**Tests Einleitung (Reihenfolge, Vorbedingungen, Funktionsweise der Tests)**

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 beide 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).

**Anwendungsfalltests: Funktionsweise**

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 bezieht 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 Löschen der Testdaten vorheriger Tests, da ein der Anwendungsfall mehrere Test Methoden enthalten kann. Zuerst wird der *superclass*Konstruktor aufgerufen (super.setUp) und anschließen folgt der Initialisierungscode. Um die UI ohne Fehler von außen Testen zu können wird der *Touch Mode mit setActivityInitialTouchMode(fasle)* 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 gehört unter anderem, die UI Elemente und das Starten der Activity.

**testUseCase Test**

Alle Tests, die auch die UI Testen laufen auf einem extra davor gesehen Thread, dem *UiThread*. Auf dem alle Interaktionen wie Buttonklicks oder Swipen simuliert werden. 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 übereinstimmen. Ist dies der Fall fährt der Test fort und endet, wenn alle asserts fehlerfrei waren. Sollten 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 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 *tetstStateDestroy* 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 dem „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.

**PublishNewInformationTest**

**Anwendungsfallbeschreibung**

**Veröffentlichen neuer Informationen 2.0**

**Beteiligte Akteure:**

Veranstalter

**Anfangsbedingungen**:

Der Veranstalter hat die „VeranstalterApp“ geöffnet, ist bereits eingeloggt und befindet sich in der HoldTabsActivity auf dem Veranstaltungen Tab.

**Ereignisfluss:**

1. Es wird auf das Tab zum Event erstellen geswipt.
2. Es werden alle Felder für das Event ausgefüllt bzw. neue Eigenschaften hinzugefügt.
3. Über „Erstellen“ wird das Event erstellt.
4. Die Erstellung wird über einen Dialog nochmal bestätigt.
5. Ab dem ersten Schritt wiederholen, um mehrere Events anzulegen.
6. Es wird zu dem Veranstaltungen Tab geswipet.
7. Von den angezeigten Events wird eines ausgewählt.
8. Es wird ein Menü zur Bearbeitung des ausgewählten Events angezeigt.
9. Bereits existierende Informationen können bearbeitet oder über den Button „Editiere Event Eigenschaften“ neue Eigenschaften hinzugefügt werden.
10. Das Event wird über den Speichern Button editiert.

**Abschlussbedingung**:

Neue Veranstaltung anlegen und diese anschließend editieren.

**Ausnahmen**:

Keine

**Spezielle Anforderungen:**

Internet Verbindung

Vorbedingung für den Test ist eine bestehende Internetverbindung, damit das anzulegende Event auf dem Server gespeichert werden kann.

Die Test Daten für die Klasse sind als private, statische sowie konstante Member der Klasse definiert (TEST\_TITLE etc.). Dabei sind die Daten mit dem Präfix *TEST\_* für den normalen useCase Test und *StatePaueResume* Test vorgesehen und die mit dem Präfix *TEST\_STATE\_DESTROY* für den *StateDestroy* Test. Zudem sind die UI Elemente als private Member deklariert.

Der Test läuft über die *HoldTabsActivity* der „VeranstalterApp“. Das hier angelegte Test Event wird an anderer Stelle in einem Test der Nutzer „App“ *ShowServeralEventsTest* verifiziert und wird somit nach Beendigung des Tests nicht gelöscht.

**setUp()**: Hier wird für den Nutzer das Google Konto ausgewählt zu Authentifizierung, zum „Veranstaltungen Erstellen“ Tab geswipt sowie die App gestartet.

**Swipe(Direction):** Da die InstrumentationTestCase2 Klasse keine Methoden zum swipen für das wechseln der Tabs bereitstellt, wird diese Funktionalität hier definiert. Es kann entweder *right* oder *left* (privates enum) der Klasse übergeben werden und führt dann die entsprechende Tätigkeit aus.

**testViews():** Activity, Testet, ob alle UI Elemente, Edits, Buttons, TimePicker etc. initialisiert wurden in der setUp Methode mit Hilfe von *assertNotNull(UIElement)*.

**testViewsVisible():** Verifiziert lediglich ob die Elemente auch auf dem Smartphone Screen sichtbar sind (assertOnScreen bereitsgellte Methode der InstrumentationTestCase2).

**testPreConditions():** Prüft, ob alle Tabas in der ActionBar (mActionBar) enthalten sind.

**testUseCase():** Kann eine Exception werfen auf Grund des UI Threads. Hier wird der Anwendungsfall verifiziert. Hierzu wird auf das Tab zum erstellen geswipt, die Test Daten eingegeben und abgebrochen um auch diese Funktionalität zu testen. Danach werden die Test Daten erneut eingegeben und auf den „Erstellen“ Button gedrückt und mit einem auftauchenden Dialog nochmal bestätigt wird. Es wird geprüft, ob das Event auf dem Server gespeichert wurde. Danach wird zu den Veranstaltungen Tab geswipt und das eben angelegt Event mit einem *Long Touch* zum Editieren ausgewählt. Die Testen zum Editieren werden eingegeben und wie vorhin wird Bestätigt und zu dem Veranstaltung Tab geswipt sowie geprüft, ob das editierte Event auch auf dem Server gespeichert wurde.

**State Tests:** Hier wird getestet, ob die Daten beim Beenden der App aus den Eingabefelder gelöscht werden, falls welche beim Erstellen schon eingegeben worden sind oder bestehen bleiben, beim Pausieren der App.

Anmerkung: Die anderen Tests werden nicht so genau geschrieben, sondern auf JavaDoc und Inline Kommentare verwiesen.

**ShowSeveralEventsTest**

**Anwendungsfallbeschreibung**

**Anzeigen mehrerer Veranstaltungen**

**Beteiligte Akteure:**

Teilnehmer

**Anfangsbedingungen**:

Der Teilnehmer hat die „App“ geöffnet, ist bereits eingeloggt und befindet sich in der S*howEventsActivity*.

**Ereignisfluss:**

1. Es wird eine Liste aller Veranstaltungen angezeigt.
2. Auswahl einer Veranstaltung.
3. Die Veranstaltung wird detailliert mit allen Informationen angezeigt.

**Abschlussbedingung**:

Anzeigen aller Veranstaltungen. Auswahl einer Veranstaltung und Anzeigen von Informationen zu dieser.

**Ausnahmen**:

Keine

**Spezielle Anforderungen:**

Internet Verbindung

Vorbedingung für den Test ist eine bestehende Internetverbindung, damit die Events von dem Server abgerufen werden können.

Für diesen Test muss zuerst der *PublishNewInformationTest* in der „VeranstalterApp“ ausgeführt worden sein. Dieser Test wird in der Nutzer „App“ ausgeführt und soll überprüfen, ob das Event das durch den *PublishNewInformationTest* angelgte Event auch in dieser App abrufbar und Fehlerfrei ist.

Es wird zuerst die App gestartet und die Liste aller verfügbaren Events angezeigt. Es wird das erste Event ausgewählt (welches das aus dem anderen Test sein sollte). Anschließend werden die Details zu dem Event genauer angezeigt in der *ShowInformationAcitivty* und die Daten aus der „Veranstalter App“ verglichen.

**SendPositionSettingstest**

**Anwendungsfallbeschreibung**

**Handhabung der aktuellen Position**

**Beteiligte Akteure:**

Teilnehmer

**Anfangsbedingungen**:

Der Teilnehmer hat die „App“ geöffnet, ist bereits eingeloggt und befindet sich in der S*howEventsActivity*.

**Ereignisfluss:**

1. Es wird mit einem Klick in dem Einstellungsmenü „Einstellungen“ ausgewählt.
2. Es wird die Checkbox zum Übertragen der Informationen an ausgewählt oder auch nicht.

**Abschlussbedingung**:

Die Teilnehmer-Position wird je nach Einstellung an den Server gesendet oder nicht.

**Ausnahmen**:

Keine

**Spezielle Anforderungen:**

Internet Verbindung, GPS

Diese Klasse besteht aus verschiedenen Tests. Den UI Tests und einem Anwendungsfalltest. Die UI Tests dienen lediglich dem Überprüfen der Funktionalität des Einstellungsmenüs. Hierzu wird in der SettingsActivity die Checkbox aktiviert, geprüft ob die Position an den Server gesendet wird. Danach wird die Checkbox wieder deaktiviert, wodurch die Position sich auf dem Server nicht verändert sollte. Vor jedem Test werden die Positionsdaten auf dem Server zurückgesetzt damit es nicht zu Fehlern kommen kann. Die unveränderte Position befindet sich in China. Der Test schlägt nur fehl wenn man sich das Testgerät genau an dieser Position aufhält, da der LocationsTransmitterService die Position nicht auf dem Server verändert.

Der Eigentliche UserCaseTest läuft wie in dem Ereignisfluss in der Anwendungsfallbeschreibung ab.

Zusätzlich gibt es noch einen Teilnehmen *testAttend* Test. Hier wird in der *ShowEventsActivity* gestartet, anschließend ein Event ausgewählt, auf die *ShowInformationActivity* gewechselt und der Teilnehmen Button bestätigt. Vor dem Test wird sichergestellt, dass die Senden in den Einstellungen aktiviert ist. Nachdem wird geprüft, ob an den Server gesendet wird. Dies hat den Hintergrund, da der *LocationTransmitterServer* zum Senden gestartet wird, wenn man an ein Event teilnimmt das bereitsgestartet ist.

**SendCurrentPositionTest**

**Anwendungsfallbeschreibung**

**Übertragung der aktuellen Position an den Server**

**Beteiligte Akteure:**

Teilnehmer

**Anfangsbedingungen**:

Der Teilnehmer hat die „App“ geöffnet, ist bereits eingeloggt und befindet sich in der S*howEventsActivity*.

**Ereignisfluss:**

1. Erfassen der Position in der App mittels GPS bzw. WLAN.
2. Regelmäßige Übertragung der Position an den Server.

**Abschlussbedingung**:

Regelmäßige Übertragung des Teilnehmerstandortes an den Server, wenn der Teilnehmer dies aktiviert hat.

**Ausnahmen**:

Keine

**Spezielle Anforderungen:**

Internet Verbindung, GPS

In diesem Test steht das Senden der Position mit Hilfe des *LocationTransmitterService* im Vordergrund und nicht der Ablauf der Anwendung des Nutzers, d.h. es wird auf eine korrekte Reihenfolge (Starten, Einstellungsmenü) verzichten.

Es wird in der *SettingsAcitivty* gestartet und die Checkbox für die Übertragung aktiviert, worauf hin der *LocationTransmitterService* gestartet wird. Anschließend wird die Checkbox wieder deaktiviert und geprüft, ob der *LocationTransmitterService*  auch wirklich gestoppt wird. Weiterhin wird getestet ob die Position an den Server gesendet wird, selbst wenn die App Pausiert.

Anmerkung: Da das Menü über eine xml. Datei generiert wird, lassen sich die einzelnen UI Elemente nicht genau ansprechen.

**Einstellungen**

Die *SettingsActivity* zeigt dem Nutzer das Einstellungsmenü an. Indem er auswählen kann ob seine Position innerhalb der App an den Server gesendet werden darf oder nicht. Zudem nutzt die *SettingsActivity* ein *PreferenceFragment*. Dies ist für Android 3.0 (API level 11) und höher bestimmt. Für Versionen vor Android 3.0 hätte die *SettingsActivity* vom der *PreferenceActivity* erben müssen, was hier jedoch nicht der Fall ist, da Fragmente eine flexiblere Architektur für die Anwendung bereitstellen im Vergleich zu der Acitivity alleine.

Das hier genutzte Fragment ist das *SettingsFragment*, welches das Menü über eine XML Datei „preferences.xml unter „res/xml/preferences.xml“ generiert.

Für die Einstellungen zu Senden wurde eine Checkbox verwendet, deren Standardwert auf *true* gesetzt ist. D.h. beim erst maligen Starten der App ist das Senden der eigenen Position aktiviert.

Die Einstellungen werden in den *SharedPreferences* als boolean gespeichert unter dem Namen „prefSendLocation“.

**Vorgehensweise**

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 Branchs angelegt.

Für das Vorgehen im Projekt wurde das Rahmenwerk *Scrum* verwendet. Die Rolle des *Product Owners* wurde von den Betreuern übernommen. Außerdem hat die Rollen 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 jeden 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.