TALLINNA TEHNIKAÜLIKOOL

Infotehnoloogia teaduskond

Arvutiteaduse instituut

Robotium integratsioonitestide juurutamine Sentab rakenduse näitel

Bakalaureusetöö

Üliõpilane: Martti Elias

Üliõpilaskood: 095808IAPB

Juhendaja: Maili Markvardt

Tallinn

2015

Autorideklaratsioon

Kinnitan, et olen koostanud antud lõputöö iseseisvalt ning seda ei ole kellegi teise poolt varem kaitsmisele esitatud. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on töös viidatud.

---------- ----------

(kuupäev) (allkiri)

Annotatsioon

Käesoleva bakalaureusetöö „Robotium integratsioonitestide juurutamine Sentab rakenduse näitel“ põhieesmärgiks on integratsioontestide raamistikuga Robotium automaattestide juurutamine. Teiseks eesmärgiks on muuta uute testide loomise ning olemasolevate testide hooldamise lihtsamaks.

Töö tulemusena on Robotium raamistikku kasutades valminud komplekt automaatteste, mis katavad Sentab rakenduse põhifunktsionaalsuse. Testid on jaotatud erinevate vaadete alla. Testide tulemuse kohta genereeritakse raport.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 37 leheküljel, 4 peatükki, 9 joonist.

Abstract

The thesis „Robotium integration tests implementation on the example of Sentab application“ primarily aims to implement integration tests with Robotium framework. Another objective is to simplify the creation of new tests and the servicing of existing tests.

As a result a set of integration tests using Robotium framework has been created, that cover the main functionality. Tests have been ordered under their respective views. A report will be generated displaying the test results.

The thesis is in Estonian and contains 37 pages, 4 chapters, 9 figures.

Sisukord

Sissejuhatus

Tarkvara testimine on tarkvara edu tagamise üks kriitilisemaid etappe. Tarkvara on vaja testida selleks, et ei jääks sisse laiskus- või hooletusvigu ning veakohti, mida pole võimalik ette näha. Olenevalt ettevõtte suurusest ja võimekusest leiavad ettevõtted inimesed, kes sõltumatult arendajatest testivad tarkvara vastavust kokkulepitud kriteeriumitele. Testimine on aeganõudev protsess, mida kiirendavad automaattestid, kui need on väljatöötatud. Automaattestide kaasamine testimisprotsessi aitab aega kokku hoida automaatselt testitud funktsionaalsuste arvelt ja keskenduda katmata funktsionaalsustele. Automaattestide väljatöötamine on aga samuti aeganõudev protsess, kuid pikemas perspektiivis, eriti kui tarkvara muutub keerulisemaks, tagab kiirema testimisprotsessi. Testimise automatiseerimine aga ei taga alati parimat kvaliteeti, sest automaattestid toimivad alati ühtmoodi – nii nagu nad on kirjutatud. Seega automaattestidega tuleb välja võrdlemisi vähe vigu.

Käesoleva töö eesmärk on juurutada integratsioontestimise raamistikuga Robotium, automaatteste Sentab rakenduse testimisprotsessi. Töö lõpptulemuseks on komplekt testjuhtumeid, mille hooldamine ja uute loomine oleks lihtne. Seda plaanin saavutada abiklassidega, milles loon erinevad meetodid, mida oleks võimalik kasutada erinevates testjuhtudes. Hinnanguliselt ei tohiks testjuhtude jooksutamine kesta kauem kui 5 minutit. Esialgselt pole automaatteste nii palju, et neid peaks jooksutama lõunapauside ajal või öösel.

Minu roll Sentabis algas ainsa tarkvara testijana, kuid töömahu ja ettevõtte kasvamine tingis kahe lisatestija vajaduse, kelle juhendajaks mind määrati, et saaksin tegeleda uute testide väljatöötamisega ning vajadusel tagada kasutajatuge.

Töö on jaotatud neljaks peatükiks. Esimeses peatükis antakse ülevaade Androidi testimisest emulaatoriga, füüsilise seadmega testimisest, käsitsi testimisest, testide disainimisest. Teises peatükis antakse ülevaade Sentab rakendusest ja testimise hetkeolukorrast. Kolmandas peatükis antakse ülevaade olemasolevatest vahenditest, kasutatud vahenditest. Neljandas peatükis tuuakse välja testjuhtumid, testraport ning selle analüüs.

1. Ülevaade Androidi testimisest

1.1 Sissejuhatus Androidi testimisse

Cem Kaneri, James Bachi ja Bret Pettichordi sõnul koosneb testimine viiest erinevast dimensioonist[12]:

- Testijad. Kes toodet testivad.
- Kattuvus. Milline osa tootest saab testitud.
- Potentsiaalsed probleemid. Milliste riskide leidmiseks toodet testitakse.
- Tegevused. Mismoodi toodet testitakse.
- Hinnang. Kuidas väljendada toote kvaliteeti.

Testimine on arendusprotsessi lahutamatu osa. Kahjuks pole mittetriviaalsete toodete täielik testimine mõistlik. Seetõttu tuleb testijatel ettenähtud aja lõppedes langetada otsus testimisel saadud informatsiooni põhjal kas toode vastab eesmärgiks seatud kriteeriumitele või mitte. Inglise keeles võetakse testijad kokku kui „*Quality Assurance*“, kuid see pole tegelikult õige viide testijatele. Testijad ei tekita toote kvaliteeti, selleks on arhitektid, analüütikud, arendajad. Kui toode jõuab testijani, siis testija võib teha selle näiliselt katki, kuid tõsiasi on see, et toode on juba enne katki kui see testijani jõuab.

Androidi rakenduste testimine erineb mõneti näiteks personaalarvuti programmide testimisest.

- Ekraani orientatsiooni muutmine. Personaalarvutid on üldjuhul alati ühesuguse orientatsiooniga, aga nutiseadmetel tuleb arvestada orientatsiooni muutmisega.
- Konfiguratsiooni muutus, näiteks klaviatuuri kättesaadavus.
- Rakenduse sõltuvus välistest ressurssidest, interneti olemasolu, SMS, GPS toetus ja muud nutiseadmele omased omadused.

Korduvatel testjuhtudel on märkimisväärne osa testimises. Seetõttu tasub mõelda automatiseerimisele. Automatiseerida tasuks kindlasti funktsionaalsus, mis nõuab andmete sisestamist, muutmist, kustutamist. Automaattestide loomisel tuleb silmas pidada, testide dünaamilisust ning, et neid oleks lihtne hooldada lähtekoodi muutumisel. Samuti tuleks silmas pidada, et uusi testjuhte oleks kerge luua näiteks kasutades abiklassidesse loodud meetodeid. Automatiseerimisega aga ei tohiks üle pingutada. Pole mõtet automatiseerida testjuhte, mida on vaja jooksutada ainult korra.

Androidil on head raamistikud, näiteks Robotium, Espresso, Selendroid, mis võimaldavad testida isegi kasutajaliideses ringiliikumist ja vajutusi. Tuleb silmas pidada, et kasutajaliidest ei saa tervenisti testida automaattestidega. Tulemused on tihti ootuspärased, kuid visuaalselt võib esineda tõrkeid, mida ei ole võimalik ette näha.

Erinevad automaattestimisviisid:

- Ühiktestid – Testimaks ühikuid lähtekoodis, tegemaks kindlaks kas neid ühikuid kasutatakse protseduurides õigesti. Sedasi on lihtne kindlaks teha, kas lähtekood töötab õigesti.
- Sqlite-s tehtavaid toiminguid – Andmetega tehtud toimingud testitakse sqlite-s. Hea raamistik selle testiseks on Robolectric.
- Integratsioon testid – Testitakse mitme mooduli koostööd. Selles töös kasutatakse integratsiooni testimisel Robotium raamistikku.

Järgnevalt kirjeldan emulaatoriga testimise positiivseid ja negatiivseid külgi.

1.2 Emulaatoriga testimine

Emulaatorid on arenenud palju paremaks kui need olid Androidi algusaastatel. Genymotion [8] näiteks käivitub kiiremini kui seda teeb reaalne seade. Seetõttu võib öelda, et testimine emulaatoril on ilmselt kõige levinum viis testimaks rakendusi Androidil. Selle kasuks räägib palju: alustades sellest, et emulaator on kättesaadav ning see on kõige odavam viis testida rakendusi paljudel seadmetel; Emulaatori testitulemused on lähedased reaalse seadme testitulemustele [11]. Testimist ei pärsi mingi konkreetse seadme või Androidi versiooni puudumine.

Emulaator aga pole hõbekuul, ka sellel on omad puudused, mida pole reaalsel seadmel. Emulaatoriga ei saa täielikku kogemust rakenduse kasutamisest.

- Pole õiget ülevaadet, kuidas rakendust oleks kasutada ringi liikumisel, kui palju on žestikuleerimine ja nuppudele vajutamine raskendatud, kui rakendust on tarvis kasutada liikudes.
- Emulaatoriga ei võimalik edasi anda tegelikku ekraani resolutsiooni ja teravust, sellepärast ei saada ka täielikku ülevaadet, kas rakendus näeb võimalikult hea välja nii tippklassi seadmetel kui ka võimalikult vanadel seadmetel.
- Emuleeritud seadme ja füüsilise seadme jõudlus ei pruugi olla sarnane.

Põhilised vead mida emulaatoriga testides ei leia, on põhiliselt kasutajamugavuse probleemid, mobiilse interneti ja wifi vaheldumisel tekkivad probleemid, sissetulevad ja väljuvad kõned.

Sentab rakenduses kasutan emulaatorina Genymotioni tasuta versiooni. Otsustavaks põhjuseks sai kiirus. Võrreldes standard emulaatoriga on Genymotion palju kiirem. Oma testides olen Genymotionit kasutanud ainult selleks, et testida rakendust erinevate ekraani suurustega ning erinevate Androidi versioonidega. Genymotionil on palju funktsionaalsusi, mis on reaalsel seadmel, kuid see on saadaval ainult litsentsi ostmisel – näiteks SMS ja telefonikõned. Sentab rakenduse automaatteste on võimalik jooksutada ka Genymotioniga, orienteeruvalt on ajakulu väiksem kui reaalse seadmega, kuid ilmselt mitte märkimisväärselt. Testid tuleb jooksutada, et anda täpsemat ülevaadet.

1.3 Füüsilise seadmega testimine

Füüsiliste seadmetega testimine on märkimisväärselt kulukam kui see on emulaatoril. Üldjuhul tasub see kulu end siiski ära, sest testitulemused on täpsemad füüsiliste seadmete peal. Arvestades, et rakendused on loodud siiski reaalsete seadmete jaoks, siis on see õigustatud kulu. Kulu saab oluliselt vähendada, kui kasutada füüsiliste seadmete kõrval emulaatorit, mida tegelikkuses ka tehakse.

Füüsilise seadme puhul pole neid puudusi, mis piiravad põhjalikku testimist emulaatoriga [3]:

- On võimalik testida käed-vabad süsteemi kasutamist
- Bluetooth ja USB ühendused
- Mälukaardi olemasolu
- Mobiilse interneti ja wifi vaheldumine
- Pole vajadust eraldiseisvate seadmete jaoks, näiteks veebikaamera ja mikrofon, et simuleerida pildistamist, lindistamist.

Sentab rakenduses eelistan testimist füüsilisel seadmel, sest nutiseadmel on vähem piiranguid võrreldes emulaatoriga ning testitulemused on täpsemad. Saab selge ülevaate kuidas rakendus käitub erinevates olukordades, mis füüsilise seadme peal võivad esineda. Kasutamismugavust saab täpselt hinnata. Sentab rakenduse automaattestide ajakulu võrreldes käsitsi testimisega on märkimisväärne. 49 testi läbimiseks kulub umbes 7 minutit, mis on võrreldes käsitsi testimisega 2 korda kiirem. Lisaväärtust lisab see, et samaaegselt on võimalik testida funktsionaalsusi, mida automaattestid ei kata. Sentab rakenduses on kasutatud palju väliseid laiendeid, siis võtab iga automaattestide jooksutamine tavapärasest kauem aega, sest rakendus on vaja valmis ehitada. Ajavõiduga on võimalik keskenduda testidele, mida ei automatiseeritud selle võimalikkusel või vajalikkusel.

1.4 Käsitsi testimine

Suur osa vigu tuleb välja just käsitsi testimisel, põhiliselt sellepärast, et kõiki vigu pole võimalik ette näha. Üldiselt sellised vead on visuaalsed, nupp on vale paigutusega, vaade pole õige suurusega, kusagil on midagi üleliigset jäänud. Selliseid testjuhte ei automatiseerita, sest need oleksid hoomamatud ja hooldamatud. Selle jaoks vaadeldakse rakendus kindlasti läbi ka käsitsi testides. Ainult käsitsi testimisel saab tajuda kasutamismugavust. Käsitsi testimine on aeganõudev tegevus, kus samade testide läbimine võrreldes automaattestidega võib võtta mitmeid kordi kauem, olenevalt testide keerukusest ja nende hulgast. Märkimisväärseima ajavahe tekitab andmete sisestamine, kus Androidi puhul tuleb oodata, kuni klaviatuur tuleb nähtavale ning alustada aeganõudvat sisestamisprotseduuri.

Käsitsi testimist ei saa võrdsustada automaattestidega ainuüksi juba sellepärast, et käsitsi testimisel on võimalik tähele panna vigu, mida ei olnud võimalik ennetada. Automaattestides tehakse täpselt etteantud liigutused iga kord samamoodi. Inimese poolt läbi viidud testid aga varieeruksid mingil määral ning just nendest variatsioonidest võivad välja tulla avastamata vead.

Sentab rakenduses kasutan käsitsi testimist seal, kus automaatteste Robotium ei toeta. Näiteks piltide ja videote üleslaadimisel, samuti häälsõnumite saatmisel. Robotium ei toeta neid sellepärast, et piltide ja videote üleslaadimiseks kasutab Sentab rakendus Androidi galeriid, mis on eraldiseisev rakendus. Häälsõnumite puhul aga ei saa sisendiks anda mingisugust heli, seega ei saa olla kindel kas sõnum on korrektselt salvestatud.

1.5 Testide disainimine

Testide disainimiseks on palju erinevaid meetodeid, kõik lähenevad erinevalt ning paljastavad erinevaid vigu. Kõige levinumad disainimistehnikad on kogemuse põhine testimine, piirjuhud, funktsionaalne-, loogika- ja „smoke“ testimine. Uute funktsionaalsuste juures testitakse kindlasti kasutatavust, kuid sellele rõhutakse kõige rohkem rakenduse alguses ning siis kui lisatakse mingi suurem funktsionaalsus kus kasutamismugavus on oluline

Sentab rakenduses testide disainimisel olen kasutanud piirjuhte, kuna piiridel tuleb kõige rohkem vigu esile. Samuti on kasutuses funktsionaalne-, loogika- ja „smoke“ testimine ning mitteformaalse testimismeetodina on kasutused uuriv testimine.

Kogemuspõhine testimine[10] – see tehnika põhineb testija teadmistele, oskustele ja tema taustale. See tehnika paneb aluse kahele olulisele testimismeetodile: veaarvamine ja uuriv testimine. Mõlemate meetodite abil üritatakse leida võimalikult palju vigu sealjuures vähe planeerides. Neid meetodid on mitteformaalsed ja peaksid olema kasutusel koos formaalsete testimismeetoditega.

Funktsionaalne testimine – tähendab Sentab rakenduses piltide ja videote üleslaadimist, vaatamist, nende andmete muutmist, kustutamist; sõnumite saatmist, kustutamist, kasutajate vahetamist. Üldiselt tähendab funktsionaalne testimine rakenduse funktsionaalsuste testimist, mis on leitavad dokumentatsioonis.

Loogika testimine – Sentab rakenduses tähendab, kasutajad peavad olema aktiivsed ja sisse logitud, et rakenduse funktsioone kasutada, teiste kasutajate poolt jagatud meediume ei saa muuta ega kustutada. Põhjuse-tagajärje graaf (*Cause-effect graph*) on levinud tehnika põhjaliku loogika põhise testide disainimisel.

„Smoke„ testimine – Sentab rakenduses tähendab erinevate sisendite ja olukordade tekitamist, mida eeldatavasti tavakasutaja ei sisesta. „Smoke“ testimist kasutatakse põhiliselt põhifunktsionaalsuste juures, tegemaks kiiresti, kuid pealiskaudselt kindlaks, et toode pole juurdearenduse käigus katki läinud.

Musta kasti testimine – rakendan Sentab rakenduses musta kasti testimist, sest testin ootuste ja spetsifikatsiooni põhjal. Kasutan ka uurivat testimist – alustades kõige elementaarsematest funktsionaalsustest ja liikudes järjest sügavamale. Nõnda on lihtne tagada sõltumatust arendajatest ja võib leida palju vigu, millele arendajad ei ole mõelnud. Kuid sellist meetodit kasutades võib

teadmatusest kirjutada mitmeid testjuhtumeid, mis tegelikult testivad täpselt sama asja ning osad funktsionaalsused võivad jääda üldse testimata.

Kasutatavuse testimine – Sentab rakenduses, kuid ka üldiselt, tähendab see kasutamismugavust ning arusaadavust rakenduse lõpp-kasutajale. Sellist testimist tehakse käsitsi ning arusaamatused ja ettepanekud tuleks dokumenteerida.

2. Sentab rakendus ja selle hetkeolukord

2.1 Sentab rakendus

Sentab on ettevõte, kes arendab tarkvara, mis on tehtud võimalikult lihtsaks, et eakad, kes ei kasuta arvutit ega nutiseadmeid ei kardaks süsteemi kasutada. See tarkvara võimaldab neil lihtsalt suhelda oma pere ja sõpradega, võimaldab perel saata pilte, videoid ja sõnumeid. See toode töötab Androidi seadme peal, mis ühildub televiisoriga. Sentab eesmärgiks on vähendada eakate üksildustunnet [19] ning anda perele parem ülevaade nende lähedaste käekäigust statistika näol. Sentab süsteemiga on võimalik olla sotsiaalne oma kodu mugavuses. Käesolevas töös aga käsitlen kõrvaltoodet – Sentab rakendus, mis on mõeldud pereliikmetele, kes kasutavad nutiseadmeid.

Sentab rakendus on loodud selleks, et jätta välja tülikas sisselogimine veebiliidesesse ning nutiseadmest piltide või videote kopeerimine arvutisse, mida saaks jagada oma pereliikmega, kes kasutab Sentab süsteemi. Rakendusega on võimalik lisaks piltide ja videote jagamisele ka hääl- ja tekstisõnumeid vahetada.

2.2 Sentab rakenduse testimise hetkeolukord

Ettevõtte siseselt on kasutusel ideaalis igakuine reliisitsükkel, kuid reaalsuses on see tavaliselt kuni kahekuune. Põhjuseks on ettenägematud vead ja komplikatsioonid, mis pahatihti pikendavad arendusprotsessi. Kuna Sentabil on hetkel peaaegu kõik testid mõeldud käsitsi testimiseks, siis algab testimine esimesel võimalusel, kui arendaja on funktsionaalsuse üle andnud. Vigade olemasolul teavitatakse arendajat kohe, kui *smoke* testimist ei läbitud, vastasel juhul koostatakse jooksvalt vearaporteid. Kui reliisitsükliks mõeldud funktsioonid on valminud ning avastatud vead on parandatud, tehakse reliis iseseisvasse keskkonda viimaseks põhjalikuks testimiseks, enne kui uued funktsionaalsused tehakse klientidele kättesaadavaks. Selles iseseisvas keskkonnas on testijatel aega tavaliselt üks nädal, et vead avastada ning anda hinnang, kas funktsionaalsused vastavad kriteeriumitele.

Sentab rakenduses on automaattestid olemas põhifunktsionaalsustel, mida on võimalik Robotiumiga testida. Enne uute testijate lisandumist jäid automaattestid suurenenud töömahu tõttu hooldamata ning neid polnud võimalik enam kasutada, sest nende hooldamine oli aeganõudev pidevalt muutuvate põhifunktsionaalsuste tõttu. Sellest tekkis idee, et testjuhtumite hooldamine võiks olla võimalikult lihtne.

Hetkel on Sentab me toodetega jõudnud nii kaugele, et automaattestide olemasolu vähendaks testimisprotsessi läbiviimisele aega, mis omakorda võimaldaks viia läbi uusi teste ning olemasolevaid teste täiendada. Olemasolu kiirendaks ka arendusprotsessi, käsitsi testimise arvelt. Aeg-ajalt juhtub, et arendajatel on uute funktsionaalsustega mingi olemasolev funktsionaalsus katki läinud. Seetõttu on

hea, kui vähemalt kõige põhilisemad funktsionaalsused oleks automaattestidega kaetud, eriti üha keerukamaks muutuvas süsteemis.

Testjuhtumite juurdearendamine ja hooldamine tuleks teha võimalikult lihtsaks, mis tähendab ajaliselt seda, et iga reliisi puhul võiks orienteeruvalt testide hooldamisele kuluda kuni päev ning uute testjuhtumite lisamiseks võiks kuluda kuni paar päeva. Sellega suudaks vältida muude kohustuste massilist kuhjumist. Juurdearendamine peaks olema jõukohane inimestele, kes on juba kokku puutunud Robotiumiga. Hooldamise käigus saaks lihtsasti Robotiumi kasutamise selgeks ka inimesed, kes sellega pole enne kokku puutunud.

3. Ülevaade automatiseerimisvahenditest

3.1 Töövahendid

Automatiseerimise, arendamise ja nende abistamise tööriistu, mida võiks kasutada, on väga palju. Järgnevalt on kirjeldatud töövahendid, mis on põhiliselt mõeldud Androidi arendamiseks ja testimiseks.

Robotium [17]– on avatud lähtekoodiga automaattestide raamistik Androidil, mis toetab nii kohalikke (*native*)kui ka hübriid rakendusi. Robotium teeb lihtsaks kasutajaliidese musta kasti testimise. Robotiumi kasutamiseks pole tarvis laialdasi teadmisi testitava rakenduse kohta. Robotiumi raamistikul põhinevad ka selles töös olevad testjuhtumid, nii Robotiumist kui ka testjuhtudest on pikemalt juttu all pool. Koodi mitteteadmine oli põhiline põhjus, Robotiumi raamistikku kasutamiseks automaattestide loomisel.

Robolectric [16]– Ühiktestide raamistik. Robolectricu teste on võimalik jooksutada Java virtuaalmasinas väga kiiresti, sest ei ole tarvis emulaatorit või reaalset seadet, et teste jooksutada. Testide jooksutamine Java virtuaalmasinas on Robolectricu põhiline pluss, lisaks sellele on olemas variobjektid, mis teevad lihtsaks osade android objektide testimise, mis ilma variobjektideta oleks väga keeruline. Nimelt on variobjektidel meetodeid, mida tegelikel objektidel pole. Võimaldab küsida variobjektilt tema tegelikku väärtust, mis tegeliku objekti puhul poleks võimalik, kuna küsimismeetodit ei ole implementeeritud. Koodi laialdane teadmine ning kasutajaliidese testimise puudumise pärast jäi Robolectricu raamistik tööst välja.

Espresso [6]– on Google poolt loodud avatud lähtekoodiga raamistik. Nii Robotium kui ka Espresso on instrumentatsiooni põhine raamistik, mis tähendab, et nad kasutavad Androidi instrumentatsioone, et inspekteerida ja suhelda testitavate tegevustega (*activities*). Suur eelis Robotiumi ees on sünkronisatsioon. Kuna vaikimisi jooksevad instrumentatsiooni testid ja kasutajaliidese operatsioonid eraldi lõimedes, tekivad sellest testide ebastabiilsus. Näiteks suvalistel hetkedel test ebaõnnestub, sest objekt polnud veel kättesaadav. Espresso oli üks valikuvariantidest, mis oleks olnud hea alternatiiv Robotiumile, kuid Sentab rakenduse puhul ei olnud seda raamistikku võimalik tööle saada. Seetõttu ei olnud võimalik seda valida.

Genymotion – parem alternatiiv standardemulaatorile, mis on juba käivitamisel 4 korda kiirem kui seda on Androidi standardemulaator. Lisaks sellele on ta ka käivitatuna stabiilsem ja kiirem. Antud töös on Genymotionit kasutatud põhiliselt selleks, et testida odavalt erinevatel Androidi versioonidel.

Android Studio [2]– on integreeritud arenduskeskkond Androidi jaoks. See vahetas välja Eclipse-i Androidi arendus töövahendid, mida Google kasutas eelnevalt peamise arenduskeskkonnana. Android Studios kasutatakse ehituse automatiseerimisel Gradle-t ning ehitusvariante mis võimaldavad apk-sid lihtsasti ehitada erinevate keskkondade vastu. Kuna Android Studio on ametlik Androidi

arenduskeskkond ning parem alternatiiv Eclipse ADT-le, siis sellepärast on seda antud töös ka kasutatud.

ADB [1]– Android Debug Bridge, on klient-server programm, mis sisaldab endas kolme komponenti:

- Klienti, mis jookseb arendusmasinas.
- Serverit, mis jookseb arendusmasina taustal. Server haldab kliendi ja deemoni suhtlust.
- Deemonit, mis jookseb Androidi seadme taustal.

ADB on arendajatele, et nad saaksid rakendustes vigu parandada.

Logcat [14]– Androidi logimisvahend, näitamaks kogutud infot süsteemi tööst. Logisid saab koguda üle terve süsteemi ning vastavalt vajadusele filtreerida. Logcati abil on lihtsam leida vigased kohad rakenduse koodis kui neid esineb.

Mockito [15]– avatud lähtekoodiga testimisraamistik Javale. Mockito põhieesmärk on võimaldada testide kirjutajal kas osaliselt või täielikult üle kirjutada konkreetset objekti, eesmärgiga selle funktsionaalsus antud testi kontekstis lukustada. Osaliselt üle kirjutamine tähendab, et on võimalik üle kirjutada konkreetseid meetodeid objekti küljes, jättes objekti ülejäänud käitumine selliseks, nagu ta peab olema toodangukoodis. Osaliselt objekti üle kirjutamine võimaldab arendatud koodi testida väiksemate meetodite testimise kaudu, kirjutades üle kõik meetodite tulemused, mida testitav meetod sisemiselt välja kutsub. Mockito pakub ka võimalust osaliselt või täielikult üle kirjutatavate objektide peal tehtavaid väljakutseid verifitseerida. Vajadusel saab kontrollida, et konkreetsed meetodite väljakutsed on tehtud kindlas järjekorras. Verifitseerimisel on võimalus kontrollida, et meetodite sisendiks on kindlad parameetrid Kui tekib olukord, kus meetodi väljakutsumisel kaasaantav parameeter on testis kättesaamatu, saab selle kinni püüda Mockito poolt pakutava argumentide kinnipüüdjaga. Kinnipüütava argumendi tüübi jaoks tehakse vastav kinnipüüdja ning meetodi väljakutse verifitseerimisel antakse kinnipüütava parameetri asemel kaasa vastava tüübi jaoks tehtud argumentide kinnipüüdja, mille käest saab hiljem omakorda küsida huvipakkuva parameetri väärtust ja seda vastavalt vajadusele omakorda verifitseerida. Kuna Mockito põhjalikuks kasutamiseks on tarvis teada rakenduse koodi, siis sellepärast jäi Mockito valikust välja.

UI Automator Viewer [18]– töövahend leidmaks kasutajaliidese objektide hierarhiat ja neile vastavaid identifikaatoreid Androidi rakenduses. Eriti kasulik on selline töövahed musta kasti testimis meetodit kasutades, kus lähtekoodile pole ligipääsu. Seda töövahendit on käesolevas töös kasutatud palju objektide identifikaatorite teada saamiseks.

BrowserStack [4]– on mitmeplatformiline töövahed veebi testimiseks. Lisaks arvutis olevate veebilehitsejatele on võimalik ka veebilehte testida erinevate platvormidega mobiilsetel seadmetel. Veebilehitsejaid on võimalik kasutada nii interaktiivselt kui ka Selenium automaattestidega.

BrowserStacki näol on tegemist põhiliselt veebi testimise töövahendiga, siis BrowserStack pole võimalik kasutada Sentab rakenduse testimises.

Testjuhtude lindistajad– Testjuhtude lindistajad on mõeldud selleks, et testjuhtude loomine oleks võimalikult lihtne. Kõik, mida on tarvis teha, on alustada lindistamist ning sooritada testjuhu jaoks vajalikud toimingud. Sellised testjuhtude lindistajad Robotiumilt, Ranorexilt on tasulised, mistõttu arendajad ilmselt eelistavad pigem raamistikke, mis oleksid tasuta.

3.2 Kasutatud töövahendid

Järgnevalt on kirjeldatud töös kasutatud töövahendid täpsemalt, mille abil on loodud töös testitav tarkvara kui ka töö tulemusena valminud automaattestid.

3.2.1 Robotium

Robotium on avatud lähtekoodiga testimisraamistik Androidi rakenduste jaoks. Seda on võimalik kasutada nii lähtekoodi olemasolul kui ka selle puudumisel. Robotiumit kasutatakse nii funktsionaal-, süsteemi- kui ka integratsioontestimiseks läbi erinevate tegevuste. Testimisraamistiku loogika põhineb instrumentatsioonidel, mis ei tööta samas lõimes kui seda teevad kasutajaliidese operatsioonid. See tähendab, et sünkroniseerimisel võivad ning aeg-ajalt esinevad vead, mis on tingitud ajastamisest. Suuresti sellepärast kukuvad testid läbi, mis tegelikkuses töötavad õigesti. Robotium on kasutusele võtnud selle vähendamiseks ooteajad ning uuesti proovimise mehhanismid. See siiski kõiki juhte ei välista ning aeglustab testide jooksutamist. Antud töös on Robotium kasutusel integratsiooni testide jaoks, see tähendab, et testitakse mitme komponendi koostööd omavahel.

Robotiumi testklassid peavad laiendama *ActivityInstrumentationTestCase2* klassi, mis võimaldab defineerida testjuhte olemasolevatele ja etteantud tegevusele. Robotiumi mitmekesised meetodid on klassis *Solo*. Sellel testimisraamistikul on suur puudus, pole võimalik testidesse kaasata teisi rakendusi, see tuleneb instrumentatsiooni raamistiku piirangust. Sentab rakenduse puhul ei ole võimalik testida pildistamist ega piltide üleslaadimist. Tema suurimaks puuduseks on kiirus, nagu ka teiste testimisraamistikega, millel on vajadus rakendus installeerida ja käivitada ning kõik sammud seadmel läbida.

Selle raamistiku kasutamisel pole vaja palju teadmisi testitava rakenduse kohta. Koodi mitteteadmise puhul on lihtne teha testjuhte ühe konkreetse seadme jaoks, kasutades meetodeid, mis kasutavad väärtuseid, mis on silmaga nähtavad ning ei tohiks muutuda, näiteks nuppude nimed, veateated jne. Kui aga rakendus on mitmekeelne, siis ei piisa sellest, et on teada silmaga nähtavad nuppude nimed, veateated jne. Testjuhtude kirjutamiseks, mis oleksid dünaamilised on tarvis kasutada objektide

identifikaatoreid, mis üldjuhul ei muutu, isegi kui funktsionaalsusi muudetakse ning tehakse juurdearendusi.

Vaikimisi jooksutatakse teste tähestikulises järjekorras. Mingi kindla järjekorra tekitamiseks luuakse testide komplekt, milles saab hõlpsalt määrata ning muuta testklasside järjekorda. Testjuhtude loetavuse ning ka töö kiirendamiseks on soovitatav kirjutada abiklass meetodite jaoks, mida kasutatakse mitmes testjuhtumis. Seda on tehtud ka antud töös, kus abiklassiks on *Utils.java*, milles on universaalsed meetodid.

3.2.2 Android Studio

Android Studio on praegune ametlik Android arenduskeskkond, mis vahetas välja eelmise arenduskeskkonna Eclipse-i poolt. Android Studio põhineb IntelliJ IDEA-le, millele on lisatud palju funktsionaalsusi Androidi arendamiseks. Ekraanitõmmis on väljatoodud lisas 2.

- Ehitus põhineb Gradle-l
- Ehistusvariandid
- ProGuard
- Lint
- Toetus Android Wear rakenduste jaoks
- Jpm.

Gradle [7]– on projekti automatiseerimise töövahend. Gradle on disainitud mitmeprojektiliste ehituste jaoks, mis võivad kiiresti kasvada suureks. Gradle toetab astmelisi ehitusi, tehes kindlaks milliseid osasid on tarvis uuendada ning millised on osad võib jätta uuendamata.

Ehitusvariandid [5]– on mõeldud selleks, et oleks lihtne luua ühe projekti põhjal erinevate seadetega rakendused. Näiteks erinevate keskkondade vastu või isegi kärbitud funktsionaalsusega.

ProGuard – on integreeritud Androidi ehitussüsteemi. See käivitub ainult siis kui rakendus ehitatakse reliisi jaoks. Seda on võimalik välja lülitada kuid pole soovitatav. ProGuard lisab turvalisust, et rakendust oleks raskem pöördprojekteerida (*reverse engineer*). Lisaks sellele kahandab, optimeerib ja kaotab kasutamata koodi, tehes nõnda rakenduse fail väiksemaks.

Lint [13]– Töövahend, mis skaneerib projektis potentsiaalseid vigu ja parandab optimeerimist. Android Studios käivitatakse lint automaatselt kui rakendus kompileeritakse.

Võrreldes Eclipse ADT-ga, eelmise põhilise Android arenduskeskkonnaga, on Android Studio stabiilsem ning jõulisem. Ehitusprotsessid on kiiremad ning ei ole tarvis aeg-ajalt keskkonda taaskäivitada. Lisaks kõigele ei ole vaja muretseda, et tehtud muudatused läheksid kaduma – Android Studio salvestab kõik tehtud muudatused automaatselt.

3.2.3 Genymotion

Genymotion on prantsuse startupi loodud hea alternatiiv standardsele Androidi emulaatorile. Selline emulaator on olnud kättesaadav 2013 aasta lõpust ja on leidnud laialdaselt arendajate poolehoidu selle aja jooksul. Genymotioni sõnul kasutab 2.5 miljonit arendajat Genymotioni emulaatorit [9]. Kasutades mõlemat pole raske näha, miks nii paljud eelistavad Genymotionit standardse emulaatori üle. Genymotionil jooksva Sentab rakenduse ekraanitõmmis on näha lisas 3.

Ta on kasutatav nii Eclipse-s kui ka Android Studios pistikprogrammi vahendusel. Genymotioni teeb jõuliseks tema x86 arhitektuuri virtualiseerimine, Genymotion töötab Oracle Virtualboxiga. Kui võrrelda standardemulaatorit ja Genymotionit, siis on silmaga näha nende jõudluse vahet, kus Genymotioni puhul pole näha nii palju kasutajaliidese ebaühtlast liikumist.

Genymotion toetab praegusel hetkel levinud Androidi versioone alustades 2.3-st kuni 5.0-ni. Sellest emulaatorist on kaks levitatavat versiooni, tasuta ning tasuline. Tasuta litsentsi alusel on võimalik kasutada emulaatorit ning lisaks sellele on võimalus kasutada GPS ja kaamera funktsionaalsust. Tasulise litsentsi alusel on funktsionaalsused silmapaistvamad, alustades akselerомeetriga, mida nad võimaldavad kasutades arvutiga ühendatud reaalset seadet ning lõpetades SMS ja kõnede emuleerimisega. Genymotioni kasutamise teeb mugavaks ka lihtne akende suuruse muutmine ning *drag and drop* funktsionaalsus, mis standardsel emulaatoril puudub.

3.2.4 UI Automator Viewer

UI Automator Viewer on Android SDK-sse sisseehitatud. Tehes UI Automator Viewer-iga ekraanitõmmis on lihtne teha kindlaks selles vaates olevate objektide identifikaatorid nende hierarhia ning nende omadused, eriti kui rakenduse kood on kellegi teise poolt kirjutatud. Saades teada objektide identifikaatorid või isegi hierarhia, on lihtsam kirjutada täpsemaid testjuhte. UI Automator Viewer-i ekraanitõmmis on väljatoodud lisas 4.

Selles töös on kasutatud UI Automator Viewer-it täpselt nendel eesmärkidel, sest seda töövahendit kasutades on palju lihtsam leida identifikaatoreid, kui seda oleks otsida koodist. Seetõttu on saadud kirjutada täpsemaid testjuhte.

4. Testlood

Järgnevalt on välja toodud testjuhud, mis on valminud selle töö raames ning kirjeldused, mida antud testjuhud teevad ning kuidas neid kontrollitakse. Samuti on välja toodud abiklass, milles peituvad meetodid, mis on Sentab rakenduse testimise lihtsustamiseks kirjutatud. Lõpetuseks on näidatud, missugune on testraport antud testide jooksutamisel.

4.1 Testlugude kirjeldus

Testid on jagatud vastavalt funktsionaalsusele eraldiseisvateks testikomplektideks. Testlood on väljamõeldud enne kui juhtumeid testima hakati.

LogoutTests – Koosneb 1-st testist.

- LogoutCorrectly() – logib rakendusest välja, kontrollib *Activity* vahetumist.

PasswordRecoveryTests – Koosneb 4-st testist.

- PasswordRecoveryActivity() – Kontrollib, kas oodatud *Activity* on aktiivne.
- RecoverPasswordWithValidEmail() – üritab salasõna taastada korrektse e-mailiga.
- RecoverPasswordWithInvalidEmail() – üritab salasõna taastada vale e-mailiga, kontrollitakse kirja kuvamist.
- RecoverPasswordWithoutInternet() – üritab salasõna taastada interneti ühenduse puudumisel, kontrollitakse *TOAST* sõnumi kuvamist. Sellised sõnumid lihtsa tagasiside saamiseks, mis ei vaja kasutaja poolset tegevust.

LoginActivityTests – Koosneb 4-st testist.

- LoginActivityActiveness() – kontrollib õige *Activity* aktiivsust.
- IncorrectEmailLogin() – üritab vale e-mailiga sisse logida. Kontrollitakse *Activity* samaksjäämist. Test kukub läbi, kui see muutub.
- IncorrectPasswordLogin() – üritab vale salasõnaga sisse logida. Kontrollitakse *Activity* samaksjäämist. Test kukub läbi, kui see muutub.
- CorrectLogin() – üritab õige e-maili ja salasõnaga sisse logida. Kontrollitakse *Activity* muutumust. Test kukub läbi, kui see jääb samaks.

DrawerActionTests – Koosneb 2-st testist.

- ChangePatient() – üritab vahetada erinevate kasutajate vahel. Kontrollitakse, et oodatud kasutaja osutub valituks.
- ChangeBetweenAlbumAndVideos() – üritab vahetada erinevate vaadete albumite ja videote vaadete vahel. Kontrollitakse, et albumid osutuvad valituks.

GalleryViewTests – Koosneb 6-st testist.

- OpenAlbums() – üritab avada Albumite vaadet. Kontrollitakse, et albumite vaade oleks aktiivne.
- ViewImage() – üritab vaadata pilti. Kontrollitakse, et pildi vaade oleks kuvatud.
- SetThumbnail() – üritab muuta *thumbnail*-i. Kontrollitakse, et valitud pildi *thumbnail* oleks albumi uus *thumbnail*.
- ChangeTitle() – üritab muuta olemasoleva albumi nime. Kontrollitakse, kas albumi nimi on edukalt muudetud.
- ChangeDate() - üritab muuta olemasoleva albumi kuupäeva. Kontrollitakse, et muudetud albumi kuupäev oleks muudetud.
- ChangeDateAndNameReset() – üritab muuta albumi nime ja kuupäeva tagasi, et testi saaks korduvalt jooksutada kõrvalise sekkumiseta. Kontrollitakse, et nime ja kuupäeva muutus oleks edukas.

AlbumViewTests – Koosneb 9-st testist.

- AddNewAlbum_DefaultDate() – loob albumi vaikimisi kuupäevaga. Kontrollib, kas album on loodud.
- AddNewAlbum_WithPastDate() – loob albumi minevikus oleva kuupäevaga. Kontrollib, kas album on loodud.
- RotateScreen_AddNewAlbum() – üritab luua albumi ekraani orientatsiooni muutes. Kontrollitakse uue albumi olemasolu.
- AddAlbumWithLongName() – üritab luua pikema kui 20 tähemärgiga albumit. Kontrollib sellise albumi puudumist. Rakenduses ei võimaldata lisada rohkem kui 20 tähemärki albumi nimeks.
- EnterGallery() –test navigeerib nimekirjas esimesse albumisse. Kontrollib galerii vaate olemasolu.
- NavigateBack() – navigeerib galeriist tagasi kasutades rakenduse sisest „tagasi“ nuppu. Kontrollitakse tegevusnupu olemasolu.
- NavigateBackButton() – test navigeerib samuti galeriist tagasi, kuid kasutades Androidi enda „tagasi“ nuppu. Kontrollitakse tegevusnupu olemasolu.
- CancelAddingNewAlbum() – test katkestab uue albumi lisamise. Kontrollitakse uue albumi lisandumist. Test kukub läbi kui uus album on lisandunud.

- RemoveAlbums() – test kustutab selles testklassis loodud uued albumid. Test kontrollib albumite arvu vähenemist vastavalt kustutatud albumite arvule.

ActionBarButtonTests – Koosneb 2-st testist.

- RotateSCreen() –test muudab n-korda ekraani orientatsiooni. Kontrollitakse, kas ekraan muudab orientatsiooni.
- Refresh() – Uuendab vaadet. Kontrollitakse, kas nupule vajutati.

VideoViewTests – Koosneb 3-st testist.

- OpenVideos() –avab videote vaate. Kontrollitakse, kas videote vaade on aktiivne.
- EditNameAndRotateScreen() –üritab muuta olemasoleva video nime, samaaegselt muutes ekraani orientatsiooni. Kontrollitakse, kas nimemuutus oli edukas.
- EditToLongName() – test üritab muuta olemasoleva video nime pikemaks kui 20 tähemärki, kontrollitakse, kas video nimi muudeti pikemaks kui 20 tähemärki ning kas uus nimi lõigati 20-nda tähemärgi juurest ära.

MessageTests – Koosneb 7-st testist.

- openMessages() – üritab avada sõnumite vaate. Kontrollitakse, et sõnumite vaade oleks aktiivne.
- NoConnectionSendMessage() – üritab saata sõnumit interneti ühenduseta. Kontrollitakse, interneti puudumise teate kuvamist.
- TrySendingUnsentMessageAgain() – üritab saata ebaõnnestunud sõnumit uuesti. Kontrollitekse veateate kuvamist.
- DeleteUnsentMessage() – üritab ebaõnnestunud sõnumi kustutada. Kontrollime, et sõnumite arv väheneks kustutatud sõnumi võrra.
- CancelSendingMessage() – üritab sõnumi saatmise katkestada. Kontrollitakse, et sõnumite arv ei oles muutunud.
- SendMessage() – üritab sõnumit edukalt saata. Kontrollitakse, et sõnumite arv suureneks ühe võrra.
- SendTextMessage() – üritab saata tekstisõnumit. Kontrollitakse, et sõnumite arv suureneks ühe võrra.

EventActionTests – Koosneb 4-st testist.

- openEvents() – üritab avada ürituste vaate. Kontrollitakse, ürituste vaade oleks aktiivne.

- ChangeEventType() – üritab vahetada vastavalt aktiivsusele ürituste kuvamise tüübi. Kontrollitakse, et ürituste tüüp oleks vahetatud.
- SelectSecondDay() – üritab vahetada nädalapäeva. Kontrollib, et valitud päeva väljendav näidik oleks õigel päeval aktiivne.
- ScrollWeek() – üritab kuvada järgmist nädalat. Kontrollitakse, et ürituste nädal oleks vahetunud ja päeva väljendav näidik oleks õigel päeval aktiivne.

GeneralPageTests – Koosneb 7-st testist.

- OpenGeneral() – üritab avada üldise lehe. Kontrollitakse, et üldine leht oleks aktiivne.
- MedicationStatistics() – üritab avada medikamentide detailsema statistika vaate. Kontrollitakse, et nupule oleks vajutatud.
- MoodStatistics() – üritab avada tuju detailsema statistika vaate. Kontrollitakse, et nupule oleks vajutatud.
- ActivityStatistics() – üritab avada aktiivsuse detailsema statistika vaate. Kontrollitakse, et nupule oleks vajutatud.
- InviteValidFamilyMember() – üritab saata kutse korrektsele e-mailile. Kontrollitakse, et pärast saatmist kuvatakse üldist lehte.
- InviteInvalidFamilyMembers() – üritab saata kutse vale vormiga e-mailile. Kontrollitakse, et kuvatakse veateade.
- InviteMultipleFamilyMembers() – üritab saata kutsed mitmele korrektsele e-mailile. Kontrollitakse, et pärast saatmist kuvatakse üldist lehte. Selle testloo kirjeldaksin ka täpsemalt järgnevalt lahti koos ekraanitõmmistega, mis on kujutatud joonisel 1.

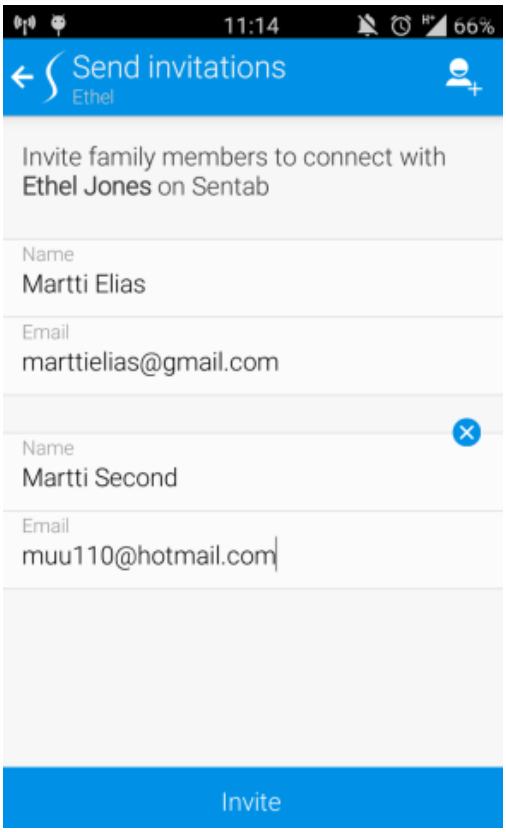
Testjuht väärtustab eelnevalt väärtustatud nime ja emaili stringile lisaks veel ühe nime ja emaili stringi paari. Käivitab abiklassis *Utils* meetodi *inviteMultipleFamilyMembers* (meetod on näidatud joonisel 2) deklareeritud nime ja emaili paaridega. Abiklassi meetodis väärtustatakse *view general_page_family_invite*-iga, millele pärast seda vajutatakse. Järgmisena väärtustatakse *view action_add*-iga, millele pärast vajutatakse ning liigutakse ekraanil üles. Sooritatakse poole sekundiline paus, et oleks võimalik ekraanil oodatud kohtadele vajutada ning sisestada kaasa antud nime ning emaili paarid (joonisel 3 on kujutatud täidetud väljad). Seejärel vajutatakse *view*-le *invite_btn*. Me teame, et edukalt saadetud kutsete puhul viiakse kasutaja tagasi üldisele lehele, kus kuvatakse kasutaja pilti. Sellepärast me saame kontrollida, kas meile kuvatakse kasutaja pilti.

```
public void  test7_inviteMultipleFamilyMembers(){
    String name2 = "Martti Second";
    String email2 = "muu110@hotmail.com";
    util.inviteMultipleFamilyMembers(solo, name, email, name2, email2);
    assertTrue(solo.waitForView(R.id.general_page_picture_thumb));
}
```

Joonis 1. InviteMultipleFamilyMembers testjuht

```
public void inviteMultipleFamilyMembers(Solo solo, String name, String email, String name2, String email2) {
    view = solo.getView(R.id.general_page_family_invite);
    solo.clickOnView(view);
    view = solo.getView(R.id.action_add);
    solo.clickOnView(view);
    solo.scrollToTop();
    solo.sleep(500);

    solo.enterText(0, name);
    solo.enterText(1, email);
    solo.enterText(2, name2);
    solo.enterText(3, email2);

    solo.clickOnView(solo.getView(R.id.invite_btn));
}
```

Joonis2. inviteMultipleFamilyMembers meetod Utils abiklassis



Joonis 3. Rakenduse sisene kutsete saatmine

AllTests – jooksutab testide komplekti kindlaks määratud järjekorras, mida saaks kergesti muuta.

Abiklass Util – Koosneb Sentab rakenduse testimise lihtsustamiseks kirjutatud meetoditest. Neid meetodeid kasutades paranes ka koodi loetavust. Abiklassi meetodid on täpselt kirjeldatud lisas 1.

Olulisemad meetodid:

- swipeSideDrawer() – Arvestame resolutsiooni ning kasutades Robotiumi meetodit *drag* avame kõrvalmenüü.

- openEvents(), openAlbums(), openMessages(), openGeneral(), openVideos() – avab kõrvalmenüü ning vastavalt meetodile avab õige vaate.
- rotateScreen() – vastavalt hetkeolekule muudab ekraani orientatsiooni vastupidiseks.
- addNewAlbum() – Loob uue albumi.
- changePatient() – avab kõrvalmenüü ning vahetab patsienti.
- clickConfirm() – vajutab kustutamise nupule ning vastavalt soovile kinnitab kustutamise.
- clickAndRemove() – sooritab pika vajutuse nimekirja objektil ning kustutab objekti.
- longClickAndEdit() – Sooritab pika vajutuse nimekirja objektil ning muudab selle nimetust.
- getListItemCount() – Tagastab nimekirjas olevate objektide summa.
- editAlbum() – muudab albumi nime ning kuupäeva.
- openImage() – avab albumis pildi
- getPictureId() – tagastab pildi identifikaatori
- getThumbnailId() – tagastab *thumbnail*-i identifikaatori
- setThumbnail() – muudab albumi *thumbnail*-i.

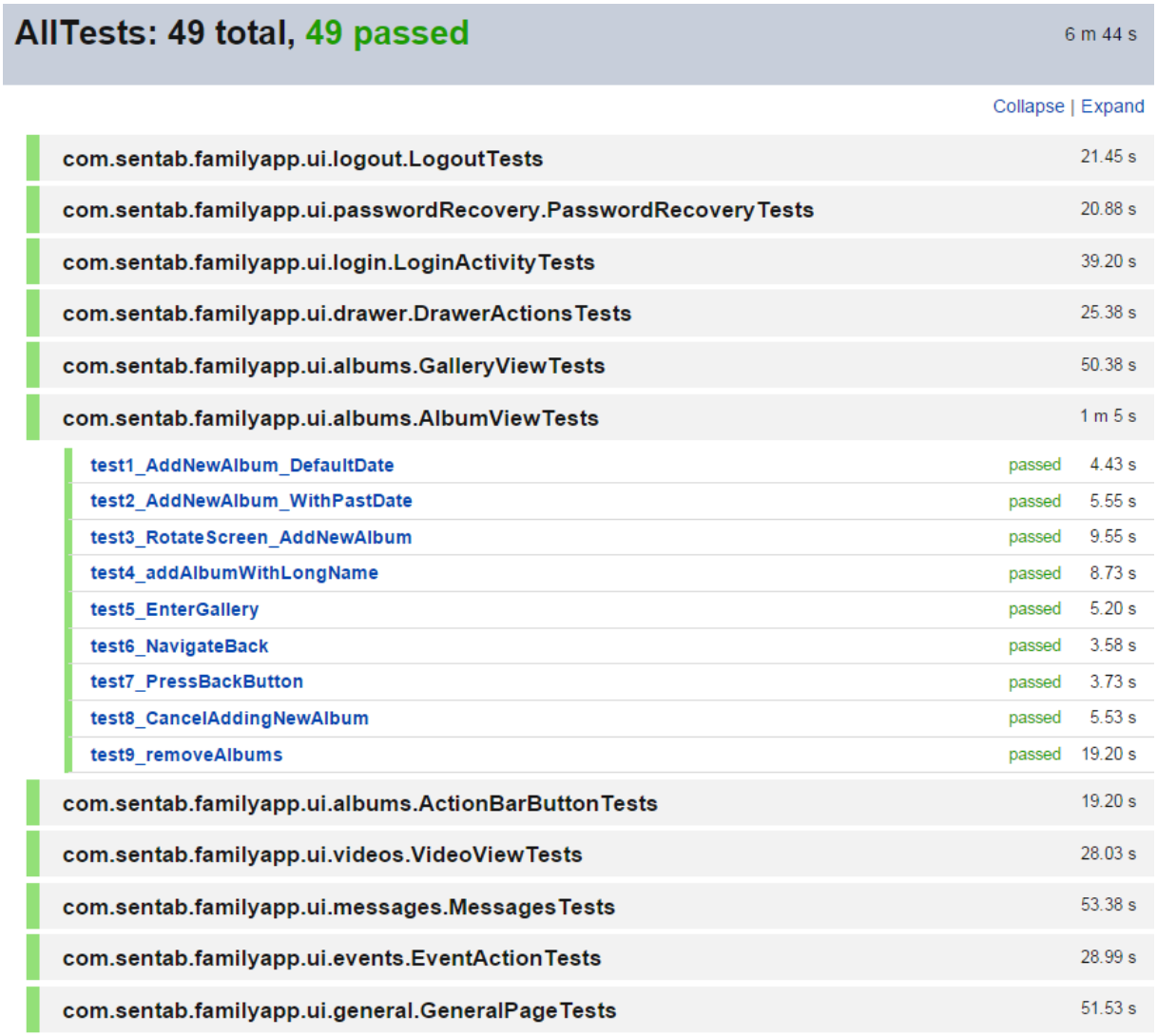
4.2 Testraport

Testraportist, mille Android Studio genereerib testjuhtude läbimisel, on näha, et kõigi, 49 testi läbimiseks kulus natuke keskmiselt alla 7 minuti, mis on aeglane võrreldes ühiktestidega, aga need pole võrreldavad integratsiooni testidega. Põhilise ajakulu tekitab rakenduse taaskäivitamine testide vahel ning vajaminevatele objektidele ligipääsu ootamine.

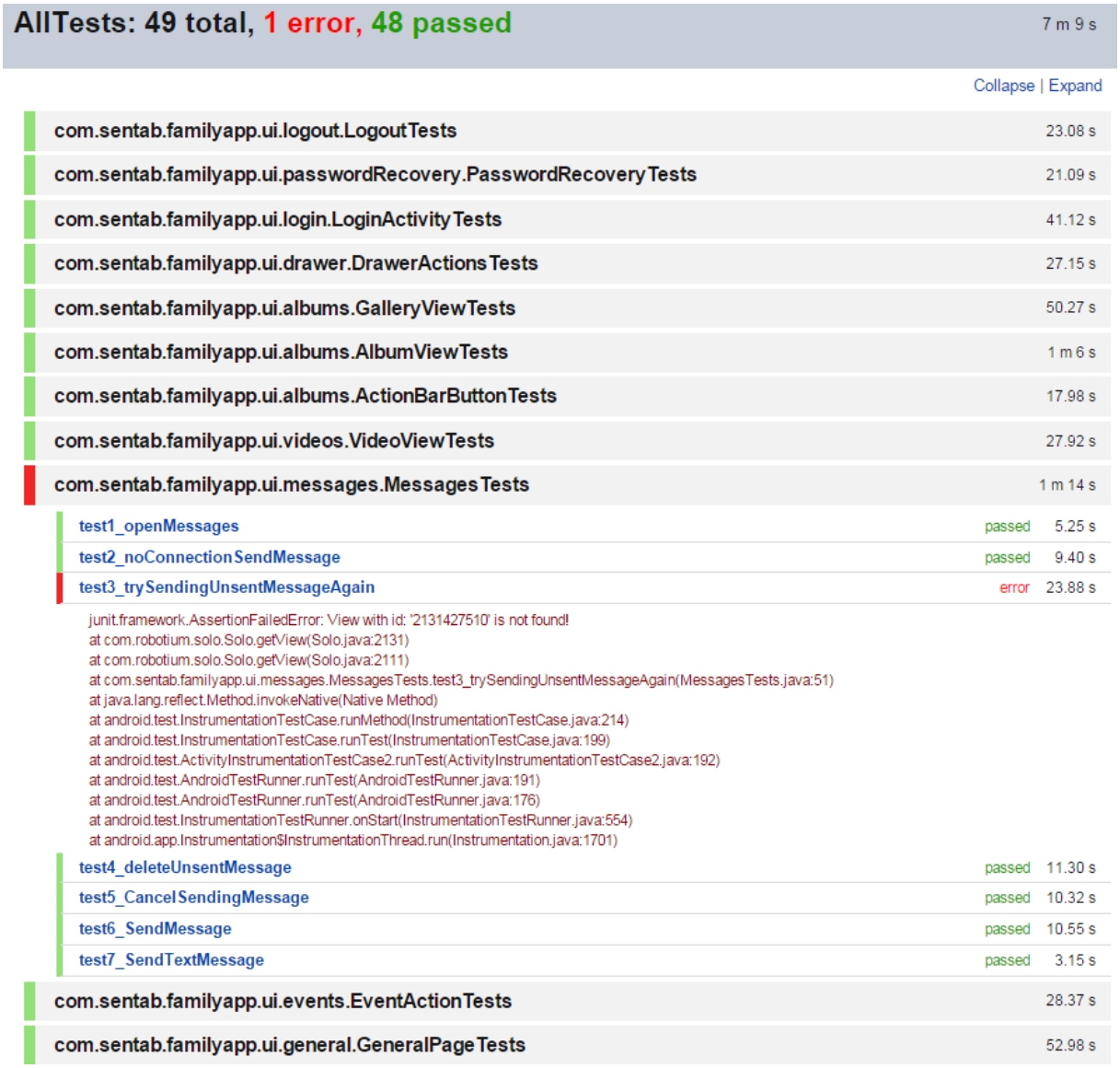
Testjuhud on osaliselt robustsed see tähendab, et osade testide eelduseks on eelnevate testide õnnestumine ning osad on eelnevatest testidest sõltumatud. Kõik testklassid on sorteeritud vaadete alusel ning on üksteisest peaaegu sõltumatud. Nad sõltuvad ainult sisse logimise õnnestumisest.

Järgnevalt on välja toodud kaks testraportit. Joonis 4 on näide, kus kõik testid õnnestusid. Joonis 5 on näide, kus osaliselt testid kukkusid läbi.

Joonis 5 näitab, et *trySendingUnsentMessageAgain* ebaõnnestus kuna ei leidnud kindla identifikaatoriga vaadet. Need identifikaatorid on alati numbriliselt ei anna see inimese jaoks piisavalt informatsiooni. Mistõttu tuleb kontrollida millised vaated testjuhus kasutusel on ning jooksutada test uuesti, vajadusel kogu testklass. See konkreetne ebaõnnestumine on aga hea näide, kuidas automaattest võib anda valesid tulemusi. Sest vaade millele oli tarvis vajutada, et kontroll õnnestuks oli kuvatud, kuid ei vajutatud sellele.



Joonis 4. Õnnestunud testraporti näide



Joonis 5. Ebaõnnestunud testraporti näide

Kokkuvõte

Käesoleva bakalaureusetöö põhieesmärgiks oli juurutada integratsioonitestid Sentab rakenduse testimisprotsessi, pidades silmas, et neid oleks tulevikus lihtne hooldada ning vajadusel juurde lisada. Töö tulemusena valmis 49 integratsioonitesti ning 1 abiklass, milles on erinevad meetodid automaattestide hooldamise ja lisamise lihtsustamiseks. Need testid hoiavad käsitsi testidega võrreldes kokku ligikaudu 14 minutit igal jooksutamisel. Käsitsi testides tekkisid suuremad ajakaod testjuhtude ja nende sammude järgimisel ning andmete sisestamisel. Abiklassi lisamine parandas testjuhtude loetavust tänu selgelt nimetatud meetoditele. Automaattestide suur pluss on nende taaskasutatavus, mis aitab aega kokku hoida võrreldes käsitsi testimisega ning võimaldab säästetud aega kulutada uute testide kirjutamisele või selle funktsionaalsuste testimisele, mida ei saa automatiseerida.

Võimalik edasiarendus oleks kasutusele võtta raamistik, mis on mõeldud ühiktestide loomisele – näiteks Mockito või Robolectric. Samuti oleks hea need testid integreerida Jenkinsiga või mõne teise järjepideva integratsiooni rakendusega, et nad käivituksid alati kui see rakendus alustab uue paketi ehitamist. See eemaldaks vajaduse käivitada teste käsitsi ning annaks testide jooksutamise unustamisel arendajatele ja testijatele automaatselt tagasisidet, kui mõni test on arenduse tõttu ebaõnnestuma hakanud. Lisaks oleks hea edasiarendus testid muuta robustsemaks, et testid, mis sõltuvad eelnevatest testidest õnnestuksid kui eelnevad kukkusid läbi. See tagaks täpsemad testitulemused.

Hea testimisplaan kataks testitava rakenduse ühik- ning integratsioonitestidega, kuid ka käsitsi testimine säilitab rakenduse testimisprotsessis väga olulise koha. Ühiktestide kasulikkus tuleneb nende kiirest läbimisest ning väiksemate osade testimisest, kuid nende kirjutamiseks on vajalikud täpsed teadmised testitavast koodist. Integratsioonitestide miinus on see, et nende läbimine võtab kaua aega võrreldes ühiktestidega. Ootuspäraselt aga on käsitsi testimine olulisel kohal, sest nii tuleb suuremal määral vigu esile. Kõiksuguseid olukordi ei ole võimalik ette näha, mistõttu tegelikkuses tuleb ainult väike osa vigu automaattestidega päevavalgele. Lisaks on käsitsi testimisel võimalik hinnata rakenduse kasutusmugavust.

Töö kirjutamise hetkel katavad töö tulemusena valminud testid umbes 50% Sentab rakenduse testimisprotsessist, kasutades Robotiumi raamistikku. Hilisemates juurdearendustes on lisandunud funktsionaalsus, mida ei ole võimalik adekvaatselt automatiseeritult testida – funktsionaalsuseks on helistamine. Uute funktsionaalsuste testjuhud automatiseeritakse, kui see on võimalik.

Töö eesmärk on saavutatud. Valmisid integratsioonitestid ning abiklass töö lihtsustamiseks ja kiirendamiseks. Integratsioonitestid on kirjutatud, kasutades meetodeid abiklassist, tänu millele on nende hooldamine tulevikus lihtsustatud. Tänu abiklassi olemasolule on võimalik ka uusi sarnaseid testjuhte lisada. Eesmärke oleks ilmselt saanud paremini saavutada, teades rakenduse koodi detailsemalt, kuid sellisel juhul oleks tegu olnud juba valge kasti testimisega.

Summary

This Bachelor thesis' main goals were to implement integration tests to Sentab application testing process while keeping in mind that the future servicing and development would be simple. As a result of this thesis, 49 integration tests were written and 1 helper class, where various methods for faster development and servicing are located. These tests help the tester save about 14 minutes of his time. A lot of the time is spent looking up test cases and its steps, also text entering. Thanks to the helper class the readability of tests was improved because of clearly named methods. Important feature of automated tests is reusability. The tester can spend the time on creating new tests or testing the features that could not be automated.

Possible future developments are to implement a unit testing framework like Robolectric or Mockito. Also it would be good to integrate the tests into Jenkins or some other continuous integration application, so the tests would be run every time Jenkins starts building new package. It would remove the need to run the tests manually and give feedback of results. Another good future development would be to make the tests more robust, so the tests that depend on the previous tests would pass even if the previous failed. This would assure more exact testing results.

Good testing plan covers the application with unit tests, as well as with integration tests and manual tests. The importance of unit tests comes from the quick run time as well as the small parts the tests cover, which makes pinpointing the bugs simpler. But it is required to know the code in detail. Downside of integration tests is the long runtime, compared to unit tests. As expected manual testing is still important because far more bugs reveal themselves with manual testing than with automated testing. It is because we cannot foresee every possible situation. In addition we can only rate the usability of an application by using it.

During the time this thesis is being written about 50% of Sentab application testing process is covered with automated tests – using Robotium. Later releases of Sentab application include calling, which cannot be adequately tested with automated tests. New functionalities will be automated if they can be.

The goal of the thesis is achieved. Integration test along with a helper class were created to ease and speed up the servicing and development of new tests. Thanks to the helper class it is simple to create new similar tests using the methods within the helper class.

Achieving the goals could have been improved when the source code would have been known in detail. But in that case it would have been white box testing.

Kasutatud kirjandus

[1] Android Debug Bridge. [WWW] http://developer.android.com/tools/help/adb.html (1.05.2015)

[2] Android Studio. [WWW] http://developer.android.com/tools/studio/index.html (1.05.2015)

[3] Androidi emulaatori piirangud. [WWW] http://developer.android.com/tools/devices/emulator.html#limitations (1.05.2015)

[4] BrowserStack. [WWW] https://www.browserstack.com/ (1.05.2015)

[5] Build Variants. [WWW] https://developer.android.com/tools/building/configuring-gradle.html#workBuildVariants (1.05.2015)

[6] Espresso. [WWW] https://code.google.com/p/android-test-kit/wiki/Espresso (1.05.2015)

[7] Gradle. [WWW] http://en.wikipedia.org/wiki/Gradle (1.05.2015)

[8] Genymotion. [WWW] https://www.genymotion.com/#!/product (1.05.2015)

[9] Genymotion. [WWW] https://www.genymotion.com/#!/ (1.05.2015)

[10] Graham, D., Veenendaal van E., Evans, I., Black, R., Foundations of Software Testing, Cengage Learning, 2008

[11] Juric-Kavelj, M. Is your Android emulator just too slow? [WWW] https://www.infinum.co/the-capsized-eight/articles/is-your-android-emulator-just-too-slow (1.05.2015)

[12] Kaner C., Bach J., Pettichord B. Lessons Learned in Software testing, John Wiley ja Sons, 2002

[13] Lint. [WWW] http://tools.android.com/tips/lint (1.05.2015)

[14] Logcat. [WWW] http://developer.android.com/tools/help/logcat.html (1.05.2015)

[15] Mockito. [WWW] http://mockito.org/ (1.05.2015)

[16] Robolectric. [WWW] https://code.google.com/p/robotium/ (1.05.2015)

[17] Robotium. [WWW] https://code.google.com/p/robotium/ (1.05.2015)

[18] UI Automator Viewer. [WWW] http://university.utest.com/android-ui-testing-uiautomatorviewer-and-uiautomator/ (1.05.2015)

[19] Vanurite üksilduse vastu võitlemine [WWW] http://www.bbc.com/news/uk-england-29063051 (1.05.2015)

Lisa 1. Abiklass

Järgnevalt on välja toodud abiklassi koodinäide, mis on kuvatud neljal leheküljel.

```
public class Util extends Activity {
    private View view;


    /deprecation/
    public int getWidth(Activity activity) {
        Display display = activity.getWindowManager().getDefaultDisplay();
        return display.getWidth();
    }

    /deprecation/
    public int getHeight(Activity activity) {
        Display display = activity.getWindowManager().getDefaultDisplay();
        return display.getHeight();
    }

    public void swipeSideDrawer(Activity activity, Solo solo) {
        int height = getHeight(activity)/3;
        float xFrom = getWidth(activity) * 0.01f;
        float xTo = getWidth(activity) * 0.75f;
        solo.drag(xFrom, xTo, height, height, 30);
    }

    public void refresh(Solo solo) {
        view = solo.getView(R.id.action_refresh);
        solo.waitForView(view);
        solo.clickOnView(view);
    }

    public void rotateScreen(Activity activity, Solo solo) {
        if (!isLandscape(activity)) {
            solo.setActivityOrientation(Solo.LANDSCAPE);
        }
        else {
            solo.setActivityOrientation(Solo.PORTRAIT);
        }
        solo.sleep(1500);
    }

    public boolean isLandscape(Activity activity) {
        return getHeight(activity) < getWidth(activity);
    }


    public void addNewAlbum(String albumName, Solo solo) {
        view = solo.getView(R.id.action_add);
        solo.clickOnView(view);
        EditText edit = (EditText) solo.getView(R.id.content_title_edit);
        solo.enterText(edit, albumName);
    }

    public void clickConfirm(Solo solo) {
        view = solo.getView(R.id.dialog_action_btn);
        solo.clickOnView(view);
    }

    public void clickCancel(Solo solo) {
        view = solo.getView(R.id.dialog_cancel_btn);
        solo.clickOnView(view);
    }
```

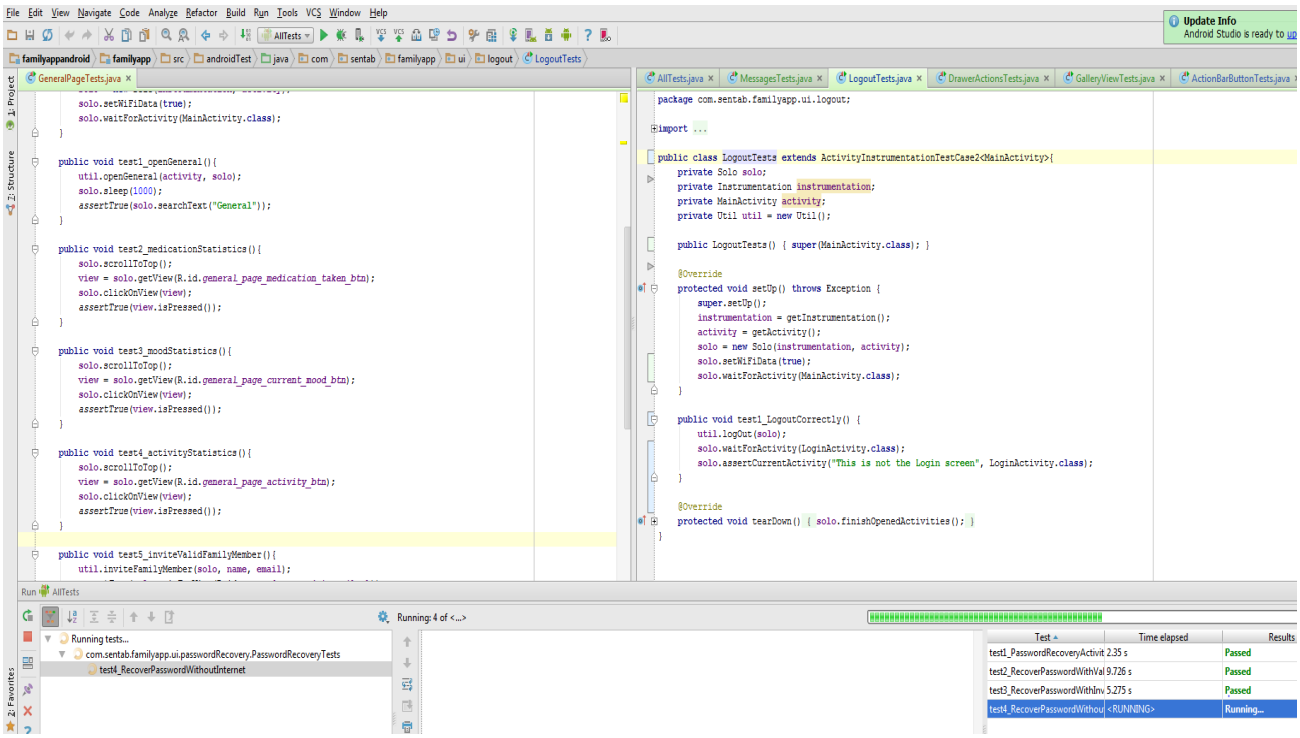
```
public void confirmDeletion(Solo solo) {
    view = solo.getView(android.R.id.button1);
    solo.clickOnView(view);
}

public void clickDelete(Solo solo, boolean confirmDelete) {
    view = solo.getView(R.id.action_delete);
    solo.clickOnView(view);
    if (confirmDelete) {
        confirmDeletion(solo);
    }
}

public void clickAndRemove(String name, Solo solo, boolean confirmDelete) {
    solo.clickLongOnText(name, 1, true);
    clickDelete(solo, confirmDelete);
}

public void longClickAndEdit(int lineNumber, String name, Solo solo) {
    solo.clickLongInList(lineNumber);
    view = solo.getView(R.id.action_edit);
    solo.clickOnView(view);
    EditText title = (EditText) solo.getView(R.id.content_title_edit);
    solo.clearEditText(title);
    solo.enterText(title, name);
    clickConfirm(solo);
}


public void logOut(Activity activity, Solo solo) {
    swipeSideDrawer(activity, solo);
    view = solo.getView(R.id.drawer_logout);
    solo.clickOnView(view);
    solo.clickOnText("Yes");
    solo.waitForActivity(LoginActivity.class);
}

public void logIn(Solo solo, String username, String pass) {
    solo.clickOnEditText(0);
    solo.clearEditText(0);
    solo.enterText(0, username);
    solo.clickOnEditText(1);
    solo.clearEditText(1);
    solo.enterText(1, pass);
    solo.clickOnButton(0);
    solo.sleep(3000);
}

public void recoverPassword(Solo solo, String email) {
    solo.clickOnEditText(0);
    solo.clearEditText(0);
    solo.enterText(0, email);
    solo.clickOnView(solo.getView(R.id.recover_password_action_button));
}

public void inviteFamilyMember(Solo solo, String name, String email) {
    view = solo.getView(R.id.general_page_family_invite);
    solo.clickOnView(view);
    solo.enterText(0, name);
    solo.enterText(1, email);
    solo.clickOnView(solo.getView(R.id.invite_btn));
}
```

```
public void inviteMultipleFamilyMembers(Solo solo, String name, String email, String name2, String email2) {
    view = solo.getView(R.id.general_page_family_invite);
    solo.clickOnView(view);
    view = solo.getView(R.id.action_add);
    solo.clickOnView(view);
    solo.scrollToTop();
    solo.sleep(500);

    solo.enterText(0, name);
    solo.enterText(1, email);
    solo.enterText(2, name2);
    solo.enterText(3, email2);

    solo.clickOnView(solo.getView(R.id.invite_btn));
}


public void openAlbums(Activity activity, Solo solo) {
    swipeSideDrawer(activity, solo);
    view = solo.getView(R.id.drawer_albums);
    solo.clickOnView(view);
}

public void openVideos(Activity activity, Solo solo) {
    swipeSideDrawer(activity, solo);
    view = solo.getView(R.id.drawer_videos);
    solo.clickOnView(view);
}

public void openMessages(Activity activity, Solo solo) {
    swipeSideDrawer(activity, solo);
    view = solo.getView(R.id.drawer_messages);
    solo.clickOnView(view);
}

public void openEvents(Activity activity, Solo solo){
    swipeSideDrawer(activity, solo);
    view = solo.getView(R.id.drawer_events);
    solo.clickOnView(view);
}

public void openGeneral(Activity activity, Solo solo) {
    swipeSideDrawer(activity, solo);
    view = solo.getView(R.id.drawer_general_page);
    solo.clickOnView(view);
}


public int getListItemCount(Solo solo) {
    ListView lw = solo.getView(ListView.class, 0);
    return lw.getCount();
}

public void changePatient(Activity activity, Solo solo, int patientNumber) {
    swipeSideDrawer(activity, solo);
    view = solo.getView(R.id.drawer_patient_name);
    solo.clickOnView(view);
    solo.clickOnRadioButton(patientNumber);
    clickConfirm(solo);
    view = solo.getView(android.R.id.home);
    solo.clickOnView(view);
}

public void recordMessage(Solo solo, int timeInSec) {
    view = solo.getView(R.id.message_action_btn);
    solo.clickLongOnView(view, timeInSec * 1000);
}
```

```
public void editAlbum(Solo solo, String albumName, int year, int month, int day) {
    view = solo.getView(R.id.action_edit);
    solo.clickOnView(view);
    solo.clearEditText(0);
    solo.enterText(0, albumName);
    solo.setDatePicker(0, year, month, day);
    view = solo.getView(R.id.dialog_action_btn);
    solo.clickOnView(view);
}

public void openImage(int pictureNumber, Solo solo) {
    solo.waitForView(RelativeLayout.class);
    view = solo.getView(RelativeLayout.class, pictureNumber);
    solo.clickOnView(view);
    solo.sleep(2000);
}

public long getPictureId(int pictureNumber, Solo solo) {
    GalleryItemView pictureView = (GalleryItemView) solo.getView(RelativeLayout.class, pictureNumber);
    FamilyPicture pictureData = pictureView.getData();
    return pictureData.getId();
}

public long getThumbnailId(int line, Solo solo) {
    View albumRow = solo.getView(RelativeLayout.class, line);
    AlbumItemData albumItemData = (AlbumItemData) albumRow.getTag();
    return albumItemData.getThumbnailId();
}


/**
 * @return 0 == picID and thumbnail are the same; 1 == they are not the same
 */
public int setThumbnail(int albumIndex, Solo solo, int pic) {
    long thumbnailBefore = getThumbnailId(albumIndex, solo);
    solo.clickInList(albumIndex + 1);
    if (getPictureId(pic, solo) == thumbnailBefore){
        pic++;
    }
    long pictureID = getPictureId(pic, solo);
    openImage(pic, solo);

    solo.clickOnImageButton(0);
    view = solo.getView(LinearLayout.class, 0);
    solo.clickOnView(view);

    solo.sendKey(KeyEvent.KEYCODE_BACK);
    solo.sendKey(KeyEvent.KEYCODE_BACK);
    solo.sleep(1500);
    long thumbnail = getThumbnailId(albumIndex, solo);
    if (pictureID == thumbnail) return 0;
    else return 1;
}

/**
 * First week layout numbers start with 3 until 15
 * Second week starts with 19 and so on.
 */
public int selectFromEventsWeek(Solo solo, int layoutNo) {
    solo.clickOnView(solo.getView(LinearLayout.class, layoutNo));
    solo.sleep(1000);
    return layoutNo;
}
```

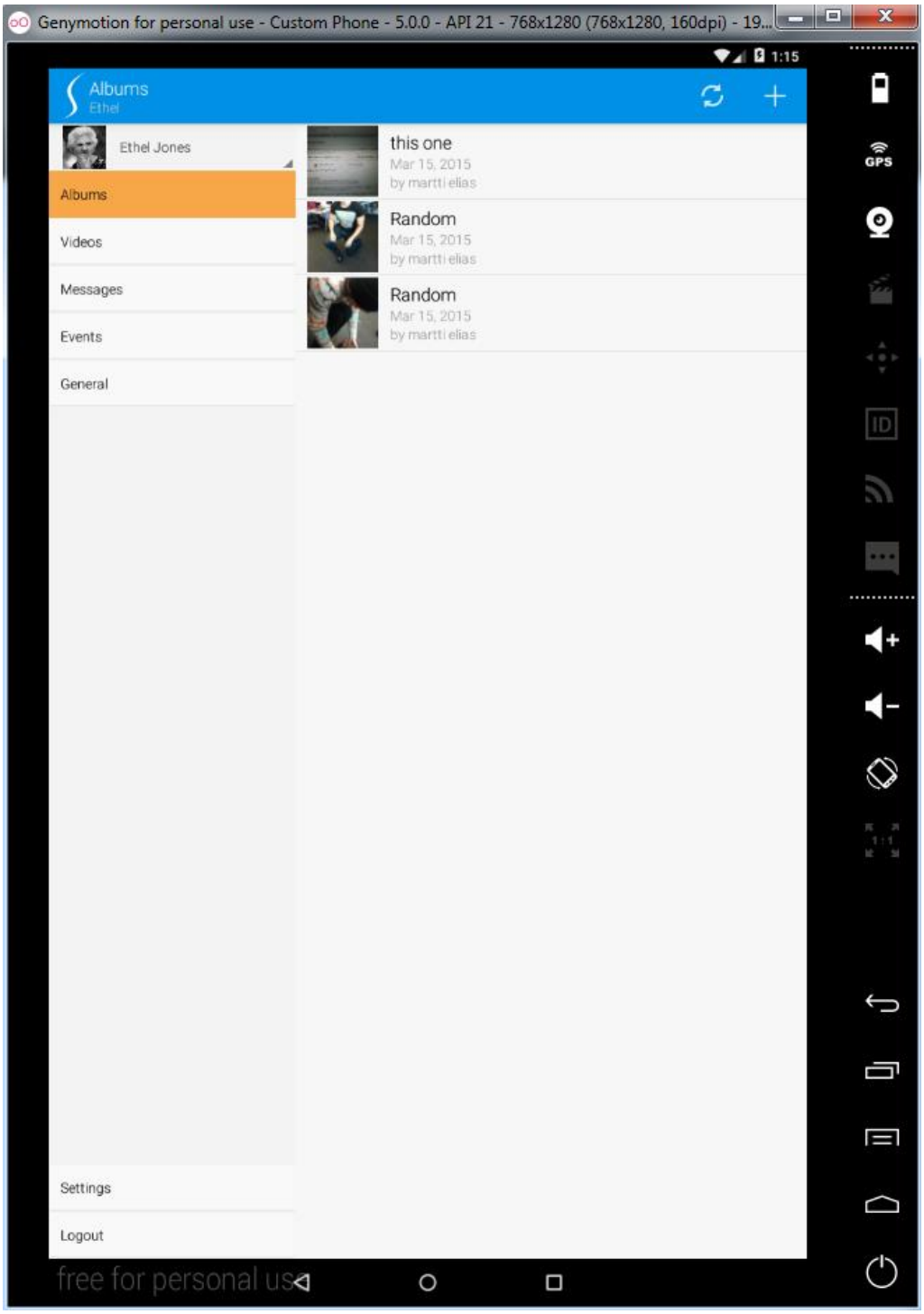
Joonis 6. Abiklassi koodinäide

Lisa 2. Android Studio



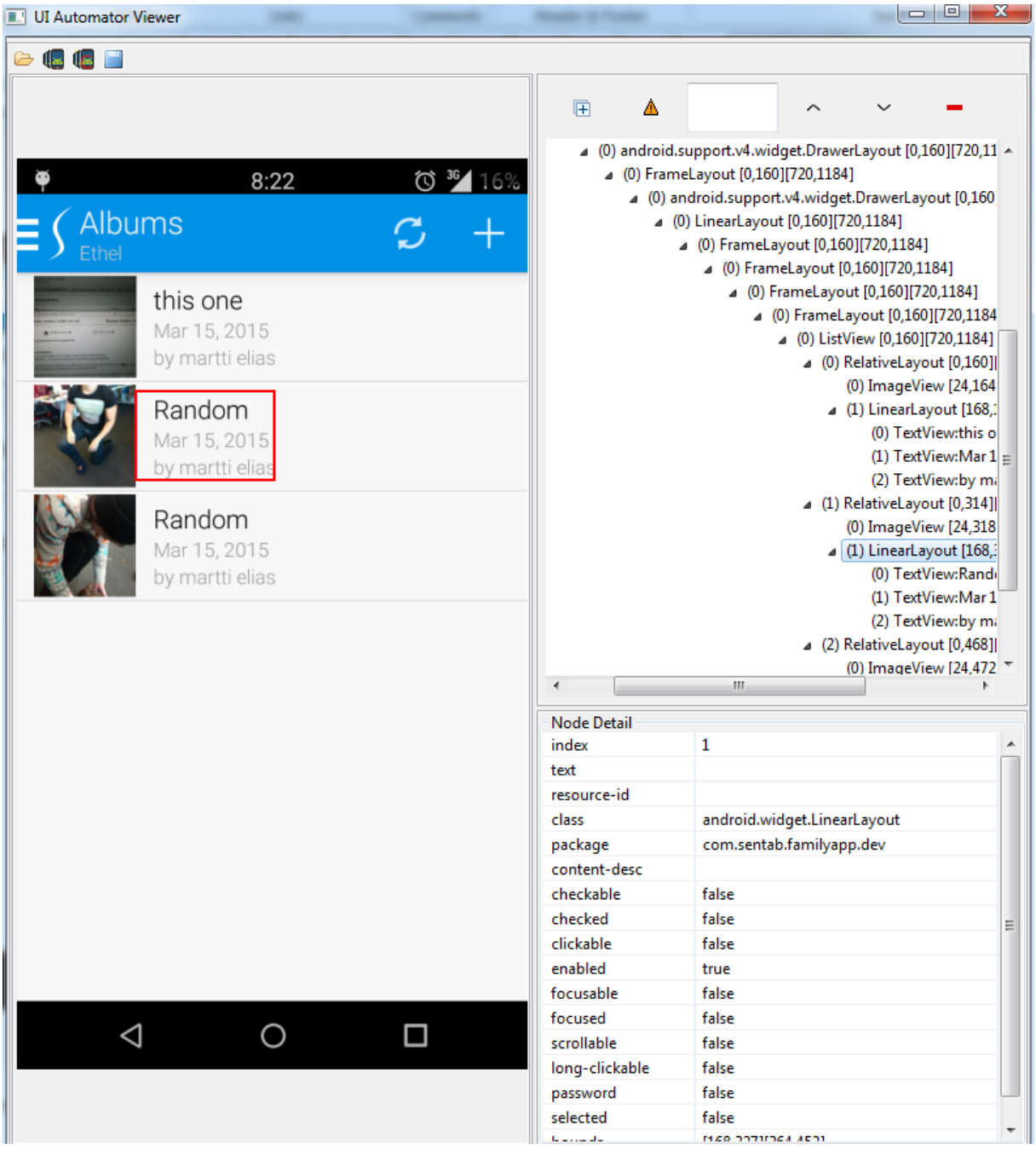
Joonis 7. Android Studios Robotium raamistikuga jooksev testide komplekt

Lisa 3. Genymotion



Joonis 8. Sentab rakendus Genymotion emulaatoris

Lisa 4. UI Automator Viewer



Joonis 9. UI Automator Viewer-is on skaneeritud Sentab rakenduse albumite vaade