

# goApp Qualitätssicherung

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

10. März 2017

# Inhaltsverzeichnis

<b>1</b>	<b>Einleitung</b>	<b>3</b>
<b>2</b>	<b>Server</b>	<b>4</b>
2.1	Änderungen . . . . .	4
2.1.1	Algorithmus . . . . .	4
2.1.2	Datenbank . . . . .	4
2.1.3	Servlet . . . . .	4
2.1.3.1	RequestServlet . . . . .	4
2.1.3.2	UserServlet . . . . .	5
2.1.3.3	EventServlet . . . . .	5
2.2	Tests . . . . .	6
2.2.1	Algorithmus . . . . .	6
2.2.2	Datenbank . . . . .	6
2.2.2.1	EventManagerment . . . . .	6
2.2.2.2	LocationDeletionTimer . . . . .	6
2.2.3	Servlet . . . . .	6
2.2.3.1	RequestServlet . . . . .	6
2.2.3.2	RequestSearchServlet . . . . .	7
2.2.3.3	UserServlet . . . . .	8
2.2.3.4	EventServlet . . . . .	9
2.2.4	Abdeckung . . . . .	9
<b>3</b>	<b>Client</b>	<b>10</b>
3.1	Lint . . . . .	10
3.2	Performance . . . . .	10
3.2.1	Monkey-Test . . . . .	10
3.2.2	Beta Tester . . . . .	10
<b>4</b>	<b>Testszenarien</b>	<b>12</b>
4.0.1	Kontoverwaltung . . . . .	12
4.0.2	Gruppenverwaltung . . . . .	12
4.0.3	Terminverwaltung . . . . .	13
4.0.4	Terminablauf . . . . .	13
4.0.5	Standort . . . . .	14
<b>5</b>	<b>Anhang</b>	<b>15</b>

# 1 Einleitung

## 2 Server

Während der Qualitätssicherung bemerkten wir durch unsere Tests noch einige kleine Fehler im Code. Diese wurden behoben und sind im folgenden zusammen mit kleineren Optimierungen aufgeführt.

### 2.1 Änderungen

#### 2.1.1 Algorithmus

#### 2.1.2 Datenbank

Es gab einen Fehler im Zusammenhang mit dem LocationDeletionTimer und dem Event-DeletionTimer, welche sich um das Löschen alter Events und alter Locations kümmern.

- Fehlersymptom: Mehrere der Timer liefen gleichzeitig. Und es gab Timer mit zum Teil veralteten Einstellungen (Dauer bis zum löschen).
- Fehlergrund: Alte Timer wurden niemals beendet und liefen auch nach erneutem Hochladen weiter. Dadurch kam bei jedem hochladen auf den Server ein weiterer Timer hinzu.
- Fehlerbehebung: Wir haben eine neue Klasse die das Interface ServletContextListener implementiert erstellt. Das Interface deklariert die zwei Methoden contextInitialized und contextDestroyed. Diese werden aufgerufen, wenn die Web-App gestartet bzw. beendet wird. Anstatt beim erstellen der SessionFactory werden die zwei Timer deshalb jetzt in dieser Klasse gestartet und dort auch wieder beendet.

Zudem zeigten neue Assertions in den Tests, dass die Parameter nicht immer auf null überprüft wurden. Zum Beispiel in der Klasse GroupManagement bei der Methode public boolean update(Group chGroup). An diesen Stellen wurden die entsprechenden Überprüfungen ergänzt.

Zusätzlich zu dem Beheben von Fehlern wurden noch folgende Sachen verändert:

- Der bei der Datenbank voreingestellte Connection Pool, welcher nur zum Testen verwendet werden sollte, wurde durch den c3p0 Pool ersetzt.
- Die Termine werden nun immer aufsteigend nach Datum sortiert zurückgegeben, damit sie auf dem Client nicht immer in einer anderen Reihenfolge erscheinen und man die aktuellsten Termine als erstes sieht.
- Gruppen und User werden nun immer aufsteigend nach dem Namen zurückgegeben.

#### 2.1.3 Servlet

**2.1.3.1 RequestServlet** Beim Ausführen der Methode create kam es zu einer NullPointerException, wenn der Nutzer in keinen Gruppen Mitglied war oder noch keine Anfragen gesendet hatte.

**2.1.3.2 UserServlet** In der Methode `changeName` trat eine `NullPointerException` auf, wenn der Nutzer nicht in der Datenbank gefunden wurde, da der Name auf dem zurückgegebenen Nutzer geändert wurde.

**2.1.3.3 EventServlet** In der Methode `change` hat eine Rückgabe des Fehlercodes gefehlt und es wurde trotz Fehlerfall weiter ausgeführt, sowie der Code `OK` zurück gegeben. Außerdem war der Umgang mit dem `Timestamp` falsch.

## 2.2 Tests

Zu den Unit-Tests aus der Implementierungsphase sind weitere Tests dazugekommen um die Abdeckung zu erhöhen, sowie Randfälle zu testen.

### 2.2.1 Algorithmus

### 2.2.2 Datenbank

Bei den Tests für die Datenbank kamen nur zwei neue Testmethoden hinzu. Es wurden aber die bestehenden um Assertions ergänzt, welche Randfälle wie eine falsche id oder einen Parameter bei dem null übergeben wird überprüfen.

#### 2.2.2.1 EventManagement

**public void testDeleteOldEvents()** Der Test überprüft, ob das löschen alter Events aus der Datenbank funktioniert.

#### 2.2.2.2 LocationDeletionTimer

**public void testTimerStarter()** Der Test stellt sicher, dass die Methoden contextInitialized und contextDestroyed des ServletContextListeners den LocationDeletionTimer korrekt starten und beenden.

### 2.2.3 Servlet

#### 2.2.3.1 RequestServlet

**public void testCreateUserIsMember()** Dieser Test überprüft das Verhalten des Servlets, wenn der Nutzer schon Mitglied in der angefragten Gruppe ist. Erwartet wird ein Interact Fehler.

**public void testCreateUserHasRequest()** Dieser Test überprüft das Verhalten des Servlets, wenn der Nutzer schon eine offene Anfrage an die Gruppe hat. Erwartet wird ein Interact Fehler.

**public void testCreateUserLimitReached()** Es wird überprüft ob ein USR\_LIMIT Fehler gesendet wird, wenn der Nutzer schon in zu vielen Gruppen Mitglied ist.

**public void testCreateGroupLimitReached()** Es wird überprüft ob ein GRP\_LIMIT Fehler gesendet wird, wenn die Gruppe ihr Mitglieder Limit schon erreicht hat.

**public void testCreateDatabaseAddFalse()** Dieser Test überprüft das Verhalten, wenn die Datenbank beim Hinzufügen der Gruppe und des Nutzers zur Datenbank false zurück gibt. Erwartet wird ein DB\_ERROR.

**public void testCreateDatabaseReqByGroupNull()** Es soll kein Fehler auftreten, wenn die Datenbank bei den Anfragen zu einer bestimmten Gruppe null zurück gibt, weil momentan keine existieren.

**public void testCreateDatabaseReqByUserNull()** Es soll kein Fehler auftreten, wenn die Datenbank bei den Anfragen zu einem bestimmten Nutzer null zurück gibt, weil momentan keine existieren.

**public void testCreateDatabaseGetUserNull()** Es soll ein Fehler auftreten, wenn der Nutzer nicht in der Datenbank existiert.

**public void testCreateDatabaseGetGroupsNull()** Es soll ein Fehler auftreten, wenn die Gruppe nicht in der Datenbank existiert.

**public void testCreateWOutJSON()** Die Methode create wird ohne alle erforderlichen JSON Parameter aufgerufen. Es soll ein READ\_JSON Fehler zurückgegeben werden.

**public void testAcceptDatabaseAddFalse()** Es soll ein DB\_ERROR entstehen, wenn die Datenbank beim Hinzufügen einer Gruppen/Nutzer Kombination false zurückgibt.

**public void testRejectDatabaseDeleteFalse()** Es soll ein DB\_ERROR entstehen, wenn die Datenbank beim Löschen einer Gruppen/Nutzer Kombination false zurückgibt.

**public void testRejectDatabaseRequestNull()** Es soll ein DB\_ERROR entstehen, wenn die Datenbank keine Anfrage vom Nutzer an die Gruppe findet.

**public void testAcceptWOutJSON()** Die Methode accept wird ohne alle erforderlichen JSON Parameter aufgerufen. Es soll ein READ\_JSON Fehler zurückgegeben werden.

**public void testRejectWOutJSON()** Die Methode reject wird ohne alle erforderlichen JSON Parameter aufgerufen. Es soll ein READ\_JSON Fehler zurückgegeben werden.

**public void testMethodNotExisting()** Wenn eine Methode aufgerufen wird, die in dem Servlet nicht existiert, soll ein METH\_ERROR zurück gegeben werden.

**public void testEmptyJSON()** Wenn ein leerer JSON String übergeben wird, soll der Fehler READ\_JSON entstehen.

#### 2.2.3.2 RequestSearchServlet

**public void testGetRequestsByUserDatabaseNull()** Wenn keine Anfragen von einem bestimmten Nutzer existieren gibt die Datenabak null zurück. Das Ergebnis der Methode soll ein Fehlercode EMPTY\_LIST sein.

**public void testGetRequestsByUserWOutJSON()** Die Methode getRequestsByUser wird ohne alle erforderlichen JSON Parameter aufgerufen. Es soll ein READ\_JSON Fehler zurückgegeben werden.

**public void testGetRequestsByGroupDatabaseNull()** Wenn keine Anfragen an eine bestimmte Gruppe existieren gibt die Datenabank null zurück. Das Ergebnis der Methode soll ein Fehlercode EMPTY\_LIST sein.

**public void testGetRequestsByGroupWOutJSON()** Die Methode getRequestsByGroup wird ohne alle erforderlichen JSON Parameter aufgerufen. Es soll ein READ\_JSON Fehler zurückgegeben werden.

**public void testMethodNotExisting()** Wenn eine Methode aufgerufen wird, die in dem Servlet nicht existiert, soll ein METH\_ERROR zurück gegeben werden.

**public void testEmptyJSON()** Wenn ein leerer JSON String übergeben wird, soll der Fehler READ\_JSON entstehen.

#### 2.2.3.3 UserServlet

**public void testMethodNotExisting()** Wenn eine Methode aufgerufen wird, die in dem Servlet nicht existiert, soll ein METH\_ERROR zurück gegeben werden.

**public void testEmptyJSON()** Wenn ein leerer JSON String übergeben wird, soll der Fehler READ\_JSON entstehen.

**public void testChangeNameDatabseNullGetUser()** Es soll ein DB\_ERROR zurückgegeben werden, wenn die Datenbank beim Aufruf der Methode getUser null zurückgibt (d.h. der Nutzer wurde nicht gefunden).

**public void testChangeNameDatabseFalseUpdate()** Es soll ein DB\_ERROR zurückgegeben werden, wenn die Datenbank beim Aufruf der Methode update vom User false zurückgibt.

**public void testChangeNameWOutJSON()** Die Methode changeName wird ohne alle erforderlichen JSON Parameter aufgerufen. Es soll ein READ\_JSON Fehler zurückgegeben werden.



#### 2.2.3.4 EventServlet

**public void testMethodNotExisting()** Wenn eine Methode aufgerufen wird, die in dem Servlet nicht existiert, soll ein METH\_ERROR zurück gegeben werden.

**public void testEmptyJSON()** Wenn ein leerer JSON String übergeben wird, soll der Fehler READ\_JSON entstehen.

**public void testCreateDatabaseNull()** Es wird ein DB\_ERROR erwartet, wenn die Datenbank das Event nicht anlegen konnte und deswegen null zurückgibt.

**public void testCreateWOutJSON()** Die Methode create wird ohne alle erforderlichen JSON Parameter aufgerufen. Es soll ein READ\_JSON Fehler zurückgegeben werden.

**public void testGetParticipatesDatabaseNull()** Es wird ein EMPTY\_LIST Fehler erwartet, wenn die Datenbank keinen Nutzer beim Event mit der übergebenen EventID gefunden hat und deswegen null zurückgibt.

**public void testGetParticipatesWOutJSON()** Die Methode getParticipates wird ohne alle erforderlichen JSON Parameter aufgerufen. Es soll ein READ\_JSON Fehler zurückgegeben werden.

**public void testChangeWOutEventId()** Wenn der JSONParameter eventId bei der Methode change fehlt, soll der Fehler READ\_JSON entstehen.

**public void testChangeDatabaseWrong()** Wenn die Datenbank das Event nicht updaten konnte und deswegen false zurück gibt, soll ein DB\_ERROR entstehen.

#### 2.2.4 Abdeckung

hier am Ende EcEmma beschreiben und %te pro Datei oder Paket? angeben

## 3 Client

### 3.1 Lint

Lint ist ein Programm zur statischen Code-Analyse der Quelltexte von Computerprogrammen und in Android Studio integriert. Lint überprüft Android-Projekten auf mögliche Bugs und liefert Vorschläge um den Code zu verbessern. Wir haben unseren Client Code mehrmals mit Lint überprüft und dadurch immer weiter optimiert, von Performance Issues über Rechtschreibfehler bis hin zur Verbesserung der XML-Dateien.

### 3.2 Performance

#### 3.2.1 Monkey-Test

#### 3.2.2 Beta Tester

Wir haben unsere App unter circa xx Beta Usern verteilt um sie unter realen Bedingungen durch Personen testen zu lassen welche nicht an der Entwicklung beteiligt waren und um Verbesserungsvorschläge an der GUI zu sammeln und umzusetzen.

Einige Beispiele:

- User bekommen jetzt bei allen „kritischen“ Aktionen wie dem Löschen einer Gruppe einen Dialog angezeigt ob sie diese Aktion wirklich durchführen wollen.
- Alle Texte der goApp sind jetzt eingerückt und kleben nicht mehr am Bildschirmrand.
- Die Reihenfolge der Buttons in der Toolbar der StartActivity wurde geändert da sie so intuitiver zu bedienen sind.
- In der NewGroupActivity erkennt ein User jetzt durch einen Cursor ob er sich wirklich im Suchfeld befindet, außerdem wird jetzt mit einem betätigen der Enter Taste ebenfalls eine Suche gestartet.

Ebenfalls waren wir an eventuellem Fehlverhalten oder gar Abstürzen der Applikation im „alltagsgebrauch“ interessiert und haben auch hier Feedback gesammelt um Bugs zu beheben.

- Problem: Teilweise stürzte die goApp ab wenn der User eine Gruppe löscht.
  - Fehler: Nachdem ein User eine Gruppe löscht soll er auf die StartActivity gelangen, weil seine Gruppe nun nicht mehr existiert. Um eine Activity zu starten muss ein Intent mit dem Context erstellt werden durch den die Activity gestartet werden soll. Die StartActivity wird hier aus dem ResultReceiver eines Fragments gestartet, hier nutzten wir die Methode getActivity() um einen Kontext zu bekommen. Diese Methode lieferte meist auch die GroupInfoActivity, manchmal wurde aber auch ein null Objekt zurückgegeben.

- Lösung: Der ResultReceiver erhält jetzt die GroupInfoActivity als Parameter für den Konstruktor und speichert sich diese, so kann er mit diesem Kontext immer eine andere Activity starten.
- Problem: Wenn ein User erfolgreich eine Gruppe gelöscht hat oder aus ihr ausgetreten ist konnte mit dem Zurückbutton wieder in die GroupInfoActivity der entsprechenden Gruppe zurückgekehrt werden.
  - Fehler: Die GroupInfoActivity war noch auf dem back stack der zuletzt geöffneten Activities abgelegt.
  - Lösung: Wenn die StartActivity gestartet wird wird der komplette back stack gelöscht.
- Problem: Keine Zuordnung der Gruppen Standorte zu den richtigen Events.
  - Fehler: Wenn ein User an (mindestens) zwei Events zur gleichen Zeit teilgenommen hat konnten die vom Server empfangenen Gruppen Standorte nicht den richtigen Events zu geordnet werden und es wurden die Karten in allen EventActivities geupdatet.
  - Lösung: Wir schicken nun bei dem Intent der die Karte updatet zusätzlich das zugehörige Event mit um sicherzustellen dass nur die Karte des richtigen Events geupdatet wird.

Im Anhang findet sich zusätzlich ein Fragebogen an die Beta User.

## 4 Testszzenarien

Im folgenden wollen wir kurz auf unsere Testszzenarien aus dem Pflichtenheft eingehen,. Trotz einer immer weiter voranschreitenden Verbesserung und dadurch teilweisen Abänderung unserer Ideen aus dem Pflichtenheft, konnten alle Testszzenarien die im Pflichtenheft definiert wurden mit positivem Ausgang durchlaufen werden.

Die Tests wurden auf dem im Pflichtenheft genannten Referenzgerät(Samsung Galaxy S4) mit einer ausreichenden Internetverbindung und funktionierendem GPS ausgeführt.

### 4.0.1 Kontoverwaltung

T10 deckt ab [FA10](#)

Registrieren: Eine bisher nicht registrierte Person mit einem Google Account registriert sich mit diesem im System und wählt zusätzlich einen Benutzernamen.

Ergebnis: Der neue Benutzer mit seinem Google Account und Benutzername ist in der Datenbank.

Bemerkung: Dieses TestszENARIO kann so nicht mehr umgesetzt werden da der Benutzername bei der Erstellung eines Accounts nicht mehr gewählt wird sondern auf den GoogleAccount Namen gesetzt wird, der Benutzername lässt sich allerdings immernoch nachträglich ändern.

T20 deckt ab [FA20](#)

Namensänderung: Ein Benutzer ändert seinen Benutzernamen.

Ergebnis: Der neue Benutzername des Benutzers ist in der Datenbank.

### 4.0.2 Gruppenverwaltung

T30 deckt ab [FA30](#), [FA35](#)

Ein beliebiger Benutzer erstellt eine neue Gruppe mit Namen.

Ergebnis: Eine neue Gruppe, mit eindeutiger ID, dem angegebenen Namen und dem Benutzer als Gründer, ist in der Datenbank.

T35 deckt ab [FA35](#)

Der Gründer ändert den Gruppennamen.

Ergebnis: Auf dem Server ist bei der Gruppe der neue Name eingetragen.

T40 deckt ab [FA40](#), [FA45](#)

Suchen der Gruppe nach Namen und eine Beitrittsanfrage an eine gefundene Gruppe stellen.

Ergebnis: Alle Gruppen-IDs der Gruppen, die die Suchanfrage im Namen enthalten. Anfrage in der Gruppe hinzugefügt.

T50 deckt ab [FA50](#)

Der Gruppengründer bestätigt eine Anfrage.

Ergebnis: Der Benutzer der Anfrage ist Mitglied der Gruppe und die Anfrage ist gelöscht.

T55 deckt ab [FA50](#)

Der Gruppengründer lehnt eine Anfrage ab.

Ergebnis: Die Anfrage ist gelöscht.

T60 deckt ab [FA60](#)

Der Gruppengründer entfernt ein Mitglied.

Ergebnis: Der Benutzer ist nicht mehr Mitglied (Alle Assoziationen zwischen Gruppe und Benutzer sind getrennt).

T70 deckt ab [FA70](#)

Der Gruppengründer löscht die Gruppe.

Ergebnis: Alle Mitgliedern sind entfernt ([FA60](#)) und die Gruppe aus der Datenbank gelöscht.

T80 deckt ab [FA80](#)

Ein Gruppenmitglied verlässt die Gruppe.

Ergebnis: Der Benutzer ist nicht mehr Mitglied (siehe T60).

#### **4.0.3 Terminverwaltung**

T90 deckt ab [FA100](#), [FA120](#)

Ein beliebiges Mitglied erstellt einen Termin mit Name, Ort, Zeitpunkt.

Ergebnis: Es gibt einen Termin mit dem genannten Namen, Ort und Zeitpunkt in der Gruppe.

T100 deckt ab [FA130](#)

Ein Mitglied sagt ab.

Ergebnis: Der Termin ist für das Mitglied gelöscht.

T105 deckt ab [FA130](#)

Ein Mitglied sagt zu.

Ergebnis: Das Mitglied ist Teilnehmer des Termins und der Termin ist auf dem Gerät des Teilnehmers gespeichert.

#### **4.0.4 Terminablauf**

T110 deckt ab [FA140](#), [FA150](#), [FA170](#)

Automatischer Ablauf eines Termins.

Ergebnis: 30 Minuten vor dem Termin werden alle Teilnehmer von dem Termin benachrichtigt.

15 Minuten vor dem Terminzeitpunkt ist die Karte für alle Teilnehmer verfügbar.

60 Minuten nach dem Terminzeitpunkt ist der Termin gelöscht (Keine Daten mehr

zu diesem Termin, weder auf dem Client noch auf dem Server). Bemerkung: Die Teilnehmer werden in der jetzigen Version 15 Minuten vor dem Termin benachrichtigt.

#### **4.0.5 Standort**

T120 deckt ab [FA110](#), [FA150](#)

Standort Übertragung der Clients an den Server.

Ergebnis: Der Standort aller Teilnehmer ist auf dem Server gespeichert.

Bemerkung: Da die Kommunikation zwischen Server und Client bei unserer App immer vom Client ausgeht lässt sich nicht sicher sagen ob zu einem gewissen Zeitpunkt die aktuellen Standorte aller User eines Termins auf dem Server gespeichert sind. Allerdings updaten die Clients bei ausreichender Internetverbindung alle 15-30 Sekunden ihre Standortinformationen auf dem Server.

T130 deckt ab [FA110](#), [FA150](#)

Der Server sendet den berechneten Gruppenmittelpunkt an den Teilnehmer.

Ergebnis: Der Client hat den aktuellen Gruppenmittelpunkt.

## 5 Anhang