goApp Qualitätssicherung

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

11. März 2017

Inhaltsverzeichnis

1 Einleitung

In der Phase der Qualitätssicherung ging es uns darum Fehler der goApp zu finden und zu beheben. Allen voran kann die App jetzt den auf dem Server berechneten Gruppenmittelpunkt bei Terminen anzeigen.
Ebenfalls haben wir weitere Unit-Tests zur Erhöhung der Code-Übderdeckung und zur Überprüfung von Randfällen implementiert um möglichst viele Fehler zu finden und eine hervorragende user experience zu bieten.
Ob uns dies gelungen ist haben wir versucht über das verbreiten der App unter Beta Usern herauszufinden. Hiervon versprachen wir uns konstruktive Kritik, insbesondere an der GUI um diese für den User noch intuitiver zu gestalten. Ansonsten haben wir unser Projekt durch Werkzeuge wie Lint, ein Tool zur statischen Ananlyse von Code, und Monkey Test überprüft um die Code Qualität noch weiter zu steigern.
Auf weitere Änderungen sowohl am Desing als auch der Funktionalität der goApp gehen wir im folgendem Dokument ein.

2 Server

2.1 SonarLint

SonarLint ist ein Programm zur statischen Code-Analyse der Quelltexte von Computerprogrammen. Mit SonarLint überprüfen wir unseren Servercode um mögliche Bugs zu finden und unsere Codequalität zu steigern. Dadurch haben wir unter anderem das Fehlen einer break Anweisung gefunden.

2.2 Änderungen

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.

2.2.1 Algorithmus

Beim Testen ist uns aufgefallen, dass die Clusterberechnung bei verschiedenen Größenordnungen an Abständen der Mitglieder schlechte Ergebnisse liefert, da der Radius, der festlegt wann man zusammen in ein Cluster gezählt wird, fest und nicht dynamisch ist. Aus diesem Grund haben wir uns nun dazu entschieden eine weitere, speziell auf den Clusteralgorithmus DBSCAN festgelegte, Methode anzubieten, die verschiedene Radien bei der Berechnung benutzt. Die Methode basiert auf der Rahmenbedingung, dass mindestens 3 von 4 Mitglieder bei der Berechnung der der Mittelpunkte berücksichtigt werden sollen, was äquivalent zu der Aussage ist, dass mindestens 3 von 4 Mitglieder sich nach der Clusterberechung in Cluster mit mehr als einer Person befinden. Wir suchen also nach den kleinsten Cluster, so dass die Rahmenbedingung erfüllt ist. Diese Suche wurde mit Hilfe einer Schleife implementiert, wobei bei jedem Schleifendurchlauf der Parameter, der angibt welchen Abstand Clustermitglieder zueinander haben dürfen, verdoppelt wird. Um unseren Server vor vor ewigen Laufzeiten zu schützen haben wir zusätzlich eine maximale Schleifendurchlaufszahl von 12 festgelegt, was einem Radius von über 200 Kilometer entspricht und somit für jede sinnvolle Nutzung unserer App ausreicht.

2.2.2 Datenbank

Es gab einen Fehler im Zusammenhang mit dem LocationDeletionTimer und dem EventDeletionTimer, 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 Interace 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.

2.2.3 Servlet

2.2.3.1 RequestServlet

- Fehlersymptom: Es tritt eine NullPointerException in der Methode create auf.
- Fehlergrund: Der Nutzer ist in keiner Gruppe Mitglied oder hat noch keine Anfrage gesendet.
- Fehlerbehebung: Es wird geprüft ob Gruppen bzw. Anfragen existieren und falls nicht entsprechend reagiert.

2.2.3.2 UserServlet

- Fehlersymptom: Es tritt in der Methode changeName eine NullPointerException auf.
- Fehlergrund: Wenn ein Nutzer in der Datenbank nicht gefunden wird, gibt diese null zurück. Dies wurde nicht geprüft.
- Fehlerbehebung: Vor der weiteren Ausführung wird die Antwort der Datenbank geprüft.

2.2.3.3 EventServlet

- Fehlersymptom: Es wurde trotz eines Fehlers, der Fehlercode für OK zurück gegeben.
- Fehlergrund: Die Rückgabe des Fehlercodes an Stelle, wo der Fehler gefunden wurde hat gefehlt.
- Fehlerbehebung: Der Fehlercode wird jetzt zurück gegeben.

2.3 Tests

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

2.3.1 Algorithmus

Wie oben beschrieben stellen wir eine weitere Methode zur Berechnung der Cluster und deren Mittelpunkte zur Verfügung. Diese Methode wurde nun auch ausführlich mit Hilfe von Unittests getestet. Dabei wurden wieder verschiedenen Testpunkte initialisiert und die Verbindung zur Datenbank mit Mockito gemockt, um den Algorithmus seperat zu testen.

2.3.2 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.

2.3.2.1 EventManagement

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

2.3.2.2 LocationDeletionTimer

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

2.3.3 Servlet

2.3.3.1 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.

2.3.3.2 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.

2.3.3.3 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.

2.3.3.4 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.

2.3.4 Abdeckung

2.3.4.1 EclEmma EclEmma ist ein Plugin für Eclipse, welches die Testabdeckung misst. Dabei wird bei der Ausführung einer Anwendung festgestellt, welche Klassen, Methoden, Blöcke und Zeilen Code ausgeführt wurden. Mit diesem Tool haben wir die Abdeckung durch unsere Tests geprüft.

2.3.4.2 XY Die Testabdeckung für den Server beträgt XX %. Nach der Implementierungsphase mit Tests zur Funktionalität betrug die Abdeckung 76,5%.

3 Client

3.1 Android Lint

Android Lint ist ein Programm zur statischen Code-Analyse der Quelltexte von Computerprogrammen und in Android Studio integriert. Wir benutzten Android Lint auf dem Client equivalent zu SonarLint auf dem Server um unser Android-Projekt auf mögliche Bugs zu überprüfen. Außerdem liefert Android Lint 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 Monkey-Test

Um die Resistenz und Stabilität unserer App zu überprüfen haben wir Monkey-Tests auf dem Android-Simulator durchgeführt. Monkey-Tests produzieren eine willkürliche Folge von Eingaben (clicks, swipes, keys) um die Reaktion der App zu testen. Unsere App ist bei jedem der 10 Testläufe mit jeweils 10.000 Eingaben nicht abgestürzt und hat eine kurze Reaktionszeit aufgewiesen.

3.3 Beta Tester

3.3.1 Verbesserungen an der GUI

Wir haben unsere App unter circa 10 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.

- User die bei einem Termin auf "GO" gedrückt haben werden in der Teilnehmeransicht nun farblich hervorgehoben.
- 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.
- Der berechnete Gruppenmittelpunkt wird dem User nun in dunkelgrün der goApp anstatt in rot angezeigt, so ist es leichter den Gruppenmittelpunkt vom Terminstandort auseinander zu halten.

3.3.2 Fehlverhalten

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: 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 gestart 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 der ausgewertete Fragebogen den wir entwickelt haben um die Beta User systematisch zu befragen und die Antworten einheitlich auswerten zu können. Insgesamt scheinen die User mit der App zufrieden zu sein.

4 Testszenarien

Im folgenden wollen wir kurz auf unsere Testszenarien aus dem Pflichtenheft eingehen,. Trotz einer immer weiter voranschreitenden Verbesserung und dadurch teilweisen Abänderung unserer Ideen aus dem Pflichtenheft, konnten alle Testszenarien die im Pflichtenheft definiert wurden mit positivem Ausgang durchlaufen werden.
Die Tests wurdem auf dem im Pflichtenheft gennanten 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 meht gewählt wird sondern auf den GoogleAccount Namen gesetzt wird, der Benutername 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 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).

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 jetztigen Version 15 Minuten vor dem Termin benarichtigt.

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