

# **goApp Implementierung**

Jörn Kussmaul, Katharina Riesterer, Julian Neubert,  
Jonas Walter, Tobias Ohlsson, Eva-Maria Neumann

18. Februar 2017

# Inhaltsverzeichnis

|          |                                     |           |
|----------|-------------------------------------|-----------|
| <b>1</b> | <b>Einleitung</b>                   | <b>3</b>  |
| <b>2</b> | <b>Änderungen am Entwurf</b>        | <b>4</b>  |
| 2.1      | Server . . . . .                    | 4         |
| 2.1.1    | Servlet . . . . .                   | 4         |
| 2.1.2    | Algorithmus . . . . .               | 5         |
| 2.2      | Model . . . . .                     | 5         |
| 2.3      | Database Management . . . . .       | 5         |
| 2.4      | Client . . . . .                    | 6         |
| 2.4.1    | View . . . . .                      | 6         |
| 2.4.2    | Controller . . . . .                | 7         |
| <b>3</b> | <b>Kriterien</b>                    | <b>8</b>  |
| 3.1      | Erfüllte Kriterien . . . . .        | 8         |
| 3.1.1    | Kontoverwaltung . . . . .           | 8         |
| 3.1.2    | Gruppenverwaltung . . . . .         | 8         |
| 3.1.3    | Terminverwaltung . . . . .          | 9         |
| 3.1.4    | Terminablauf . . . . .              | 9         |
| 3.1.5    | Qualitative Anforderungen . . . . . | 9         |
| 3.2      | Nicht erfüllte Kriterien . . . . .  | 10        |
| 3.2.1    | Kontoverwaltung . . . . .           | 10        |
| 3.2.2    | Gruppenverwaltung . . . . .         | 10        |
| 3.2.3    | Terminverwaltung . . . . .          | 10        |
| 3.2.4    | Terminablauf . . . . .              | 10        |
| <b>4</b> | <b>Implementierungsplan</b>         | <b>11</b> |
| <b>5</b> | <b>Unit-Tests</b>                   | <b>12</b> |
| 5.1      | Server . . . . .                    | 12        |
| 5.1.1    | Datenbank . . . . .                 | 12        |
| 5.1.2    | Algorithmus . . . . .               | 16        |
| 5.1.3    | Servlet . . . . .                   | 16        |
| 5.2      | Client . . . . .                    | 19        |
| 5.2.1    | HTTPConnection . . . . .            | 19        |
| 5.2.2    | View . . . . .                      | 19        |
| 5.2.3    | Controller . . . . .                | 19        |

# 1 Einleitung

Der folgende Bericht beschreibt den Ablauf der Implementierungsphase der goApp, sowie Änderungen im Vergleich zum Pflichtenheft und dem Entwurf. Die im Entwurf festgelegten Klassen wurden zum Großteil so umgesetzt wie im Entwurfsdokument beschrieben, wobei einzelne Änderungen im nächsten Abschnitt beschrieben werden. Dabei ist Anzumerken, dass das Programmgerüst anhand der entworfenen UML-Diagramme erzeugt wurde. Die im Pflichtenheft festgelegten Kriterien wurden bis auf einige Wunschkriterien umgesetzt, worauf im Abschnitt "Kriterien" näher eingegangen wird.

## 2 Änderungen am Entwurf

Während der Implementierungsphase ergaben sich einige notwendige Änderungen gegenüber dem Entwurf. Diese sind im folgenden aufgeführt.

### 2.1 Server

#### 2.1.1 Servlet

**2.1.1.1 GoServlet** Das GoServlet wird nicht mehr gebraucht und wird gelöscht. Die Funktionalität dieser Klasse wurde in das ParticipateServlet, bzw. EventServlet integriert.

**2.1.1.2 EventServlet** Die Methode `getEvent` wurde durch `getParticipant` ersetzt. Im momentanen Zustand braucht unsere App keine eigene Methode `getEvents` über die `eventId`, da die Events über die Gruppe gelesen werden. Es fehlen nur die Informationen über die dazugehörigen Teilnehmer, weswegen dafür eine eigene Methode benötigt wird.

**2.1.1.3 GroupServlet** Aus `getGroups` wurde `getMembers`, da die Gruppe schon beim Erstellen auf den Client geholt wird und somit nur die Mitglieder der Gruppe gebraucht werden.

**2.1.1.4 LocationServlet** Die beiden Methoden `setGPS`, sowie `setCluster` werden zusammen über `syncLocation` aufgerufen.

**2.1.1.5 Participate Servlet** Die beiden Methoden `accept` und `reject` wurden durch die Methode `setStatus` ersetzt. Diese setzt den Status im Event-User Management, wodurch die Redundanz von zwei Methoden vermieden wird. Falls der Status `Reject` ist, wird der Teilnehmer gelöscht.

**2.1.1.6 UserServlet** Die Nutzer werden an den Client beim Einloggen bzw. Registrieren übermittelt und bis zum nächsten Einloggen gespeichert. Deswegen ist die Methode `getUser` überflüssig geworden.

**2.1.1.7 ServletUtils** Beim Implementieren der Servlets gibt es einige Schritte, die von jedem Servlet durchgeführt werden müssen. Dazu gehören zum Beispiel Methoden, um JSONObjekte aus der HTTP Anfrage auszulesen, oder das Verfassen der Antwort des Servers. Deswegen haben wir uns dazu entschieden eine statische Hilfsklasse zu nutzen, um diesen redundanten Code zu vermeiden. Außerdem definieren JSONObjekte eine wichtige Schnittstelle zum Client, da das Senden & Empfangen von Informationen durch diese geregelt wird. In unserem Fall bleiben Änderungen der Schnittstelle somit lokal.

**2.1.1.8 JSONParameter** Wir brauchen viel mehr Parameter um die JSONObjekte zu erstellen, als in der Entwurfsphase angenommen. Außerdem gibt es ein weiteres Enum für die vom Client aufrufbaren Methoden, sowie eines für Fehlercodes. Die JSONParameter sind auf dem Client, sowie Server gleich.

## 2.1.2 Algorithmus

**2.1.2.1 Clusteralgorithmus** Die Clusterfassade bietet nun neben der Methode getClusteredPoints auch die Methode getClusters und getCenter um direkt auf den Clusterer bzw. den Mittelpunktalgorithmus zugreifen zu können. Außerdem kann nun über den Konstruktor ein beliebiger Clusterer und Mittelpunktalgorithmus übergeben werden.

## 2.2 Model

**2.2.0.1 User** Zusätzlich zu der id des Users in der Datenbank, die Google-Id als String.

**2.2.0.2 Group** Konstruktor hat nun zusätzlich den Gründer (User) als Parameter.

**2.2.0.3 Event** Konstruktor hat nun zusätzlich den Ort (Location), den Ersteller (User) und die Gruppe (Group) in welcher der Termin erstellt wird als Parameter. Die Zeit eines Events wird nicht als Time sondern als Timestamp gespeichert, dann ist das Datum mit dabei.

**2.2.0.4 Participant & Status** Der Status in Participant ist zu einem Enum geworden. Der Status kann Invited, Participate, Started oder Rejected sein.

## 2.3 Database Management

**2.3.0.1 EventDeletionTimer** Läuft im Hintergrund und löscht Events nachdem deren Startzeitpunkt eine Stunde in der Vergangenheit liegt.

**2.3.0.2 LocationDeletionTimer** Läuft im Hintergrund und löscht veraltete Locations von Usern.

**2.3.0.3 GroupManagement** Die Methoden getGroupsByMember, addMember und deleteMember wurden durch Methoden getGroups, add und delete in GroupUserManagement ersetzt.

**2.3.0.4 UserManagement** Die Methode getUserByGoogleId kam hinzu um beim Starten der App den User der App zu laden.

#### 2.3.0.5 EventManagement

- Die Methode `updateStatus` wurde durch die Methode `updateStatus` in `EventUserManagement` ersetzt.
- Die Methoden `getUserLocations` und `setClusterPoints` wurden hinzugefügt um dem Algorithmuspaket den Zugriff auf die Daten zu erleichtern.
- Die Methode `deleteOldEvents` wurde hinzugefügt damit der `EventDeletionTimer` alte Events löschen kann.

#### 2.3.0.6 EventUserManagement

- Die Methode `getEventsByStatus` wurde hinzugefügt damit die App die Events in getrennten Listen erhält.
- Die Methoden `getParticipant` and `getParticipants` wurden hinzugefügt.

**2.3.0.7 RequestManagement** Die Methode `getRequest` wurde hinzugefügt. Diese gibt die Request aus der Datenbank mit der übergebenen `userId` und `groupId` zurück.

### 2.4 Client

#### 2.4.1 View

Die GUI die der User zu Gesicht bekommt weist bis auf die Farbgebung kaum Unterschiede zum GUI Design aus der Entwurfsphase auf und an der Navigation zwischen den einzelnen Activities hat sich nichts geändert. Im folgenden findet sich für jede Activity Änderungen welche Services von ihr gestartet werden können.

##### 2.4.1.1 StartActivity

Zusätzliche Service Aufrufe:

- `RequestService`

Wegfallende Service Aufrufe:

- `GroupService`

##### 2.4.1.2 GroupActivity

Zusätzliche Service Aufrufe:

- `GroupService`

Wegfallende Service Aufrufe:

- `EventService`
- `GoService`

### 2.4.1.3 EventActivity

Zusätzliche Service Aufrufe:

- EventService

### 2.4.2 Controller

Im Entwurf haben wir unsere Services wie folgt beschrieben:

Sie alle erben von der Klasse IntentService, weshalb wir zusätzlich zum Konstruktor nur eine public Methode für jeden Service implementieren und zwar die Methode onHandleIntent(Intent intent). Weil diese Methode bei all unseren Services nur je nach Intent entscheidet, welche private Methode aufgerufen wird und das Ergebnis weiterleitet, haben wir uns dazu entschieden im folgenden nicht für jeden Service auf die besagte Methode einzugehen sondern jeweils die Methoden zu beschreiben welche durch onHandleIntent(Intent intent) aufgerufen werden, da dies unserer Meinung nach eine bessere Definition der Services Klassen liefert und damit für bessere Verständlichkeit sorgt.

Zwar existieren die Methoden die wir daraufhin beschrieben haben nach wie vor(bis auf Änderungen die ebenfalls in diesem Kapitel erläutert werden). Allerdings sind die meisten der Methoden nicht mehr private sondern public. Bei den Services, wo dies der Fall ist werden besagte Methoden von den Activities direkt aufgerufen und erstellen den Intent, mit welchem erst im nächsten Schritt OnHandleIntent(Intent intent) des Services und damit ein neuer Thread gestartet wird. Die Funktionalität und der Rückgabewert der Services nach außen bleibt genau gleich, aber für die Activities ändert sich die Schnittstelle wie entsprechende Services aufgerufen werden. So ist es jetzt schon bei Betrachtung des Codes einer Activity durch den expliziten Aufruf einer standardisierten Methode zum erstellen eines speziellen Intents klar, welche Daten der Client an dieser Stelle mit dem Server austauschen soll.

Da die Servlets wie oben beschrieben verändert wurden mussten selbstverständlich auch die Services angepasst werden, da die Services des Clienten das Gegenstück zu den Servlets bilden wenn es um die Kommunikation zwischen Server und Clienten geht. Da eine Veränderung des einen Teils zwangsläufig eine äquivalente Veränderung des Gegenstücks hervorruft wird hier nicht auf weitere Änderungen der Services eingegangen da diese aus Kapitel 2.1.1 hervorgehen.

**2.4.2.1 UtilService** Die Klasse UtilService vereinigt Methoden die von verschiedene Services immer wieder aufgerufen werden müssen um JSON Parameter auszulesen. Um redundanten Code zu vermeiden haben wir deshalb auch auf dem Clienten diese statische Hilfsklasse eingeführt.

## 3 Kriterien

Im Pflichtenheft wurden Kriterien für unsere App definiert. Im folgenden unterteilen wir diese in erfüllt bzw. nicht erfüllt.

### 3.1 Erfüllte Kriterien

Alle Muss-Kriterien wurden erfüllt und außerdem noch einige von unseren Wunsch-Kriterien.

#### 3.1.1 Kontoverwaltung

FA10 Anmeldung eines Benutzers in der App über Google Services

FA20 Ersterfassung und Änderung des Benutzernamens

#### 3.1.2 Gruppenverwaltung

FA30 Jeder Benutzer kann eine Gruppe erstellen

FA35 Ersterfassung und spätere Änderung des Gruppennamens durch den Gruppengründer

FA40 Gruppen können über den Gruppennamen gesucht werden

FA45 Benutzer können Beitrittsanfragen an eine Gruppe senden

FA50 Der Gruppengründer kann Beitrittsanfragen der Gruppe verwalten:

- Anfragen bestätigen, wodurch der Anfragende ein Mitglied der Gruppe wird und die Anfrage gelöscht wird
- Anfragen ablehnen, wodurch die Anfrage gelöscht wird
- Anfragen ignorieren, wodurch die Anfrage bestehen und sichtbar bleibt

FA60 Der Gruppengründer kann Mitglieder aus der Gruppe entfernen

FA70 Die Gruppe kann durch den Gruppengründer gelöscht werden

FA80 Mitglieder der Gruppen, ausgenommen des Gruppengründers, können die Gruppe verlassen

FA90 Gruppenmitglieder können sich andere Gruppenmitglieder anzeigen lassen



### **3.1.3 Terminverwaltung**

FA100 Jedes Mitglied einer Gruppe kann einen Termin erstellen

- Ersterfassung von Terminnamen, Terminzeit (in der Zukunft) und Terminort
- Visuelle Darstellung des gewählten Terminortes

WFA105 Der Terminersteller kann den Terminnamen, Terminzeit und Terminort nachträglich ändern

FA110 Der Gruppenmittelpunkt wird anhand der Standorte aller Teilnehmer berechnet

FA120 Jeder Termin wird jedem Gruppenmitglied angezeigt

FA130 Jedes Gruppenmitglied kann bei einem Termin zu- oder absagen

### **3.1.4 Terminablauf**

FA140 Jeder Teilnehmer wird 30 Minuten vor Beginn des Termins benachrichtigt

WFA145 Teilnehmer werden auch bei geschlossener App benachrichtigt

FA150 Jeder Teilnehmer kann den Gruppenmittelpunkt ab 15 Minuten vor Beginn des Termins in einer Karte darstellen lassen

WFA155 Vom Gruppenmittelpunkt entfernte Teilgruppen erhalten einen eigenen Standort

WFA160 Teilnehmer können in der App anderen Teilnehmern mitteilen,  
– dass sie bereits unterwegs sind

FA170 Der Termin löscht sich nach der Dauer des Termins (standardmäßig 60 Minuten)

### **3.1.5 Qualitative Anforderungen**

Die Qualitativen Anforderungen beziehen sich auf unser Referenzgerät „Samsung Galaxy S4“(GT-I9505) mit Android 5.0.1

QA10 Die App soll auf jede Anfrage in durchschnittlich unter 5 Sekunden reagieren

QA20 Die App fährt in durchschnittlich unter 10 Sekunden hoch

QA30 Jede Funktion der App ist mit höchstens 5 Eingaben vom Startbildschirm der App zu erreichen

QA50 Unterstützt bis zu 50 Benutzer pro Gruppe

QA60 Jeder Benutzer kann in bis zu 20 Gruppen Mitglied sein

QA70 Der Server unterstützt das Anlegen von mindestens 30000 Benutzern

### **3.2 Nicht erfüllte Kriterien**

Wir haben es nicht geschafft alle Wunsch-Kriterien zu implementieren. Diese können allerdings in zukünftigen Updates der goApp noch erfüllt werden.

#### **3.2.1 Kontoverwaltung**

WFA15 Der Benutzer kann zwischen Sprachen wählen

#### **3.2.2 Gruppenverwaltung**

WFA85 Der Gruppengründer kann seinen Status als Gruppengründer an ein Mitglied übergeben

WFA95 Gruppenmitglieder können die Teilnahmequoten anderer Gruppenmitglieder einsehen

#### **3.2.3 Terminverwaltung**

WFA105 Der Terminersteller kann den Termin löschen

#### **3.2.4 Terminablauf**

WFA160 Teilnehmer können in der App anderen Teilnehmern mitteilen,

- dass sie zu spät kommen
- dass sie schon am Terminort angekommen sind

WFA175 Die Dauer des Termins kann individuell festgelegt werden

## 4 Implementierungsplan

Während der Implementierungsphase haben sich einige Änderungen gegenüber unserem geplanten Verlauf ergeben.

Unsere Woche Puffer am Ende haben wir durch eine Unterschätzung des Gesamtaufwandes doch gebraucht. Insbesondere die Arbeit auf dem Client wurde unterschätzt, da sich aufgrund der direkten Interaktion mit Benutzern viele Randfälle ergeben.

Zusätzlich ergab sich eine große Änderung fast direkt zu Beginn, da wir gemerkt haben, da das HTTPConnectionPackage weniger Arbeit braucht als gedacht und wir somit die Aufgaben Database(Model) und Management ganz Jonas zugeordnet haben und Eva dafür an der Servlets gearbeitet hat.

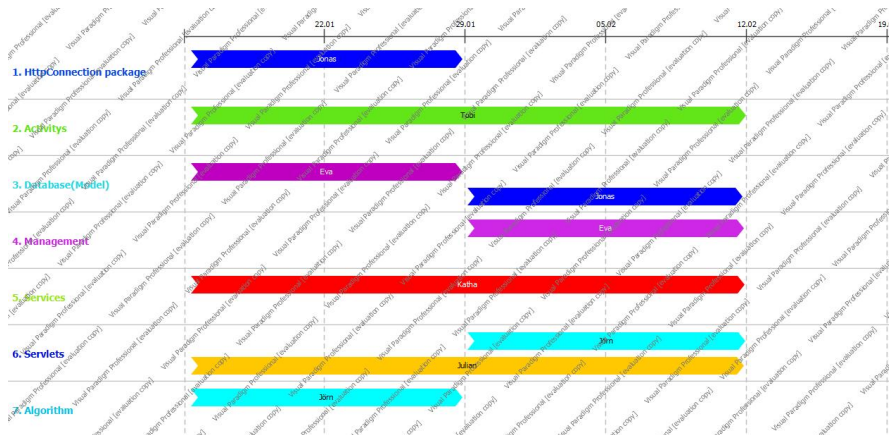


Abbildung 1: Das Implementierungsplandiagramm, vor der Implementierung.

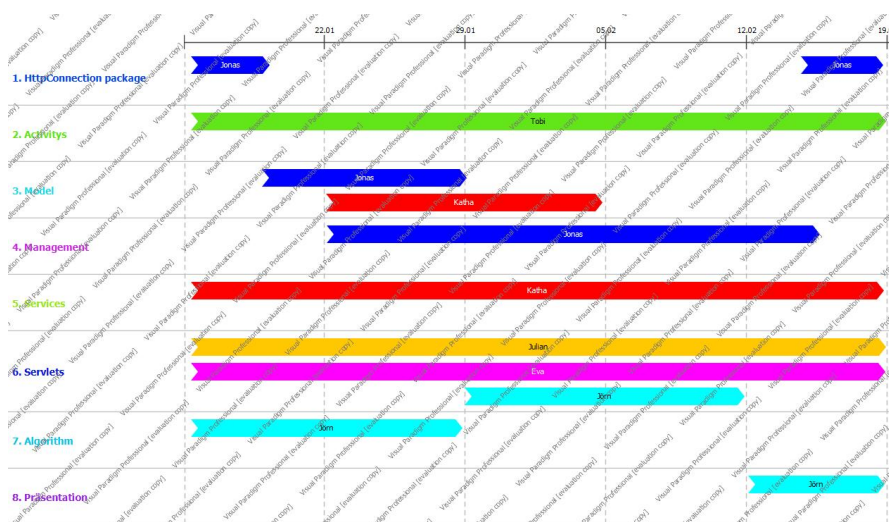


Abbildung 2: Das Implementierungsdiagramm.

## 5 Unit-Tests

Zum Testen unseres Codes haben wir Unit-Tests benutzt. In dieser Phase prüfen wir nur die Funktionalität. Weitere Tests zu Fehlerfällen folgen in der Testphase.

### 5.1 Server

#### 5.1.1 Datenbank

Um die Management Klassen, welche über Hibernate mit der Datenbank kommunizieren, zu testen haben wir auf dem PC eine MySQL-Datenbank erstellt. Jeder Testfall erstellt erst alle Daten, die er zum Testen benötigt, macht dann Datenbankabfragen und löscht die Daten wieder. Dadurch ist der Ausgangszustand für jeden Testfall immer klar definiert.

**5.1.1.1 DatabaseInitializer - DatabaseInitializerTest** Testet ob das initialisieren der Datenbank klappt.

**5.1.1.2 EventDeletionTimer - EventDeletionTimerTest** Die EventDeletionTimerTest Klasse stellt sicher, dass das periodische löschen von veralteten Events funktioniert.

**5.1.1.3 LocationDeletionTimer - LocationDeletionTimerTest** Die LocationDeletionTimerTest Klasse stellt sicher, dass das periodische löschen von veralteten Locations funktioniert.

#### 5.1.1.4 EventManagement - EventManagementTest

**public void testUpdate()** Der Test überprüft, dass die update Methode die Änderungen an einem Event in die Datenbank überträgt.

**public void testUpdateName()** Der Test überprüft, dass die updateName Methode nach der Änderung des Namens eines Events den neuen Namen in die Datenbank abspeichert.

**public void testAdd()** Der Test stellt sicher, dass das hinzufügen eines Events funktioniert.

**public void testDelete()** Der Test stellt sicher, dass das löschen eines Events funktioniert.

**public void testGetCreator()** Der Test stellt sicher, dass beim Aufruf der getCreator Methode der richtige Ersteller zurück gegeben wird.

**public void testGetEvent()** Der Test stellt sicher, dass beim Aufruf von getEvent das richtige Event zurück gegeben wird.

**public void testGetUserLocations()** Der Test stellt sicher, dass beim Aufruf der `getGetUserLocations` Methode alle Positionen der Eventteilnehmer zurück gegeben werden.

**public void testSetClusterPoints()** Der Test stellt sicher, dass beim Aufruf der `setClusterPoints` Methode die neuen Positionen korrekt in der Datenbank abgespeichert werden.

#### 5.1.1.5 EventUserManagement - EventUserManagementTest

**public void testAddUser()** Der Test stellt sicher, dass das hinzufügen eines Users zu einem Event funktioniert.

**public void testUpdateStatus()** Der Test überprüft, dass die `update` Methode die Änderungen in die Datenbank überträgt.

**public void testGetParticipant()** Der Test stellt sicher, dass beim Aufruf von `getParticipant` der richtige Participant zurück gegeben wird.

**public void testGetParticipants()** Der Test stellt sicher, dass beim Aufruf von `getParticipants` der richtigen Participants zurück gegeben werden.

**public void testDeleteParamsEventIdUserId()** Der Test stellt sicher, dass das löschen eines Participants anhand der id des Users und des Events funktioniert.

**public void testDeleteParamsParticipantId()** Der Test stellt sicher, dass das löschen eines Participants anhand der id des Participants funktioniert.

**public void testGetUsers()** Der Test stellt sicher, dass beim Aufruf von `getUsers` die richtigen User zu einem Event zurück gegeben werden.

**public void testGetUserByStatus()** Der Test stellt sicher, dass beim Aufruf von `getUserByStatus` alle User mit dem Status bei dem Event zurück gegeben werden.

**public void testGetEventsByStatus()** Der Test stellt sicher, dass beim Aufruf von `getEventsByStatus` alle Events mit dem Status von einem User zurück gegeben werden.

**public void testGetEvents()** Der Test stellt sicher, dass beim Aufruf von `getEvents` die richtigen Events zu einem User zurück gegeben werden.

#### 5.1.1.6 GroupManagement - GroupManagementTest

**public void testAdd()** Der Test stellt sicher, dass das hinzufügen einer Gruppe in der Datenbank funktioniert.

**public void testGetGroup()** Der Test stellt sicher, dass beim Aufruf von `getGroup` die richtige Gruppe zurück gegeben wird.

**public void testDelete()** Der Test stellt sicher, dass das löschen einer Gruppe funktioniert.

**public void testGetEvents()** Der Test stellt sicher, dass beim Aufruf von `getEvents` alle Events der Gruppe zurück gegeben werden.

**public void testUpdate()** Der Test überprüft, dass die `update` Methode die Änderungen in die Datenbank überträgt.

**public void testGetGroupsByName()** Der Test stellt sicher, dass die Methode `getGroupsByName` alle Gruppen aus der Datenbank deren Namen den übergebenen Namen enthält zurück gibt.

**public void testUpdateFounder()** Der Test überprüft, dass die `updateFounder` Methode den neuen Gründer in die Datenbank überträgt.

**public void testUpdateName()** Der Test überprüft, dass die `updateName` Methode den neuen Namen in die Datenbank überträgt.

#### 5.1.1.7 GroupUserManagement - GroupUserManagementTest

**public void testAdd()** Der Test stellt sicher, dass das hinzufügen eines Users zu einer Gruppe funktioniert.

**public void testDelete1()** Der Test stellt sicher, dass man nicht den Gründer der Gruppe aus der Gruppe entfernen kann.

**public void testDelete2()** Der Test stellt sicher, dass das löschen eines Users aus einer Gruppe funktioniert.

**public void testGetGroups()** Der Test stellt sicher, dass die `getGroups` Methode alle Gruppen eines Users zurück gibt.

**public void testGetUsers()** Der Test stellt sicher, dass die `getUsers` Methode alle User einer Gruppe zurück gibt.

#### 5.1.1.8 RequestManagement - RequestManagementTest

**public void testAdd()** Der Test stellt sicher, dass das hinzufügen einer Request in der Datenbank funktioniert.

**public void testDeleteParamsRequestId()** Der Test stellt sicher, dass das löschen einer Request anhand der und der Request funktioniert.

**public void testDeleteParamsGroupIdUserId()** Der Test stellt sicher, dass das löschen einer Request anhand der id des Users und der Gruppe funktioniert.

**public void testGetRequestByGroup()** Der Test stellt sicher, dass die getRequestByGroup Methode alle User, die eine Anfrage an die Gruppe gestellt haben, zurück gibt.

**public void testGetRequestByUser()** Der Test stellt sicher, dass die getRequestByUser Methode alle Gruppen, an die der User eine Anfrage gestellt hat, zurück gibt.

#### 5.1.1.9 UserManagement - UserManagementTest

**public void testAdd()** Der Test stellt sicher, dass das hinzufügen eines Users in der Datenbank funktioniert.

**public void testDelete()** Der Test stellt sicher, dass das löschen eines Users funktioniert.

**public void testGetUser()** Der Test stellt sicher, dass die getUser Methode den User mit der entsprechenden id zurück gibt.

**public void testGetUserByGoogleId()** Der Test stellt sicher, dass die getUser Methode den User mit der entsprechenden googleId zurück gibt.

**public void testUpdateLocation()** Der Test überprüft, dass die updateLocation Methode die neue Position in der Datenbank abspeichert.

**public void testUpdate()** Der Test überprüft, dass die update Methode die Änderungen in die Datenbank überträgt.

**public void testUpdateName()** Der Test überprüft, dass die updateName Methode den neuen Namen in der Datenbank abspeichert.

### 5.1.2 Algorithmus

Zum Einen wird hier überprüft ob der selbstgeschriebene Mittelpunktalgorithmus das korrekte Ergebnis ausgibt und auch Nullparameter richtig verarbeitet, zum Anderen wird die Klasse ClusterFacade durch Benutzung von Mockito auf die richtige Arbeitsweise überprüft, dabei wird die Richtigkeit der importierten Clusteralgorithmen vorausgesetzt.

### 5.1.3 Servlet

Zum Testen der Servlets werden die jeweiligen Management Klassen der Datenbank, sowie HTTP requests gemockt.

#### 5.1.3.1 EventServlet

**public void testCreate()** Dieser Test prüft das erfolgreiche erstellen eines Events.

**public void testGetParticipates()** Dieser Test prüft das erfolgreiche Auslesen aller Teilnehmer der übergebenen Events.

**public void testChange()** Dieser Test prüft das erfolgreiche Ändern des Namens des Events.

#### 5.1.3.2 GroupSearchServlet

**public void testMemberSearch()** Überprüft das erfolgreiche Einfordern einer Liste aller Gruppen, in denen ein gegebener Benutzer mitglied ist.

**public void testNameSearch()** Überprüft das erfolgreiche Abrufen einer Liste aller Gruppen, in deren Name die gegebene Suchanfrage enthalten ist.

#### 5.1.3.3 LocationServlet

**public void testSyncPos()** Testet ob die gegebene Position des Benutzers in der Datenbank aktualisiert wird und der geclusterte Gruppenstandort des Events korrekt zurückgegeben wird.

**5.1.3.4 LoginServlet** Da jede der beiden Methoden auf den GoogleServer zum Verifizieren des GoogleTokens zugreift und man dies nicht testen kann, wird dieses Servlet erst manuell über den Client mit einem echten GoogleAccount getestet. Dort funktioniert das Registrieren, sowie auch Anmelden.

#### 5.1.3.5 ParticipateServlet



**public void testSetStatus()** Testet das korrekte setzen des Status' auf "teilnehmen".

#### 5.1.3.6 GroupServlet

**public void testGroupCreating()** In diesem Test wird eine Gruppe erstellt.

**public void testGroupDeleting()** Dieser Test versucht eine Gruppe zu entfernen.

**public void testNameChanges()** Hier wird eine Anfrage vom Client zum Ändern des Namens einer Gruppe simuliert.

**public void testMemberKicking()** Es wird das Entfernen von Mitgliedern aus einer Gruppe getestet.

**public void testEventRequesting()** Diese Methode testet das Abfragen aller zu einer Gruppe und einem Benutzer zugehörigen Events. Hierbei werden die Events, die bereits vom Nutzer bestätigt wurden separat von denen, auf die der Nutzer noch antworten muss, zurückgegeben.

**public void testMemberRequesting()** Es wird das Abrufen einer Liste mit allen Mitgliedern einer Gruppe getestet.

**public void testFounderChanging()** In diesem Test wird der founder einer Gruppe aktualisiert.

#### 5.1.3.7 RequestSearchServlet

**public void testGetRequestsByGroup()** Dieser Tests prüft das erfolgreiche holen aller Gruppen, zu dem von einem gegeben User, eine Anfrage existiert.

**public void testGetRequestsByUser()** Dieser Tests prüft das erfolgreiche holen aller Nutzer, zu dem von einem gegeben Gruppe, eine Anfrage existiert.

#### 5.1.3.8 RequestServlet

**public void testCreate()** Dieser Test prüft das erfolgreiche Anlegen eines neuen Requests.

**public void testAccept()** Dieser Test prüft das erfolgreiche Akzeptieren eines Requests.

**public void testReject()** Dieser Test prüft das erfolgreiche Ablehnen eines Requests.

**5.1.3.9 ServletUtils** In den ServletUtils werden die 3 Methoden `testIsUserAlreadyRegistered()`, `testGetGoogleIdByToken()` und `testGetGoogleNameByToken()` nicht getestet, da diese auf GoogleServer zugreifen und ein gültiges `GoogleToken` benötigen.

**public void testCreateJSONParticipate()** Dieser Test prüft das erfolgreiche Erstellen eines `JSONObjektes` aus einem `Participate` Objekt.

**public void testCreateJSONListPart()** Dieser Test prüft das erfolgreiche Erstellen eines `JSONObjektes`

**public void testCreateJSONEventID()** Dieser Test prüft das erfolgreiche Erstellen eines `JSONObjektes` mit nur der `EventID` aus einem `Event` Objekt.

**public void testCreateJSONEvent()** Dieser Test prüft das erfolgreiche Erstellen eines `JSONObjektes` aus einem `Event` Objekt.

**public void testCreateJSONLocation()** Dieser Test prüft das erfolgreiche Erstellen eines `JSONObjektes` aus einem `Location` Objekt.

**public void testCreateJSONGroup()** Dieser Test prüft das erfolgreiche Erstellen eines `JSONObjektes` aus einem `Group` Objekt.

**public void testCreateJSONUser()** Dieser Test prüft das erfolgreiche Erstellen eines `JSONObjektes` aus einem `User` Objekt.

**public void testCreateJSONListEvent()** Dieser Test prüft das erfolgreiche Erstellen eines `JSONObjektes` aus einem `User` Objekt.

**public void testCreateJSONDoubleListEvent()** Dieser Test prüft das erfolgreiche Erstellen eines `JSONObjektes` aus 2 Listen mit `Event` Objekten

**public void testCreateJSONListUsr()** Dieser Test prüft das erfolgreiche Erstellen eines `JSONObjektes` aus einer Liste von `User` Objekten.

**public void testCreateJSONListGrp()** Dieser Test prüft das erfolgreiche Erstellen eines `JSONObjektes` aus einer Liste von `Group` Objekten.

**public void testCreateJSONGroupID()** Dieser Test prüft das erfolgreiche Erstellen eines `JSONObjektes` mit nur der `GroupID` aus einem `Group` Objekt.

**public void testCreateJSONListLoc()** Dieser Test prüft das erfolgreiche Erstellen eines `JSONObjektes` aus einer Liste von `Location` Objekten.

**public void testCreateJSONError()** Dieser Test prüft das erfolgreiche Erstellen eines JSONObjektes aus einem ErrorCodes Objekt.

#### 5.1.3.10 UserServlet

**public void testChangeName()** Dieser Test überprüft das erfolgreiche Ändern des Namens des Nutzers.

## 5.2 Client

### 5.2.1 HTTPConnection

**public void testSendGetRequest()** Der Test überprüft, dass die get-Anfragen an den Server funktionieren.

**public void testSendPostRequest()** Der Test überprüft, dass die post-Anfragen an den Server funktionieren.

**public void testConnectionFailed()** Der Test überprüft, dass wenn ein Service nicht erreichbar ist ConnectionFailed zurück gegeben wird.

### 5.2.2 View

Für die View haben wir keine Testfälle implementiert da diese sich besser "von Hand" testen lässt. Durch einfaches benutzen der goApp können wir überprüfen, dass die View sich so verhält wie wir als Implementierer es erwarten. Auch in der nächsten Phase werden wir die View nicht mit klassischen Unit-Tests abdecken, sondern durch Monkey-Tests die Robustheit unserer Applikation bei Eingaben jeglicher Art sicherstellen. Ebenfalls wird die goApp an die ersten User verteilt um uns Feedback über die hoffentlich intuitive Bedienbarkeit der GUI einzuholen.

### 5.2.3 Controller

Es ist nicht möglich Intent Services mit Unit-Tests zu testen. Da unser Controller wie im Entwurf beschrieben nur Intent Services implementiert existieren hier also keine Unit-Tests. Allerdings ist der Großteil der Logik in die Klasse UtilService ausgelagert.

#### 5.2.3.1 UtilService

**public void getGroups()** Dieser Test überprüft das erfolgreiche erstellen eines Group Arrays aus einem JSON Objekt.

**public void getUsers()** Dieser Test überprüft das erfolgreiche erstellen eines User Arrays aus einem JSON Objekt.

**public void getParticipants()** Dieser Test überprüft das erfolgreiche erstellen eines User Arrays, aus einem JSON Objekt wobei hier jeder User einen Status hat.

**public void getLocations()** Dieser Test überprüft das erfolgreiche erstellen eines Location Arrays aus einem JSON Objekt.

**public void isErrorTrue()** Dieser Test überprüft das erkennen von Fehlern.

**public void isErrorFalse()** Dieser Test überprüft das erkennen einer Fehlerfreien Kommunikation mit dem Server.

**public void getError()** Dieser Test überprüft ob ein Error Code als der richtige erkannt wird.

**public void getEvents()** Dieser Test überprüft das erfolgreiche erstellen eines Event Arrays aus einem JSON Objekt.