goApp Qualitätssicherung

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

4. März 2017

Inhaltsverzeichnis

1 Einleitung

2 Änderungen

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

2.1 Server

2.1.1 Algorithmus

2.1.2 Datenbank

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 zurückgegeben 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 Client

3 Testbericht

3.1 Testszenarien

hier unere Szenarien aus dem Pflichtenheft rein und testen - beschreiben ob es ok war oder nicht, momentan sind diese nur kopiert aus dem Pflichtenheft. Irgendwie unter jedes noch Ergebnis: machen und Auffälligkeiten bei der Durchführung beschreiben oder das alles ok war oder so.

Die Tests werden auf dem Referenzgerät mit einer ausreichenden Internetverbindung und funktionierendem GPS ausgeführt.
Die folgenden Testfälle stellen sicher, dass die genannten Funktionalen Anforderungen erfüllt sind.

3.1.1 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.

T20 deckt ab FA20
Namensänderung: Ein Benutzer ändert seinen Benutzernamen.
Ergebnis: Der neue Benutzername des Benutzers ist in der Datenbank.

3.1.2 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 Gruppengrnder 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 Gruppengrnder 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).

3.1.3 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.

3.1.4 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).

3.1.5 Standort

T120 deckt ab FA110, FA150
Standort Übertragung der Clients an den Server.
Ergebnis: Der Standort aller Teilnehmer ist auf dem Server gespeichert.

T130 deckt ab FA110, FA150
Der Server sendet den berechneten Gruppenmittelpunkt an den Teilnehmer.
Ergebnis: Der Client hat den aktuellen Gruppenmittelpunkt.

3.2 Server

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

3.2.1 Algorithmus

3.2.2 Datenbank

3.2.3 Servlet

3.2.3.1 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 augerufen 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.

3.2.3.2 RequestSearchServlet

public void testGetRequestsByUserDatabaseNull() Wenn keine Anfragen von einem bestimmten Nutzer existieren gibt die Datenabnak 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 augerufen 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.

3.2.3.3 UserServlet

public void testMethodNotExisting() Wenn eine Methode augerufen 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.

3.2.3.4 EventServlet

public void testMethodNotExisting() Wenn eine Methode augerufen 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 die Datenbank das Event nicht angelegen 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 die Datenbank keiner 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.

3.2.4 Abdeckung

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

3.3 Client

3.3.1 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.

3.3.2 Performance

3.3.2.1 Monkey-Test

3.3.2.2 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 umzusetzten.
Einige Beispiele:

- User bekommen jetzt bei „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.

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 Mehtode 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: Teilweise stürzte die goApp ab wenn der User eine Gruppe verlässt. Fehler und Lösung sind wie beim Löschen einer Gruppe.

- Problem: Wenn ein User erfolgreich eine Gruppe gelöscht hat 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 gestart wird wird der komplette back stack gelöscht.

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

4 Anhang