    - `key` – Der Primärschlüssel der Entity, die aus der Datenbanktabelle gelöscht werden soll. Der Datentyp wird durch das Generic PK bei der Implementierung der Klasse spezifiziert.
  - **Throws**
    - `EntityNotFoundException` –

- **get**

  ```
  public edu.kit.pse17.go_app.PersistenceLayer.GoEntity get(java.
      lang.Long key)
  ```

  - **Parameters**
    - `key` – Der Primärschlüssel der Entity, die aus der Datenbank geholt werden soll. Der Datentyp wird von dem Generic PK bestimmt, mit dem das Interface implementiert wird.
  - **Returns** –

- **notify**

  ```
  public void notify(java.lang.String impCode,edu.kit.pse17.go_app.
      ServiceLayer.Observable observable,edu.kit.pse17.go_app.
      PersistenceLayer.GoEntity goEntity)
  ```

  - **Parameters**
    - `impCode` – Ein Code, der angibt, welche Observer-Implementierung benachrichtigt werden soll. dabei handelt es sich immer um ein öffentliches statisches Attribut in der Observer-Klasse. Handelt es sich um keinen gültigen Implementierungs-Code, wird kein Observer auf das notify() reagieren.
    - `observable` – Eine Instanz des Observables, das die notify()-Methode aufgerufen hat. Durch diese Referenz weiß der Observer, von wo er eine Benachrichtigung bekommen hat.
    - `goEntity` – Das Go an dem Änderungen vorgenommen wurden

- **persist**

  ```
  public void persist(edu.kit.pse17.go_app.PersistenceLayer.
      GoEntity entity)
  ```

  - **Parameters**
    - `entity` – Das Entity-Objekt, das in der Datenbank gespeichert werden soll. Es wird garantiert, dass das Objekt, welches der Methode

- **register**

  ```
  public void register(edu.kit.pse17.go_app.ServiceLayer.Observer
      observer)
  ```

  - **Parameters**
    - `observer` – der Observer, der registriert werden soll. Dabei spielt es keine Rolle, um welche Implementierung eines

- **unregister**

```
public void unregister(edu.kit.pse17.go_app.ServiceLayer.Observer
    observer)
```

- **Parameters**
  - `observer` – Der Observer der aus der Liste entfernt werden soll. es muss vor dem Aufruf dieser Methode sichergestellt werden, dass

- **update**

```
public void update(edu.kit.pse17.go_app.PersistenceLayer.GoEntity
    goEntity) throws EntityNotFoundException
```

- **Parameters**
  - `goEntity` –
- **Throws**
  - `EntityNotFoundException` –

18.6 Klasse GroupDaoImp

Diese Klasse implementiert die Interfaces GroupDao, AbstractDao und Observable. Sie übernimmt die konkreten Datenbankzugriffe auf die Tabelle "groups". Dazu werden alle Methoden aus den DAO Interfaces entsprechend implementiert. Aufgerufen werden die Methoden dieser Klasse von den RestController-Klassen, wenn ein Client dem Server eine Anfrage zur Manipulation seiner Daten geschickt hat. Die Klasse gehört außerdem zu einer Implementierung des Beobachter-Entwurfsmusters und übernimmt dabei die Rolle des konkreten Subjekts. Die Klasse hat eine Liste von Beobachtern, die benachrichtigt werden, wenn sich in der Datenbank eine Änderung ergibt. Es ist die Verantwortung der Beobachter zu entscheiden, ob die Änderung eine Folgeaktion auslöst oder nicht.

18.6.1 Deklaration

```
public class GroupDaoImp
 extends java.lang.Object implements AbstractDao, GoDao, edu.kit.
    pse17.go_app.ServiceLayer.Observable
```

18.6.2 Konstruktoren Auflistung

GroupDaoImp()

18.6.3 Methoden Auflistung

changeStatus(String, long, Status)
delete(Long)
get(Long)
notify(String, Observable, GroupEntity)
persist(GroupEntity)
register(Observer)
unregister(Observer)
update(GroupEntity)

18.6.4 Attribute

private SessionFactory sessionFactory Eine Sessionfactory, die Sessions bereitstellt. Die Sessions werden benötigt, damit die Klasse direkt mit der Datenbank kommunizieren kann und dort die Änderungen vornehmen. Das Attribut ist mit "@Autowiredännotiert, damit es automatisch mit einem gültigen Objekt instanziiert wird.

Auf dieses Feld darf nur innerhalb dieser Klasse zugegriffen werden. Nach der Instanzizerung ist diese Objekt unveränderbar und bleibt bestehen, bis die Instanz dieser Klasse wieder zerstört wird.

private List<Observer> observer Eine Liste mit Observern, die benachrichtigt werden, sobald eine Änderung an der Datenbank vorgenommen wird, die auch die Daten anderer Benutzer betrifft.

18.6.5 Konstruktoren

- **GroupDaoImp**

```
public GroupDaoImp()
```

18.6.6 Methoden

- **changeStatus**

```
public void changeStatus(java.lang.String userId,long goId,edu.
    kit.pse17.go_app.PersistenceLayer.Status status)
```

  - **Parameters**
    - `userId` – Die ID des Benutzers, dessen Teilnahmestatus geändert werden soll. Dabei handelt es sich um eine gültige Id, ansonsten kann die Methode nicht erfolgreich ausgeführt werden.
    - `goId` – Die des GOs, für den der Teilnahmestatus geändert werden soll. Dabei handelt es sich um eine gültige Id, ansonsten kann die Methode nicht erfolgreich ausgeführt werden.
    - `status` – Der neue Status des Benutzers.

- **delete**

```
public void delete(java.lang.Long key) throws
    EntityNotFoundException
```

  - **Parameters**
    - `key` – Der Primärschlüssel der Entity, die aus der Datenbanktabelle gelöscht werden soll. Der Datentyp wird durch das Generic PK bei der Implementierung der Klasse spezifiziert.
  - **Throws**
    - `EntityNotFoundException` –

- **get**

  public edu.kit.pse17.go_app.PersistenceLayer.GroupEntity get(java.lang.Long key)

  - **Parameters**
    - `key` – Der Primärschlüssel der Entity, die aus der Datenbank geholt werden soll. Der Datentyp wird von dem Generic PK bestimmt, mit dem das Interface implementiert wird.
  - **Returns** –

- **notify**

  public void notify(java.lang.String impCode,edu.kit.pse17.go_app.ServiceLayer.Observable observable,edu.kit.pse17.go_app.PersistenceLayer.GroupEntity groupEntity)

  - **Parameters**
    - `impCode` – Ein Code, der angibt, welche Observer-Implementierung benachrichtigt werden soll. dabei handelt es sich immer um ein öffentliches statisches Attribut in der Observer-Klasse. Handelt es sich um keinen gültigen Implementierungs-Code, wird kein Observer auf das notify() reagieren.
    - `observable` – Eine Instanz des Observables, das die notify()-Methode aufgerufen hat. Durch diese Referenz weiß der Observer, von wo er eine Benachrichtigung bekommen hat.
    - `groupEntity` –

- **persist**

  public void persist(edu.kit.pse17.go_app.PersistenceLayer.GroupEntity entity)

  - **Parameters**
    - `entity` – Das Entity-Objekt, das in der Datenbank gespeichert werden soll. Es wird garantiert, dass das Objekt, welches der Methode

- **register**

  public void register(edu.kit.pse17.go_app.ServiceLayer.Observer observer)

  - **Parameters**
    - `observer` – der Observer, der registriert werden soll. Dabei spielt es keine Rolle, um welche Implementierung eines

- **unregister**

```
public void unregister(edu.kit.pse17.go_app.ServiceLayer.Observer
    observer)
```

- **Parameters**
  - `observer` – Der Observer der aus der Liste entfernt werden soll. es muss vor dem Aufruf dieser Methode sichergestellt werden, dass

- **update**

```
public void update(edu.kit.pse17.go_app.PersistenceLayer.
    GroupEntity groupEntity) throws EntityNotFoundException
```

- **Parameters**
  - `groupEntity` –
- **Throws**
  - `EntityNotFoundException` –

18.7 Klasse UserDaoImp

Diese Klasse implementiert die Interfaces UserDao, AbstractDao und Observable. Sie übernimmt die konkreten Datenbankzugriffe auf die Tabelle üsers". Dazu werden alle Methoden aus den DAO Interfaces entsprechend implementiert. Aufgerufen werden die Methoden dieser Klasse von den RestController-Klassen, wenn ein Client dem Server eine Anfrage zur Manipulation seiner Daten geschickt hat. Die Klasse gehört außerdem zu einer Implementierung des Beobachter-Entwurfsmusters und übernimmt dabei die Rolle des konkreten Subjekts. Die Klasse hat eine Liste von Beobachtern, die benachrichtigt werden, wenn sich in der Datenbank eine Änderung ergibt. Es ist die Verantwortung der Beobachter zu entscheiden, ob die Änderung eine Folgeaktion auslöst oder nicht.

18.7.1 Deklaration

```
public class UserDaoImp
 extends java.lang.Object implements UserDao, AbstractDao, edu.kit.
    pse17.go_app.ServiceLayer.Observable
```

18.7.2 Konstruktoren Auflistung

UserDaoImp()

18.7.3 Methoden Auflistung

addUser(UserEntity)
delete(String)
deleteUser(String)
get(String)
getGroups(String)
getRequests(String)
getUserByEmail(String)
notify(String, Observable, UserEntity)
persist(UserEntity)
register(Observer)
unregister(Observer)
update(UserEntity)

18.7.4 Attribute

private SessionFactory sessionFactory Eine Sessionfactory, die Sessions bereitstellt. Die Sessions werden benötigt, damit die Klasse direkt mit der Datenbank kommunizieren kann und dort die Änderungen vornehmen. Das Attribut ist mit "@Autowiredännotiert, damit es automatisch mit einem gültigen Objekt instanziiert wird.

Auf dieses Feld darf nur innerhalb dieser Klasse zugegriffen werden. Nach der Instanzizerung ist diese Objekt unveränderbar und bleibt bestehen, bis die Instanz dieser Klasse wieder zerstört wird.

private List<Observer> observer Eine Liste mit Observern, die benachrichtigt werden, sobald eine Änderung an der Datenbank vorgenommen wird, die auch die Daten anderer Benutzer betrifft.

18.7.5 Konstruktoren

- **UserDaoImp**

```
public UserDaoImp()
```

18.7.6 Methoden

- **addUser**

```
public void addUser(edu.kit.pse17.go_app.PersistenceLayer.
    UserEntity user)
```

  - **Parameters**
    - `user` – Die Entity, die in die Datenbank eingefügt werden soll. Dieses Objekt muss eine in der Datenbank noch nicht

- **delete**

```
public void delete(java.lang.String key) throws
    EntityNotFoundException
```

  - **Parameters**
    - `key` – Der Primärschlüssel der Entity, die aus der Datenbanktabelle gelöscht werden soll. Der Datentyp wird durch das Generic PK bei der Implementierung der Klasse spezifiziert.
  - **Throws**
    - `EntityNotFoundException` –

- **deleteUser**

```
public void deleteUser(java.lang.String userId)
```

  - **Parameters**

* `userId` – Die userId des Benutzers, dessen Account gelöscht werden soll. Es wird garantiert, dass es sich beim Aufruf

- **get**

```
public edu.kit.pse17.go_app.PersistenceLayer.UserEntity get(java.
    lang.String key)
```

  - **Parameters**
    * `key` – Der Primärschlüssel der Entity, die aus der Datenbank geholt werden soll. Der Datentyp wird von dem Generic PK bestimmt, mit dem das Interface implementiert wird.
  - **Returns** –

- **getGroups**

```
public java.util.List getGroups(java.lang.String userId)
```

  - **Parameters**
    * `userId` – Die ID des Benutzers, dessen Gruppen zurückgegeben werden sollen. Es wird garantiert, dass es sich beim Aufruf der Methode um eine gültige userid handelt.
  - **Returns** –

- **getRequests**

```
public java.util.List getRequests(java.lang.String userId)
```

  - **Parameters**
    * `userId` – Die ID des Benutzers, dessen Gruppenanfragen zurückgegeben werden sollen. Es wird garantiert, dass es sich beim Aufruf der Methode um eine gültige userId handelt.
  - **Returns** –

- **getUserByEmail**

```
public edu.kit.pse17.go_app.PersistenceLayer.UserEntity
    getUserByEmail(java.lang.String mail)
```

  - **Parameters**
    * `mail` – Die E-Mailadresse, anhand derer der Benutzer gesucht werden soll. Der String muss keinem besonderen Muster entsprechen, damit diese Methode fehlerfrei ausgeführt werden kann.
  - **Returns** –

- **notify**

public void notify(java.lang.String impCode,edu.kit.pse17.go_app.ServiceLayer.Observable observable,edu.kit.pse17.go_app.PersistenceLayer.UserEntity userEntity)

- **Parameters**
  - `impCode` – Ein Code, der angibt, welche Observer-Implementierung benachrichtigt werden soll. dabei handelt es sich immer um ein öffentliches statisches Attribut in der Observer-Klasse. Handelt es sich um keinen gültigen Implementierungs-Code, wird kein Observer auf das notify() reagieren.
  - `observable` – Eine Instanz des Observables, das die notify()-Methode aufgerufen hat. Durch diese Referenz weiß der Observer, von wo er eine Benachrichtigung bekommen hat.
  - `userEntity` –

- **persist**

public void persist(edu.kit.pse17.go_app.PersistenceLayer.UserEntity entity)

- **Parameters**
  - `entity` – Das Entity-Objekt, das in der Datenbank gespeichert werden soll. Es wird garantiert, dass das Objekt, welches der Methode

- **register**

public void register(edu.kit.pse17.go_app.ServiceLayer.Observer observer)

- **Parameters**
  - `observer` – der Observer, der registriert werden soll. Dabei spielt es keine Rolle, um welche Implementierung eines

- **unregister**

public void unregister(edu.kit.pse17.go_app.ServiceLayer.Observer observer)

- **Parameters**
  - `observer` – Der Observer der aus der Liste entfernt werden soll. es muss vor dem Aufruf dieser Methode sichergestellt werden, dass

- **update**

public void update(edu.kit.pse17.go_app.PersistenceLayer.UserEntity userEntity) **throws** EntityNotFoundException

- **Parameters**

* `userEntity` – Die Entity des Users, der geändert werden soll. Dabei muss es sich um eine vorhandene Entity handeln, ansonsten schlägt die Ausführung der Methode fehl.

– **Throws**

* `EntityNotFoundException` –

19 Package ClientCommunication.Downstream

Package Contents *Page*

Classes

EventArg 106
Dieses Enum enthält String-Konstanten, die vom FcmClient in die Nachrichten an Clients eingefügt werden, damit der Client anhand der Nachricht feststellen kann, welches Ereignis eingetreten ist.

FcmClient 108
Client-Klasse, die ein HTTP POST-Request an den FCM-Server schickt, wo die Nachricht wiederum an das User-Endgerät weitergeleitet wird.

19.1 Enum EventArg

Dieses Enum enthält String-Konstanten, die vom FcmClient in die Nachrichten an Clients eingefügt werden, damit der Client anhand der Nachricht feststellen kann, welches Ereignis eingetreten ist.

19.1.1 Deklaration

```
public final class EventArg
 extends java.lang.Enum
```

19.1.2 Attribute

ADMIN_ADDED_EVENT Event wird ausgelöst, wenn ein Adminsitrator zu einer Gruppe hinzugefügt wurde.

GO_ADDED_EVENT Event wird ausgelöst, wenn ein neues GO in einer Gruppe erstellt wurde.

GO_EDITED_COMMAND Event wird ausgelöst, wenn die Daten in einem GO verändert werden.

GO_REMOVED_EVENT Event wird ausgelöst, falls eine Go-Entität gelöscht wird.

GROUP_EDITED_COMMAND Event wird ausgelöst, wenn die Daten in einer gruppe verändert werden.

GROUP_REMOVED_EVENT Event wird ausgelöst, falls eine Gruppenentität gelöscht wird.

GROUP_REQUEST_RECEIVED_EVENT Event wird ausgelöst, wenn ein Benutzer eine Anfrage für eine Gruppe bekommen hat.

MEMBER_ADDED_EVENT Event wird ausgelöst, falls ein neues Mitglied zu einer Gruppe hinzugefügt wurde (also ein Benutzer ein gruppenanfrage bestätigt hat).

MEMBER_REMOVED_EVENT Event wird ausgelöst, falls ein Mitglied aus einer Gruppe gelöscht wird.

STATUS_CHANGED_COMMAND Event wird ausgelöst, wenn ein GO-Teilnehmer seinen Teilnehmerstatus verändert hat.

19.1.3 Methoden Auflistung

valueOf(String)
values()

19.1.4 Felder

- `public static final EventArg` **GROUP_REMOVED_EVENT**
  - Event wird ausgelöst, falls eine Gruppenentität gelöscht wird. Das Event wird nur an Clients gesendet, die Mitglied in der Gruppe waren bzw. eine Anfrage für diese Gruppe erhalten haben.
- `public static final EventArg` **MEMBER_REMOVED_EVENT**
  - Event wird ausgelöst, falls ein Mitglied aus einer Gruppe gelöscht wird. Dies ist auch der Fall, wenn ein Benutzer seinen Benutzeraccount gelöscht hat. Das Event wird an alle Mitglieder der Gruppe gesendet und an Benutzer, die eine Anfrage für die Gruppe erhalten haben.
- `public static final EventArg` **GO_REMOVED_EVENT**
  - Event wird ausgelöst, falls eine Go-Entität gelöscht wird. Die Benachrichtigung muss an alle Benutzer geschickt werden, die Mitglied in der Gruppe des GOs sind bzw. eine Gruppenanfrage für diese Gruppe haben.
- `public static final EventArg` **MEMBER_ADDED_EVENT**
  - Event wird ausgelöst, falls ein neues Mitglied zu einer Gruppe hinzugefügt wurde (also ein Benutzer ein Gruppenanfrage bestätigt hat). Die Benachrichtigung muss an alle Benutzer geschickt werden, die Mitglied in der Gruppe sind bzw. eine Gruppenanfrage für diese Gruppe haben.
- `public static final EventArg` **GROUP_REQUEST_RECEIVED_EVENT**
  - Event wird ausgelöst, wenn ein Benutzer eine Anfrage für eine Gruppe bekommen hat. Die Benachrichtigung wird an diesen Benutzer und an alle Mitglieder der Gruppe gesendet bzw. Benutzer, die eine Gruppenanfrage für diese Gruppe haben.
- `public static final EventArg` **GO_ADDED_EVENT**
  - Event wird ausgelöst, wenn ein neues GO in einer Gruppe erstellt wurde. Die Benachrichtigung muss an alle Benutzer geschickt werden, die Mitglied in der Gruppe sind bzw. eine Gruppenanfrage für diese Gruppe haben.
- `public static final EventArg` **GO_EDITED_COMMAND**
  - Event wird ausgelöst, wenn die Daten in einem GO verändert werden. Die Benachrichtigung muss an alle Benutzer geschickt werden, die Mitglied in der Gruppe sind bzw. eine Gruppenanfrage für diese Gruppe haben.
- `public static final EventArg` **GROUP_EDITED_COMMAND**
  - Event wird ausgelöst, wenn die Daten in einer Gruppe verändert werden. Die Benachrichtigung muss an alle Benutzer geschickt werden, die Mitglied in der Gruppe sind bzw. eine Gruppenanfrage für diese Gruppe haben.

- `public static final EventArg` **ADMIN_ADDED_EVENT**
  - Event wird ausgelöst, wenn ein Administrator zu einer Gruppe hinzugefügt wurde. Die Benachrichtigung muss an alle Benutzer geschickt werden, die Mitglied in der Gruppe sind bzw. eine Gruppenanfrage für diese Gruppe haben.
- `public static final EventArg` **STATUS_CHANGED_COMMAND**
  - Event wird ausgelöst, wenn ein GO-Teilnehmer seinen Teilnehmerstatus verändert hat. Die Benachrichtigung muss an alle Benutzer geschickt werden, die Mitglied in der Gruppe des GOs sind bzw. eine Gruppenanfrage für diese Gruppe haben.

19.1.5 Methoden

- **valueOf**

```
public static EventArg valueOf(java.lang.String name)
```

- **values**

```
public static EventArg[] values()
```

19.1.6 von Enum geerbte Methoden

`java.lang.Enum`

- `protected final Object` **clone**`() throws CloneNotSupportedException`
- `public final int` **compareTo**`(Enum` **arg0**`)`
- `public final boolean` **equals**`(Object` **arg0**`)`
- `protected final void` **finalize**`()`
- `public final Class` **getDeclaringClass**`()`
- `public final int` **hashCode**`()`
- `public final String` **name**`()`
- `public final int` **ordinal**`()`
- `public String` **toString**`()`
- `public static Enum` **valueOf**`(Class` **arg0**`, String` **arg1**`)`

19.2 Klasse FcmClient

Client-Klasse, die ein HTTP POST-Request an den FCM-Server schickt, wo die Nachricht wiederum an das User-Endgerät weitergeleitet wird. Dadurch kann der Server eine Nachricht an einen Client schicken, ohne dass dieser zuvor den Server angesprochen haben muss. Die Methoden der Klasse werden aufgerufen von den Observer-Klassen der Anwendung. Dabei werden die Nachrichten, die an die Clients gesendet werden müssen, sowie die Adressdaten bereits in den aufrufenden Methoden bestimmt. Diese Klasse muss sich nur um das eigentliche Senden der HTTP-Requests kümmern, der Inhalt der Nachricht spielt dabei keine Rolle.

19.2.1 Deklaration

```
public class FcmClient
 extends java.lang.Object
```

19.2.2 Konstruktoren Auflistung

FcmClient() Die Klasse bietet einen Konstruktor an, der keine Argumente entgegen nimmt.

19.2.3 Methoden Auflistung

send(JsonObject, String, List) Die Methode einen POST-Request an den FCM-Server, der diese an das User-Endgerät weiterleitet.

19.2.4 Attribute

private static final String BASE_URL Base URL des FCM-Servers an den die Requests geschickt werden müssen. Dieser Wert darf sich nicht ändern.

private HttpClinet httpClient Ein Http-Client, der für das Senden der HTTP-Requests zuständig ist. Die Konfiguration des HttpClients wird bei der Erstellung des FcmClient-Objekts vorgenommen. Die Konfiguration wird von dem Firebase Cloud Messaging Service vorgegeben.

19.2.5 Konstruktoren

- **FcmClient**

```
public FcmClient()
```

  - **Description**
    Die Klasse bietet einen Konstruktor an, der keine Argumente entgegen nimmt. In dem Konstruktor wird die Konfiguration des HttpClients standardmäßig implementiert, sodass er Anfragen an die von FCM definierte URL schicken kann.

19.2.6 Methoden

- **send**

```
public void send(JsonObject data,java.lang.String command,java.
    util.List receiver)
```

  - **Description**
    Die Methode einen POST-Request an den FCM-Server, der diese an das User-Endgerät weiterleitet. Dafür wird der HttpClient der FcmClient-Instanz benutzt. Diese Methode wird von den Observer-Klassen aufgerufen, um die Änderungen, die dort behandelt wurden mithilfe dieser Klasse an die Clients zu schicken. Es wird vorausgesetzt, dass die Daten und vor allem die InstanceIDs, die dieser Methode übergeben werden, gültig sind.
  - **Parameters**
    * `data` – Dieses Objekt enthält die Daten, die an den Client geschickt werden sollen
    * `command` – Ein String, der anzeigt, um was für eine Nachricht es sich handelt, also zu welchem Serverereignis sie gehört. Dieser String bestimmt, an welche Command-Klasse auf dem Client die Nachricht weitergeleitet wird.
    * `receiver` – Eine Liste mit den InstaceIds der Clients,an die die Nachricht geschickt werden soll. dabei muss es sich um gültige InstanceIds handeln, sonst kann die Methode nicht fehlerfrei ausgeführt werden.

20 Package ClientCommunication.Upstream

Package Contents *Page*

Classes

GoRestController 110
Die Klasse GORestController gehört zum Upstream ClientCommunication Modul und bildet einen Teil der REST API, die der Tomcat Server den Clients zur Kommunikation anbietet.

GroupRestController 115
Die Klasse GroupRestController gehört zum Upstream ClientCommunication Modul und bildet einen Teil der REST API, die der Tomcat Server den Clients zur Kommunikation anbietet.

UserRestController 120
Die Klasse UserRestController gehört zum Upstream ClientCommunication Modul und bildet einen Teil der REST API, die der Tomcat Server den Clients zur Kommunikation anbietet.

20.1 Klasse GoRestController

Die Klasse GORestController gehört zum Upstream ClientCommunication Modul und bildet einen Teil der REST API, die der Tomcat Server den Clients zur Kommunikation anbietet. Die Aufgabe dieser Klasse ist die Abwicklung von REST-Requests, die User-spezifische Anfragen beinhalten. Dazu gehört: - das Empfangen und Senden von HTTP-Requests - das Parsen der empfangenen / zu sendenden Daten von bzw. nach JSON - das Weiterleiten der Anfragen zur Bearbeitung an die richtige Stelle im Programm (das UserDAO) Das REST API wird umgesetzt von dem Java Framework Spring, anhand der Annotationen der Methoden in dieser Klasse. Die Klasse selbst ist annotiert mit "@RestController", um zu signalisieren, dass es sich um eine Klasse handelt, deren Methoden REST Resourcen beschreiben. Die Methoden dieser Klasse sind auf die URL {Base_URL}/gos gemappt. Die Methoden der Klasse werden aufgerufen, von den Methoden des Interfaces TomcatRestApi", das von den Clients des Systems verwendet wird. Bei einem Methodenaufruf in dieser Klasse, wird die Anfrage an die DAOs der MySQL Datenbank der Anwendung weitergeleitet. Von dort werden die richtigen Daten geholt (falls der Client bestimmte Daten in der Antwort erwartet). Danach werden die Daten von dieser Klasse in JSON-Objekte umgewandelt (mithilfe der Gson Library) und dem Client in der Antwort zugesendet. Nähere Erläuterungen zum JSON-Schema und der Konvertierung finden sich im Entwurfsdokument.

20.1.1 Deklaration

```
public class GoRestController
 extends java.lang.Object
```

20.1.2 Konstruktoren Auflistung

GoRestController()

20.1.3 Methoden Auflistung

changeStatus(long, String, Status) Diese Methode wird von einem Client aufgerufen, um seinen Teilnahmerstatus in einem Go zu ändern.

createGo(String, String, Date, Date, double, double, int, long, String) Diese Methode wird von einem Client aufgerufen, wenn eine neue Gruppe erstellt werden soll.

deleteGo(String) Diese Methode wird von einem Client aufgerufen, wenn er ein GO löschen möchte.

editGo(String, String, String, Date, Date, long, long, int) Diese Methode wird von einem Benutzer aufgerufen, wenn er die Daten eines GOs ändern will.

getLocation(String) Die Methode gibt eine Liste mit Cluster-Objekten zurück, die die aktuellen Positionen der Go-Mitglieder beschreiben.

setLocation(String, long, long, String) Diese Methode wird von einem Client aufgerufen, um seinen Standort für das Clustering dem Server mitzuteilen.

20.1.4 Attribute

private GoDao goDao Ein Objekt einer Klasse, die das Interface GoDao implementiert. Dieses Objekt besitzt Methoden, um auf die Datenbank des Systems zuzugreifen und Daten zu manipulieren. Es wird benötigt, um die Anfragen, die durch die REST Calls an den Server gestellt werden, umzusetzen.

20.1.5 Konstruktoren

- **GoRestController**

```
public GoRestController()
```

20.1.6 Methoden

- **changeStatus**

```
public void changeStatus(long goId,java.lang.String userId,edu.
    kit.pse17.go_app.PersistenceLayer.Status status)
```

  - **Description**

    Diese Methode wird von einem Client aufgerufen, um seinen Teilnehmerstatus in einem Go zu ändern. IN der Methode werden die Anfrage-Daten aus dem Request Body ausgewertet und an das goDao weitergegeben, um die entsprechenden Änderungen in der Datenbank vorzunehmen. Es ist garantiert, dass der Benutzer ein Mitglied des GOs ist und das er die geforderte Statusänderung vornehmen darf. Der Aufruf dieser Methode entspricht einem HTTP PUT-Request an den Server an die URL {Base_URL}/gos/status.

  - **Parameters**

    * `goId` – Die ID des GOs, in dem der Benutzer seinen Teilnahmestatus ändern will. es muss sich dabei um eine gültige Go-ID sein, ansonsten schlägt die Anfrage fehl. Die ID muss sich zu einem long casten lassen.
    * `userId` – Die ID des Users, der seinen Teilnahmestatus ändern will. Diese ID muss eine gültige, auf dem System registrierte User ID sein.
    * `status` – Der neue Status des Clients. Dieser hat entweder den Wert Äbgelehnt", "Bestätigtöder Losgegangen".

- **createGo**

```
public long createGo(java.lang.String name,java.lang.String
    description,java.util.Date start,java.util.Date end,double lat
    ,double lon,int threshold,long groupId,java.lang.String userId
    )
```

- **Description**

  Diese Methode wird von einem Client aufgerufen, wenn eine neue Gruppe erstellt werden soll. Die Methode liest die Argumente aus dem Request Body der HTTP Anfrage aus und übergibt diese an das goDao zur Erzeugung des GOs in der Datenbank. Zusätzlich zur Erzeugung des GOs wird der Ersteller als Verantwortlicher des GOs gespeichert und für jedes Gruppenmitglied der Teilnahmestatus 'Abgelehnt' gespeichert. es ist garantiert, dass der Client, der die Methode aufruft eine Mitglied in der Gruppe ist, in der das GO erstellt werden soll. Der Aufruf dieser Methode entspricht einem HTTP POST-Request an den Server an die URL {Base_URL}/gos.

- **Parameters**
  - `name` – Der Name des GOs. Es handelt sich um einen String, der bis zu 50 Zeichen enthält.
  - `description` – Eine Beschreibung für das GO. Diese Argument dar den wert null annehmen. Ist der Wert nicht null, darf der String bis zu 140 Zeichen enthalten.
  - `start` – Ein Datum mit Uhrzeit an dem das GO beginnt. Dieses Datum darf nicht in der Vergangenheit liegen.
  - `end` – Ein Datum mit Uhrzeit an dem das GO zu Ende ist. Dieses Datum darf nicht vor dem Startdatum liegen.
  - `lat` – Der geographische Breitengrad des Zielorts des GOs. Der Wert muss als Breitengrad interpretierbar sein, muss also zwischen +90 und -90 liegen. Der Wert darf außerdem nullßein, falls kein Zielort für das GO ausgewählt wurde.
  - `lon` – Der geographische Längengrad des Zielorts des GOs. Der Wert muss als Breitengrad interpretierbar sein, muss also zwischen +180 und -180 liegen. Der Wert darf außerdem nullßein, falls kein Zielort für das GO ausgewählt wurde.
  - `threshold` – Ein Schwellwert für die Genauigkeit, mit der der Clustering-Algorithmus ausgeführt wird. Der Wert liegt zwischen 1 (sehr ungenau) und 10 (sehr genau). Wird der Wert in dem Request Body nicht spezifiziert wird default-mäßig ein Wert von 5 gespeichert.
  - `groupId` – Die ID der Gruppe, in der das GO angelegt werden soll. Der Wert muss eine gültige Group-ID sein und sich zu einem Long casten lassen.
  - `userId` – Die ID des Benutzers, der das GO erstellt. Der Wert muss eine gültige UserID sein. Dieser Benutzer wird als GO-Verantwortlicher gespeichert.
- **Returns** – Die Methode gibt in der Antwort die im System eindeutige ID des Gos zurück. Diese wird im Header der HTTP-Response im Location-Feld an den Client zurückgesendet, also : {Base_URL}/gos/{goId} und kann dort vom Client ausgelesen werden. Der Wert ist eine positive ganze Zahl, die im Wertebereich des primitiven Datentyps long liegt.

- **deleteGo**

```
public void deleteGo(java.lang.String goId)
```

- **Description**

Diese Methode wird von einem Client aufgerufen, wenn er ein GO löschen möchte. Durch einen Methodenaufruf bei dem goDao wird das GO entsprechend aus der Datenbank entfernt. Es ist garantiert, dass der Client, der die Gruppe aufruft dazu berechtigt ist, d.h. er der GO-Verantwortliche des GOs ist. Der Aufruf dieser Methode entspricht einem HTTP DELETE-Request an den Server an die URL {Base_URL}/gos/{goId}.

- **Parameters**
  - `goId` – Die ID des GOs, das gelöscht werden soll. Der Wert dieses Arguments ist Teil der URL der REST Resource und wird entsprechend von Spring extrahiert und der Methode bereitgestellt. Die ID muss gülti sein und zu einem Long-Datentyp gecastet werden können.

- **editGo**

```
public void editGo(java.lang.String goId,java.lang.String name,
    java.lang.String description,java.util.Date start,java.util.
    Date end,long lat,long lon,int threshold)
```

  - **Description**

    Diese Methode wird von einem Benutzer aufgerufen, wenn er die Daten eines GOs ändern will. Zu den Daten, die mit dieser Methode geändert werden können, gehören: - der GO-Name - die GO-Beschreibung - Der Anfangs- und Endzeitpunkt - Der Zielort - Der Clustering-Schwellwert Es ist garantiert, dass dieser Aufruf nur von einem Go-Verantwortlichen des zu ändernden GOs kommt. Der Aufruf dieser Methode entspricht einem HTTP PUT-Request an den Server an die URL {Base_URL}/gos/{goId}. Abgesehen von der Go ID, können sämtliche Argumente dieser Methode den Wert null annehmen. Dies signalisiert der Methode, das der Wert nicht geändert wurde und die bisherigen Daten beibehalten werden sollen.

  - **Parameters**
    - `goId` – Die ID des GOs, das gelöscht werden soll. Der Wert dieses Arguments ist Teil der URL der REST Resource und wird entsprechend von Spring extrahiert und der Methode bereitgestellt. Die ID muss gültig sein und zu einem Long-Datentyp gecastet werden können.
    - `name` – Der Name des GOs. Es handelt sich um einen String, der bis zu 50 Zeichen enthält.
    - `description` – Eine Beschreibung für das GO. Diese Argument dar den wert null annehmen. Ist der Wert nicht null, darf der String bis zu 140 Zeichen enthalten.
    - `start` – Ein Datum mit Uhrzeit an dem das GO beginnt. Dieses Datum darf nicht in der Vergangenheit liegen.
    - `end` – Ein Datum mit Uhrzeit an dem das GO zu Ende ist. Dieses Datum darf nicht vor dem Startdatum liegen.
    - `lat` – Der geographische Breitengrad des Zielorts des GOs. Der Wert muss als Breitengrad interpretierbar sein, muss also zwischen +90 und -90 liegen. Der Wert darf außerdem nullßein, falls kein Zielort für das GO ausgewählt wurde.
    - `lon` – Der geographische Längengrad des Zielorts des GOs. Der Wert muss als Breitengrad interpretierbar sein, muss also zwischen +180 und -180 liegen. Der Wert darf außerdem nullßein, falls kein Zielort für das GO ausgewählt wurde.
    - `threshold` – Ein Schwellwert für die Genauigkeit, mit der der Clustering-Algorithmus ausgeführt wird. Der Wert liegt zwischen 1 (sehr ungenau) und

10 (sehr genau). Wird der Wert in dem Request Body nicht spezifiziert wird default-mäßig ein Wert von 5 gespeichert.

- **getLocation**

```
public java.util.List getLocation(java.lang.String goId)
```

  - **Description**
    Die Methode gibt eine Liste mit Cluster-Objekten zurück, die die aktuellen Positionen der Go-Mitglieder beschreiben. Diese Methode, wird von Clients periodisch aufgerufen, um während eines GOs die Standorte der anderen Mitglieder zu erfahren. Um den eigenen Standort mit den anderen Mitgliedern zu teilen, wird nicht diese Methode verwendet, sondern die Methode setLocation(). Im Gegensatz zu den meisten anderen Methoden der restController-Klassen, wird diese Anfrage nicht an eine DAO Objekt weitergeleitet. Da die Standort-Daten nicht langfristig gespeichert werden müssen, wird die Anfrage an ein locationService Objekt gegeben und dort behandelt. Es ist garantiert, dass der Benutzer, der diese Methode aufruft, dazu berechtigt ist, die Standorte der anderen Mitglieder zu erfahren. Der Aufruf dieser Methode entspricht einem HTTP GET-Request an den Server an die URL {Base_URL}/gos/location/{goId}.
  - **Parameters**
    - `goId` – Die ID des GOs, dessen Location-Daten angefragt werden. Dabei muss es sich um eine gültige GO ID handeln, die sich zu einem Long casten lässt. Der Wert dieses Arguments ist Teil der URL der REST Resource und wird entsprechend von Spring extrahiert und der Methode bereitgestellt.
  - **Returns** – Eine Liste mit Clustern. Ein Cluster besteht dabei aus drei Feldern: Der Längen- und Breitengrad der Position des Clusters und der Größe, also der Anzahl an Personen, die sich in diesem Cluster befinden. Der Rückgabewert dieser Methode kann auch null sein, z.B. dann wenn für das Clustering zu wenig Personen an dem GO teilnehmen. Maximal besteht die Liste aus 50 Clustern, da die Anzahl der Gruppenmitglieder auf 50 beschränkt ist.

- **setLocation**

```
public void setLocation(java.lang.String userId,long lat,long lon
    ,java.lang.String goId)
```

  - **Description**
    Diese Methode wird von einem Client aufgerufen, um seinen Standort für das Clustering dem Server mitzuteilen. Der übermittelte Standort wird aus dem RequestBody ausgelesen und zur Weiterverarbeitung an den locationService weitergeleitet. Es wird garantiert, dass der Client, der diese Methode aufruft, ein aktiver Teilnehmer des entsprechenden GOs ist. Der Aufruf dieser Methode entspricht einem HTTP PUT-Request an den Server an die URL {Base_URL}/gos/location/{goId}.
  - **Parameters**
    - `userId` – Die ID des Benutzers, der seinen Standort teilen will. Dabei muss es sich um eine gültige, im System registrierte UserID handeln.
    - `lat` – Der geographische Breitengrad des Standorts des Benutzers. Der Wert muss als Breitengrad interpretierbar sein, muss also zwischen +90 und -90 liegen.

* `lon` – Der geographische Längengrad des Standorts des Benutzers. Der Wert muss als Längengrad interpretierbar sein, muss also zwischen +180 und -180 liegen.
* `goId` – Die ID des GOs, zu dessen Location-Daten der Standort des Benutzers gehört. Dabei muss es sich um eine gültige GO ID handeln, die sich zu einem Long casten lässt. Der Wert dieses Arguments ist Teil der URL der REST Resource und wird entsprechend von Spring extrahiert und der Methode bereitgestellt.

20.2 Klasse GroupRestController

Die Klasse GroupRestController gehört zum Upstream ClientCommunication Modul und bildet einen Teil der REST API, die der Tomcat Server den Clients zur Kommunikation anbietet. Die Aufgabe dieser Klasse ist die Abwicklung von REST-Requests, die Gruppen-spezifische Anfragen beinhalten. Dazu gehört: - das Empfangen und Senden von HTTP-Requests - das Parsen der empfangenen / zu sendenden Daten von bzw. nach JSON - das Weiterleiten der Anfragen zur Bearbeitung an die richtige Stelle im Programm (das GroupDAO) Das REST API wird umgesetzt von dem Java Framework Spring, anhand der Annotationen der Methoden in dieser Klasse. Die Klasse selbst ist annotiert mit "@RestController", um zu signalisieren, dass es sich um eine Klasse handelt, deren Methoden Rest Resourcen beschreiben. Die Methoden dieser Klasse sind auf die URL {Base_URL}/groups gemappt. Die Methoden der Klasse werden aufgerufen, von den Methoden des Interfaces TomcatRestApi", das von den Clients des Systems verwendet wird. Bei einem Methodenaufruf in dieser Klasse, wird die Anfrage an die DAOs der MySQL Datenbank der Anwendung weitergeleitet. Von dort werden die richtigen Daten geholt (falls der Client bestimmte Daten in der Antwort erwartet). Danach werden die Daten von dieser Klasse in JSON-Objekte umgewandelt (mithilfe der Gson Library) und dem Client in der Antwort zugesendet. Nähere Erläuterungen zum JSON-Schema und der Konvertierung finden sich im Entwurfsdokument.

20.2.1 Deklaration

```
public class GroupRestController
 extends java.lang.Object
```

20.2.2 Konstruktoren Auflistung

GroupRestController()

20.2.3 Methoden Auflistung

acceptRequest(Long, String) Diese Methode wird dann aufgerufen, wenn ein Benutzer eine bestehende Gruppenmitgliedschaftsanfrage bestätigt und somit zu einem vollwertigen Mitglied der Gruppe wird.

addAdmin(String, String) Diese Methode wird aufgerufen, wenn ein Administrator einer Gruppe ein anderes Gruppenmitglied zu administrator ernennen will.

createGroup(String, String, String) Diese Methode wird von einem Client aufgerufen, wenn eine neue Gruppe erstellt werden soll.

deleteGroup(Long) Diese Methode wird von einem Client aufgerufen, wenn er eine Gruppe löschen möchte.

denyRequest(String, String) Diese Methode wird aufgerufen, wenn ein Benutzer eine Gruppenmitgliedschaftsanfrage ablehnt.

editGroup(Long, String, String) Diese Methode wird von einem Benutzer aufgerufen, wenn er die Daten der Gruppe ändern will.

inviteMember(Long, String) Diese Methode wird von einem Client aufgerufen, der einen Benutzer zu einer Gruppe einladen will.

removeMember(String, String) Diese Methode kann von einem Client aufgerufen werden, wenn ein Gruppenmitglied aus einer Gruppe entfernt werden soll.

20.2.4 Attribute

private GroupDao groupDao Ein Objekt einer Klasse, die das Interface GroupDao implementiert. Dieses Objekt besitzt Methoden, um auf die Datenbank des Systems zuzugreifen und Daten zu manipulieren. Es wird benötigt, um die Anfragen, die durch die REST Calls an den Server gestellt werden, umzusetzen.

20.2.5 Konstruktoren

- **GroupRestController**

```
public GroupRestController()
```

20.2.6 Methoden

- **acceptRequest**

```
public void acceptRequest(java.lang.Long groupId,java.lang.String
    userId)
```

  - **Description**
    Diese Methode wird dann aufgerufen, wenn ein Benutzer eine bestehende Gruppenmitgliedschaftsanfrage bestätigt und somit zu einem vollwertigen Mitglied der Gruppe wird. Bei einem Aufruf, müssen zwei Methoden des GroupDaos aufgerufen werden: Zunächst muss die Mitgliedschaftsanfrage, die soeben beantwortet wurde, gelöscht werden, danach muss der Benutzer als Mitglied in die Gruppe eingefügt werden. Der Aufruf dieser Methode entspricht einem HTTP PUT-Request an den Server an die URL {Base_URL}/groups/members/{groupId}/{userId}.
  - **Parameters**
    * `groupId` – Die ID der Gruppe, zu der der Benutzer hinzugefügt werden soll. Der Wert dieses Arguments ist Teil der URL der REST Resource und wird entsprechend von Spring extrahiert und der Methode bereitgestellt. Die ID muss zu einem Long-Datentyp gecastet werden können.
    * `userId` – Die ID des Benutzers, der der Gruppe hinzugefügt werden soll. Der Wert dieses Arguments ist Teil der URL der REST Resource und wird entsprechend von Spring extrahiert und der Methode bereitgestellt.

- **addAdmin**

```
public void addAdmin(java.lang.String groupId,java.lang.String
    userId)
```

  - **Description**
    Diese Methode wird aufgerufen, wenn ein Administrator einer Gruppe ein anderes Gruppenmitglied zu Administrator ernennen will. In der Methode wird eine Methode des groupDaos aufgerufen, die einen Datenbankzugriff ausführt und

den entsprechenden Benutzer zu den Administratoren der Gruppe hinzufügt. Es ist garantiert, dass der aufrufende Client ein Administrator der Gruppe ist und der neue Administrator bereits ein Gruppenmitglied ist. Der Aufruf dieser Methode entspricht einem HTTP POST-Request an den Server an die URL {Base_URL}/groups/admins/{groupId}/{userId}.

- **Parameters**
  - `groupId` – Die ID der Gruppe, in der der neue Administrator hinzugefügt werden soll. Der Wert dieses Arguments ist Teil der URL der REST Resource und wird entsprechend von Spring extrahiert und der Methode bereitgestellt. Die ID muss zu einem Long-Datentyp gecastet werden können.
  - `userId` – Die ID des Benutzer, der zum Administrator ernannt wird. Der Wert dieses Arguments ist Teil der URL der REST Resource und wird entsprechend von Spring extrahiert und der Methode bereitgestellt.

- **createGroup**

```
public long createGroup(java.lang.String name,java.lang.String
    description,java.lang.String userId)
```

  - **Description**
    Diese Methode wird von einem Client aufgerufen, wenn eine neue Gruppe erstellt werden soll. Die Methode liest die Argumente aus dem Request Body der HTTP Anfrage aus und übergibt diese an das groupDao zur Erzeugung der Gruppe in der Datenbank. Zusätzlich zur Erzeugung der Gruppe wird der Ersteller als Gruppenmitglied und Administrator zur Gruppe hinzugefügt. Der Aufruf dieser Methode entspricht einem HTTP POST-Request an den Server an die URL {Base_URL}/groups.
  - **Parameters**
    - `name` – Der Name, den die Gruppe haben soll. Der String darf bis zu 50 Zeichen lang sein.
    - `description` – Eine Gruppenbeschreibung. Dieser Wert ist möglicherweise nicht im Body der HTTP Nachricht enthalten. Das bedeutet der Benutzer hat keine Beschreibung eingegeben. Die Variable wird daraufhin auf null gesetzt. ist der Wert nicht null, darf der String maximal 140 Zeichen enthalten.
    - `userId` – Die ID des Benutzers, der die Gruppe erstellt hat.
  - **Returns** – Die global eindeutige ID, die der Gruppe zugewiesen wurde. Diese wird im Header der HTTP-Response im Location-Feld an den Client zurückgesendet, also : {Base_URL}/gos/{goId} und kann dort vom Client ausgelesen werden. Der Wert ist eine positive ganze Zahl, die im Wertebereich des primitiven Datentyps long liegt.

- **deleteGroup**

```
public void deleteGroup(java.lang.Long groupId)
```

  - **Description**
    Diese Methode wird von einem Client aufgerufen, wenn er eine Gruppe löschen möchte. Durch einen Methodenaufruf bei dem groupDao wird die Gruppe entsprechend aus der Datenbank entfernt. Durch Konsistenzkriterien in der Datenbank werden zusätzlich alle GOs, die es in der Gruppe gab ebenfalls entfernt. Es ist garantiert, dass

der Client, der die Gruppe aufruft dazu berechtigt ist, d.h. er ein Administrator der Gruppe ist. Der Aufruf dieser Methode entspricht einem HTTP DELETE-Request an den Server an die URL {Base_URL}/groups/{groupId}.

- **Parameters**
  - `groupId` – Die ID der Gruppe, die gelöscht werden soll. Der Wert dieses Arguments ist Teil der URL der REST Resource und wird entsprechend von Spring extrahiert und der Methode bereitgestellt. Die ID muss zu einem Long-Datentyp gecastet werden können.

- **denyRequest**

```
public void denyRequest(java.lang.String groupId,java.lang.String
    userId)
```

  - **Description**
    Diese Methode wird aufgerufen, wenn ein Benutzer eine Gruppenmitgliedschaftsanfrage ablehnt. Beim Aufruf wird das groupDAO dazu veranlasst, die Anfrage aus der Datenbank zu löschen. Der Aufruf dieser Methode entspricht einem HTTP POST-Request an den Server an die URL {Base_URL}/groups/requests/{groupId}/{userId}.
  - **Parameters**
    - `groupId` – Die ID der Gruppe, zu die der Benutzer eingeladen war. Der Wert dieses Arguments ist Teil der URL der REST Resource und wird entsprechend von Spring extrahiert und der Methode bereitgestellt. Die ID muss zu einem Long-Datentyp gecastet werden können.
    - `userId` – Die ID des Benutzers, der die Anfrage abgelehnt hat. Der Wert dieses Arguments ist Teil der URL der REST Resource und wird entsprechend von Spring extrahiert und der Methode bereitgestellt.

- **editGroup**

```
public void editGroup(java.lang.Long groupId,java.lang.String
    name,java.lang.String description)
```

  - **Description**
    Diese Methode wird von einem Benutzer aufgerufen, wenn er die Daten der Gruppe ändern will. Zu den Daten, die mit dieser Methode geändert werden können, gehören: - der Gruppenname - die Gruppenbeschreibung Es ist garantiert, dass dieser Aufruf nur von einem Administrator der zu ändernden Gruppe kommt. Der Aufruf dieser Methode entspricht einem HTTP PUT-Request an den Server an die URL {Base_URL}/groups/{groupId}.
  - **Parameters**
    - `groupId` – Die ID der Gruppe, die geändert werden soll. Der Wert dieses Arguments ist Teil der URL der REST Resource und wird entsprechend von Spring extrahiert und der Methode bereitgestellt. Die ID muss zu einem Long-Datentyp gecastet werden können.

* `description` – Die neue Beschreibung, die die Gruppe erhalten soll. Dieser Wert kann null sein, falls die Beschreibung nicht geändert wird. Der Wert des Attributs ist im request Body der Anfrage gespeichert und wird von Spring ausgelesen und der Methode zur Verfügung gestellt.
* `name` – Der neue Name, den die Gruppe erhalten soll. Dieser Wert kann null sein, falls der Name nicht geändert wird. Der Wert des Attributs ist im request Body der Anfrage gespeichert und wird von Spring ausgelesen und der Methode zur Verfügung gestellt.

- **inviteMember**

```
public void inviteMember(java.lang.Long groupId,java.lang.String
    userId)
```

  - **Description**

    Diese Methode wird von einem Client aufgerufen, der einen Benutzer zu einer Gruppe einladen will. Bei Aufruf dieser Methode wird mittels des groupDAOs die Information über den Group Request in der Datenbank gespeichert. Es ist garantiert, dass der Client, der diese Methode aufruft ein Administrator ist und der eingeladene Benutzer nicht bereits Mitglied der Gruppe ist. Diese Vorbedingungen müssen in der Methode nicht überprüft werden. Der Aufruf dieser Methode entspricht einem HTTP POST-Request an den Server an die URL {Base_URL}/groups/requests/{groupId}/{userId}.
  - **Parameters**
    * `groupId` – Die ID der Gruppe, zu der der Benutzer eingeladen werden soll. Der Wert dieses Arguments ist Teil der URL der REST Resource und wird entsprechend von Spring extrahiert und der Methode bereitgestellt. Die ID muss zu einem Long-Datentyp gecastet werden können.
    * `userId` – Die ID des Benutzers, der zu der Gruppe eingeladen werden soll. Der Wert dieses Arguments ist Teil der URL der REST Resource und wird entsprechend von Spring extrahiert und der Methode bereitgestellt.

- **removeMember**

```
public void removeMember(java.lang.String userId,java.lang.String
    groupId)
```

  - **Description**

    Diese Methode kann von einem Client aufgerufen werden, wenn ein Gruppenmitglied aus einer Gruppe entfernt werden soll. Dies kann der Fall sein, wenn ein Benutzer freiwillig aus einer Gruppe austritt oder wenn er von einem Administrator aus der Gruppe entfernt wird. Bei einem Aufruf leitet die Methode die Anfrage an die entsprechende Methode des groupDAOs weiter. Dies entfernt den Benutzer aus der Gruppe. Durch Foreign Key Constraints in der Datenbank wird der Benutzer auch aus allen GOs der Gruppe entfernt. Darum muss sich diese Methode demnach nicht kümmern. Der Aufruf dieser Methode entspricht einem HTTP POST-Request an den Server an die URL {Base_URL}/groups/members/{groupId}/{userId}.
  - **Parameters**

* `userId` – Die ID des Benutzers, der aus der Gruppe entfernt werden soll. Der Wert dieses Arguments ist Teil der URL der REST Resource und wird entsprechend von Spring extrahiert und der Methode bereitgestellt.
* `groupId` – Die ID der Gruppe, aus der der Benutzer entfernt werden soll. Der Wert dieses Arguments ist Teil der URL der REST Resource und wird entsprechend von Spring extrahiert und der Methode bereitgestellt. Die ID muss zu einem Long-Datentyp gecastet werden können.

20.3 Klasse UserRestController

Die Klasse UserRestController gehört zum Upstream ClientCommunication Modul und bildet einen Teil der REST API, die der Tomcat Server den Clients zur Kommunikation anbietet. Die Aufgabe dieser Klasse ist die Abwicklung von REST-Requests, die User-spezifische Anfragen beinhalten. Dazu gehört: - das Empfangen und Senden von HTTP-Requests - das Parsen der empfangenen / zu sendenden Daten von bzw. nach JSON - das Weiterleiten der Anfragen zur Bearbeitung an die richtige Stelle im Programm (das UserDAO) Das REST API wird umgesetzt von dem Java Framework Spring, anhand der Annotationen der Methoden in dieser Klasse. Die Klasse selbst ist annotiert mit "@RestController", um zu signalisieren, dass es sich um eine Klasse handelt, deren Methoden Rest Resourcen beschreiben. Die Methoden dieser Klasse sind auf die URL {Base_URL}/user gemappt. Die Methoden der Klasse werden aufgerufen, von den Methoden des Interfaces TomcatRestApi", das von den Clients des Systems verwendet wird. Bei einem Methodenaufruf in dieser Klasse, wird die Anfrage an die DAOs der MySQL Datenbank der Anwendung weitergeleitet. Von dort werden die richtigen Daten geholt (falls der Client bestimmte Daten in der Antwort erwartet). Danach werden die Daten von dieser Klasse in JSON-Objekte umgewandelt (mithilfe der Gson Library) und dem Client in der Antwort zugesendet. Nähere Erläuterungen zum JSON-Schema und der Konvertierung finden sich im Entwurfsdokument.

20.3.1 Deklaration

```
public class UserRestController
 extends java.lang.Object
```

20.3.2 Konstruktoren Auflistung

UserRestController()

20.3.3 Methoden Auflistung

createUser(String, String) Diese Methode wird aufgerufen, wenn ein Benutzer sich zum ersten Mal in der App anmeldet.

deleteUser(String) Diese Methode wird aufgerufen, wenn ein Benutzer seinen Benutzeraccount löschen möchte.

getData(String) Diese Methode liefert dem Anfragenden eine Liste aller Gruppen, in der der Benutzer mit der User ID {userId} Mitglied ist, bzw. zu denen er eine Anfrage bekommen hat.

registerDevice(String) Diese Methode wird aufgerufen, um das Gerät, dass ein Benutzer aktuell benutzt auf dem Server mit seiner InstanceId zu registrieren.

20.3.4 Attribute

private UserDao userDao Ein Objekt einer Klasse, die das Interface UserDao implementiert. Dieses Objekt besitzt Methoden, um auf die Datenbank des Systems

zuzugreifen und Daten zu manipulieren. Es wird benötigt, um die Anfragen, die durch die REST Calls an den Server gestellt werden, umzusetzen.

20.3.5 Konstruktoren

- **UserRestController**

```
public UserRestController()
```

20.3.6 Methoden

- **createUser**

```
public void createUser(java.lang.String email,java.lang.String
    userId)
```

  - **Description**
    Diese Methode wird aufgerufen, wenn ein Benutzer sich zum ersten Mal in der App anmeldet. Die Methode veranlasst das userDAO einen neuen Eintrag in der Datenbank anzulegen. Dazu überträgt der Client die benötigten Daten im Request Body der HTTP-Anfrage, verpackt als JSON-Objekt. Der Aufruf dieser Methode entspricht einem HTTP POST-Request an die URL {base_URL}/user/{userId}. Die Methode besitzt keinen Rückgabewert, lediglich einen Statuscode in der HTTP-Antwort, die an den Anfragenden gesendet wird. Der Statuscode gibt an, ob die Transaktion erfolgreich war.
  - **Parameters**
    * `userId` – Die ID des Benutzers, der sich registriert. Diese muss eindeutig sein. Die ID wird generiert von dem Firebase Authentication Service, der auch die Eindeutigkeit derselben sicherstellt. Diese wird von Spring aus der URL extrahiert und als Argument der Methode verwendet.
    * `email` – Die E-Mailadresse des Benutzers, die mit dem Google-Account assoziiert ist, mit dem er sich angemeldet hat.

- **deleteUser**

```
public void deleteUser(java.lang.String userId)
```

  - **Description**
    Diese Methode wird aufgerufen, wenn ein Benutzer seinen Benutzeraccount löschen möchte. In der Methode wird das UserDAO dazu aufgerufen, das Tupel aus der User-Relation zu entfernen. Durch Fremdschlüssel-Constraints in der Datenbank, werden alle dem User gehörenden Gruppen (in denen er Admin war) , GOs (in denen er Go-Verantwortlicher war), Gruppenmitgliedschaften sowie Gruppenanfragen an den User automatisch gelöscht. Der Aufruf dieser Methode entspricht einem HTTP DELETE-Request an die URL {base_URL}/user/{userId}. Die Methode besitzt keinen Rückgabewert, lediglich einen Statuscode in der HTTP-Antwort, die an den Anfragenden gesendet wird. Der Statuscode gibt an, ob die Transaktion erfolgreich war.
  - **Parameters**

* `userId` – die ID des Benutzers, dessen Konto entfernt werden soll. Diese wird von Spring aus der URL extrahiert und als Argument der Methode verwendet.

- **getData**

```
public java.util.List getData(java.lang.String userId)
```

  - **Description**
    Diese Methode liefert dem Anfragenden eine Liste aller Gruppen, in der der Benutzer mit der User ID {userId} Mitglied ist, bzw. zu denen er eine Anfrage bekommen hat. Sie wird genau dann von einem Client aufgerufen, wenn ein Benutzer sich in der App anmeldet. Der Aufruf dieser Methode entspricht einem HTTP GET-Request an den Server an die URL {Base_URL}/user/{userId}, die {userId} Da in den Gruppen die einzelnen GOs dieser Gruppe gespeichert sind, erhält der Anfragende mit dem Aufruf dieser Methode sämtliche Daten, die den Benutzer mit der User ID {userID} betreffen und für das Navigieren und Benutzen der App benötigt werden (angesehen von Änderungen der Daten, die zu einem späteren Zeitpunkt stattfinden).
  - **Parameters**
    * `userId` – Die ID des Benutzers, dessen Daten zurückgegeben werden sollen. Diese wird von Spring aus der URL extrahiert und als Argument der Methode verwendet.
  - **Returns** – eine Liste aller Gruppen von Gruppenobjekten. Der Rückgabewert dieser Methode wird innerhalb der Methode in ein JSON-Objekt geparst und in der empfangenden Methode des Clients zu Java Objekten konvertiert. Die Konvertierung nach JSON und zurück ändert nicht den Inhalt der Daten. Die Liste kann leer sein, für den Fall dass ein Benutzer nicht Mitglied in irgendeiner Gruppe ist. In diesem Fall wird in dem JSON-Objekt ein leerer Data-Block übertragen. Die Länge der Liste ist auf 300 Gruppen beschränkt (dies ist die Gesamtanzahl an Gruppen, die von dem System unterstützt werden)

- **registerDevice**

```
public void registerDevice(java.lang.String instanceId)
```

  - **Description**
    Diese Methode wird aufgerufen, um das Gerät, dass ein Benutzer aktuell benutzt auf dem Server mit seiner InstanceId zu registrieren. Die InstanceId wird vom Server benötigt, um das Gerät des Benutzers identifizieren zu können, um Kommunikationsströme zu initiieren. Da diese InstanceId sich von Gerät zu Gerät unterscheidet bzw. sich durch Konfigurationsänderungen ändern kann, sollte diese Methode zusätzlich zu getData() bei jeder Anmeldung von dem Client aufgerufen werden. Der Aufruf dieser Methode entspricht einem HTTP PUT-Request an die URL {base_URL}/user/device/{instanceId}.
  - **Parameters**
    * `instanceId` – Die InstanceID des Geräts, an dem sich der User angemeldet hat. Diese wird von Spring aus der URL extrahiert und als Argument der Methode verwendet. Generiert wird die InstanceId von dem Service Firebase Cloud Messaging, der auch benutzt wird, um Downstream-Kommunikation zu realisieren.

21 Package edu.kit.pse17.go_app

Package Contents *Page*

Classes

Main 123
Created by tina on 29.06.17.

21.1 Klasse Main

Created by tina on 29.06.17.

21.1.1 Deklaration

```
public class Main
 extends java.lang.Object
```

21.1.2 Konstruktoren Auflistung

Main()

21.1.3 Methoden Auflistung

main(String[])

21.1.4 Konstruktoren

- **Main**

```
public Main()
```

21.1.5 Methoden

- **main**

```
public static void main(java.lang.String[] args)
```

22 Package ServiceLayer

Package Contents *Page*

Interfaces

ClusterStrategy 124
Dieses Interface definiert die Schnittstelle, die eine Klasse, die einen Clustering-Algorithmus implementiert, anbieten.
Observable 125
Dieses Interface ist Teil einer Implementierung eines Beobachter-Entwurfsmusters.
Observer 126
Dieses Interface gehört zu einer Implementierung des Entwurfsmusters Beobachter.

Classes

Cluster 128
Bei dieser Klasse handelt es sich um eine Datenhaltungsklasse, die dem Clustering-Algorithmus das hantieren mit den Standorten erleichtert.
EntitiyRemovedObserver 129
Bei dieser Klasse handelt es sich um eine Implementierung des Observer-Interfaces.
EntityAddedObserver 132
Bei dieser Klasse handelt es sich um eine Implementierung des Observer-Interfaces.
EntityChangedObserver 135
Bei dieser Klasse handelt es sich um eine Implementierung des Observer-Interfaces. Dementsprechend ist diese Klasse Teil des Observer- Entwurfsmusters und übernimmt die Rolle des konkreten Observers.
GoClusterStrategy 138
In dieser Klasse wird der in er Anwendung verwendete Clustering-Algorithmus implementiert.
LocationService 139
Diese Klasse bietet eine Schnittstelle für den GOrestController, an die Anfragen, die den User- bzw.
UserLocation 142
Bei dieser Klasse handelt es sich um eine Datenhaltungsklasse, die dem Clustering-Algorithmus das hantieren mit den Standorten erleichtert.

22.1 Interface ClusterStrategy

Dieses Interface definiert die Schnittstelle, die eine Klasse, die einen Clustering-Algorithmus implementiert, anbieten. muss. Die Anzahl der Teilnehmer eines GOs liegt zwischen 3 und 50. Der implementierende Algorithmus muss mit dieser Anzahl an Benutzern umgehen können. Das Interface ist Teil eines Strategie-Entwurfsmusters und übernimmt die Rolle der allgemeinen Strategie.

22.1.1 Deklaration

```
public interface ClusterStrategy
```

22.1.2 All known subinterfaces

GoClusterStrategy (in 22.8, page 138)

22.1.3 Klassen, die das Interface implementieren

GoClusterStrategy (in 22.8, page 138)

22.1.4 Methoden Auflistung

calculateCluster(List) Diese Methode muss von jeder konkreten Algorithmus-Klasse implementiert werden.

22.1.5 Methoden

- **calculateCluster**

```
java.util.List calculateCluster(java.util.List userLocationList)
```

– **Description**

Diese Methode muss von jeder konkreten Algorithmus-Klasse implementiert werden. Ein Aufruf dieser Methode führt zu einer Ausführung des konkreten Algorithmus. Dabei ist es egal, wie der Algorithmus beim Clustering konkret vorgeht. In dem Entwurfsmuster Strategie übernimmt diese Methode die Rolle der führeAus()MMethode in der abstrakten Strategie.

– **Parameters**

* `userLocationList` – Eine Liste mit den aktuellen Standorten der einzelnen GO-Teilnehmer. Die Länge der Liste beträgt dabei mindestens drei Objekte und maximal 50 Objekte.

– **Returns** – eine Liste von Cluster-Objekten, die den aktuellen Standort der Gruppe beschreiben. (Die Länge der Liste liegt zwischen...)

22.2 Interface Observable

Dieses Interface ist Teil einer Implementierung eines Beobachter-Entwurfsmusters. Es übernimmt die Rolle des abstrakten Subjekts. Es muss von allen Klassen, die beobachtet werden müssen implementiert werden. In dieser Anwendung sind dies die DAO Klassen. Diese werden beobachtet, um Änderungen am Datenbestand zu bemerken und diese Änderungen an betroffene Clients weiterleiten zu können. Das bedeutet bei jeder Änderung an dem Datenbestand muss anschließend die notify()-Methode aufgerufen werden, um den Beobachtern die Änderungen zu übergeben. Das Generic T gibt an, welcher Datentyp von den Änderungen betroffen ist und von den Beobachtern an die Clients weitergeleitet werden muss.

22.2.1 Deklaration

```
public interface Observable
```

22.2.2 All known subinterfaces

GoDaoImp (in 18.5, page 96), UserDaoImp (in 18.7, page 102), GroupDaoImp (in 18.6, page 99)

22.2.3 Klassen, die das Interface implementieren

GoDaoImp (in 18.5, page 96), UserDaoImp (in 18.7, page 102), GroupDaoImp (in 18.6, page 99)

22.2.4 Methoden Auflistung

notify(String, Observable, T) Mit dieser Methode können Observer über eine Änderung benachrichtigt werden. es muss dabei nicht angegeben werden, welche Änderung vorgenommen wurde, dies wissen die Observer selbst.

register(Observer) Mit dieser Methode kann man einen neuen Observer registrieren.

unregister(Observer) ein zuvor registrierter Observer kann wieder entfernt werden, indem diese Methode aufgerufen wird.

22.2.5 Methoden

- **notify**

```
void notify(java.lang.String impCode, Observable observable, java.
    lang.Object t)
```

- **Description**

  Mit dieser Methode können Observer über eine Änderung benachrichtigt werden. es muss dabei nicht angegeben werden, welche Änderung vorgenommen wurde, dies wissen die Observer selbst. Die Methode löst nur dann eine Aktion bei einem der Observer aus, wenn zuvor ein zu der Änderung passender Observer registriert wurde.

- **Parameters**

  * `impCode` – Ein Code, der angibt, welche Observer-Implementierung benachrichtigt werden soll. dabei handelt es sich immer um ein öffentliches statisches Attribut in der Observer-Klasse. Handelt es sich um keinen gültigen Implementierungs-Code, wird kein Observer auf das notify() reagieren.
  * `observable` – Eine Instanz des Observables, das die notify()-Methode aufgerufen hat. Durch diese Referenz weiß der Observer, von wo er eine Benachrichtigung bekommen hat.
  * `t` – Das Objekt das die Änderung enthält bzw. an dem die Änderung durchgeführt wurde.

- **register**

```
void register(Observer observer)
```

  - **Description**

    Mit dieser Methode kann man einen neuen Observer registrieren. Er wird zu einer Liste von Observern hinzugefügt, falls diese Liste noch nicht vorhanden ist, wird sie erstellt. Der Beobachter ist nach dem Hinzufügen zu der Liste funktionsfähig.

  - **Parameters**

    * `observer` – der Observer, der registriert werden soll. Dabei spielt es keine Rolle, um welche Implementierung eines Observers es sich handelt.

- **unregister**

```
void unregister(Observer observer)
```

  - **Description**

    Ein zuvor registrierter Observer kann wieder entfernt werden, indem diese Methode aufgerufen wird. Er wird aus der Liste entfernt.

  - **Parameters**

    * `observer` – Der Observer der aus der Liste entfernt werden soll. es muss vor dem Aufruf dieser Methode sichergestellt werden, dass es sich bei dem Objekt um einen vorher registrierten, noch existenten Observer handelt.

22.3 Interface Observer

Dieses Interface gehört zu einer Implementierung des Entwurfsmusters Beobachter. Es übernimmt dabei die Rolle des abstrakten Beobachters. Jede konkrete Beobachter-Klasse muss dieses Interface implementieren und über eine update-Methode verfügen. Die Aufgabe der Beobachter in dieser Anwendung ist das Beobachten der DAO-Klassen und bei Änderungen im Datenbestand, diese Änderungen an die Clients der betroffenen Benutzer weiterzuleiten. Dazu werden

aus dem übergebenen Objekt die wichtigen Daten extrahiert und in ein JSON-Objekt umgewandelt. Dieses kann an das Downstream-ClientCommunication Modul übergeben werden, wo es an die betroffenen Clients geschickt wird. Die Implementierung des Entwurfsmusters benutzt ein push-Modell, d.h. die Änderungen werden den Beobachtern bei einem notify()-Aufruf gleich mit übergeben. Die Beobachter müssen sich diese Änderungen nicht selbst holen.

22.3.1 Deklaration

```
public interface Observer
```

22.3.2 All known subinterfaces

EntitiyRemovedObserver (in 22.5, page 129), EntityChangedObserver (in 22.7, page 135), EntityAddedObserver (in 22.6, page 132)

22.3.3 Klassen, die das Interface implementieren

EntitiyRemovedObserver (in 22.5, page 129), EntityChangedObserver (in 22.7, page 135), EntityAddedObserver (in 22.6, page 132)

22.3.4 Methoden Auflistung

update(String, Observable, Object) Die update()-Methode, mit der die Beobachter die beobachteten Änderungen an die Clients weitergeben.

22.3.5 Methoden

- **update**

  ```
  void update(java.lang.String arg,Observable observable,java.lang.
      Object o)
  ```

  - **Description**
    Die update()-Methode, mit der die Beobachter die beobachteten Änderungen an die Clients weitergeben. Wie dieses Update genau aussieht, wird von der konkreten Implementierung des Beobachters bestimmt.
  - **Parameters**
    - `arg` – Ein Argument, das die aufgetretene Änderung beschreibt. Ein Beobachter reagiert nur dann auf die Änderung, wenn dieses Argument mit dem statischen Feld OBSERVER_CODE, das jeder Beobachter besitzt übereinstimmt.
    - `observable` – Eine Instanz des Observable-Objekts, dass den Beobachter benachrichtigt hat. Dadurch kann der Beobachter zurückverfolgen von wo in der Anwendung er benachrichtigt wurde.
    - `o` – Ein Objekt, das die Änderungen, um die der Beobachter sich kümmern muss enthält. Da es sich um den Datentyp Öbject"handelt, ist der Beobachter sehr flexibel, welche Änderungen ihm übergeben werden können. Dies erleichtert auch das Überladen der Methode, wodurch ein Beobachter mehrere ähnliche Ereignisse beobachten kann.

22.4 Klasse Cluster

Bei dieser Klasse handelt es sich um eine Datenhaltungsklasse, die dem Clustering-Algoeithmus das hantieren mit den Standorten erleichtert. Ein Objekt dieser Klasse beschreibt dabei ein Cluster, bestehend aus mehreren GO-Teilnehmern, die sich nahe genug beieinander befinden, um von dem benutzten Clustering-Algorithmus als Cluster erkannt worden zu sein.

22.4.1 Deklaration

```
public class Cluster
 extends java.lang.Object
```

22.4.2 Konstruktoren Auflistung

Cluster(int, long, long)

22.4.3 Methoden Auflistung

getLat()
getLon()
getSize()
setLat(long)
setLon(long)
setSize(int)

22.4.4 Attribute

private int size Anzahl an Personen, die sich in dem Cluster befinden. Der Wert liegt zwischen 1 und 60 Teilnehmern.

private int lat Der geographische Breitengrad des Standorts des Clusters. Der Wert muss als Breitengrad interpretierbar sein, muss also zwischen +90 und -90 liegen.

private int long Der geographische Längengrad des Standorts des Clusters. Der Wert muss als Längengrad interpretierbar sein, muss also zwischen +180 und -180 liegen.

22.4.5 Konstruktoren

- **Cluster**

```
public Cluster(int size,long lat,long lon)
```

22.4.6 Methoden

- **getLat**

```
public long getLat()
```

- **getLon**

```
public long getLon()
```

- **getSize**

```
public int getSize()
```

- **setLat**

```
public void setLat(long lat)
```

- **setLon**

```
public void setLon(long lon)
```

- **setSize**

```
public void setSize(int size)
```

22.5 Klasse EntitiyRemovedObserver

Bei dieser Klasse handelt es sich um eine Implementierung des Observer-Interfaces. Dementsprechend ist diese Klasse Teil des Observer- Entwurfsmusters und übernimmt die Rolle des konkreten Observers. Die Aufgabe dieser Klasse ist das Beobachten der DAO-Klassen und auf Entfernen von Entitäten zu reagieren. Dies schließt folgende Ereignisse mit ein: - Entfernen eines GOs - Entfernen eines Gruppenmitglieds / einer Gruppenanfrage - Entfernen einer Gruppe Um auf diese verschiedenen Änderungen reagieren zu können, muss bei der Implementierung die update()-Methode überladen werden oder innerhalb der Methode anhand der übergebenen Änderung entschieden werden, welche weitere Vorgehensweise gewählt werden muss.

22.5.1 Deklaration

```
public class EntitiyRemovedObserver
 extends java.lang.Object implements Observer
```

22.5.2 Field summary

OBSERVER_CODE Der Code anhand dem der Observer erkennt, dass er auf ein notify() reagieren soll.

22.5.3 Konstruktoren Auflistung

EntitiyRemovedObserver()

22.5.4 Methoden Auflistung

update(String, Observable, GoEntity) Überladung der update()-Methode des Observers.
update(String, Observable, GroupEntity) Überladung der update()-Methode des Observers.
update(String, Observable, List) Überladung der update()-Methode des Observers.
update(String, Observable, Object) Implementierung der update()-Methode.

22.5.5 statische Felder

- `public static final java.lang.String` **OBSERVER_CODE**
  - Der Code anhand dessen der Observer erkennt, dass er auf ein notify() reagieren soll. Bei jeglichen Änderungen wird jeder Observer benachrichtigt, wer regieren muss wird anhand dieses Codes entschieden. Er wird als erstes Argument der update()-Methode verwendet.

22.5.6 Attribute

private FcmClient fcmClient Eine Instanz eines FcmClients, der dafür verwendet wird, Nachrichten an die Clients zu schicken. Das Attribut wird bei der Erzeugung eines Observer Objekts automatisch instanziiert (durch Benutzung des einzigen, argumentlosen Konstruktors der FcmClient-Klasse). Danach kann das von außen Attribut nicht mehr verändert werden.

22.5.7 Konstruktoren

- **EntitiyRemovedObserver**

```
public EntitiyRemovedObserver()
```

22.5.8 Methoden

- **update**

```
public void update(java.lang.String arg,Observable observable,edu
    .kit.pse17.go_app.PersistenceLayer.GoEntity go)
```

  - **Description**
    Überladung der update()-Methode des Observers. Diese Methode kümmert sich um das Entfernen eines GOs. Die Daten des GOs werden in dieser Methode zu einem passenden JSON-Objekt umgewandelt und an den FcmClient weitergegeben, um von dort an die entsprechenden Clients geschickt zu werden. Wer diese Clients sind wird ebenfalls in dieser Methode bestimmt und kann anhand der übergebenen Go-Entity ermittelt werden.
  - **Parameters**
    - `arg` – in Argument, das die aufgetretene Änderung beschreibt. Ein Beobachter reagiert nur dann auf die Änderung, wenn dieses Argument mit dem statischen Feld OBSERVER_CODE, das jeder Beobachter besitzt übereinstimmt.
    - `observable` – Eine Instanz des Observable-Objekts, dass den Beobachter benachrichtigt hat. Dadurch kann der Beobachter zurückverfolgen von wo in der Anwendung er benachrichtigt wurde.
    - `go` – Das GO, das aus der Datenbank entfernt wurde.

- **update**

```
public void update(java.lang.String arg,Observable observable,edu
    .kit.pse17.go_app.PersistenceLayer.GroupEntity group)
```

- **Description**

  Überladung der update()-Methode des Observers. Diese Methode kümmert sich um das Entfernen einer Gruppe. Die Daten der Gruppe werden in dieser Methode zu einem passenden JSON-Objekt umgewandelt und an den FcmClient weitergegeben, um von dort an die entsprechenden Clients geschickt zu werden. Wer diese Clients sind wird ebenfalls in dieser Methode bestimmt und kann anhand der übergebenen Go-Entity ermittelt werden.

- **Parameters**

  * `arg` – Ein Argument, das die aufgetretene Änderung beschreibt. Ein Beobachter reagiert nur dann auf die Änderung, wenn dieses Argument mit dem statischen Feld OBSERVER_CODE, das jeder Beobachter besitzt übereinstimmt.
  * `observable` – Eine Instanz des Observable-Objekts, dass den Beobachter benachrichtigt hat. Dadurch kann der Beobachter zurückverfolgen von wo in der Anwendung er benachrichtigt wurde.
  * `group` – Die Gruppe, die aus der Datenbank entfernt wurde.

- **update**

```
public void update(java.lang.String arg,Observable observable,
    java.util.List changes)
```

  - **Description**

    Überladung der update()-Methode des Observers. Diese Methode kümmert sich um das Entfernen eines Gruppenmitglieds. Die Daten der Änderung werden in dieser Methode zu einem passenden JSON-Objekt umgewandelt und an den FcmClient weitergegeben, um von dort an die entsprechenden Clients geschickt zu werden. Wer diese Clients sind wird ebenfalls in dieser Methode bestimmt und kann anhand der übergebenen übergebenen Daten ermittelt werden.

  - **Parameters**

    * `arg` – in Argument, das die aufgetretene Änderung beschreibt. Ein Beobachter reagiert nur dann auf die Änderung, wenn dieses Argument mit dem statischen Feld OBSERVER_CODE, das jeder Beobachter besitzt übereinstimmt.
    * `observable` – Eine Instanz des Observable-Objekts, dass den Beobachter benachrichtigt hat. Dadurch kann der Beobachter zurückverfolgen von wo in der Anwendung er benachrichtigt wurde.
    * `changes` – Eine Liste mit Objekten, die die Änderungen beschreiben. Dabei muss die Liste folgenden Aufbau haben: 1. GroupEntity – Gruppe, aus der der Benutzer entfernt werden soll 2. UserEntity – Benutzer, der aus der Gruppe entfernt werden soll

- **update**

```
public void update(java.lang.String arg,Observable observable,
    java.lang.Object o)
```

  - **Description**

    Implementierung der update()-Methode. Wird überladen, um die unterschiedlichen Ereignisse, auf die dieser Observer reagieren kann zu unterscheiden.

– **Parameters**

* `arg` – Ein Argument, das die aufgetretene Änderung beschreibt. Ein Beobachter reagiert nur dann auf die Änderung, wenn dieses Argument mit dem statischen Feld OBSERVER_CODE, das jeder Beobachter besitzt übereinstimmt.
* `observable` – Eine Instanz des Observable-Objekts, dass den Beobachter benachrichtigt hat. Dadurch kann der Beobachter zurückverfolgen von wo in der Anwendung er benachrichtigt wurde.
* `o` – Ein Objekt, das die Änderungen, um die der Beobachter sich kümmern muss enthält. Da es sich um den Datentyp Öbject"handelt, ist der Beobachter sehr flexibel, welche Änderungen ihm übergeben werden können. Dies erleichtert auch das Überladen der Methode, wodurch ein Beobachter mehrere

22.6 Klasse EntityAddedObserver

Bei dieser Klasse handelt es sich um eine Implementierung des Observer-Interfaces. Dementsprechend ist diese Klasse Teil des Observer- Entwurfsmusters und übernimmt die Rolle des konkreten Observers. Die Aufgabe dieser Klasse ist das Beobachten der DAO-Klassen und auf das Erstellen neuer Entitäten zu reagieren. Dies schließt folgende Ereignisse mit ein: - Hinzufügen eines GOs - Hinzufügen einer Gruppenanfrage - Hinzufügen eines Gruppenmitglieds Um auf diese verschiedenen Änderungen reagieren zu können, muss bei der Implementierung die update()-Methode überladen werden oder innerhalb der Methode anhand der übergebenen Änderung entschieden werden, welche weitere Vorgehensweise gewählt werden muss.

22.6.1 Deklaration

```
public class EntityAddedObserver
 extends java.lang.Object implements Observer
```

22.6.2 Field summary

OBSERVER_CODE Der Code anhand dem der Observer erkennt, dass er auf ein notify() reagieren soll.

22.6.3 Konstruktoren Auflistung

EntityAddedObserver()

22.6.4 Methoden Auflistung

update(String, Observable, GoEntity) Überladung der Update-Methode des Observers.
update(String, Observable, GroupEntity) Überladung der Update-Methode des Observers.
update(String, Observable, Object) Implementierung der update()-Methode.
update(String, Observable, UserEntity) Überladung der update-Methode des Observers.

22.6.5 statische Felder

- `public static final java.lang.String` **OBSERVER_CODE**
  – Der Code anhand dem der Observer erkennt, dass er auf ein notify() reagieren soll. Bei jeglichen Änderungen wird jeder Observer benachrichtigt, wer regieren muss wird

anhand dieses Codes entschieden. Er wird als erstes Argument der update()-Methode verwendet.

22.6.6 Attribute

private FcmClient fcmClient Eine Instanz eines FcmClients, der dafür verwendet wird, Nachrichten an die Clients zu schicken. Das Attribut wird bei der Erzeugung eines Observer Objekts automatisch instanziiert (durch Benutzung des einzigen, argumentlosen Konstruktors der FcmClient-Klasse). Danach kann das von außen Attribut nicht mehr verändert werden.

22.6.7 Konstruktoren

- **EntityAddedObserver**

```
public EntityAddedObserver()
```

22.6.8 Methoden

- **update**

```
public void update(java.lang.String arg,Observable observable,edu
    .kit.pse17.go_app.PersistenceLayer.GoEntity go)
```

  - **Description**
    Überladung der Update-Methode des Observers. Diese Implementierung der Methode wird aufgerufen, wenn ein neues GO in einer Gruppe erstellt wurde. Die Methode wandelt das GO daraufhin in ein passendes JSON-Objekt um, um es an den FcmClient weiterzugeben, der es wiederum an die passenden Clients schickt. Wer diese Clients sind wird ebenfalls in dieser Methode bestimmt und kann anhand der übergebenen Go-Entity ermittelt werden.
  - **Parameters**
    - `arg` – Ein Argument, das die aufgetretene Änderung beschreibt. Ein Beobachter reagiert nur dann auf die Änderung, wenn dieses Argument mit dem statischen Feld OBSERVER_CODE, das jeder Beobachter besitzt übereinstimmt.
    - `observable` – Eine Instanz des Observable-Objekts, dass den Beobachter benachrichtigt hat. Dadurch kann der Beobachter zurückverfolgen von wo in der Anwendung er benachrichtigt wurde.
    - `go` – Das Go, das neu erstellt wurde. Es enthält alle Daten die wichtig sind für das GO und die an die Clients weitergegeben werden müssen.

- **update**

```
public void update(java.lang.String arg,Observable observable,edu
    .kit.pse17.go_app.PersistenceLayer.GroupEntity group)
```

  - **Description**
    Überladung der Update-Methode des Observers. Diese Implementierung der Methode wird aufgerufen, wenn eine neue Gruppenanfrage in einer Gruppe erstellt wurde. Die

Methode wandelt die Anfrage daraufhin in ein passendes JSON-Objekt um, um es an den FcmClient weiterzugeben, der es wiederum an die passenden Clients schickt. Wer diese Clients sind wird ebenfalls in dieser Methode bestimmt und kann anhand der übergebenen Go-Entity ermittelt werden.

- **Parameters**
  - `arg` – Ein Argument, das die aufgetretene Änderung beschreibt. Ein Beobachter reagiert nur dann auf die Änderung, wenn dieses Argument mit dem statischen Feld OBSERVER_CODE, das jeder Beobachter besitzt übereinstimmt.
  - `observable` – Eine Instanz des Observable-Objekts, dass den Beobachter benachrichtigt hat. Dadurch kann der Beobachter zurückverfolgen von wo in der Anwendung er benachrichtigt wurde.
  - `group` – Die Gruppe in der die Anfrage neu erstellt wurde. Sie enthält alle Daten die wichtig sind für das GO und die an die Clients weitergegeben werden müssen. Da die Gruppenanfragen in einer Liste gespeichert werden, wird immer das letzte Listenelement als die neu hinzugefügte Anfrage betrachtet. Darauf muss geachtet werden, wenn in der DAO-Klasse eine neue Anfrage angelegt wird.

- **update**

```
public void update(java.lang.String arg,Observable observable,
    java.lang.Object o)
```

  - **Description**
    Implementierung der update()-Methode. Wird überladen, um die unterschiedlichen Ereignisse, auf die dieser Observer reagieren kann zu unterscheiden.
  - **Parameters**
    - `arg` – Ein Argument, das die aufgetretene Änderung beschreibt. Ein Beobachter reagiert nur dann auf die Änderung, wenn dieses Argument mit dem statischen Feld OBSERVER_CODE, das jeder Beobachter besitzt übereinstimmt.
    - `observable` – Eine Instanz des Observable-Objekts, dass den Beobachter benachrichtigt hat. Dadurch kann der Beobachter zurückverfolgen von wo in der Anwendung er benachrichtigt wurde.
    - `o` – Ein Objekt, das die Änderungen, um die der Beobachter sich kümmern muss enthält. Da es sich um den Datentyp Öbject"handelt, ist der Beobachter sehr flexibel, welche Änderungen ihm übergeben werden können. Dies erleichtert auch das Überladen der Methode, wodurch ein Beobachter mehrere

- **update**

```
public void update(java.lang.String arg,Observable observable,edu
    .kit.pse17.go_app.PersistenceLayer.UserEntity user)
```

  - **Description**
    Überladung der Update-Methode des Observers. Diese Implementierung der Methode wird aufgerufen, wenn ein neuer Benutzer zu einer Gruppe hinzugefügt wurde. Die Methode wandelt die Änderung daraufhin in ein passendes JSON-Objekt um, um es an den FcmClient weiterzugeben, der es wiederum an die passenden Clients schickt. Wer diese Clients sind wird ebenfalls in dieser Methode bestimmt und kann anhand der übergebenen User-Entity ermittelt werden.

– **Parameters**

* `arg` – Ein Argument, das die aufgetretene Änderung beschreibt. Ein Beobachter reagiert nur dann auf die Änderung, wenn dieses Argument mit dem statischen Feld OBSERVER_CODE, das jeder Beobachter besitzt übereinstimmt.
* `observable` – Eine Instanz des Observable-Objekts, dass den Beobachter benachrichtigt hat. Dadurch kann der Beobachter zurückverfolgen von wo in der Anwendung er benachrichtigt wurde.
* `user` – Der Benutzer, der der Gruppe hinzugefügt wurde. Die Entität enthält alle Daten die wichtig sind für die Änderung und die an die Clients weitergegeben werden müssen. Da die Gruppenmitgliedschaften in einer Liste gespeichert werden, wird immer das letzte Listenelement als die neu hinzugefügte Gruppe betrachtet. Darauf muss geachtet werden, wenn in der DAO-Klasse eine neue Anfrage angelegt wird.

22.7 Klasse EntityChangedObserver

Bei dieser Klasse handelt es sich um eine Implementierung des Oberver-Interfaces. Dementsprechend ist diese Klasse Teil des Observer- Entwurfsmusters und übernimmt die Rolle des konkreten Observers. Die Aufgabe dieser Klasse ist das Beobachten der DAO-Klassen und auf Änderungen einer bestehenden Entität zu reagieren. Dies schließt folgende Ereignisse mit ein: - Änderung von GO-Daten - Änderung von Gruppendaten - Änderung des Teilnahmestatus - Änderung der Administratoren einer Gruppe Um auf diese verschiedenen Änderungen reagieren zu können, muss bei der Implementierung die update()-Methode überladen werden oder innerhalb der Methode anhand der übergebenen Änderung entschieden werden, welche weitere Vorgehensweise gewählt werden muss.

22.7.1 Deklaration

```
public class EntityChangedObserver
 extends java.lang.Object implements Observer
```

22.7.2 Field summary

OBSERVER_CODE Der Code anhand dem der Observer erkennt, dass er auf ein notify() reagieren soll.

22.7.3 Konstruktoren Auflistung

EntityChangedObserver()

22.7.4 Methoden Auflistung

update(String, Observable, GoEntity) Überladung der Methode update() des Observers.
update(String, Observable, GroupEntity) Überladung der Methode update() des Observers.
update(String, Observable, List) Überladung der Methode update() des Observers.
update(String, Observable, Object) Implementierung der update()-Methode.

22.7.5 statische Felder

- `public static final java.lang.String` **OBSERVER_CODE**
  - Der Code anhand dem der Observer erkennt, dass er auf ein notify() reagieren soll. Bei jeglichen Änderungen wird jeder Observer benachrichtigt, wer regieren muss wird anhand dieses Codes entschieden. Er wird als erstes Argument der update()-Methode verwendet.

22.7.6 Attribute

private FcmClient fcmClient Eine Instanz eines FcmClients, der dafür verwendet wird, Nachrichten an die Clients zu schicken. Das Attribut wird bei der Erzeugung eines Observer Objekts automatisch instanziiert (durch Benutzung des einzigen, argumentlosen Konstruktors der FcmClient-Klasse). Danach kann das von außen Attribut nicht mehr verändert werden.

22.7.7 Konstruktoren

- **EntityChangedObserver**

```
public EntityChangedObserver()
```

22.7.8 Methoden

- **update**

```
public void update(java.lang.String arg,Observable observable,edu
    .kit.pse17.go_app.PersistenceLayer.GoEntity go)
```

  - **Description**
    Überladung der Methode update() des Observers. In dieser Methode werden Datenänderungen an einem GO behandelt. Das geänderte Go wird in ein JSON-Objekt umgewandelt und an den FcmClient weitergegeben, um es an die entsprechenden Clients weiterzuleiten. Wer diese Clients sind wird ebenfalls in dieser Methode bestimmt und kann anhand der übergebenen Go-Entity ermittelt werden.
  - **Parameters**
    - `arg` – Ein Argument, das die aufgetretene Änderung beschreibt. Ein Beobachter reagiert nur dann auf die Änderung, wenn dieses Argument mit dem statischen Feld OBSERVER_CODE, das jeder Beobachter besitzt übereinstimmt.
    - `observable` – Eine Instanz des Observable-Objekts, dass den Beobachter benachrichtigt hat. Dadurch kann der Beobachter zurückverfolgen von wo in der Anwendung er benachrichtigt wurde.
    - `go` – Das Go, dessen Daten verändert wurden.

- **update**

```
public void update(java.lang.String arg,Observable observable,edu
    .kit.pse17.go_app.PersistenceLayer.GroupEntity group)
```

- **Description**
  Überladung der Methode update() des Observers. In dieser Methode werden Datenänderungen an einer Gruppe behandelt. Die geänderte gruppe wird in ein JSON-Objekt umgewandelt und an den FcmClient weitergegeben, um es an die entsprechenden Clients weiterzuleiten. Wer diese Clients sind wird ebenfalls in dieser Methode bestimmt und kann anhand der übergebenen Group-Entity ermittelt werden.
- **Parameters**
  * `arg` – Ein Argument, das die aufgetretene Änderung beschreibt. Ein Beobachter reagiert nur dann auf die Änderung, wenn dieses Argument mit dem statischen Feld OBSERVER_CODE, das jeder Beobachter besitzt übereinstimmt.
  * `observable` – Eine Instanz des Observable-Objekts, dass den Beobachter benachrichtigt hat. Dadurch kann der Beobachter zurückverfolgen von wo in der Anwendung er benachrichtigt wurde.
  * `group` – Die Gruppe, deren Daten verändert wurden.

- **update**

```
public void update(java.lang.String arg,Observable observable,
    java.util.List changes)
```

  - **Description**
    Überladung der Methode update() des Observers. In dieser Methode wird das Ereignis eines neu hinzugefügten Admins behandelt und an die entsprechenden Clients weitergeleitet. Wer diese Clients sind wird ebenfalls in dieser Methode bestimmt und kann anhand der übergebenen Group-Entity ermittelt werden.
  - **Parameters**
    * `arg` – Ein Argument, das die aufgetretene Änderung beschreibt. Ein Beobachter reagiert nur dann auf die Änderung, wenn dieses Argument mit dem statischen Feld OBSERVER_CODE, das jeder Beobachter besitzt übereinstimmt.
    * `observable` – Eine Instanz des Observable-Objekts, dass den Beobachter benachrichtigt hat. Dadurch kann der Beobachter zurückverfolgen von wo in der Anwendung er benachrichtigt wurde.
    * `changes` – Eine Liste von Objekten, die die Änderung beschreiben. Dabei muss die Liste folgende Struktur haben: 1. String – ÄDMIN"2. GroupEntity – Gruppe, um die es sich handelt 3. UserEntity – Benutzer, der zum Administrator gemacht wurde.

- **update**

```
public void update(java.lang.String arg,Observable observable,
    java.lang.Object o)
```

  - **Description**
    Implementierung der update()-Methode. Wird überladen, um die unterschiedlichen Ereignisse, auf die dieser Observer reagieren kann zu unterscheiden.
  - **Parameters**
    * `arg` – Ein Argument, das die aufgetretene Änderung beschreibt. Ein Beobachter reagiert nur dann auf die Änderung, wenn dieses Argument mit dem statischen Feld OBSERVER_CODE, das jeder Beobachter besitzt übereinstimmt.

* `observable` – Eine Instanz des Observable-Objekts, dass den Beobachter benachrichtigt hat. Dadurch kann der Beobachter zurückverfolgen von wo in der Anwendung er benachrichtigt wurde.
* `o` – Ein Objekt, das die Änderungen, um die der Beobachter sich kümmern muss enthält. Da es sich um den Datentyp Öbject"handelt, ist der Beobachter sehr flexibel, welche Änderungen ihm übergeben werden können. Dies erleichtert auch das Überladen der Methode, wodurch ein Beobachter mehrere

22.8 Klasse GoClusterStrategy

In dieser Klasse wird der in er Anwendung verwendete Clustering-Algorithmus implementiert. Die Ausführung des Algorithmus wird von der Klasse LocationService aufgerufen. Diese Klasse ist Teil einer Implementierung des Entwurfsmusters SStrategieünd übernimmt dabei die Rolle der konkreten Strategie. Das Interface der abstrakten Strategie, in diesem Fall ClusterStrategy"wird implementiert. Die führeAus()Methode der Strategie ist die Methode calculateCluster()"

22.8.1 Deklaration

```
public class GoClusterStrategy
 extends java.lang.Object implements ClusterStrategy
```

22.8.2 Konstruktoren Auflistung

GoClusterStrategy(int)

22.8.3 Methoden Auflistung

calculateCluster(List) Methode des Interfaces, die hier implementiert wird.
getThreshold()
setThreshold(int)

22.8.4 Attribute

private int threshold Ein Schwellwert für die Genauigkeit, mit der der Clustering-Algorithmus ausgeführt wird. Der Wert liegt zwischen 1 (sehr ungenau) und 10 (sehr genau). Dieser Wert kann nach der Instanziierung des GoClusterStrategy-Objekts nicht mehr verändert werden. Wird der Wert nicht wenigstens einmal spezifiziert wird default-mäßig ein Wert von 5 benutzt.

22.8.5 Konstruktoren

- **GoClusterStrategy**

```
public GoClusterStrategy(int threshold)
```

22.8.6 Methoden

- **calculateCluster**

```
public java.util.List calculateCluster(java.util.List
    userLocationList)
```

– **Description**

Methode des Interfaces, die hier implementiert wird. Der Aufruf dieser Methode stößt die Ausführung des Algorithmus an und sie liefert die Ergebnisse des Clustering-Vorgangs an den Aufrufer zurück.

– **Parameters**

* `userLocationList` – Eine Liste mit den aktuellen Standorten der einzelnen GO-Teilnehmer. Die Länge der Liste beträgt dabei mindestens drei Objekte und maximal 50 Objekte.

– **Returns** – eine Liste von Cluster-Objekten, die den aktuellen Standort der Gruppe beschreiben. Die Länge der Liste liegt zwischen 1 und 50.

- **getThreshold**

```
public int getThreshold()
```

- **setThreshold**

```
public void setThreshold(int threshold)
```

22.9 Klasse LocationService

Diese Klasse bietet eine Schnittstelle für den GOrestController, an die Anfragen, die den User- bzw. Gruppenstandort betreffen, weitergeleitet werden können. Die Aufgabe dieser Klassen ist das Verwalten und Bearbeiten dieser Anfragen. Um die Anfragen bearbeiten zu können, bedient sich die die Klasse dem Clustering Algorithmus, der implementiert ist mit der Schnittstelle, wie sie in dem Interface ClusterStrategy beschrieben ist. Alle Programmteile, die Funktionalitäten aus dieser Klasse benötigen, stellen ihre Anfragen an statische Methoden. Erst innerhalb der Klasse wird die anfrage dem richtigen LocationService-Objekt zugeordnet. Dies erlaubt eine klare Trennung der Teile des GOs, die in der Datenbank verwaltet werden und denen, die in dieser Klasse verwaltet werden. Diese Klasse ist Teil einer Implementierung des Entwurfsmusters SStrategie". Sie übernimmt die Rolle des Aufrufers, d.h. sie ruft eine Implementierung einer abstrakten Strategie (hier: eine Implementierung von ClusterStrategie) auf, ohne dass des dabei eine Rolle spielt, wie genau die Implementierung aussieht.

22.9.1 Deklaration

```
public class LocationService
 extends java.lang.Object
```

22.9.2 Konstruktoren Auflistung

LocationService() Der einzige Konstruktor dieser Klasse nimmt keine Argumente entgegen.

22.9.3 Methoden Auflistung

getGroupLocation(long) Gibt die aktuelle GroupLocation des spezifizierten GOs an den Aufrufer zurück.

setUserLocation(long, String, long, long) Diese Methode speichert eine User-Location in der acitveUsers Liste des entsprechenden GOs.

22.9.4 Attribute

private static List<LocationService> activeServices Diese Map enthält für jedes gerade aktive GO ein LocationService Objekt, welches die alle dieses GO betreffende Anfragen übernimmt. Der Schlüssel der Map ist die ID des GOs zu dem das LocationService-Objekt gehört. Die maximale Länge der Map beträgt 3000 Wertepaare. Ein neues Objekt wird in die Liste eingefügt, wenn die getLocationService()-Methode aufgerufen wird, und dabei kein passender Service gefunden wird. Die Erstellung der LocationService-Objekte findet ausschließlich in dieser Klasse statt.

private final ClusterStrategy strat Ein Clustering-Strategie, die den Algorithmus, der für das Clustering benutzt wird festlegt. Das Attribut ist final, da es, nachdem es einmal festgelegt wurde nicht mehr verändert werden sollte. Ein GO sollte stattdessen immer den gleichen Algorithmus benutzen.

private List<UserLocation> activeUsers Eine Liste mit den UserLocations aller Benutzer, die momentan ihren Standort mit den anderen teilen. Die Länge der Liste liegt zwischen 0 und 50 UserLocation-Objekten.

private List<Cluster> groupLocation Eine Liste mit den aktuellen Clustern der Standorte des GOs. Diese Liste repräsentiert die Ergebnisse des Clustering für den Input ächtiveUsers". Die Länge der Liste liegt zwischen 0 und 50 Cluster-Objekten.

private private int newLocationCounter Eine Zählvariable, um sich zu merken, wie viele neue Locations übermittelt wurden, seit das letzte Mal die groupLocation des GOs berechnet wurde. Die Berechnung findet nur statt, wenn diese Variable einen Wert größer als 5 hat. Danach wird der Zähler wieder auf 0 gesetzt. Sämtliche Manipulationen an diesem Attribut finden innerhalb dieser Klasse statt. Nach außen hin sit diese Variable nicht sichtbar und insbesondere nicht veränderbar.

private private int userCounter Eine Zählvariable, um sich zu merken, wie viele verschiedene Benutzer bereits ihren Standort geteilt haben. Ist diese Zahl kleienr als 3, so wird keine groupLocation berechnet. Dies dient der Anonymisierung der einzelnen Benutzer, was bei einer Anzahl von UserLocations kleiner als 3 nicht mehr garantiert werden kann.

22.9.5 Konstruktoren

- **LocationService**

```
public LocationService()
```

  - **Description**

    Der einzige Konstruktor dieser Klasse nimmt keine Argumente entgegen. Sämtliche Attribute werden nur innerhalb dieser Klasse gesetzt und verändert. Die default-Werte der Attribute sind: - activeUsers: leere Liste - groupLocation: leere Liste - strat: Objekt einer Klasse, die ClusterStrategy implementiert. Als threshold-Wert wird dem Konstruktor 5 übergeben. - newLocationCounter: 0 - userCounter: 0

22.9.6 Methoden

- **getGroupLocation**

```
public static java.util.List getGroupLocation(long goId)
```

- **Description**

  Gibt die aktuelle GroupLocation des spezifizierten GOs an den Aufrufer zurück. Aufgerufen wird diese Methode von einem GoRestController, der von einem Client eine Anfrage nach der groupLocation eines GOs bekommen hat. Bei einem Methodenaufruf wird das entsprechende GO aus der Map activeGos anhand der goId herausgesucht. Es ist dabei garantiert, dass das GO in der Map zu finden ist. Aus dem locationService-Objekts des GOs wird zuerst der Wert des newLocationCounters betrachtet. Ist dieser größer oder gleich 5, wird die ClusterStrategy aufgerufen und die groupLocation neu berechnet. Anschließend wird der newLocationCounter zurückgesetzt und die groupLocation wird zurückgegeben.

- **Parameters**

  * `goId` – Die ID des GOs, dessen groupLocation gesucht wird. Dabei handelt es sich um eine gültige GO-ID, die ein Schlüssel in der Map active GOs ist.

- **Returns** – Eine Liste mit Cluster-Objekten. Diese stellt den aktuellen Standort der GO-Teilnehmer dar. Dabei ist die Länge der Liste zwischen 0 und 50. Ist die Liste leer, heißt das, dass noch nicht genügend Teilnehmer ihren Standort übermittelt haben, um eine groupLocation ausrechnen zu können, ohne die Anonymisierungs-Vorschriften der Anwendung zu verletzen.

- **setUserLocation**

```
public static void setUserLocation(long goId,java.lang.String
    userId,long lat,long lon)
```

- **Description**

  Diese Methode speichert eine UserLocation in der activeUsers Liste des entsprechenden GOs. Sie wird aufgerufen von der GoRestController Klasse, wenn diese eine setLocation-Anfrage von einem Benutzer bekommt. Bei einem Methodenaufruf wird aus der Liste der activeGos das richtige locationService Objekt ausgewählt und dort die userLocation in der activeUsers Liste aktualisiert. Sollte dieses Objekt in der Liste nicht existierten, wird es erzeugt und der Liste hinzugefügt. In diesem Fall wird die Variable userCounter um eines erhöht, da ein neuer Benutzer angefangen hat, seinen Standort zu teilen. Bei jedem Aufruf dieser Methode wird außerdem der Wert von newLocationCounter um 1 erhöht. Sollte das GO in der Map activeGos nicht zu finden sein, wird ein neues LocationService-Objekt erzeugt und der Map hinzugefügt.

- **Parameters**

  * `goId` – Die ID des GOs zu dem die Location gehört. Es muss eine gültige ID eines GOs sein. Sie wird verwendet, um den richtigen locationService aus einer statischen Map zu finden.
  * `userId` – Die ID des Benutzers, zu dem der Standort gehört. Es muss sich um eine gültige Benutzer-ID handeln.
  * `lat` – Der geographische Breitengrad des Standorts des Benutzers. Der Wert muss als Breitengrad interpretierbar sein, muss also zwischen +90 und -90 liegen.
  * `lon` – Der geographische Längengrad des Standorts des Benutzers. Der Wert muss als Längengrad interpretierbar sein, muss also zwischen +180 und -180 liegen.

22.10 Klasse UserLocation

Bei dieser Klasse handelt es sich um eine Datenhaltungsklasse, die dem Clustering-Algorithmus das hantieren mit den Standorten erleichtert. Ein Objekt dieser Klasse beschreibt dabei einen Benutzerstandort.

22.10.1 Deklaration

```
public class UserLocation
 extends java.lang.Object
```

22.10.2 Konstruktoren Auflistung

UserLocation(String, long, long)

22.10.3 Methoden Auflistung

getLat()
getLon()
getUserId()
setLat(long)
setLon(long)
setUserId(String)

22.10.4 Attribute

private String userId Die ID des Benutzers, um dessen Standort es sich handelt. Es handelt sich um eine gültige Benutzer-ID, die von anderen Klassen der anwendung erkannt werden kann.

private long lat Der geographische Breitengrad des Standorts des Benutzers. Der Wert muss als Breitengrad interpretierbar sein, muss also zwischen +90 und -90 liegen.

private long lon Der geographische Längengrad des Standorts des Benutzers. Der Wert muss als Längengrad interpretierbar sein, muss also zwischen +180 und -180 liegen.

22.10.5 Konstruktoren

- **UserLocation**

```
public UserLocation(java.lang.String userId,long lat,long lon)
```

22.10.6 Methoden

- **getLat**

```
public long getLat()
```

- **getLon**

```
public long getLon()
```

- **getUserId**

  ```
  public java.lang.String getUserId()
  ```

- **setLat**

  ```
  public void setLat(long lat)
  ```

- **setLon**

  ```
  public void setLon(long lon)
  ```

- **setUserId**

  ```
  public void setUserId(java.lang.String userId)
  ```

23 Client-Server-Schnittstelle

Dieser Abschnitt erläutert die Schnittstelle zwischen dem Client und dem Server. Diese Schnittstelle besteht aus zwei Teilen:

Zum Einen bietet der Server eine REST API an, über die der Client die Dienste des Servers in Anspruch nehmen kann. Zum Anderen gibt es eine Schnittstelle, die über Firebase Cloud Messaging realisiert ist, damit der Server Nachrichten an bestimmte Clients schicken kann.

23.1 REST API des Servers

Folgende Grafik zeigt, welche Methoden unter welchen URL des Servers zu erreichen sind:

	GET	POST	PUT	DELETE
/gos		JA		
/gos/status			JA	
/gos/location/{goId}	JA		JA	
/gos/{goId}			JA	JA
/groups		JA		
/groups/{groupId}			JA	JA
/groups/members/{groupId}/{userId}			JA	JA
/groups/requests/{groupId}/{userId}		JA		JA
/groups/admins/{groupId}/{userId}		JA		
/users/{userId}	JA	JA		JA
/users/device/{instanceId}			JA	

Abbildung 3: Übersicht über die Rest Api des Servers

Aufrufe der Request-Methoden DELETE und GET enthalten keinen Content-Body. Sämtliche Informationen, die der Server braucht, um richtig auf die anfrage antworten zu können, sind in der URL kodiert. Bei den Methoden POST und PUT, sowie bei Antworten des Servers, die einen Content-Body erfordern, ist sind die Daten in einem JSON-Objekt gekapselt. Dieses Objekt wird von dem Framework Gson aus der entsprechenden Entity-Klasse erzeugt. die Anwendung muss den genauen Aufbau des JSON Objekts nicht kennen. Die Verantwortung für die Verwaltung derselben wird hier vollständig an Gson übergeben.

Bei sämtlichen Requests kann der Client anhand des HTTP-Statuscodes der Server-Response erkennen, ob die Anfrage erfolgreich ausgeführt werden konnte.

23.2 FCM Schnittstelle

Die Schnittstelle zwischen Server und Client, die zum Senden von Downstream-Nachrichten verwendet werden kann, wird über Firebase Cloud Messaging realisiert. Der Server sendet einen HTTP Post Request an den Firebase Server. Dabei besteht der Content-Body dieser HTTP-Anfrage aus einem JSON-Objekt indem der Empfänger und die zu übermittelnden Daten spezifiziert sind.

Folgende Grafik [1] zeigt den Aufbau eines HTTP-Requests, wie er an den FCM Server gesendet werden muss für eine erfolgreiche Weiterleitung der Nachricht:

[1] Quelle: https://firebase.google.com/docs/cloud-messaging/send-message

```
https://fcm.googleapis.com/fcm/send
Content-Type:application/json
Authorization:key=AIzaSyZ-1u...0GBYzPu7Udno5aA

{ "data": {
    "score": "5x1",
    "time": "15:10"
  },
  "to" : "bk3RNwTe3H0:CI2k_HHwgIpoDKCIZvvDMExUdFQ3P1..."
}
```

Abbildung 4: Beispiel für ein HTTP-Request an den FCM Server

Das in grün markierte JSON-Objekt kann dabei je nach Anwendungsfall ein anderes Data-Field enthalten. Das toFeld enthält die instanceId des Empfängers der Nachricht.

Inhalt des 'data'-Felds für die verschiedenen Anwendungsfälle der App: Zunächst enthalten sämtliche Nachrichten unter dem Tag tagëinen String der signalisiert, was der Anlass zum Senden der Nachricht war. Bei diesen Strings handelt es sich um Elemente des Enums EventArg.

- *Go Added*
  Ein aus einer GO-Entität erzeugtes JSON-Objekt unter dem Tag 'go'. Dieses wird automatisch durch das Framework Gson erzeugt.

- *Go Edited*
  Ein aus einer GO-Entität erzeugtes JSON-Objekt unter dem Tag 'go'. Dieses wird automatisch durch das Framework Gson erzeugt. Es werden allerdings die Listen der Go-Teilnehmer aus dem Objekt entfernt, da Änderungen derselben von diesem Anwendungsfall nicht betroffen sind und die Daten somit nicht übertragen werden müssen.

- *Go Removed*
  Die ID des entfernten Gos unter dem Tag 'id'

- *Group Edited*
  Ein aus einer Group-Entität erzeugtes JSON-Objekt unter dem Tag 'group'. Dieses wird automatisch durch das Framework Gson erzeugt. Es werden allerdings die Listen der Gruppenmitglieder und Administratoren aus dem Objekt entfernt, da Änderungen derselben von diesem Anwendungsfall nicht betroffen sind und die Daten somit nicht übertragen werden müssen.

- *Group Removed*
  Die ID der entfernten Gruppe unter dem Tag 'id'

- *Group Request Received*
  Ein aus einer Group-Entität erzeugtes JSON-Objekt unter dem Tag 'group'. Dieses wird automatisch durch das Framework Gson erzeugt

- *Member Added*
  Die ID der Gruppe zu der der Benutzer hinzugefügt werden soll unter dem Tag 'id'. Unter dem Tag 'user' ist ein JSON-Objekt gespeichert, das aus einer User-Entität erzeugt wurde. Dies geschieht automatisch durch das Framework Gson.

- *Member Removed*
  Die ID des Benutzers, der aus der Gruppe entfernt werden soll unter dem Tag 'user_id' und die ID der Gruppe, aus der der Benutzer entfernt werden soll unter dem Tag 'group_id'.

- *Admin Added*
  Die ID des Benutzers, der als Adminsitrator hinzugefügt werden soll unter dem Tag 'user_id' und die ID der Gruppe, in der dies geschehen soll unter dem Tag 'group_id'. Es ist nicht nötig das vollständige User-Objekt zu senden, da dies bereits auf den Clients in dem entsprechenden Gruppen-Objekt gespeichert ist.

- *Status Changed*
  Die ID des Benutzers, der seinen Status geändert hat unter dem Tag 'user_id', die ID des GOs in der die Statusänderung stattgefunden hat unter dem Tag 'go_id' und eine Zahl, die den neuen Status repräsentiert, unter dem Tag 'status'. Es gilt '0': ABELEHNT, '1': BESTÄTIGT, '2': UNTERWEGS.

Bei den Clients kommt die gesendete Nachricht als remoteMessage-Objekt an. Durch die getData()-Methode kann auf den Content-Body, also das JSON-Objekt, das den eigentlichen Inhalte der Nachricht entält zugegriffen werden.

24 Klassendiagramme

24.1 Server

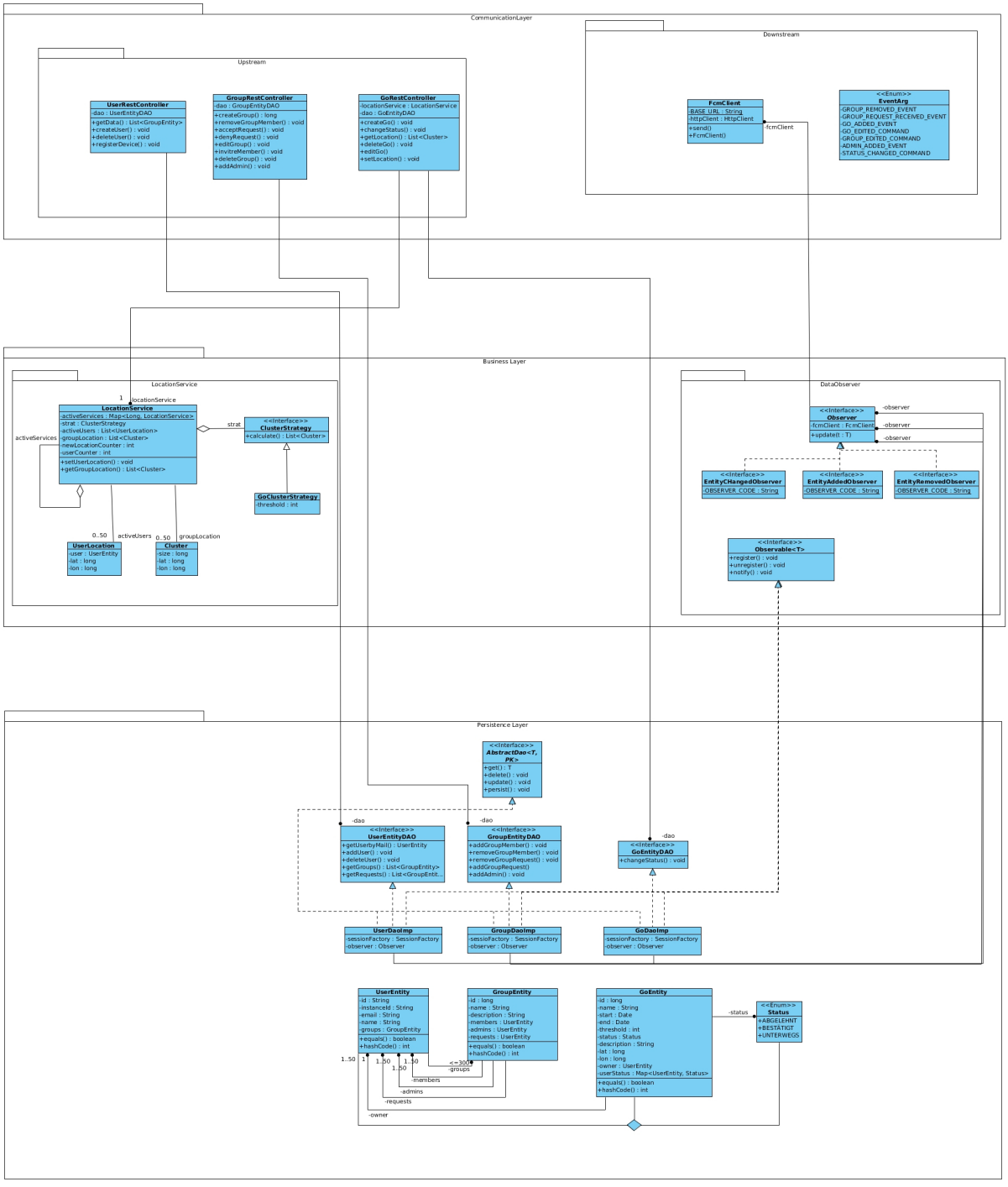

Abbildung 5: Klassendiagramm der Serveranwendung

Bemerkung: Die Argumente der Funktionen wurden im Klassendiagramm zur besseren Übersichtlichkeit ausgelassen. Sie können den Klassenbeschreibungen entnommen werden.

25 Client

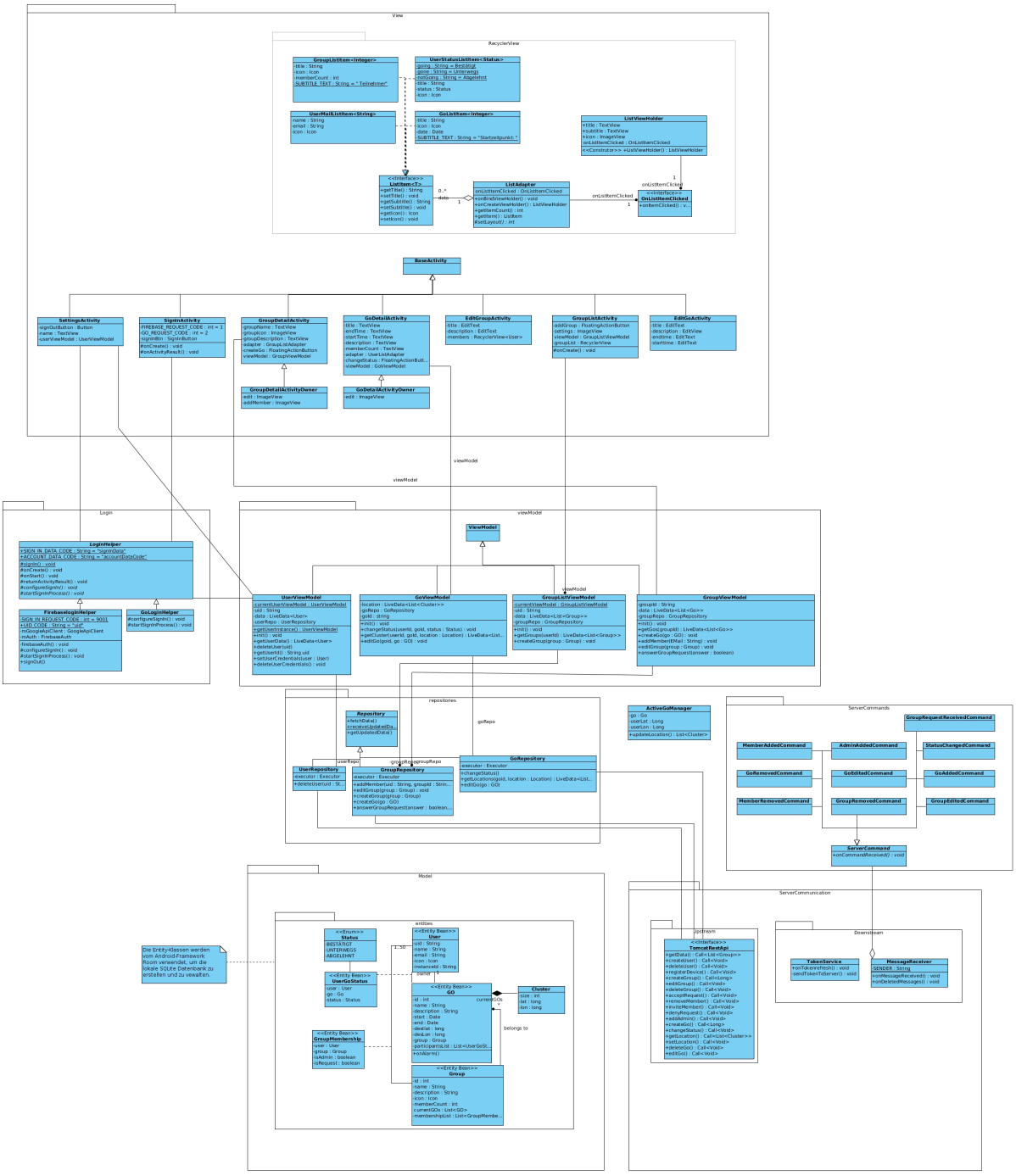

Abbildung 6: Klassendiagramm der Serveranwendung

26 Sequenzdiagramme

26.1 Hinzufügen eines Gruppenmitglieds

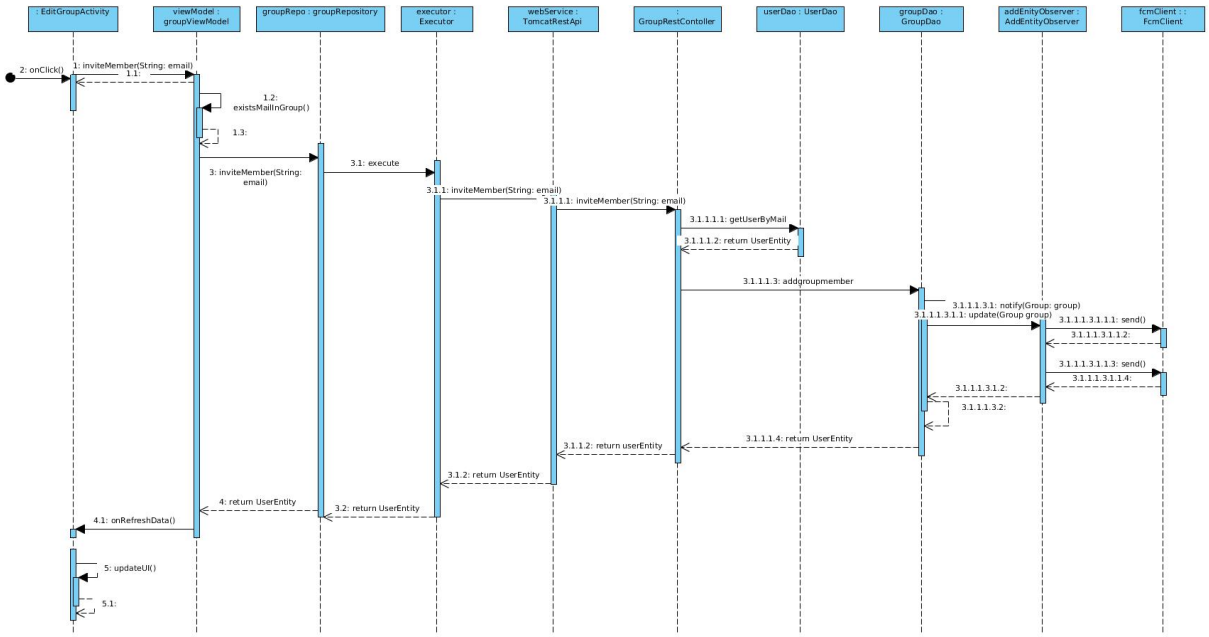

Abbildung 7: Sequenzdiagramm - Hinzufügen eines Gruppenmitglieds Teil 1

Das obige Sequenzdiagramm zeigt, was während der Ausführung des Programms passiert, wenn ein Benutzer die Funktion ïnviteMemberäusführt. Das User Interface stellt dem Benutzer ein Textfeld zur Eingabe der E-Mailadresse und einen Button zum Bestätigen zur Verfügung. Bei Klick dieses Buttons extrahiert die Activity-Klasse die eingegebene Mail-Adresse aus dem Textfeld und übergibt diese an das ViewModel über den Methodenaufruf ïnviteMember". Das ViewModel überprüft zunächst ob es bereits einen Benutzer in Gruppe gibt, der diese E-Mailadresse besitzt. Falls nicht, wird die Gruppeneinladung an die Grouprepository weitergeleitet und von dort über die Klasse TomcatrestApi an den Server gesendet.

Empfängt der Server eine Anfrage, einen User zu einer Gruppe hinzuzufügen, wird diese Anfrage zunächst an das UserDao weitergegeben. Dort wird zuerst die Methode getUserByMail() aufgerufen, um den richtigen Benutzer aus der Datenbank zu finden. Danach wird die addGroupmember-methode des GroupDaos aufgerufen. In dieser Methode wird die neue Gruppenanfrage in der Datenbank gespeichert und es werden die Observer benachrichtigt, dass sich Daten geändert haben.

Der AddEntityObserver erkennt, dass es sich um eine Änderung handelt, die seinen Verantwortungsbereich betrifft. Er bekommt beim Aufruf der update()-Methode die Gruppe mit der zusätzlichen Gruppenanfrage übergeben. Der Observer extrahiert alle Gruppenmitglieder aus dem Gruppenobjekt und ruft die send()-Methode des FcmClients auf, um das geänderte Gruppenobjekt an alle Gruppenmitglieder zu senden. Danach wird die send()-Methode ein zweites Mal aufgerufen, um dem neuen Gruppenmitglied die neue Gruppenanfrage zu übermitteln.

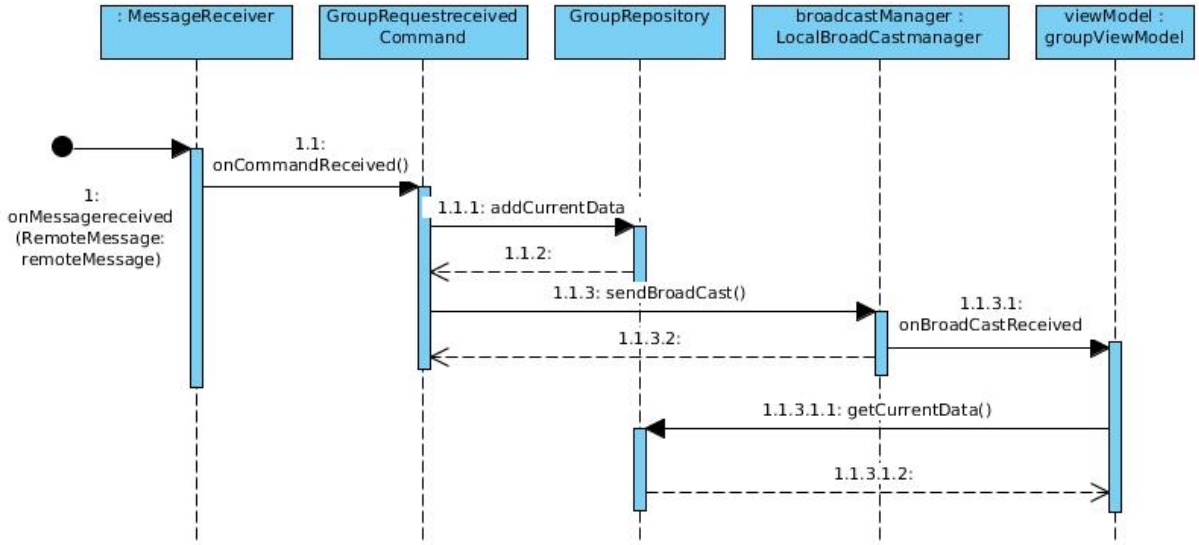

Abbildung 8: Sequenzdiagramm - Hinzufügen eines Gruppenmitglieds Teil 2

Das zweite Sequenzdiagramm zeigt, was passiert, wenn an einen Benutzer eine Gruppenmitgliedschaftsanfrage gesendet wird. Die Nachricht, die von dem Server, über dem Firebase Cloud Messaging Server, an den Client gesendet wird, löst einen Aufruf der Methode onMessageReceived() in der Klasse MessageReceiver aus. Diese Klasse extrahiert das JSON-Feld COMMAND_CODE aus dem empfangenen JSON-Objekt und findet so heraus, an welches ServerCommand-Objekt die Anfrage weitergeleitet werden muss.

Nach Weiterleitung der Anfrage an den GroupRequestReceivedCommand wird dort das Datenfeld aus der JSON-Nachricht extrahiert. dort ist die Gruppe gespeichert, zu der er Benutzer eingeladen wurde. Diese Gruppe wird in dem öffentlichen CurrentDataField des GroupRepository gespeichert. Danach schickt das GroupRequestReceivedCommand-Objekt einen Broadcast an alle ViewModels. Das GroupViewModel erkennt, dass der Broadcast eine Änderung der Gruppen des Benutzers betrifft. Daher wird dort die onBroadcastReceived()-Methode aufgerufen. Daraufhin holt sich das ViewModel die aktualisierten Daten von der GoupRepository ab, durch einen Aufruf der getCurrentData()-Methode. Da das UI die LiveData der ViewModels beobachtet, wird automatisch bei einer Aktualisierung des ViewModels auch das UI aktualisiert und zeigt die neuen Daten an.

26.2 Entfernen einer Gruppe

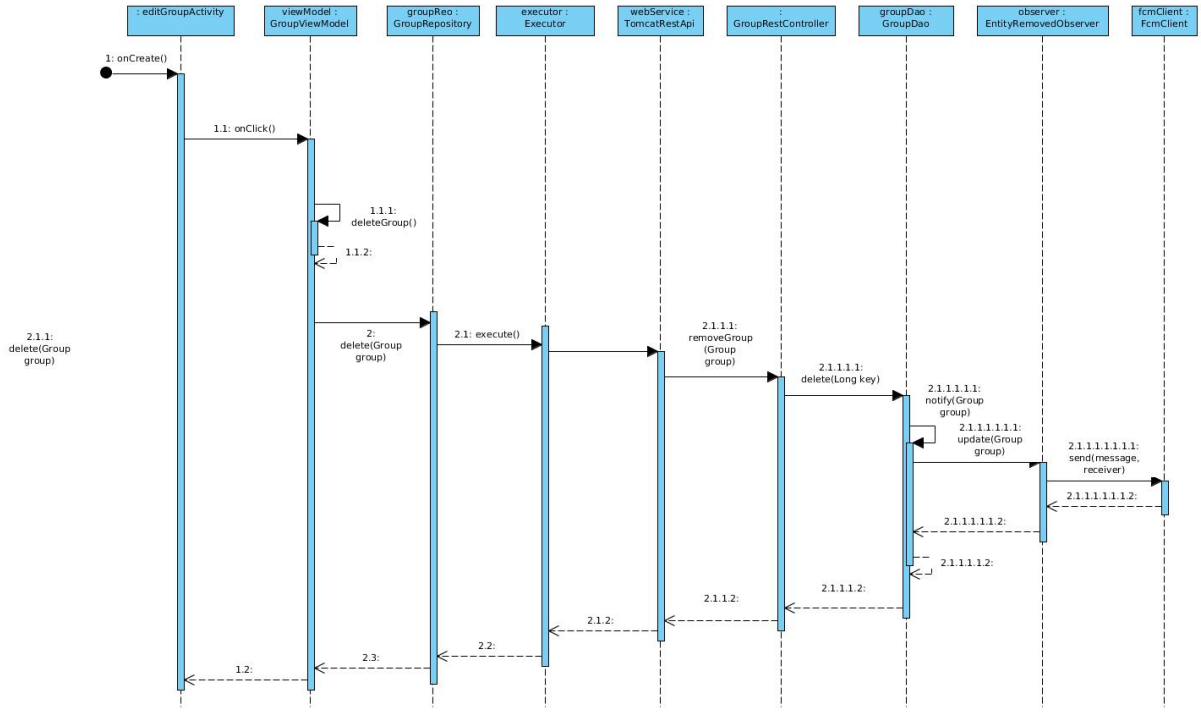

Abbildung 9: Sequenzdiagramm - Entfernen einer Gruppe

Das obige Sequenzdiagramm zeigt den Programmablauf, nachdem ein Benutzer die "Gruppe löschenFunktion ausgelöst hat. Das UI gibt den Button Press an das GroupViewModel weiter. Dort wird die Gruppe zunächst in den lokalen Daten gelöscht. Dabei muss sichergestellt werden, dass die Daten nach dem Löschen konsistent sind, also z.B. auch alle GOs der Gruppe gelöscht wurden.

Danach wird die Anfrage über die GroupRepository und das Rest API an den Server übergeben, wo sie durch den Methodenaufruf deleteGroup() in der GroupRestController-Klasse ankommt. Von dort aus wird die Anfrage an das GroupDao gegeben, welches die Gruppe in der Datenbank löscht. Auch hier muss auf die Konsistenz der Daten geachtet werden. Danach werden die Observer des GroupDaos benachrichtigt, dass eine Änderung stattgefunden hat. Da die Änderung nur den EntityRemovedObserver betrifft, wird bei diesem Objekt die Methode update() aufgerufen. Mit dem Methodenaufruf wird auch die gelöschte Gruppe übergeben.

Der Observer baut ein Message-Objekt aus der erhaltenen Gruppe und extrahiert eine Liste aller Gruppenmitglieder aus dem Gruppenobjekt. Diese Daten werden weitergegeben an dem FcmClient über die Methode send(). Dadurch werden die Nachrichten über die Löschung der Gruppe an die Gruppenmitglieder geschickt, damit diese ihre lokalen Daten anpassen können.

26.3 Teilnahmestatus ändern

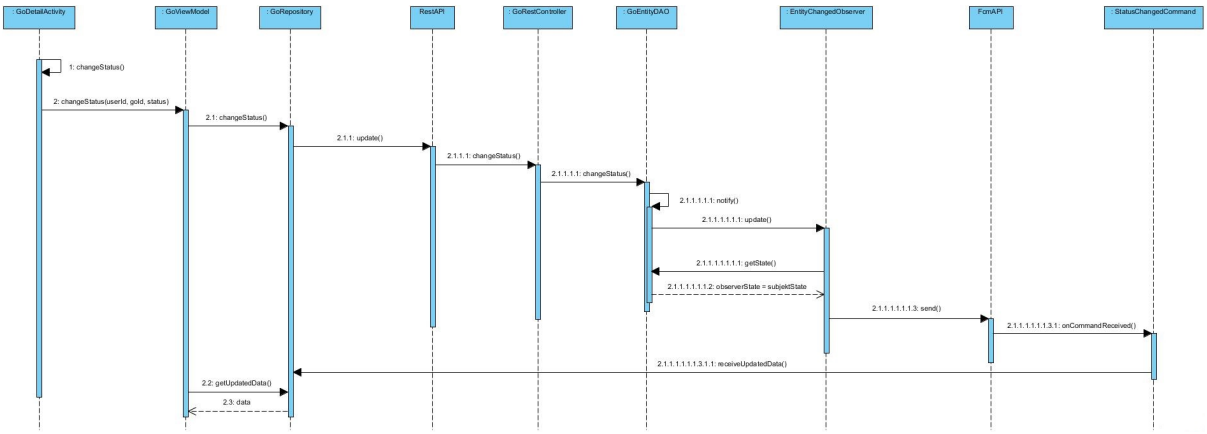

Abbildung 10: Sequenzdiagramm - Teilnahmestatus des Benutzers innerhalb eines GOs ändern

Das obige Sequenzdiagramm zeigt, was während der Ausführung des Programms passiert, wenn ein Benutzer seinen Teilnahmestatus innerhalb eines GOs ändert. Das User Interface stellt dem Benutzer ein Button in der GO-Detail-Ansicht zur Verfügung. Beim Anklicken des Buttons wird dem Benutzer ein Feld mit der Wahlmöglichkeit eines Status gezeigt. Activity-Klasse extrahiert den neuen Status und übergibt diesen an das GoViewModel über den Methodenaufruf 'changeStatus()' mit userId, goId und status als Arguments enthalten. Das ViewModel überprüft zunächst ob es ein gültiger Status für diesen Benutzer ist. Falls ja, wird die Anfrage an das GoRepository weitergeleitet und von dort über die Klasse TomcatRestApi (RestAPI) an den Server gesendet.

Die Anfrage kommt durch den Methodenaufruf 'changeStatus()' in der GoRestController-Klasse an und wird dann an das GoEntityDAO weitergegeben, wobei die Methode 'changeStatus()' des GoEntityDAOs aufgerufen wird. Der neue Status des Benutzers beim aktuellen GO wird in der Datenbank ('UserGoStatus' Entity Bean) gespeichert und es werden die Observer benachrichtigt, dass sich die Daten geändert haben.

Der EntityChangedObserver merkt sich diese Änderung des betroffenen GOs, das er beim Aufruf der 'update()'-Methode bekommt. Der Observer baut ein Message-Objekt aus dem erhaltenen GO und extrahiert eine Liste aller GO-Teilnehmer. Diese Daten werden weitergegeben an dem FcmClient über die Methode 'send()'. Die Nachricht, die von dem Server, über dem Firebase Cloud Messaging Server, an den Client gesendet wird, löst einen Aufruf der Methode 'onMessageReceived()' in der Klasse MessageReceiver aus. Diese Klasse findet heraus, an welches ServerCommand-Objekt die Anfrage weitergeleitet werden muss.

Nach Weiterleitung der Anfrage an den StatusChangedCommand werden die neuen Daten (das GO) im GoRepository gespeichert. Danach schickt das StatusChangedCommand-Objekt einen Broadcast an alle ViewModels. Das GoViewModel erkennt, dass der Broadcast eine Änderung des GOs des Benutzers betrifft. Daraufhin holt sich das ViewModel die aktualisierten Daten von der GoRepository ab, durch einen Aufruf der 'getCurrentData()'-Methode. Da das UI die LiveData der ViewModels beobachtet, wird automatisch bei einer Aktualisierung des ViewModels auch das UI aktualisiert und zeigt die neuen Daten an.

26.4 GPS-Standort ermitteln

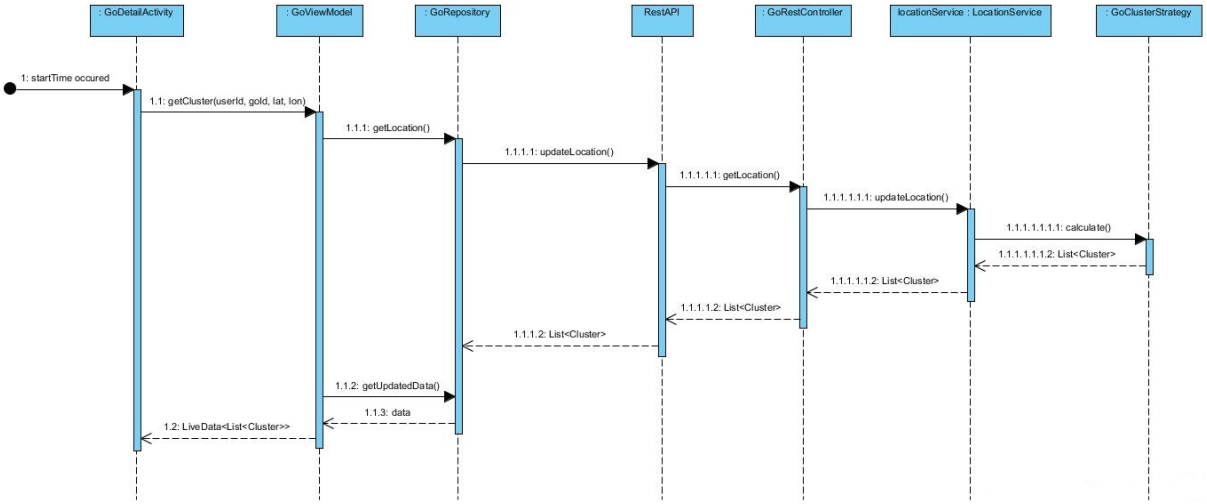

Abbildung 11: Sequenzdiagramm - Anonymisierte und gemittelte GPS-Standort (Cluster der Standorte) der Teilnehmer eines GOs ermitteln

Das obige Sequenzdiagramm zeigt, was während der Ausführung des Programms passiert, wenn der Startzeitpunkt eines bestimmten GOs eintritt. Zu dem Zeitpunkt wird der Teilnahmestatus aller Teilnehmer des GOs mit dem Status 'Bestätigt' auf 'Unterwegs' gesetzt. Dabei wird die Methode 'getCluster()' der Klasse GoViewModel mit Parametern userId, goId und Location (latitude und longitude) aller Teilnehmer des GOs mit dem Status 'Unterwegs' aufgerufen. Das ViewModel überprüft zunächst ob die Daten für diesen Benutzer gültig sind. Falls ja, wird die Anfrage an das GoRepository weitergeleitet und von dort über die Klasse TomcatRestApi (RestAPI) an den Server gesendet.

Die Anfrage kommt durch den Methodenaufruf 'getLocation()' in der GoRestController-Klasse an und wird dann an das LocationService weitergegeben. LocationService spielt dabei die Rolle von Context des Strategie-Entwurfsmusters. Die konkrete Strategie 'GoClusterStrategy' wird benutzt, um mithilfe der Methode 'calculate()' das Cluster aus den GPS-Standorten aller Teilnehmer des GOs zu berechnen und zurückzugeben.

Das berechnete Cluster wird weiter zurückgegeben, bis die neuen Daten (Cluster der Standorte) im GoRepository gespeichert werden. Danach wird ein Broadcast an alle ViewModels geschickt, dabei erkennt das GoViewModel, dass der Broadcast eine Änderung des GOs (also GPS-Standort aller Teilnehmer) betrifft. Daraufhin holt sich das ViewModel die aktualisierten Daten von der GoRepository durch einen Aufruf der 'getUpdatedData()'-Methode ab. Da das UI die LiveData der ViewModels beobachtet, wird automatisch bei einer Aktualisierung des ViewModels auch das UI aktualisiert und zeigt die neuen Daten an.