Projektbericht Skatenight-App

Projektseminar „Android-Programmierung“ im WS 14/15

Bernd Eissing, Pascal Otto, Daniel Papoutzis,
Tristan Rust, Richard Schulze, Martin Wrodarczyk

Inhaltsverzeichnis

1 Einleitung

von allen

2 Vorgehensweise

von Tristan, Richard

Am Anfang des Projektes haben wir uns darauf geeinigt alle die gleiche Integrierte Entwicklungsumgebung (IDE) „Android Studio“ von Google zu nutzen, welches die offizielle Entwicklungsumgebung für Android ist, damit es nicht zu Problemen innerhalb des Projektes kommt. Weiterhin haben wir uns für die freie Software „Git“ zur Versionsverwaltung des Projekts entschieden. Hierfür gab es einen Master Branch auf dem sich immer eine lauffähige Version der Apps befunden hat. Für die Entwicklung einzelner Funktionen, wie zum Beispiel die „lokale Auswertung“wurden extra Branches angelegt. Für das Vorgehen im Projekt wurde das Rahmenwerk Scrum verwendet. Die Rolle des Product Owners wurde von den Betreuern übernommen. Außerdem hat die Rolle des Scrum Masters nach jedem einzelnen Sprint, der in der Regel 4 Wochen betrug, ein anderes Mitglied des Entwicklungsteams übernommen. Zudem wurden die einzelnen Aufgaben für das Sprint Backlog vor jedem Sprint bzw. nach dem Sprint Review in Tasks aufgeteilt. Damit ein Task wirklich als bearbeitet („done“) gilt wurde vor dem Projekt definiert, ab wann ein Task als dies gelten darf:

- Programmierung abgeschlossen
- Funktion getestet
- Ein Code Review durchgeführt wurde
- Die Feature-Branch auf den master-Branch gemerged wurden
- Die Dokumentation (Java Doc) geschrieben wurde

Außerdem wurde regelmäßig das Daily Scrum durchgeführt, wodurch wir uns gegenseitig Informiert haben, wer gerade an was arbeitet, ob Probleme aufgetreten sind oder Hilfe benötigt wird.
Zum kontinuierlichen Bauen der Apps und des Backends wurde ein Jenkinsserver eingesetzt. Da es einige Schwierigkeiten bei der Durchführung von Tests auf dem Jenkinsserver gab, haben wir bei unserer Projektarbeit nur sehr selten den Jenkinsserver in Anspruch genommen. Dennoch besteht eine Möglichkeit das Projekt auf dem Jenkinsserver zu bauen. Der Ablauf ist dabei wie folgt:

1. Säubern des Projekts mit dem Befehl „clean“
2. Anpassen der Versionsnummer der Build-Tools von 1.9.17 auf 1.9.8 in der Gradle-Builddatei für das Backend, da das Bauen des Backends sonst fehlschlägt.
3. Bauen und Hochladen des Backend auf den Release-Server
4. Zurücksetzen der Änderungen in der Build-Datei.
5. Bauen und Signieren der User- und Veranstalter-App in der Release-Konfiguration.

3 Features der Apps

von Daniel

4 Vorstellung von Android

von Pascal

5 Server-Backend

5.1 Anforderungen

von Richard

Zur Kommunikation der Apps, aber auch zur zentralen Speicherung der Veranstaltungen, Routen, Nutzern und Nutzergruppen ist ein Backend, im folgenden auch Skatenight API genannt, notwendig, das eine öffentlich erreichbare Schnittstelle zur Verfügung stellt. Neben der Speicherung von Daten ist aber auch notwendig, dass der Server eigene Berechnungen durchführen kann. So wird zum Beispiel das Feld, das den Benutzern in der User-App angezeigt wird, zentral auf dem Server berechnet und von den User-Apps abgerufen (siehe Abschnitt 5.7).
Eine weitere wichtige Anforderung war für uns, dass der Server kostenlos nutzbar ist und nach Möglichkeit bereits ein Grundgerüst bietet, über das aus Android-Apps möglichst einfach auf die Server-API zugegriffen werden kann. Wir wollten Hauptaugenmerk auf die Entwicklung der Apps legen und die Zeit zur Entwicklung des Backends möglichst gering halten. Das Google App Engine Projekt bietet diese Möglichkeit, da es von den realen Servern vollkommen abstrahiert. Zusätzlich dazu werden einige nützliche APIs angeboten, die unter anderem Funktionalität zur Speicherung von Daten bereitstellen und die Definition von Kommunikations-Endpunkten, sowie die Kommunikation vom Server zum Handy ermöglichen.

5.2 Server, Keys und Build-Varianten

von Richard

Während der Entwicklung der Apps ist uns schnell aufgefallen, dass aufgrund der verschiedenen Features, die jeweils auf eigenen Branches entwickelt wurden, auch der Server von mehr als einer Person angepasst wurde. Damit einher ging, dass zwischenzeitlich verschiedene Versionen des Servers notwendig waren, damit jede Teilgruppe des Teams das eigene Feature implementieren konnte, ohne die Entwicklung anderer Teammitglieder zu stören. Aus diesem Grund haben wir mehrere Projekte bei der Google App Engine angelegt, zwischen denen relativ einfach gewechselt werden kann. In unserem Projekt in den Modulen „app“ und „veranstalterapp“ gibt es entsprechende Build-Varianten, mit denen der gewünschte Server ausgewählt werden kann. Da es im Modul „SkatenightBackend“ nicht die Möglichkeit gibt Build-Varianten zu definieren, gibt es dort die Befehle „applyProductionserverConfig“, „applyTestserverConfig“ und „applyJenkinsserverConfig“. Diese kopieren jeweils aus den entsprechenden Unterordnern des src-Ordners die Konfigurationsdateien, die für einen Server-Wechsel notwendig sind, in den main-Ordner. Bei einem Serverwechsel wird die *Constants.java* und die Projekt-ID in der Datei *appengine-web.xml* ausgetauscht. Wenn anschließend der Befehl „appengineUpdate“ zur Aktualisierung des Servers aufgerufen wird, wird der Server aktualisiert, dessen Projekt-ID in der *appengine-web.xml* steht.

Web-Client ID: Dient der Authentifizierung über OAuth2, das die gesicherten API-Funktionen, wie beispielsweise das Anlegen eines Events schützt.		
Key	Release	37947570052-dk3rjhgran1s38gscv6va2rmmv2bei8r.apps.googleusercontent.com
	Debug	644721617929-unb9em0kl73b9evdv2h52ufn26fao20p.apps.googleusercontent.com
	Jenkins	1032268444653-7agre4q3eosqhlh92sq62hf5fan9jbv5.apps.googleusercontent.com
Client-ID der User-App: Dient der Identifikation der User-App am Backend.		
Key	Release	37947570052-g006o3ovfotnjqreltom6c7cbktm7dap.apps.googleusercontent.com
	Debug	644721617929-mk0do3jm5lec1dasijdj3220ot87gbn7.apps.googleusercontent.com
	Jenkins	1032268444653-kj1mpisvlrpl67e7db2fu2005aube7mu.apps.googleusercontent.com
Client-ID der Veranstalter-App: Dient der Identifikation der Veranstalter-App am Backend.		
Key	Release	37947570052-4ru7asfhnrjmmqvj3qdpo2rrp31fudf3.apps.googleusercontent.com
	Debug	644721617929-kbcha00vb3j30sh9at05sagm73iltqqo.apps.googleusercontent.com
	Jenkins	1032268444653-83ih5mp5mguh66c012u0sobqe4oqoupr.apps.googleusercontent.com
Google-Maps API-Key für die User-App: Wird zur Nutzung der Google-Maps API in der User-App benötigt.		
Key	Release	AIzaSyBKGVQHij0JcjgHyCnT0ZrLihbCn8Av2jg
	Debug	AIzaSyDQgEKF42jjm57x7kgnY7EI62CcXW_zDS0
	Jenkins	AIzaSyAQ47zRjgPRk1sxJYifd2URQdEnk5HvnRw
Google-Maps API-Key für die Veranstalter-App: Wird zur Nutzung der Google-Maps API in der Veranstalter-App benötigt.		
Key	Release	AIzaSyBedRi4A–p3LNQLy5lgOsBEdIdKuLbi2U
	Debug	AIzaSyAO4zGlaKexByZmEiFKLbo18Y_I-t9KbMc
	Jenkins	AIzaSyBes8RSzVdqcQY-vnj9GLWxCLBKAFCzh98
Google Cloud Messaging API-Key: Dient der Registrierung beim GCM-Dienst von Google in der User-App.		
Key	Release	AIzaSyDO8mosWwYXjjGZ9besu9CZw1LDEEXrFXE
	Debug	AIzaSyCpwhxda1Lb5E61_fybZq2iSJgViZu3QNM
	Jenkins	AIzaSyBjeKzx_vzkUgQZdT83BB0BMD3gFebpet0

Tabelle 1: Eine Liste aller verwendeter IDs und Keys, die zur Authentifizierung der Apps und Server notwendig sind.

Tabelle 1 listet nocheinmal die Details zu den einzelnen Google App Eingine Projekten und auch die notwendigen Keys auf, mit denen sich die Apps gegenüber dem Backend und anderen Diensten identifizieren.

5.3 Klassenstruktur der Skatenight API

von Richard

Das Backend ist eine in Java geschriebene Anwendung, die sich im Projektordner in dem Modul „SkatenightBackend“ befindet. Das für die Apps zugängliche Interface wird in der Klasse *SkatenightServerEndpoint* beschrieben. Alle darin implementierten öffentlichen Methoden sind über die generierten Client-Libraries aufrufbar.
Die Klasse *EventStartServlet* wird über einen Cron-Job, der in der Datei *cron.xml* im WEB-INF-Ordner des Moduls definiert ist, alle zwei Minuten aufgerufen. Ihre Aufgabe ist es nach Events zu suchen, die kürzlich begonnen haben und für die noch keine GCM-Nachricht an die teilnehmenden Benutzer versandt wurde. Problematisch war, dass Cron-Jobs bei einem Google App Engine Projekt nicht dynamisch zur Laufzeit erstellt werden können. Die zunächst angedachte Lösung für jedes Event einen eigenen Cron-Job anzulegen, der genau zum Startzeitpunkt

des Events aufgerufen wird, ist dadurch nicht möglich. Wir haben uns deshalb dafür entschieden, die Events mit einem Boolean zu markieren, wenn sie gestartet sind, sodass in der *doGet()* -Methode nach Events gesucht werden kann, für die diese Markierung noch nicht gesetzt wurde. Für jedes startende Event kann dann eine Liste der GCM-IDs der Benutzer erstellt werden, die über das Event benachrichtigt werden müssen. Gleichzeitig wird für die Benutzer auch die aktuelle Event-ID(*currentEventId*) gesetzt, die für die Berechnung des Feldes relevant ist.

5.4 Datenmodell

von Richard

Abbildung 1 zeigt das auf dem Server repräsentierte Datenmodell. Die Attribute, die mit einer dickeren Linie gestrichelt umrandet sind, sind Listen von Objekten, die serialisiert werden und nicht als eigener Typ in der Datenbank existieren. Die Veranstalter werden als Hosts in der Datenbank abgespeichert, haben jedoch keine weiteren Beziehungen zu anderen Entitätsklassen.

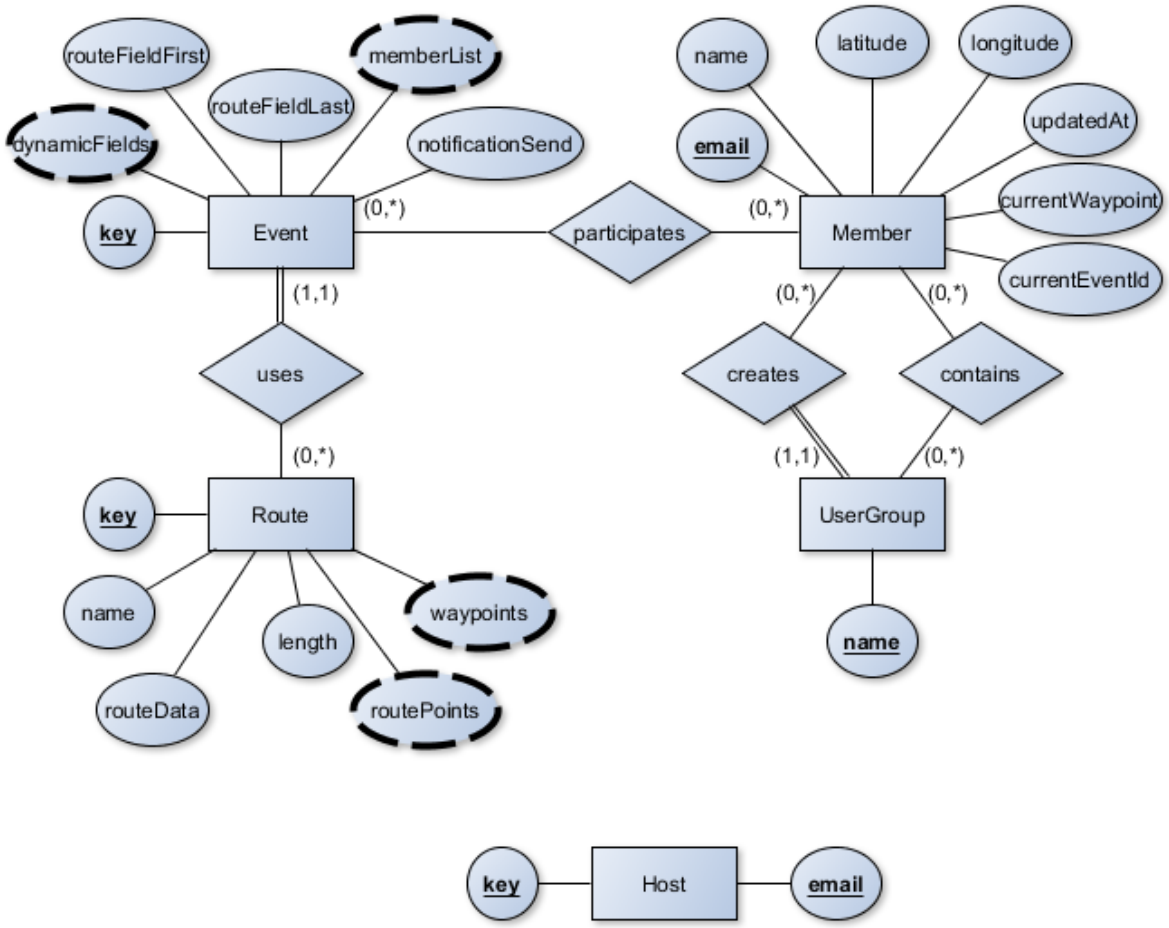

Abbildung 1: ER-Modell der Daten, die auf dem Server gespeichert werden

Wie auf der Abbildung gezeigt, werden zu einem Event auch die teilnehmenden *Member* gespeichert. In dem Event existieren dazu jedoch nicht direkt die Member-Objekte, sondern nur Strings, die auf den Primary Key der Klasse *Member* (*email*) zeigen. Dies ist notwendig, da aufgrund der Struktur des Google Datastore die Member nicht existentiell von den Events abhängen dürfen. Im Datastore kann für jeden Datensatz ein übergeordneter Datensatz definiert werden. Würde dies das Event sein, so würden beim Löschen des Events auch alle teilnehmenden

Member-Datensätze gelöscht werden. Analog sind auch die Benutzergruppen implementiert. Damit beim Löschen einer Benutzergruppe die Benutzer nicht ebenfalls gelöscht werden, enthalten die Benutzergruppen ebenfalls nur eine Liste von Strings, die auf die E-Mails der enthaltenen Benutzer zeigen.
Die Umsetzung des Klassenmodells des Backends weicht etwas von der in der Datenbank gewählten Struktur ab. Hauptunterschied ist, dass mehr Klassen zur Repräsentation der Daten definiert wurden. So gibt es beispielsweise zusätzlich zu den Routen auch die Klassen *RoutePoint* zur Repräsentation eines feinen Wegpunktes auf der Strecke, sowie die Klasse *ServerWaypoint* , die die beim Erstellen der Strecke gesetzten Marker speichert. Einen genauen Überblick über das Klassenmodell bietet Abbildung 2.

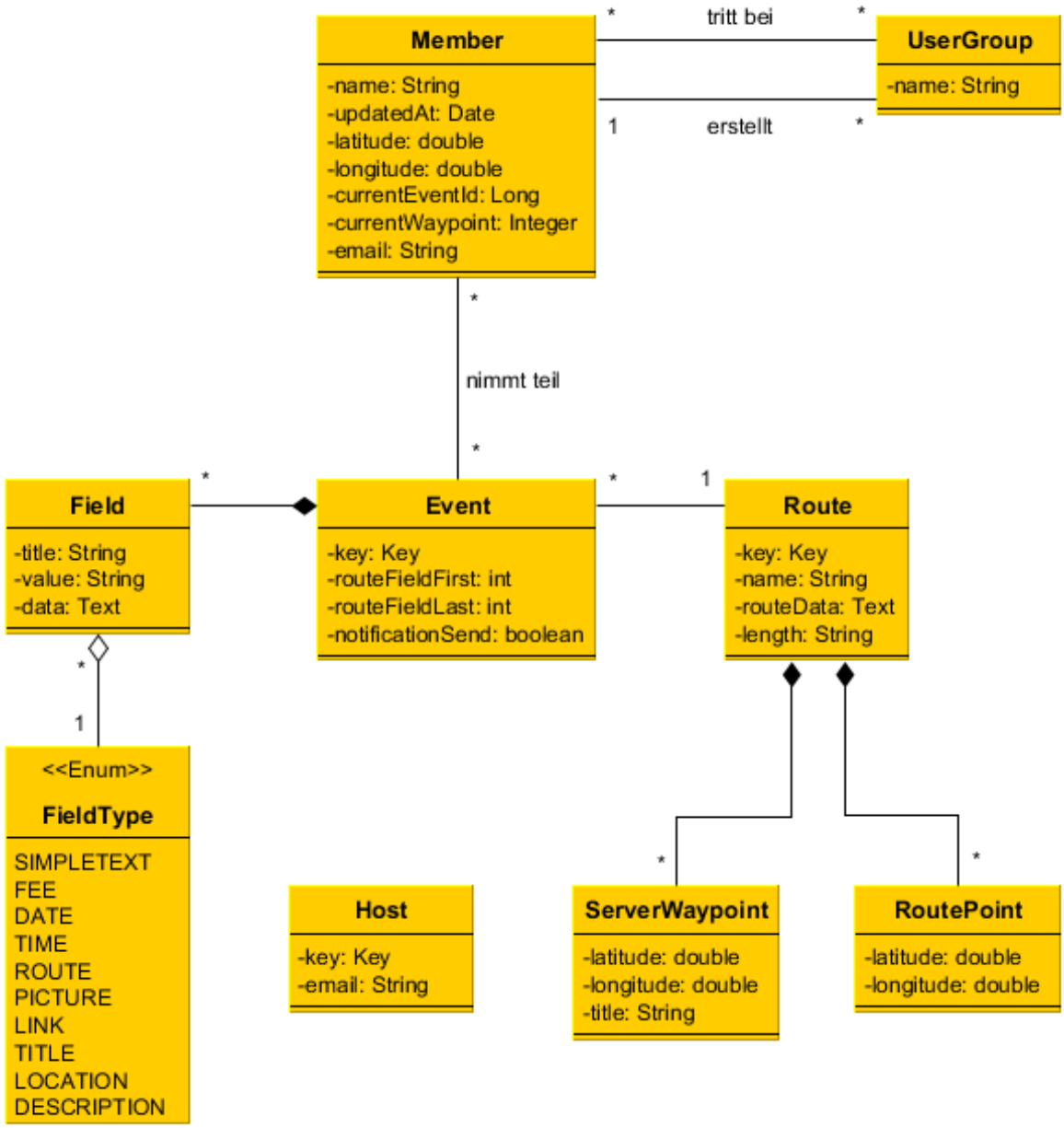

Abbildung 2: UML-Modell der Klassenstruktur des Backends

5.5 Verfügbare API-Aufrufe

von Richard

Zur Kommunikation der User- und Veranstalter-App mit dem Backend existieren einige API-Aufrufe auf dem Backend. Ein wichtiges Kriterium bei der Implementierung war, dass nicht jeder Benutzer die gleichen Rechte hat. So kann ein Veranstalter z.B. Events erstellen und löschen, ein normaler Skatenight-Teilnehmer sollte dies jedoch nicht können. Aufgrund dieser Anforderungen haben wir uns für die OAuth2-API von Google entschieden. Mit ihr ist es einfach, eine Authentifizierung der Benutzer zu implementieren, da sie in Google Cloud Endpoints bereits integriert ist. Auf dem Server werden alle Methoden, die eine Authentifizierung benötigen, um einen Parameter vom Typ *com.google.appengine.api.users.User* erweitert. Beim Verbindungsaufbau in der App wird der am Handy angemeldete Benutzer ausgelesen und bei jedem Aufruf an das Backend nun automatisch übergeben. Hinzu kommt, dass die Authentifikation über den Google-Account geschieht. Da die App für Android-Geräte entwickelt wurde, ist sichergestellt, dass jeder Benutzer der App auch ein entsprechendes Konto besitzt. Die folgende Liste beschreibt die verfügbaren API-Aufrufe und stellt auch kurz dar, in welchem Zusammenhang diese in den Apps verwendet werden.

addHost

Die Methode *addHost* dient dem Hinzufügen von Veranstaltern. Durch den User-Parameter in der Signatur der Methode ist der Aufruf so geschützt, dass nur bereits eingetragene Veranstalter neue Veranstalter hinzufügen können. In der Veranstalter-App ist diese Funktion in der Rechteverwaltung zugänglich.

removeHost

Ähnlich wie die *addHost* -Methode ist die *removeHost* -Methode wieder über ein User-Parameter geschützt. Die Methode löscht den angegebenen Veranstalter, sofern der aufrufende Benutzer ein bereits eingetragener Veranstalter ist. Auch diese Funktion ist in der Rechteverwaltung der Veranstalter-App zugänglich.

isHost

Die *isHost* -Methode prüft für den angegebenen Benutzer, ob dieser ein eingetragener Veranstalter ist. Dies ist z.B. bei der Anmeldung in der Veranstalter-App notwendig.

getHosts

Diese Methode liefert eine Liste aller Veranstalter zurück, die zurzeit auf dem Server hinterlegt sind. Die Liste der Veranstalter kann nur von bereits eingetragenen Veranstaltern abgerufen werden.

createEvent

Zum Erstellen eines Events dient die Methode *createEvent* , die von eingetragenen Veranstaltern aufgerufen werden kann. Zusätzlich zu dem Event kann angegeben werden, ob es sich um einen Editiervorgang handelt und, ob das Datum des Events geändert wurde. Diese Daten sind notwendig, da Benutzer der User-App so über die Änderungen per GCM benachrichtigt werden können.

createMember

Die *createMember* -Methode wird aus der User-App heraus aufgerufen. Über diese Funktion wird sichergestellt, dass auf dem Server für den Benutzer ein *Member* -Objekt existiert, das unter anderem die Liste der Events speichert, an dem der Benutzer teilnimmt, und auch die aktuell gemeldete Position des Benutzers.

updateMemberLocation

Über die Methode *updateMemberLocation* können die Positionen der Benutzer der User-App an den Server übermittelt werden. Neben der neuen Position des Benutzers wird auch die ID des Events, an dem der Benutzer gerade teilnimmt, übertragen. Dies ist für die Feldberechnung auf dem Server notwendig, die ebenfalls über diese Methode angestoßen wird, falls die letzte Feldberechnung länger als einen gewissen Wert zurückliegt.

simulateMemberLocations

Für Präsentationen der App und der Feldauswertung haben wir neben den beiden Apps einen Simulator für Benutzer der Skatenight-App entwickelt. Zunächst war geplant, dass dieser Simulator genau wie die User-App die *updateMemberLocation* -Methode aufruft. Da bei bis zu 100 simulierten Teilnehmern die Anzahl der Serverzugriffe jedoch sehr hoch und damit nicht mehr im kostenlosen Rahmen der Google App Engine lag, haben wir uns dafür entschieden eine gesonderte Methode zur Simulation der Positionen zu erstellen. Diese kann mit nur einem Aufruf eine Liste von Teilnehmern und deren Positionen entgegen nehmen, sodass nun die Anzahl der Serveraufrufe unabhängig von der simulierten Teilnehmerzahl ist. Intern wird jedoch nach wie vor die *updateMemberLocation* -Methode aufgerufen.

getCurrentEventsForMember

Da im Datenmodell in den *Member* -Objekten die Liste der Events, an denen der Benutzer teilnimmt, nicht gespeichert werden, kann über diese Methode eine Liste der entsprechenden Events für einen Teilnehmer abgerufen werden.

getMember

Die *getMember* -Methode dient dem Abrufen der Member-Objekte zu entsprechenden E-Mail-Adressen.

addMemberToEvent

Die *addMemberToEvent* -Methode fügt einen Teilnehmer anhand seiner Mail zu dem Event mit der angegebenen ID hinzu. Dabei wird außerdem geprüft, ob das Event bereits begonnen hat und der Teilnehmer in diesem Fall per GCM informiert.

removeMemberFromEvent

Neben dem Teilnehmen an einem Event muss der Benutzer auch die Möglichkeit haben, das Event wieder zu verlassen. Die Methode *removeMemberFromEvent* realisiert diese Funktion und wird in der User-App aufgerufen.

getMembersFromEvent

Analog zur *getCurrentEventsForMember* -Methode ruft die Methode *getMembersFromEvent* die Liste der Teilnehmer eines Events ab. Zwar werden diese in der Klasse *Event* gespeichert,

jedoch nur als Referenz über die Mail-Adresse. Diese Methode wandelt die Mail-Adressen in eine Liste von *Member* -Objekten um, die anschließend zurück gegeben werden.

addRoute

Die *addRoute* -Methode ist eine Methode, die nur von eingetragenen Veranstaltern aufgerufen werden kann. Sie speichert die übergebene Route in der Datenbank und wird aus der Veranstalter-App nach dem Anlegen einer neuen Route aufgerufen.

getRoutes

Diese Methode gibt eine Liste der auf dem Server gespeicherten Routen zurück. Sie wird sowohl aus der User- als auch aus der Veranstalter-App aufgerufen, da in beiden Apps eine Liste der Routen angezeigt werden kann.

deleteRoute

Die *deleteRoute* -Methode bildet das Gegenstück zur *addRoute* -Methode. Sie löscht die angegebene Route, falls es sich bei dem aufrufenden Benutzer um einen eingetragenen Veranstalter handelt.

getEvent

getEvent gibt das *Event* -Objekt mit der angegebenen ID zurück. Die Methode wird sowohl aus der Veranstalter- als auch aus der User-App aufgerufen, wenn der Anwender auf ein Event klickt.

deleteEvent

Die *deleteEvent* -Methode kann von eingetragenen Veranstaltern zum Löschen eines Events aus der Veranstalter-App heraus aufgerufen werden. Die Methode gibt als Boolean-Wert zurück, ob das Event gelöscht werden konnte oder nicht. Da durch die Google Cloud Endpoints die Menge der verwendbaren Rückgabetypen für Methoden eingeschränkt ist, sind unter anderem primitive Datentypen wie boolean oder der durch Java vorgegebene Standardwrapper Boolean nicht möglich. Aus diesem Grund haben wir einen eigenen Boolean-Wrapper implementiert, der als einzigen Zweck das Kapseln eines Boolean-Wertes erfüllt.

editEvent

Um ein Event anzupassen, können Veranstalter die *editEvent* -Methode aufrufen. Die Implementation dieser Methode löscht zunächst das alte Event und legt es anschließend über die *createEvent* -Methode mit den neuen Daten wieder an.

getAllEvents

Für Benutzer der beiden Apps ist die Methode *getAllEvents* ohne Authentifikation zugänglich. Sie gibt eine Liste aller auf dem Server gespeicherter Events zurück. Da durch die Nuztung von JDO und DataNucleus die in den Events eingebetteten Routen jedoch nicht abgerufen werden konnten, wird nach dem Ermitteln der Eventliste jedes Event nocheinmal einzeln abgerufen. So sind auch die Routen in den Event-Objekten initialisiert.

registerForGCM

Von der User-App wird bei jedem Appstart die *registerForGCM* -Methode aufgerufen. Sie übermittelt die aktuelle GCM-ID an den Server, die zur Kommunikation über den Google Cloud Messaging Dienst benötigt wird. Die GCM-ID eines Teilnehmers wird in seinem *Member* -Objekt gespeichert.

getAllUserGroups

getAllUserGroups ruft die Liste aller Benutzergruppen ab. Sie wird von der User-App benutzt.

fetchMyUserGroups

Diese Methode ruft ebenfalls eine Liste der Benutzergruppen ab, schränkt diese jedoch auf Benutzergruppen des angegebenen Benutzers ein. Zusätzlich prüft die Methode, ob *Member* -Objekte auf Gruppen verweisen, die nicht mehr existieren und bereinigt die Daten, falls entsprechende Gruppen gefunden werden.

createUserGroup

Die *createUserGroup* -Methode legt eine Benutzergruppe mit dem angegebenen Namen und dem aufrufenden Benutzer als Ersteller an. Zusätzlich werden die User-Apps der Benutzer per GCM über die neue Gruppe informiert, damit die Liste der Benutzergruppen in bereits geöffneten Apps aktualisiert werden kann. Für den Benutzer ist diese Nachricht nicht sichtbar.

deleteUserGroup

Die *deleteUserGroup* -Methode löscht die Benutzergruppe mit dem angegebenen Namen, sofern der aufrufende Benutzer Ersteller der Gruppe ist.

joinUserGroup

Über die *joinUserGroup* -Methode haben Benutzer der User-App die Möglichkeit, Benutzergruppen anderer Benutzer beizutreten.

leaveUserGroup

Die *leaveUserGroup* -Methode entfernt den Benutzer wieder aus der angegebenen Gruppe. Dies ist nur dann Möglich, wenn der aufrufende Benutzer nicht Ersteller der Gruppe ist, aus der er austreten möchte.

fetchGroupMembers

Mithilfe der *fetchGroupMembers* -Methode kann eine Liste aller Member abgerufen werden, die sich in der angegebenen Benutzergruppe befinden. Dies wird benötigt, wenn die Gruppenmitglieder der eigenen Gruppen auf der Karte während eines Events angezeigt werden sollen.

5.6 Google Cloud Messaging

von Richard

An mehreren Stellen der User-App wird die Google Cloud Messaging API genutzt. Sie ermöglicht es, Nachrichten vom Backend an registrierte Handys zu senden. Dies ist vor allem dann nützlich, wenn sich Daten auf dem Server ändern und der Benutzer darüber informiert werden muss. An folgenden Stellen benutzen wir das Google Cloud Messaging:

- Zur Benachrichtigung aller Benutzer, wenn ein neues Event erstellt wurde. In diesem Fall wird die Liste der Events in der User-App automatisch aktualisiert, falls sie geöffnet ist. Ebenfalls wird dem Benutzer eine Notification-Nachricht angezeigt, die von überall im Handy über die obere Statusleiste abrufbar ist.
- Zur Benachrichtigung aller Benutzer, wenn eine neue Benutzergruppe erstellt wurde. Dabei wird jedoch keine Notification erstellt, sondern lediglich die Liste der Gruppen aktualisiert.
- Zu Benachrichtigung aller Benutzer, wenn eine Benutzergruppe gelöscht wurde. Dies ist notwendig, damit gegebenenfalls die Einstellung beim Benutzer, dass die gelöschte Gruppe auf der Karte angezeigt werden soll, ebenfalls gelöscht wird. Bei dieser Benachrichtigung erhält der Benutzer außerdem eine Notification.

Die ID, die zur Identifikation und damit zur Kontaktaufnahme mit den Benutzern per GCM notwendig ist, wird beim Start der User-App auf dem Handy generiert. Dieses sendet die ID dann an den Server, der die ID für den aufrufenden Benutzer hinterlegt. Zur Verwaltung der GCM-IDs gibt es auf dem Server die Klasse *RegistrationManager* . Diese speichert zu jeder Nutzer-Mail-Adresse eine ID ab, kann aber auch eine Liste aller registrierter GCM-IDs zurückgeben.
Zur Unterscheidung der einzelnen Nachrichten-Typen, die vom Server zu der User-App gesendet werden, existiert die Enumeration *MessageType* . Alle vom Skatenight-Backend generierten Nachrichten enthalten als Payload mindestens das Attribut *type* , das als Wert den Namen des Enumeration-Eintrags hat. Anhand dieses Typs wird auf dem Handy die Nachricht dann durch den GcmIntentService weiter verarbeitet.
Um eine GCM-Nachricht vom Backend zu Senden, wird zunächst ein *Message* -Objekt erstellt. Dieses erhält Daten wie den *collapseKey* , über den mehrere Nachrichten zusammengefasst werden können, falls sie stoßweise beim Benutzer ankommen. Ebenfalls kann die Gültigkeitsdauer der Nachricht angegeben werden. Falls die Nachricht innerhalb dieser Zeitspanne nicht ausgeliefert wurde, wird sie verworfen. Als weitere Einstellung kann angegeben werden, ob die Nachricht direkt zugestellt werden soll oder erst, wenn der Benutzer sein Handy aktiviert. Über ein Objekt vom Typ *Sender* kann die Nachricht dann an die gewünschten GCM-IDs verschickt werden.

5.7 Feldberechnung

von Pascal

6 Frontend

7 Testing

7.1 Einleitung

von Tristan

In dem Projekt gibt es zwei Arten von Tests, die „Task Tests“ und die „Anwendungsfalltests“ (Use Case Tests). Die Tasks Tests dienen hierbei zum Testen einer bestimmten Funktionalität, wie zu Beispiel dem Hinzufügen oder Löschen einer Route (*AddRouteTaskTest* und *DeleteRouteTaskTest*). Die Anwendungsfalltests sollen im Gegensatz zu den Tasks Tests nicht nur eine Funktionalität sondern ganze Anwendungsabläufe Testen, was die UI mit einschließt. Beide Tests, der Task Test und Anwendungsfalltest bestehen aus einem Konstruktor zum Initialisieren oder Aufsetzen der Testdaten, welcher einmalig im Test aufgerufen wird und einer *setUp* -Methode, die vor jedem einzelnen Test aufgerufen wird. Sie dient dazu Variablen zu initialisieren und Daten

von vorherigen Tests „aufzuräumen“. Weiterhin gibt es bei dem Task Test eine einzige Methode, die ausschließlich die Funktion testet, für die der Test bestimmt ist (*testTask* -Methode).

7.2 Funktionsweise der Anwendungsfalltests

von Tristan

Da die TaskTests an anderer Stelle genauer beschrieben werden und nicht die Komplexität der Anwendungsfalltests aufweisen, wird hier der allgemeine Ablaufe sowie die Funktionsweise dieser Tests genauer beschrieben.
Einfachheitshalber werden die Anwendungsfalltests ab jetzt nur noch als Tests bezeichnet und beziehen sich nur auf diesen Abschnitt. Die Testklassen verwenden als „Base Test Case Class“ die *ActivityInstrumentationTestCase2* Klasse, welche Methoden zur Interaktion mit der UI unter Testbedingungen zur Verfügung stellt und in dem Paket „android.test“ enthalten ist. Der Test erbt also von dieser Klasse. Zusätzlich wird hierbei die Activity angegeben, in der der Test starten soll. Zum Beispiel startet der Test bei *ActivityInstrumentationTestCase2<HoldTabsActivity>* in der *HoldtabsActivity* . Zudem ist die Reihenfolge, in der die Tests ausgeführt werden nicht vorgegeben und kann variieren.

Konstruktor

Weiterhin wird im Konstruktor lediglich der superclass Konstruktor auf gerufen und die Klasse der Start Activity übergeben (HoldtabsActivity.class).

setUp Methode

Die *setUp* -Methode wird vor jedem Test aufgerufen und dient wie bereits erwähnt der Initialisierung und dem Löschen der Testdaten vorheriger Tests, da ein Anwendungsfall mehrere Test Methoden enthalten kann. Zuerst wird die superclass-Methode aufgerufen (super.setUp) und anschließen folgt der Initialisierungscode. Um die UI ohne Fehler von außen Testen zu können wird der Touch Mode mit *setActivityInitialTouchMode(false)* ausgeschaltet. In der Regel wird hier auch die Activity gestartet, falls dies noch nicht der Fall sein sollte (*getActivity()*) und einem Klassenattribut *mActivity* zugewiesen.

testPreConditions Test

Stellt sicher, dass die Vorbedingungen für die Anwendung fehlerfrei initialisiert wurden. Dazu gehören unter anderem die UI Elemente und das Starten der Activity.

testUseCase Test

Alle Tests, die auch die UI Testen, laufen auf einem extra dafür vorgesehen Thread, dem Ui-Thread. Auf diesem werden alle Interaktionen wie Buttonklicks oder Swipen simuliert. In den meisten Fällen werden Key Events simuliert mit *sendKeys* in dem UIThread und anschließen folgt die Überprüfung mit Hilfe von „Asserts“ wie in Junit. Diese prüfen, ob die Werte zum Beispiel die über den UI Thread eingegebenen Daten in Editfelder mit den erwarteten Werten übereinstimmen. Ist dies der Fall fährt der Test fort und endet, wenn alle Asserts fehlerfrei waren. Sollte jedoch ein Assert fehlschlagen, wird die Methode *testUseCase* abgebrochen und gilt somit als nicht bestanden. Außerdem können innerhalb des Tests auch andere Activities gestartet werden. Diese laufen dann über einen *Instrumentation.ActivityMonitor* über den die gestartete Activity überwacht wird, anschließen wird der Test Code ausgeführt und die Activity wieder geschlossen mit dem Befehl *finish()* . Sollte diese Activiy an dieser Stelle nicht über *finish()*

beendet werden und der *testUseCase()* erfolgreich abgeschlossen worden sein, läuft diese Activity im Hintergrund über den Monitor weiter und kann zu fehlerhaften Ergebnissen in anderen Tests führen.

State Management Tests

Diese Art von Tests verifizieren die Activity in den Status „Pausieren“ oder „Beendet“. Folglich das Verhalten der Activity nach dem Fortsetzen oder Neustarten. Diese Art von Tests sind nicht in allen Anwendungsfalltests enthalten. Der Test *testStateDestroy* verwendet die von der Klasse *InstrumentationTestCase2* bereit gestellte Methode *fininsh* (*mActivity.finish()*) zum Beenden der Activity und der Test *testStatePause* die Methoden *callActivityOnPause* und *callActivityOnResume* . Diese halten die App an, was beispielsweise ein Klick des Nutzers auf den „Home Screen Button“ sein könnte.
Innerhalb des Tests wurde häufiger der Befehl *Thead.sleep(Zeitangabe)* verwendet, was den Hintergrund hat bei zeitkritischen Abschnitten eine bestimmte Zeit lang zu warten, damit zum Beispiel Daten vom Server über Netzwerkverbindungen, UI Elemente rechtzeitig initialisiert werden (die Zeit kann Testgeräte abhängig sein) oder die GPS Position ermittelt werden kann.
Auch kommen in den Testklassen die Begriffe „Small“, „Medium“ und „Large“ – Test vor. Deren Zweck ist es über den „Scope“, Abhängigkeiten und Performance Faktoren der Tests zu informieren, die Code-Qualität sowie System Instandhaltung zu verbessern.
Small (Unit): Verifiziert kleine „low-level“ Logik meist im Bereich einer Methode oder Klasse und bezieht sich auf keine externen Ressourcen. Die Ausführungszeit sollte im Sekunden (besser Millisekunden) Bereich liegen.
Medium (Integration): Kann mehrere Interaktionen zwischen Komponenten enthalten und auch auf externe Datei Systeme zugreifen oder mehrere Prozesse laufen lassen. Die Ausführungszeit kann Minuten betragen (vorzugsweise Sekunden).
Large (System): Hier kann es sein das der Test auch auf externe Ressourcen, wie Server über das Internet zu greifen muss. Die Ausführungszeit kann hier also etwas länger betragen, von Sekunden, Minuten oder Stunden.

7.3 Simulator

von Richard

Zu besseren Testbarkeit der App ist während der Entwicklung auch ein Simulator für eine Skatenight entstanden. Der Simulator ist in JavaScript geschrieben und benutzt JQuery, sowie die Google Maps API v3 als Bibliotheken.
Im Quelltext des Simulators kann in Zeile 50 die Adresse des Backends eingegeben werden, dass zur Simulation genutzt werden soll. So hat man die Möglichkeit, nicht nur auf dem Debug-Server sondern auch auf dem Jenkins- und Release-Server Felder zu simulieren. Nach dem Starten des Simulators werden dann die Events des entsprechenden Servers automatisch heruntergeladen und in dem Dropdown-Menü angezeigt. Wenn ein Event ausgewählt wird, wird die Strecke des Events als roter Polygonenzug auf der Karte dargestellt. Es kann unterhalb der Dropdownliste angegeben werden, wieviele Skater simuliert werden sollen, wie schnell sie fahren, wie groß das Feld ist, also auf welcher Fläche sie sich verteilen sollen, und in welchen Abständen die Position der Skater auf den Server übertragen werden soll. Mit den Schaltfläche „Start“, „Pause“ und „Stop“ kann die Simulation gesteuert werden.
Zur Simulation der Skater wird auf dem Server die Methode *simulateMemberLocations* aufgerufen. Diese nimmt die Positionen für viele Skater gleichzeitig entgegen und ruft anschließend jeweils die eigentliche Methode zur Aktualisierung der Position *updateMemberLocation* auf. Wir haben uns für diese Lösung entschieden, da Probleme auftraten, wenn der Simulator direkt die *updateMemberLocation* -Methode nutzte. Die zu große Anzahl Anfragen hat den kostenlosen

Rahmen der Google App Engine innerhalb einer sehr kurzen Zeit aufgebraucht. Über die Methode zur Simulation der Positionen wird unabhängig von der Anzahl der Skater immer nur eine Anfrage gestellt.

7.4 Task-Tests

von Bernd

7.5 Use-Case-Tests: Veranstalter-App

7.5.1 AddAndDeleteHostTest

von Bernd

7.5.2 AuthenticationOrganizerTest

von Bernd

7.5.3 CreateAndDeleteRouteTest

von Richard

Der *CreateAndDeleteRouteTest* legt, wie der Name schon sagt, eine Route an und löscht sie anschließend wieder. Da es sich um einen Use-Case-Tests handelt, werden die Abläufe vollständig über die GUI gesteuert. Lediglich beim Löschen der Route wird nicht über die GUI-Elemente das Löschen angestoßen, sondern direkt die entsprechende Methode auf dem Fragment zur Verwaltung der Routen aufgerufen. Dies ist notwendig, da wir in den Tests das Auswählen von Optionen in einem Kontextmenü nicht an allen Stellen simulieren konnten. Der weitere Ablauf des Löschens wird aber exakt wie in der Veranstalter-App abgearbeitet.
Im Test wird nach einer kurzen Wartezeit in der initialisierten View auf den Plus-Button im *ManageRoutesFragment* gedrückt. Es öffnet sich dadurch der Dialog, der den Namen der neuen Route entgegennimmt. Nach Eintragen eines Teststrings, wird mit der OK-Taste bestätigt und es öffnet sich der Routeneditor. Nach einer erneuten Wartezeit ist die Route vollständig geladen und es werden zwei Wegpunkte über den Plus-Button in der ActionBar erstellt. Nach Erstellung des ersten Wegpunktes wird die Karte ein Stück verschoben, damit sich der zweite Wegpunkt an einer anderen Position befindet. Durch die Simulation eines Zurück-Tastenklicks wird der Sicherungsdialog angezeigt, der bestätigt und damit geschlossen wird. Der Test wartet dann, bis die neue Route angelegt wurde und löscht diese über den bereits erwähnten Aufruf beim *ManageRoutesFragment* .

7.5.4 PublishNewInformationTest

von Tristan

7.6 Use-Case-Tests: User-App

7.6.1 CreateJoinLeaveDeleteUserGroupTest

von Bernd

7.6.2 SendCurrentPositionTest

von Tristan

7.6.3 SendPositionSettingsTest

von Tristan

7.6.4 ShowSeveralEventsTest

von Tristan