TARTU ÜLIKOOL
Arvutiteaduse instituut
Informaatika õppekava

Sander Roosalu

Automaattestidel põhinev lahendus Androidi kodutööde automaatseks hindamiseks õppetöös

Bakalaureusetöö (9 EAP)

Juhendaja:
Jakob Mass MSc

Tartu 2022

Automaattestidel põhinev lahendus Androidi kodutööde automaatseks hindamiseks õppetöös

Lühikokkuvõte:

Lõputöö eesmärgiks oli luua Tartu Ülikooli mobiilirakenduste arendamise ainele automaatne Androidi kodutööde hindamissüsteem, kuhu kasutaja saaks Androidi kodutöid üles laadida ning saada kohest tagasisidet oma üleslaetud tööle. Süsteem loodi veebirakendusena kasutades Pythonit ja Flaski. Kodutööde hindamist viidi läbi skriptidega emuleeritud Androidi seadmel.

Töö annab ka ülevaade, kuidas Android platvormil rakenduste arendamine ja nende testimine käib ning uuriti, kuidas automaatseid hindamissüsteeme on kasutatud Tartu Ülikoolis ja teistes ülikoolides.

Loodud süsteem suudab täita oma eesmärki anda kohest tagasisidet. Selleks, et aga süsteem kursuse siseselt ka kasutust leiaks, peaks lisama juurde mitmeid uuendusi nt ümber kujundama püsiprogrammeeritud koodiridu ning täiustada kasutajaga suhtlust.

Võtmesõnad:

Mobiilirakenduste arendus, Android, automaathindajad, testimine, veebirakendus.

CERCS:

P175 Informaatika, süsteemiteooria

A intracurricular solution based on automated tests for grading Android applications

Abstract:

The aim of this Bachelor's thesis was to develop an automated grading system for Android applications in University of Tartu’s mobile app development course. The user could submit his or her Android homework for grading and receive near-instant feedback about the homework. The system was developed as a web application using Python and Flask. The homework grading process was carried out by scripts on an emulated Android device.

The thesis also gives an overview about how application development and testing on the Android platform could be conducted and how other automated grading systems have been used in University of Tartu and other universities.

The developed system is able to fulfil its intended goal of delivering near-instant feedback. For the system to be used at a scale within the course, several changes and updates must be made to the system beforehand, for example redesigning hardcoded lines of code and improving user interaction.

Keywords:

Mobile application development, Android, automated graders, testing, web application

CERCS:

P175 Informatics, systems theory

Sisukord

1. Sissejuhatus

Aastast 2017 tõusis Android maailma kõige kasutatumaks operatsioonisüsteemiks [1] ja on seda staatust hoidnud tänase päevani. Seda peamiselt seetõttu, et erinevad nuti- ja elektroonikaseadmed on muutunud iga aastaga tarbijale kättesaadavamaks ning nendes seadmetes on populaarseks operatsioonisüsteemiks Android. Samal aastal tõusis ka Androidi populaarseimas rakenduste teegis *Google Play Store* olevate rakenduste arv kolme miljonini [2]. Sellise kasvu juures peab aina enam rõhku panema rakenduste loomise õpetamisele, et olla kindel rakenduste kvaliteedis [3].

Tartu Ülikool on alates 2019. a sügissemestrist pakkunud ainet "Mobiiliarvutus ja asjade internet" (edaspidi MIoT), mis annab praktilise ülevaate mobiilirakenduste arendamisest ja asjade interneti (ingl k *Internet of Things*, IoT) süsteemide ehitamisest [4]. Aine on mõeldud bakalaureuse-, magistri- ja doktoriõppe õppeastmetes õppivatele tudengitele. Android hõlmas aines 2020/21 õppeaastal 10 nädala jagu teemasid. Aine on ajamahukas ning iga kodutöö korrektselt lahendamisele kulub palju tunde programmeerimist. Sisuka ja kiire tagasiside väärtus seega on väga suur.

MIoT Androidi kodutööd mida tudengid hetkel aines esitavad, ei ole mitte mõned kokku-pakitud failid, vaid terved rakenduste lähtekoodid. Tavaline kodutöö koosneb vaadetest (*layout*), mis määravad ära kasutajaliidese komponentide paigutuse ning toimingutest (*activity*), mis kirjeldavad ära loogika, kuidas kasutajaliidese elemendid omavahel suhtlevad.

Kodutööde kontrollimine võtab õppejõududel palju aega. Tööd on mahukad ja võimalikke veakohti on palju. Hetkel toimub õppejõududel kodutöö kontrollimine ja tagasiside loomine täiesti käsitsi. Kontrollimine hõlmab endas rakenduse käivitamist seadmel, kasutajaliidese funktsionaalsuse järgi proovimist, erinevate sisenditega töö kontrollimist jne. Samuti ei ole tudengitel muud võimalust kodutööde tegemise ajal oma tehtud töö korrektsuse kontrolli-miseks, kui ise rakendus käivitada oma seadmes või küsida küsimusi õppejõududelt või kursusekaaslastelt.

Mõlemat probleemi saaks potentsiaalselt lahendada ja/või leevendada, kui oleks olemas keskkond, kuhu saaks tehtud kodutöö üles laadida ja keskkond annab vastu tagasisidet tehtud tööle. Peale kodutöö üles laadimist paigaldatakse see Androidi emulaatorisse, kus saab jooksutada vastava kodutöö automaatteste. Peale testide jooksutamist saab tulemusi töödelda ja need kasutajale loetaval kujul esitada. Sellise keskkonnaga säästaksid aine õppejõud aega lihtsamate asjade kontrollimise pealt ja tudengid saaksid kindlustunnet oma töö osas.

Eelnevalt kirjeldatud regulaarsest kodutööst saaks automaattestidega ära kontrollida vaadetele paigutatud kasutajaliidese komponente (*UI Design*) ning kasutajaga suhtlemisel nende õigesti toimimist (*UI Interaction*). Elemente, mida rakendusel potentsiaalselt vaadata ja automaatselt kontrollida, on palju - vaadete komponentide asetus ja funktsionaalsus, rakendusele antud load, suhtlus sensoritega jne. Käesolev töö keskendub eelkõige kodutööde UI testimisele automaattestidega. Töö skoobist jääb välja kodutööde hindamine staatilise koodianalüüsi meetoditega.

Käesoleva töö eesmärk on luua MIoT ainele automaatne kodutööde hindamissüsteem, kuhu kasutaja saaks Androidi kodutöid üles laadida ning saada kohest tagasisidet oma üleslaetud kodutööle. Tehakse ülevaade, kuidas Android platvormil testimine käib ning uuritakse, kuidas automaatseid hindamissüsteeme on kasutatud Tartu Ülikoolis ja teistes ülikoolides.

Bakalaureusetöö on jaotatud kolmeks suuremaks peatükiks.

Esimene peatükk kirjeldab töö teoreetilist tausta. Tutvustatakse, mida automaathindajad ennast kujutavad, vaadatakse, mida on TÜ ja teised ülikoolid automaatse hindamise valdkonnas teinud ning kirjeldatakse, kuidas Androidil testimine töötab. Teine peatükk kirjeldab automaatse kodutööde hindamissüsteemiga seotud tehtud tööd. Kirjeldatakse loodud süsteemi ennast, selle arhitektuuri, nõudeid ning kasutatud tehno- loogiaid. Samuti teostati süsteemi jõudlusanalüüs. Kolmandas peatükis arutletakse süsteemi kasutamise üle MIoT aines ning kuidas saaks süsteemi ainele paremini kohandada.

Lõputöö lisades on olemas ka tabel kasutatud terminite kohta, mis võivad lugemise ajal ebaselgeks jääda.

2. Teoreetiline ülevaade

Teoreetilise ülevaate peatükk jaguneb kolmeks alapeatükiks. Esimes alapeatükis kirjeldatakse, mida automaathindajad endast kujutavad. Teises alapeatükis tehakse ülevaade, kuidas automaathindajaid on kasutatud Tartu Ülikoolis ja teistes maailma ülikoolides. Kolmandas peatükis kirjeldatakse, kuidas Android platvormil testimist läbi viia. Samuti tehakse lühiülevaate, missugune on Android Studioga rakenduste loomise protsess.

2.1 Automaathindajad üldiselt

Automaathindajad on tarkvaralised tööriistad, millele saab ette anda mingi sisendi ning seejärel väljundina tagasi saada mingi hinne või hinnang esitatud sisendi kohta. Üks automaatse hindamissüsteemi kasutusnäide on nt kui tudeng sisestab oma vastustega kodutöö süsteemi ning saab tagasisideks, mitu tema kodutöös esitatud vastust õiged on. Vastavalt sisendi struktuurile ja hindamiskriteeriumitele saab väljund olla see, mis olukorras vajalik on.

Automaathindajate kasutamine on kasulik nii tudengitele kui ka õppejõududele. Tudengid saavad aimu, et kas nad on saavutamas oma õpiväljundeid ning õppejõud saavad silma peal hoida iga tudengi õppeprotsessil iga antud teema kohta [5]. Tudengitel on võimalik saada tagasisidet oma tehtud tööle ka siis, kui õppejõud ei ole parasjagu kättesaadavad ning õppejõududel kulub vähem aega kodutööde lihtsamate osade kontrollimisega. Automaathindajate efektiivsus suureneb lineaarselt kasutajate arvuga, seega annab selliste süsteemide olemasolu võimaluse välja õpetada rohkem tudengeid korraga.

Automaathindajaid on olemas kahte tüüpi, staatilised analüsaatorid ja dünaamilised analüsaatorid. Staatiliste analüsaatorite eesmärk on vaadelda sisendit ning markeerida ära kõik võimalikud veakohad ja probleemid enne, kui sisendit edasi töödeldakse [6]. Sellised süsteemid on nt õigekirjakontrollid lingvistika kontekstis ja staatilised koodianalüsaatorid nagu SpotBugs [7] informaatika kontekstis. Dünaamiliste hindajate eesmärk on sisendi peal läbi jooksutada erinevaid testimismalle ning samal ajal vaadelda, mis sisendiga käitamise jooksul juhtub, kas sisend valideeritakse või mitte. Sellised süsteemid on näiteks programmeerimisülesannete automaatkontrollid. Edaspidi käesolevas töös mõeldakse automaathindajate all dünaamilisi analüsaatoreid informaatika kontekstis, kui ei ole eraldi mainitud.

Automaatkontrollides kasutatavaid ja funktsionaalsust kontrollivaid testimismalle saab skoobi järgi jaotada neljaks: üksuste testimine (*unit testing*), integratsiooni testimine (*integration testing*), süsteemi testimine (*system testing*) ning vastuvõtutestimine (*acceptance testing*) [8]. Üksuste testidega tehakse kindlaks, kas üks üksus programmist (nt üks meetod või funktsioon) toimib nii nagu peab. Integratsiooni testide eesmärk on kontrollida, kas mitu üksust suhtlevad omavahel õigesti. Süsteemi testidega kontrollitakse, kas kogu süsteem

käitub vastavalt eelnevalt kirja pandud spetsifikatsioonidele. Vastuvõtutestimisega verifitseeritakse, et tarkvara on valmis kasutamiseks.

2.2 Ülevaade kodutööde hindajatest

Peatükk annab ülevaate, kuidas on automaatseid hindajaid kasutatud Tartu Ülikoolis ning kuidas mobiilirakenduste automaatseid hindajaid on kasutatud välisülikoolides.

2.2.1 Automaathindajad Tartu Ülikooli Arvutiteaduse instituudis

Tartu Ülikooli Arvutiteaduse instituut (ATI) pakub mitmeid tarkvara arendusega seotud aineid, kus tudengitel on võimalik enda tehtud tööd automaattestidega kontrollida. Sellised ained on näiteks Programmeerimine ja Programmeerimine II. Mõlemas aines on loodud "automaatkontrollid", mis käivituvad, kui tudeng on esitamise keskkonda oma kodutöö ülesse laadinud. Automaatkontroll proovib kodutöö peal läbi mitmeid sisendeid ning seejärel annab tudengile tagasisidet väljundite õigsuse kohta.

Tudengite lahenduste automaatse hindamise võimalusi ainetes on uuritud veel mitmes bakalaureusetöös. Mikk Õunmaa [9] lõi ainele "Sissejuhatus andmebaasidesse" automaattestid, mida oli võimalik panna õppekeskkonda Moodle. Töö käigus samuti loodi ka testimisvahend, mis lihtsustas õppejõul tudengite andmebaasi kontrollimist. Siim Plangi [10] kirjeldab, kuidas tema koos 4 kaastudengiga lõid ainele "Algoritmid ja andmestruktuurid" automaatse programmeerimisülesannete kontrollija. Süsteemis saavad õppejõud luua programmeerimise ülesandeid ning tudengitel on võimalus esitada oma lahendusi antud ülesannetele.

Mobiilirakenduste automaatseid hindamissüsteeme ATI ainetes käesoleva töö kirjutamise ajal ei leidu.

2.2.2 Mobiilirakenduste automaatsed hindajad välisülikoolides

Kuna ATI ainetes mobiilirakenduste automaatseid hindajaid hetkel ei leidu, siis lisaks ATIs pakutavatele ainetele otsiti juurde ka näiteid ja teadustöid teistelt ülikoolidelt.

Teadusartiklite päringuid tehti otsingumootorites Scopus ja Google Scholar. Scopuse lõplik otsingutulemus saadi päringuga:

> 18 document results (27.03.2022)
>
> TITLE-ABS-KEY ((android OR ios) AND (automated AND (grad* OR homework OR test* OR assessment)) AND education) AND (PUBYEAR > 2014) AND (LIMIT-TO (SUBJAREA , "COMP") OR LIMIT-TO (SUBJAREA , "ENGI") OR LIMIT-TO (SUBJAREA , "SOCI")).

Filtreerimis-kriteeriumitest võeti kasutusele 2 valikut, mis andsid otsingule fookust juurde. Esimesena pandi päringule juurde avaldamisaasta > 2014, sest arenduskeskkond Android Studio avalikustati selle aasta lõpus. Teisena määrati tööde alateemadeks arvutiteadus, inseneriteadus või sotsiaalteadus. Viimane lisati juurde, sest loodeti saada teadustöid, mis sisaldasid tagasiside andmist. Päringule "educationi" juurde lisamine samuti vähendas tulemusi mitmetest sadadest kümneteni.

18st leitud tööst leidus üks töö [11], mis kattub käesoleva töö temaatikaga praktiliselt üleni. Seda tööd Google Scholaris tsiteerinud tööde hulgas leidub ka kaks teist teadustööd [5, 12], kust võeti käesolevale tööle inspiratsiooni.

Paiva et al. [5] Porto ülikoolist on koostanud süstemaatilise ülevaate-artikli automaatsetest hindamissüsteemidest informaatika hariduse õpetamises. Töös tehakse ülevaateid erinevatest valdkondadest nagu nt testimise meetodid ja tagasiside andmine. Töö lõpus võetakse vaatlustulemused kokku ja antakse hinnang automaatsete hindamissüsteemide seisukorrast valdkonnas. Töö tundub kindlasti vajalik selleks, et saada aru eelnevalt nimetatud süsteemide olulisusest valdkonnas ning annab vihjeid, kuidas kõige moodsamal kujul võimalikele probleemidele läheneda.

Tagasiside andmise teemal on nii Paiva et al. [5] kui ka Keuning et al. [13] tähele pannud, et populaarsemad tagasiside tüübid on teadvustamine, mis läks valesti (*knowledge about mistakes*) ning teadvustamine, kuidas edasi tegutseda (*knowledge about how to proceed*). Need on ka kaks tagasiside tüüpi, mille kallal on ka rohkem uurimusi tehtud. Kui loodavas hindamissüsteemis saab ebaeduka testi tulemuse korral tudengile näidata veateadet ning vihjet, miks viga potentsiaalselt tekkis, siis on ära kasutatud mõlemad tagasiside tüübid. Bruzual et al. [11] loodud hindamissüsteemi tagasisides tõid tudengid eraldi välja, et veateade oli eriti väärtuslik, sest sealt suutsid nad välja lugeda, kuidas nad peaksid oma kodutööd täpselt siluma.

Bruzual et al. [11] Aalto ülikoolist kirjeldavad, kuidas nad lõid Androidi ülesannete automaatse hindamissüsteemi, mille nad mahutasid ära ühte Dockeri konteinerisse. Testimis-raamistikuna kasutati Appiumi ning üksuste testimiseks Mavenit. Loodud süsteemi katsetati ka õppetöös magistikraadi omandavate tudengite peal. Töös mainitakse, et loodud lahendus oli efektiivne, sest tudengid väga väärtustasid saadud tagasisidet ning see toetas tudengite iseseisvat mobiilirakenduste loomise protsessi. Töö lõpus samuti räägitakse loomisprotsessi jooksul ilmnenud tähelepanekutest. Bruzual et al. tehtud töö väärtus käesolevale tööle on väga suur, sest ka selle töö üheks eesmärgiks on luua sarnase sihtotstarbega keskkond, mis annaks tudengitele tagasisidet nende kodutöödele.

Bruzual et al. [11] palusid tudengitel esitada oma tehtud tööd binaarfailidena. See tõi kaasa olukorra, kus tudengid said ise valida raamistiku ja keele, milles oma kodutöö luua. Lisaks Javas loodud lahendustele, leidus ka edukaid lahendusi, mis olid loodud kasutades nt Kotlinit või React Native'it. MIoT aine õpetab tudengeid looma lahendusi kasutades Kotlinit ja Android Studiot. Kuna need kaks tööriista on ka ametlikud tööriistad, mida Android platvormi haldaja Google soovitab kasutada rakenduste arendamiseks, siis ei pea töö autor võimalikuks, et MIoT aines teistsugused lahendused prevaleerima hakkavad.

Modesti, P [12] Teesside'i ülikoolist kirjeldab, kuidas skripte saaks ära kasutada Androidi rakenduste loomise õpetamises ja hindamises. Tuuakse välja ja hinnatakse, kui efektiivsed olid skriptid, mis aitasid rakenduse arendusprotsessile kaasa ning õppejõule mõeldud skriptid tudengite tööde hindamiseks. Samuti kirjeldatakse tudengitööde hindamise skripti töövoogu ja kuidas seda on võimalik pakkfailidega (*batch files*) saavutada. Töö lõpus jagatakse soovitusi neile, kes soovivad skripte kasutada oma tulevastes projektides. Skriptide kasutamine töövoo loomiseks kodutööde hindamises on üks võimalik viis, kuidas käesoleva töö käigus loodava keskkonna tagaliides lahendada. Skriptidega saaks kodutöö Androidi emulaatorisse paigaldada ning seejärel nendega juhendada hindamise läbiviimist. Ka praegu kasutavad MIoT aine õppejõud enda loodud ja personaalseks kasutamiseks mõeldud skripte tudengite kodutööde paremaks haldamiseks.

Modesti, P [12] demonstreerib oma töös tudengi kodutöö hindamisskripti töövoogu. Skripti käitab õppejõud manuaalselt, kui ta on tudengi lähtekoodi kätte saanud. Loodava hindamissüsteemi tagaliideses võiks sarnane skript automaatselt tööle minna kohe, kui töö on süsteemi kohale jõudnud.

Eelnevalt pikemalt kirjeldatud kolm teadustööd on leitud töödest enim seotud käesolevas töös pakutava lahendusega. Otsingu tulemusena üles leitud lahendusi, mis olid loodavale süsteemile sarnased, oli vähe, sest mobiiliarendus ei ole veel nii populaarne ega laialt levinud aine ülikoolides, kui nt veebiarendus. TÜ veebiarenduse ainele LTAT.05.004 "Veebirakenduste loomine" pakutakse nt kolm korda rohkem õppekohti kui MIoT aines, samal ajal olles mitte ainus TÜ veebiarendusega seotud aine. Teine põhjus, mida ka Modesti, P [12] toob oma töös välja, et miks selliseid süsteeme ei ole palju loodud, on, et Androidi arenduskeskkonnad muutuvad väga tihedalt ja sellega koos võivad olemasolevad automaatsed hindamissüsteemid lakata töötamast. Nutiseadmete populaarsuse kasvuga maailmas kasvab nõudlus kvaliteetsemate rakenduste järele, seega automaatsed rakenduste hindamissüsteemid muutuvad aina enam aktuaalsemaks.

2.3 Rakenduse arendamine ja testimine platvormil Android

Alapeatükk kirjeldab vajalikke aspekte Androidi rakenduste arendusprotsessist ja testimisest eesmärgiga anda konteksti lõputöö praktilise osa kohta. Samuti tehakse alapeatükis ülevaade kasutatavatest tehnoloogiatest.

2.3.1 Lühiülevaade Androidi rakenduse arendusprotsessist

Kõige tavalisem viis uue rakenduse loomiseks on kasutada arenduskeskkonda Android Studio (rohkem peatükis 2.3.3). Uue projekti (rakenduse) loomisel antakse kasutajale mitmeid valikuid projekti eelsätestamiseks. Esimesena on kasutajal võimalus valida riistvara aluse ja toimingu malli, mille järgi oma rakendust arendama hakata. Riistvara aluste valikute seas on mobiiliseadmed, nutikellad, nutitelerid ja auto keskkonsoolid. Toimingu mallide valikus on nt täisekraani toiming ja Google Mapsi toiming. Võimalik on edasi minna ka ilma malli valimata. Sagedaseim valik MIoT aine kodutöödes on tühi toiming.

Peale riistvara aluse ja toimingu malli valimist peab kasutaja valima oma projektile pealkirja, paketi nime (tähtis testimisel), projekti asukoha andmekandjal, programmeerimiskeele ning minimaalse Androidi SDK, mida rakendus toetama peab. Mida uuem SDK valitakse, seda rohkem uusi funktsioone saab rakendusele lisada, aga suureneb ka tõenäosus, et rakendus ei tööta vanematel Androidi seadmetel. Peale nende valikute tegemist viiakse kasutaja oma projekti juurkausta ning projektiga sünkroniseeritakse Gradle tööriistad. Seejärel on võimalik kasutajal oma rakendust looma hakata.

Esimese kaks rakenduse faili, mis kasutajal vaikimisi avatud on, on esimese vaate (*layout*) fail ning seda kontrolliva toimingu (*activity*) fail, mis on eelväärtustatud valitud malli "Tere, maailm!" programmiga. Rakenduse vaadete failid on XML-failid, kuhu kasutaja saab XML-märgenditega vaate komponente juurde lisada. Alternatiivselt on komponente juurde võimalik lisada neid pukseerides komponentide paletist vaatele. Rakenduse loogikat hõlmavad failid on eelnevalt valitud programmeerimiskeele failid.

Et kasutaja saaks kuskil käivitada enda loodud rakendust, on tal vaja Androidi seadet. Kasutaja saab ühendada enda seadme Android Studioga juhtme või WiFi kaudu. Alternatiivselt saab kasutaja Android Studios luua emuleeritud Androidi virtuaalse seadme ning seda

oma arvuti ekraanil kuvada. Kui rakenduse kood kompileerub, siis tekitatakse rakenduse binaarfail (.apk laiendiga fail, edaspidi APK-fail), mis paigaldatakse seadmesse. Peale paigaldamist avaneb rakendus seadme esiplaanile ning kasutajal on võimalik sellega nüüd oma seadmes suhelda.

Rakenduse APK-fail on olemuselt tihendatud fail, mis sisaldab vastava rakenduse binaarfaile. Rakenduse APK-faili on vaja vastava rakenduse paigaldamiseks seadmesse. APK-fail omab endas ka infot paketi nime kohta, kus rakendus loodud on. APK-faili metaandmetes on vastava info välja nimi *applicationId*. See infoväli on tähtis hiljem, kui rakendust on vaja testima hakata.

2.3.2 Androidi rakenduse testimine ja testi tüübid

Sarnaselt tavalistele tarkvara testidele, saab ka Androidi rakenduste teste analoogsel viisil jaotada vastavalt keskkonnast isoleerituse tasemele (skoobile): üksuste testimine, integratsiooni testimine ning läbivtestimine *(end-to-end testing*), mis on sarnane süsteemi testimisele [14].

Androidi arendaja jaoks on olulisem testide jaotamine selle järgi, kus neid käitatakse: kohalikud testid (*local tests,* näide joonisel 1) ja instrumenteeritud testid (*instrumented tests,* näide joonisel 2). Kohalike testide hulka kuuluvad tavaliselt väiksema-mahulised testid nagu üksuste testid ning nende käivitamiseks piisab ainult Java virtuaalmasinast (JVM), mis asub kohalikus masinas [15]. Instrumenteeritud testide alla kuuluvad tavaliselt integratsiooni testid ja läbivtestid ning nende testide jooksutamiseks on lisaks JVM-ile vaja ka parasjagu töötavat Androidi seadet.

```
@Test
fun addition_isCorrect() {
    assertEquals( expected: 4, add( x: 2,  y: 2))
}
```

Joonis 1. Näide kohalikust testist keeles Kotlin.

```
@Test
fun prevButtonTestsL() {
    val device = UiDevice.getInstance(getInstrumentation())
    device.setOrientationLeft()

    onView(withId(R.id.land_prev_button)).check(isCompletelyLeftOf(withId(R.id.land_next_button)))
    onView(withId(R.id.land_prev_button)).check(isTopAlignedWith(withId(R.id.land_next_button)))
    onView(withId(R.id.land_prev_button)).check(isBottomAlignedWith(withId(R.id.land_next_button)))

    device.setOrientationNatural()
}
```

Joonis 2. Näide instrumenteeritud testist keeles Kotlin. Lillad kirjed on viited vaate elementidele.

Teste saab jooksutada mitmel viisil. Üksikuid teste saab käitada testi klassidest endast. Mitmete testi klasside korraga jooksutamiseks on aga loodud automatiseerimise tööriistad nagu Gradle ja ADB, mida on pikemalt lahti seletatud peatükis 2.3.3.

Kuna instrumenteeritud testid (edaspidi lihtsalt testid) jooksutatakse parasjagu töötaval Androidi seadmel, siis sarnaselt rakenduse jooksutamisele on ka instrumenteeritud testide jooksutamiseks vaja testide APK-fail seadmesse paigaldada. Testide APK-failil on samuti olemas info oma paketi nime kohta infoväljal *applicationId*.

Instrumenteeritud testide jooksutamiseks on seega vaja testitava rakenduse APK-faili ning testide APK-faili. Kui seadmele, kuhu need kaks APK-faili eelnevalt paigaldatud on, tuleb sisse käsklus teste läbi viia, siis need kaks APK-faili leiavad üksteist ülesse just paketi nimede järgi. Testide paketi nimi peab olema sama, mis testitaval rakendusel, ainult et sellele on lõppu veel lisatud *.test*. Näiteks, kui testitava rakenduse paketi nimi on *com.example.homework1*, siis sellele vastava testide paketi nimi peab olema *com.example.homework1.test*. Hiljem on võimalik paketi nime muuta projekti mooduli-tasemel olevas *build.gradle* failis, muutes välja *android{defaultConfig{applicationId}}* väärtust.

2.3.3 Tehnoloogiate kirjeldus

Ametlik integreeritud arenduskeskkond (IDE) Androidi rakenduste loomiseks on Android Studio, mis on kõrvalearendus populaarsest JetBrainsi IDEst IntelliJ IDEA. Android Studio sisaldab endas ka virtuaalsete Androidi seadmete e. emulaatorite loomise võimalust. Emulaatorid funktsioneerivad analoogselt tavalistele Androidi seadmetele, mõlemale saab loodud rakendusi peale installeerida ning ka neid testida. Seega ei ole tingimata vaja füüsilist Androidi seadet, et rakenduste testimist läbi viia. Samuti tekitab Android Studio automaatselt uuele projektile eraldi kaustad kohalikele testidele (nimi *test*) ning instrumenteeritud testidele (nimi *androidTest*).

Testide loomiseks on Androidil mitmeid raamistikke. Nii kohalike kui ka instrumenteeritud testide loomiseks ja jooksutamiseks on vaja kasutada raamistikku JUnit, mis vastutab loodud testide struktuuri ja reeglistiku eest [16, 17]. JUnit sisaldab endas ka standartsemaid tööriistu testide läbiviimiseks nt *assertTrue()* ja *assertEquals()*. Kohalike testide loomisel on abiks ka raamistikud nagu Robolectric ja Mockito, mis testivad spetsiifilisemaid olukordi, mida JUniti pakutud vahendid ei võimalda [16]. Näited sellistest olukordadest on nt UI elementidega suhtlemine ilma töötava masinata ning elementide jäljendamine. Instrumenteeritud testide loomisel on abiks raamistikud nagu Espresso ja UI Automator [17]. Espresso on spetsialiseerunud UI testide loomiseks, UI automator võimaldab suhelda ka rakenduse väliste elementidega nt kodunupud ekraani all servas. Androidi testimisvahenditest on ülevaate teinud ka Sinaga et al. [3]. Nii Espressot kui ka UI Automatorit kasutatakse lisaks JUnitile käesoleva töö jooksul loodava hindamissüsteemi näidis-testide loomiseks.

Testide käitamiseks käsurealiideselt (*command line interface*, CLI) on Google soovitanud arendajatel kasutada kahte võimalikku tööriista, Gradle ja Android Debug Bridge (ADB).

Gradle on Android Studiosse sisse integreeritud riistakomplekt, mis automatiseerib ja haldab rakenduse arendustsükli protsesse nt rakenduse kompileerimine ja automaattestide käitamine [18]. Gradle käsuga *connectedAndroidTest*, mis jooksutab parasjagu töötaval seadmel läbi kõik instrumenteeritud testid (testid kaustas *androidTest*), saab minimaalse tööga automaatset testimist läbi viia. Peale käsu käivitamist genereeritakse automaatselt testimiseks vajalikud

binaarfailid ning peale testide sooritamist ka testi tulemuste raportid nii XML kui ka ja HTML-formaadis. Gradle-i käsud kutsutakse välja puustruktuuris deklaratiivselt, s.t kõrge-tasemelised keerulised käsud kutsuvad välja mitu lihtsamat madalatasemelist käsku. Näiteks sama eelnevalt välja toodud käsk *connectedAndroidTest* käitab automaatselt alamkäsud *processDebugAndroidTestResources*, *preDebugAndroidTestBuild*, *packageDebugAndroidTest*, *packageDebug*, *compileDebugAndroidTestKotlin* ja *connectedDebugAndroidTest*, millest mõned käitavad veel omakorda alamkäske. Sellisel viisil läbi viia automaatteste on väga töökindel, aga nõuab palju aega ja lähtekoodi olemasolu. ADB-ga testide läbiviimine leevendab mõlemat probleemi.

ADB on Androidi arendus-tarkvara komplektist (SDK) paigaldatav riistakomplekt, mis annab võimaluse kasutajal suhelda Androidi seadmega käsurealiidese abil [19]. ADB abil saab nt otse paigaldada seadmesse rakendusi, seadet rootida või kohalikus masinas pääseda ligi seadme failidele. Et töötaval seadmel ADB abil instrumenteeritud teste jooksutada, peab kõigepealt jooksutama kaks käsku testitava rakenduse APK-faili ning testide APK-faili seadmesse paigaldamiseks. Seejärel saab jooksutada käsku, mis käivitab testid seadmes. Käsule peab argumentidena kaasa andma veel testide paketi nime ja testide jooksutaja (tavaliselt JUnit). Neid kolme käsku üksteise järel välja kutsudes saab rakenduse peal automaatteste läbi viia kiiremini, kui Gradle-ga.

3. Automaatne kodutööde hindaja

Peatükk jaguneb neljaks alapeatükiks. Esimene alapeatükk räägib süsteemi funktsionaalsetest nõuetest. Teises alapeatükis kirjeldatakse loodud süsteemi arhitektuuri ja töövoogu ning räägitakse kasutatud tehnoloogiatest. Kolmandas alapeatükis kirjeldatakse süsteemi kasutus-nõudeid kasutajale ja serverile, kus süsteem jookseb. Neljandas alapeatükis tuuakse välja, mis asjaolud mõjutavad süsteemi jõudlust.

3.1 Hindaja funktsionaalsed nõuded

Selleks, et automaatse kodutööde hindamissüsteemi loomisel oleks selge siht ja eesmärk, mille poole pürgida, on hindajale määratud funktsionaalsed nõuded.

Kasutajal peab olema võimalus:

FN1.1 - jõuda kohani, kus saaks süsteemile päringuid teha;

FN1.2 - sisestada süsteemi oma hinnatava kodutöö APK-fail;

FN1.3 - valida, millist kodutööd peab käitama tema hinnatava töö peal;

FN1.4 - saata päring koos hinnatava tööga serverisse.

Süsteemi tagaliides peab:

FN2.1 - saabunud kodutöid testima Androidi seadmel endal:

FN2.2 - saabunud kodutöid hoidma järjekorras, kuni avaneb võimalus kodutööd hinnata;

FN2.3 - hindamistulemuste pealt koostatud tagasiside andma edasi granulaarselt, s.t tuuakse välja vead alamülesannetes, kui viga oli ainult seal, mitte kogu ülesandes.

Peale kodutöö hindamist peab kasutaja olema võimalus:

FN3.1 - näha oma kodutöö hindamistulemusi;

FN3.2 - näha, missuguseid nõudeid hinnatud kodutöö ei läbinud;

FN3.3 - ebaedukate alamülesannete puhul näha ja aru saada, mis valesti läks;

FN3.4 - ebaedukate testide puhul näha ja aru saada, kuidas tegutseda edasi nii, et testi oleks võimalik edukalt läbida.

Kirjeldatud funktsionaalsed nõuded ei sisalda endas kõiki aspekte, mis võiks loodaval süsteemil olemas olla. Nõuded on loodud eesmärgiga olla suunisteks loodavale süsteemile.

3.2 Kodutööde hindaja loomine

Alapeatüki esimene jaotis kirjeldab loodud hindaja arhitektuuri ja töövoogu sissetuleva päringu korral. Teine jaotis kirjeldab, mis tehnoloogiaid kasutati hindaja komponentide loomisel.

3.2.1 Arhitektuur ja töövoog

Loodud lahendusena valmis hindamissüsteem, mis koosneb neljast suuremast omavahel suhtlevast üksusest. Kasutajaliides on loodud programmeerimiskeele Python mikroraamistikuga Flask. Kasutaja päringuid ja teisi tagaliideses sooritatavaid operatsioone samuti hallatakse Pythonis. Kodutööde testimist viiakse läbi kasutades virtuaalset Androidi emulaator-seadet. Testilogisid salvestatakse ja testide vihjeid loetakse sisse andmebaasist, mida hallatakse MongoDB-s. Eelmisel lehel on kujutatud ja edasi siin alapeatükis kirjeldatakse, kuidas hindamissüsteem toimib sissetuleva päringu korral. Päringu töövoo skeem on kujutatud joonisel 3.

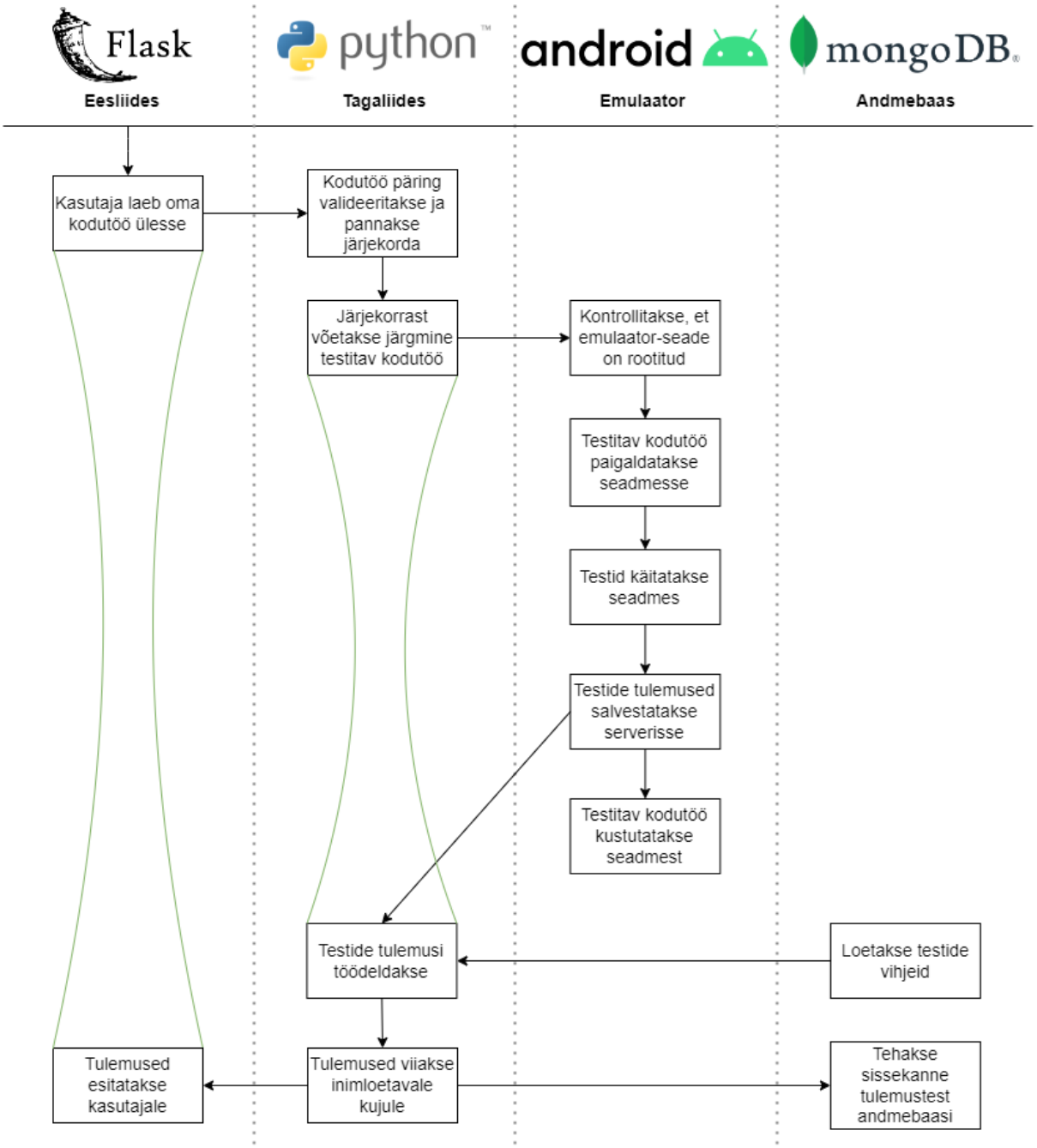
Päring sisaldab endas kolm tähtsat infokildu: testitav kodutöö, esitatud kodutöö number ning esitaja matrikkel. Esitatava kodutöö number on vajalik päringu valideerimisprotsessis, matriklit kasutatakse automaathindamise tulemuse salvestamisel.

Kui hindamissüsteemi saabub päring, siis selle õigsus kõigepealt valideeritakse. Valideerimise jooksul kontrollitakse kolme erinevat kriteeriumit:

1. kas tööde järjekorras on ruumi;
2. kas valitud kodutööd saab hetkel esitada;
3. kas testitava rakenduse APK-failis infoväli *applicationId* on korrektne.

Valideerimisprotsess on edukas, kui kõik kolm kriteeriumit kehtivad. Protsessi ebaõnnestumise järel antakse kasutajale koheselt vastus koos selgitusega, mis valesti läks ning töö selle päringuga lõpetatakse. Edukuse korral pannakse päring tehtavate tööde järjekorda ning jääb oma korda ootama (FN2.2). Et rakendus ei hõivaks serveris liigselt muutmälu, siis on järjekorras ruumi korraga kuni kümnele päringule.

Kui parasjagu tööd mingi päringuga ei tehta, siis võetakse järjekorrast järgmine päring. Päringuid võetakse järjekorrast välja FIFO (*first in, first out*) printsiibil, st alustatakse tööd päringu kallal, mis on kõige kauem pidanud ootama. Uuest päringust salvestatakse maha testitav rakendus ning matrikkel. Seejärel käitatakse skriptid, mis hoolitsevad testimisprotsessi eest.



Joonis 3. Hindamissüsteemi sissetuleva päringu töövoo skeem. Rohelised jooned tähistavad ühe protsessi teise järele ootamist.

Skripte mida käitatakse on viis. Esimene skript kontrollib, kas aktiivne Androidi seade on rootitud. Kui ei, siis seade rootitakse. Teine skript paigaldab testitava kodutöö seadmesse. Kolmas skript käivitab seadmes testitava kodutöö peal vastavad testid (FN2.1). Selle skripti töö moodustab ajaliselt kõige suurema osa kogu töövoost ning võib kulutada kuni mitukümmend sekundit vastavalt testide arvule ja mahukusele (täpsemalt alapeatükis 3.4). Kolmanda skripti töö lõpuks on genereeritud testide tulemused Androidi seadmesse. Neljas skript tõmbab seadmest testide tulemused serverisse ning viies skript eemaldab seadmest just-testitud kodutöö. Neljandas skriptis tulemuste genereerimiseks seadmesse kasutatakse

teeki XmlRunListener [20]. Skriptide töö lõpuks on olemas töötlemata kujul testide tulemused ja Androidi seadme olek on sama, mis enne skriptide käitamist.

Kätte saadud tulemusi ei ole mõistlik kohe esitada; need on vaja ära töödelda ja inimloetavaks teha. Iga jooksutatud testi kohta pannakse kirja testi nimi, klassi nimi mille alla test kuulub, testile kulunud aeg, ning tulemus. Negatiivse tulemusega testidele lisatakse juurde ka veateade ja vihje korrektse lahenduse kohta. Igale testile on kohandatud oma vihje, mis juhendab kasutajat üle vaatama sagedaseid veakohti testi mitteläbimise kohta. Testide vihjeid loetakse sisse MongoDB andmebaasist.

Kui tulemused on töödeldud, siis viimased kaks operatsiooni on logi kandmine andmebaasi ja tulemuste esitamine kasutajale. Logi sisaldab andmeid testimisprotsessi kohta: matrikkel, kodutöö number, tulemus ning ajad päringu saabumise, testimisprotsessi alguse ja protsessi lõpu kohta. Tulemused esitatakse kasutajale uuel lehel, kus need on tabelis mugavalt vaadeldavad. Peale andmebaasi logimist on töö päringuga lõppenud ning on võimalus järjekorrast uus päring võtta.

3.2.2 Kasutatud tehnoloogiad

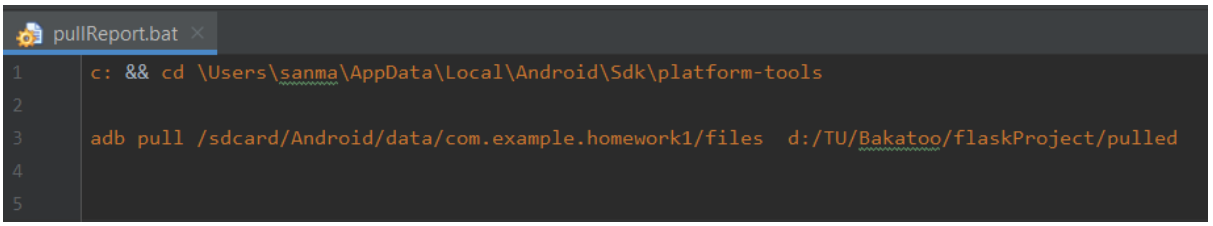
Veebirakenduse eesliidese teostuseks valiti Pythoni mikroraamistik Flask [21]. Flask lubab luua lihtsa veebiserveri kasutades vähe trafarett-koodi ning olles ühtlasi teistest teekidest sõltumatu. See annab arendajale vabad käed veebiserveri lahenduse implementeerimiseks. HTML-failidega suhtlemiseks kasutab Flask keelt Jinja. Pythoni veebiraamistik Django või Javascripti raamistikud nagu React või Vue.js olid tehnoloogia valimise ajal liiga keerukad veebirakenduse mahu kohta. Kuna veebirakenduse eesliides ei tegele keeruliste eeskomponentide haldamisega, vaid konkreetselt kasutajalt kodutöö kätte saamisega ja testide info edastamisega, siis sobib Flask antud veebirakendusele väga hästi. Kasutatud Flaski versioon oli 2.0.3.

Hindamissüsteemi tagaliidese teostuseks valiti keel Python [22]. Python on populaarne keel tagaliideste lahenduseks, kui tagaliideses tehtav töö ei nõua suurt kogust arvutuslikku ressurssi. Veebirakendustele tagaliidese loomine Pythonis on aga vähem levinud, kui nt Node.js-iga. Pythonist erineva tagaliidese arenduskeele valimine oleks aga teinud arendusprotsessi ebavajalikult keeruliseks, sest käesoleva töö autor rohkem kokku puutunud Pythoniga, kui Javascriptiga või Javaga. Samuti kasutati veebirakenduse eesliidese loomiseks Flaski, mis lubas väikese vaevaga päringuid kanda üle tagaliidesele töötlemiseks. Kasutatud Pythoni versioon oli 3.9.0.

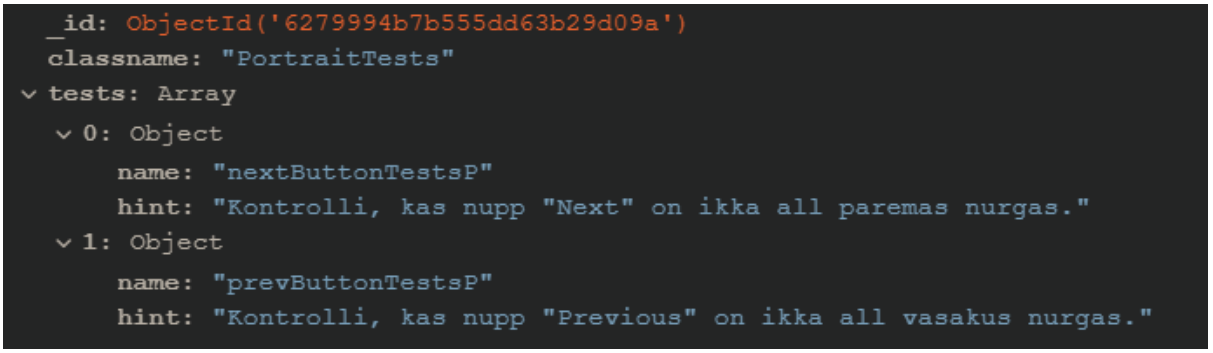
Hindamissüsteemi Pythoni programmide arenduskeskkonnaks oli PyCharm [23], mis on analoogselt Android Studiole samuti üks paljudest JetBrainsi IDEdest. Kasutatud PyCharmi versioon oli 2022.1.

Androidi emulaator koos näidis kodutööde lahenduste ja näidistestidega loodi Android Studio [24] abiga. Loodud Androidi seade on ilma Google APIdeta virtuaalne kloon telefonist Google Pixel 2. Seadme peal jooksis Androidi versioon 11 API tasemega 30 ehk Android R ning omas 1.5GB muutmälu. Ilma Google APIdeta on võimalik loodud seadet rootida, mida on omakorda vaja, et ligi pääseda skriptide poolt seadmesse genereeritud testide tulemustele. Analoogne seade koos Google APIdega on vaikimisi emulaator, mida kasutatakse praktikumi- ja kodutöödes MIoT aines. Kasutatud Android Studio versioon oli 2021.1.1 “Bumblebee” Patch 3 ning Android Emulatori versioon oli 31.2.10.

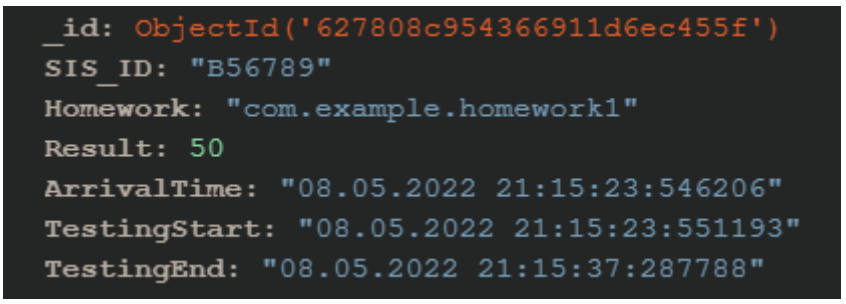
Skriptid, mis haldavad kodutööde testimisprotsessi Androidi seadmel, olid loodud Windowsi pakkfailidena (joonis 4). Pakkfaili käitamisel kutsutakse välja käske parasjagu aktiivsest kataloogist. Iga skripti alguses on kataloogipuu õige kataloogini, kust kutsutakse välja ülejäänud skripti käsud. ADB käskude abil seadmega suheldes on kogu testimisprotsess võimalik skriptidega läbi viia efektiivsemalt, kui Gradle-i käskudega. ADB ja Gradle kohta on võimalik täpsemalt lugeda peatükist 2.3.3. Kasutatud ADB versioon oli 1.0.41 ning Android SDK Platform-Tools versioon oli 33.0.1.



Joonis 4. Skripti näide. Skript sooritab testide tulemuste tõmbamist serverisse.



Joonis 5. Näide ühte testi klassi kuuluvate testide vihjetest andmebaasi dokumendis.



Joonis 6. Näide testitulemuste logist andmebaasi dokumendis.

Testid, millega kodutööde hindamist teostati, loodi kasutades teeke Espresso ja UI automator. Näidet testist, kus on kasutatud tööriistu mõlemast teegist, on näha joonisel 2.

Testide vihjete hoiustamiseks (joonis 5) ja logide üles kirjutamiseks (joonis 6) kasutab hindamissüsteem MongoDB-s loodud andmebaasi. MongoDB [25] andmebaas on struktuurilt NoSQL ehk andmeid hoitakse dokumentides ning nendel ei ole kindlat struktuuri. Kuna andmebaas tagataustal keerulisemaid operatsioone ei tee kui sealt lugemine ja sinna kirjutamine, siis relatsioonilise mudeliga andmebaasi loomine oleks arendusprotsessi muutnud asjatult keeruliseks. Kasutatud MongoDB versioon oli 5.0.7 Community.

Tarkvaralised tööriistad hindamissüsteemi loomiseks olid valitud nii, et oleks olnud võimalik luua süsteem piisavalt lihtsalt ning, et ei oleks kadusid valitud tööriistade funktsionaalsuses.

3.3 Süsteemi kasutusnõuded

Alapeatükis kirjeldatakse nõudeid nii kasutajale kui ka serverile. Nõuded kasutajale hõlmavad ennas korrektse keskkonna kirjeldust, mis on tingimused veebirakendusele ligipääsuks ning selle õigesti toimimiseks. Nõuded serverile hõlmavad endas riistvara nõudeid selleks, et süsteemi hostida.

3.3.1 Nõuded kasutajale

Kuna kasutaja suhtleb hindamissüsteemiga veebirakenduse kaudu, siis ühed kasutajapoolsed nõuded on seotud veebilehe jooksutamisega. Veebibrauseris, kus veebirakendus avada, peab lubatud olema Javascripti elementide kasutus, et veebirakendus toimiks korrektselt. Veebirakenduse korrektset toimimist on läbi proovitud veebibrauserites Google Chrome, Microsoft Edge ja Mozilla Firefox.

Teised nõuded on seotud esitatava kodutööga. Kasutaja peab jälgima, et tema esitatud kodutöö käivituks edukalt. Et selles veenduda, saab kasutaja käivitada oma kodutöö Androidi emulaatoris. MIoT aine siseselt on selline emulaator kasutajal juba loodud. Eduka käivitamise järel saab kasutaja oma kodutööst teha APK-faili. Android Studioga on APK-fail võimalik tekitada kaht viisi. Esimene meetod on lihtsalt rakendus jooksutada oma seadmel. Teine meetod on valida Android Studio ülevalt valikurealt Build > Build Bundle(s) / APK(s) > Build APK(s). Vaikimisi kataloogipuu tekitatud APK-failini on *<paketi_nimi>\build\intermediates\apk\debug\app-debug.apk*. Lisaks peab kasutaja tähele panema, et tema kodutöö vaadete elementide ID-d oleksid õiged, et automaatsed testid vastavad elemendid ülesse leiaksid.

Kui kasutaja saadab veebirakenduse kaudu oma kodutöö hindamissüsteemi, siis ta peab esitama just oma kodutöö APK-faili, mitte terve kodutöö lähtekoodi. APK-faili on võimalik otse kasutada ADB käske testide läbiviimiseks. Lähtekoodist APK-faili tekitamine lisaks hindamisprotsessile juurde liigset aega. Kiireks hindamiseks saaks kasutaja selle osa ise ära teha. Bruzual et al. [11] samuti palusid oma tudengitel esitada kodutöid APK-failidena eesmärgiga, et tudengid saaksid veenduda, et nende tehtud rakendus töötab nii nagu vaja enne automaathindamist. Selline lähenemine vähendab ka serveri poolt nõutud arvutuslikku ressurssi.

3.3.2 Nõuded serverile

Serverile kõikide vajalike failide olemasolu nõuab kokku ligikaudu 13GB andmekandja mälu. Tagaliidese failid, mis hõlmavad Pythoni faile, skripte ja veebilehtede (HTML, CSS, JavaScript) faile, nõuavad kokku umbes 19MB. Android Studio nõuab 1.7GB, Androidi

emulaator 9.6GB (!) ning MongoDB 1.2GB. Märkimisväärselt mahukad on ka Java ja Androidi arendus-tarkvara komplektid JDK ja Android SDK, millest viimane hõivab käesolevas töös kasutatud serveris 10.7GB andmekandja mälu.

Server peab hindamissüsteemi jooksutamiseks olema võimeline eraldama kuni 2.7GB vaba muutmälu. Süsteemi töötamise ajal peavad olema aktiivsed Pythoni ees- ja tagaliidesed, Androidi emulaator seade, ning MongoDB andmebaas. Pythoni pool nõuab 33MB muutmälu, Androidi emulaator seade nõuab kuni 2.5 GB muutmälu (1GB vaikimisi + 1.5GB virtualiseeritud muutmälu) ning MongoDB andmebaas nõuab 15 MB muutmälu.

Esitatud numbrid on loetavad ka allolevas tabelis 1.

Tabel 1. Ülevaade serverile vajaminevatest mälu ressurssidest.

Tehnoloogia	Andmekandja mälu	Muutmälu
Tagaliidese failid	19MB	33MB
Android Studio	1.7GB	-
Androidi emulaator	9.6GB	2.5GB
MongoDB	1.2GB	15MB

Esitatud numbritest on kõige suurema muutlikkusega emulaatoriga seotud numbrid. Vastavalt loodud emulaatorile võivad suureneda nii andmekandjal hõivatav mälu kui ka muutmälu. Siinkohal välja toodud mälukasutus on käesoleva töö jooksul kasutatud emulaatori omad. Iga nimetatud tarkvara puhul on vastavad arendajad samuti pannud soovituslikus riistvaralised nõuded nende tarkvara kasutamiseks. Need nõuded on saadaval vastava tarkvara haldaja veebilehel.

Tagaliidese failid olid paigaldatud serveri kõvakettale (HDD), ülejäänud komponendid olid paigaldatud serveri pooljuhtkettale (SSD).

3.4 Jõudlusanalüüs

Hindamissüsteemi jõudlust peamiselt mõjutavad testide arv ja kompleksus ning serveri jõudlus.

Seadme rootimis-kontrollile ei lähe üle viiendiku sekundist. Kui avastatakse, et seade ei ole rootitud, siis sellele võib kuluda kuni 1 sekund.

Testitava rakenduse paigaldamisele ei tohiks kuluda üle poole sekundi. See maht aga võib kiiresti kasvada sõltuvalt rakenduse suurusest. Esimese kodutöö APK-faili, millega mõõtmisi teostati, suurus oli 2.89MB.

Androidi seadmes jooksutatakse teste kodutöö peal läbi ükshaaval ja üksteise järel, mitte paralleelselt. Seega tõuseb ajakulu iga testi kohta, mida on tarvis jooksutada hinnangu saamiseks. Esimese kodutöö näidisteste jooksutamiseks näidis kodutöö peal kulus keskmiselt

13±2 sekundit, sh igale üksikule testile, mida oli kokku neli, kulus varieeruvalt 1 kuni 2 sekundit ning harva kuni 3 sekundit.

Seadmest testimistulemuste serverisse tõmbamisele kulub triviaalselt vähe aega. Esimese kodutöö tulemuste, mille faili suurus oli 12.1KB, kulus stabiilselt 0.005 sekundit. Isegi, kui teste on rohkem ja tulemuste fail muutub suuremaks, siis selle aja mitmekordset suurenemist ei ole võimalik tajuda. Samuti kulub triviaalselt vähe aega ka rakenduse eemaldamisele seadmest.

Serveri jõudlus on piiratud riistvarast ning parasjagu aktiivsetest protsessidest. Mida rohkem protsesse on serveris paralleelselt tööl ning mida vähem vabasid ressursse saadakse riistvaralt hindamiseks allokeerida, seda rohkem aega võib kuluda kodutöö hindamisele. Töö käigus kasutusel olnud serveris jooksis keskmiselt 290 protsessi (saadud PowerShelli käsuga (Get-Process).Count), kui hindamissüsteem oli aktiivne. Süsteemi arendamise jooksul töötasid serveris kõik neli suuremat süsteemi komponenti paralleelselt. Serveri operatsioonisüsteem oli Windows 10 Home versioon 21H2, protsessor oli Intel Core i7-9750H, mille tuumad jooksid baassagedusel 2.60GHz ning videokaart oli NVIDIA GeForce GTX 1650 Max-Q. Serverisse oli paigaldatud 2x8GB DDR4 muutmälu, mis jooksis sagedusel 2667 MHz.

4. Hindaja kasutamine MIoT aine kontekstis

Peatükk on jagatud neljaks alapeatükiks. Esimeses alapeatükis kirjeldatakse, kuidas aine tudeng peaks loodud hindamissüsteemi veebirakendust kasutama. Teises peatükis kirjeldatakse, kuidas aine õppejõud hetkel MIoT aines kodutöid hindavad ning kuidas nad saavad loodud hindamissüsteemi kasutada. Samuti arutletakse, kuidas nad saaksid aine kodutöid kohendada automaatteste silmas pidades. Kolmandas peatükis arutletakse, missuguseid parandusi on vaja teha ning uuendusi vaja lisada süsteemile, et selle saaks õppetöösse kasutusele võtta. Viimane alapeatükk hõlmab endas töö autori viimaseid kommentaare ja arutelu loodud hindamissüsteemi osas.

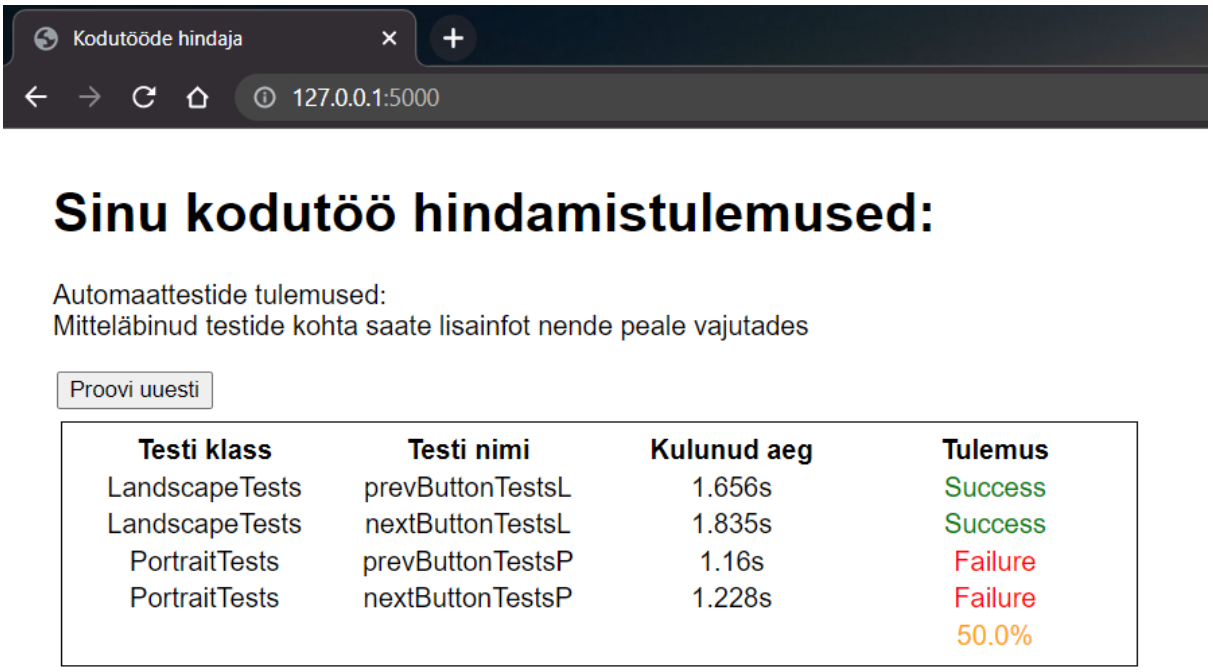
4.1 Hindaja kasutamine tudengi vaatepunktist

Loodud automaatse kodutööde hindajast on veebirakendust mõeldud kasutama just tudeng.

Automaatse kodutööde hindaja veebirakendus koosneb kahest vaatest. Esimene vaade, kuhu tudeng suunatakse veebirakendusse saabumisel, on avaleht (joonis 7, FN1.1). Avalehel on kirjed, kuhu täpselt tudeng saabunud on ning juhised edasiseks tegevuseks. Tudengil on palutud veebivormi kaudu sisestada hinnatava kodutöö APK-fail (FN1.2), hinnatav kodutöö number (FN1.3) ning tema matrikli number. Peale nõutud asjade sisestamist veebivormi on tudengil võimalus esitada päring serverile vajutades nuppu “Esita hindamiseks”, et tema kodutööd hinnata saaks (FN1.4). Lehe allpool on teade, mis palub tudengil veebilehte mitte värskendada peale töö hindamiseks esitamist, sest siis ei ole hindamistulemustel võimalik jõuda tudengini tagasi. Lehe viimasel real on ka näidik, mitu kodutööd parasjagu ootavad oma hindamist. Selle järgi on tudengil võimalik kaudselt hinnata, kui kaua tudeng peab oma kodutöö hindamistulemuste järele ootama.

Joonis 7. Automaatse kodutööde hindaja veebirakenduse avaleht veebilehitsejas Google Chrome.

Kui hindamissüsteem on on jõudnud kodutöö ära hinnata ning tulemused ära töödelda, siis suunatakse tudeng teisele veebirakenduse vaatele (joonis 8), kus tehakse tulemused tudengile avalikuks (FN3.1).

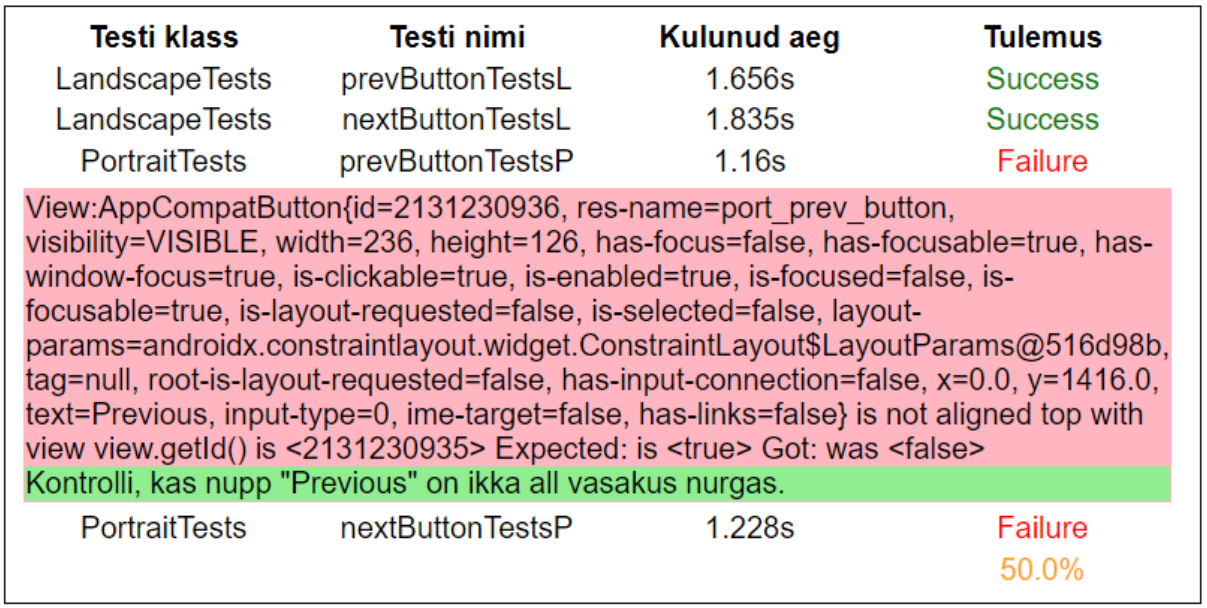


Joonis 8. Automaatse kodutööde hindaja veebirakenduse hindamistulemuste leht veebilehitsejas Google Chrome.

Tulemusi kujutatakse veebilehel tabelis (joonis 9), kus igas reas on info ühe kontrolli kohta (FN2.3). Tulpasid on tabelil neli: “Testi klass”, “Testi nimi”, “Kulunud aeg” ja “Tulemus”. Tulbas “Testi klass” on kirjas, missugusesse testi klassi käitatud test kuulus. Tulbas “Testi nimi” on kirjas, mis nimi testi nimi, mis kodutöö peal läbi prooviti. Tulbas “Kulunud aeg” näitab sekundites, kui kaua aega kulus testi jooksutamiseks Androidi emulaatoris. Viimases tulbas “Tulemus” on kirjas, kas esitatud kodutöö läbis vastava kontrolli edukalt või mitte. Roheline kirje “Success” viitab kontrolli edukale läbimisele, punane kirje “Failure” viitab kontrolli ebaedukale läbimisele (FN3.2). Tabeli viimasel real on täidetud ainult tulba “Tulemus” väärtus protsenti väljendava kirjega. Protsendi väärtus on vastavalt sellele, mitu protsenti testidest kodutöö läbis edukalt. Vahemikus 0 kuni 49.(9) on protsendi kirje punane, 50 kuni 99.(9) oranž ning 100 korral roheline.

Ebaedukate testide peale on tudengil võimalus klikkida. Peale klikki ilmub tabelisse testi alla täpsem informatsioon, mis sisaldab endas ilmnenud veateadet (FN3.3) ning vihjet (FN3.4). Veateade on kujutatud heleroosal taustal ning vihje helerohelisel taustal.

Tabeli kohal on ka olemas nupp “Proovi uuesti”, mille peale vajutades suunatakse tudeng tagasi eelmisele vaatele. Eelmise vaate veebivormis olevad väljad on eeltäidetud tudengi varasemalt valitud valikutega, mida ta ka vajadusel muuta saab.



Testi klass	Testi nimi	Kulunud aeg	Tulemus
LandscapeTests	prevButtonTestsL	1.656s	Success
LandscapeTests	nextButtonTestsL	1.835s	Success
PortraitTests	prevButtonTestsP	1.16s	Failure

View:AppCompatButton{id=2131230936, res-name=port_prev_button, visibility=VISIBLE, width=236, height=126, has-focus=false, has-focusable=true, has-window-focus=true, is-clickable=true, is-enabled=true, is-focused=false, is-focusable=true, is-layout-requested=false, is-selected=false, layout-params=androidx.constraintlayout.widget.ConstraintLayout$LayoutParams@516d98b, tag=null, root-is-layout-requested=false, has-input-connection=false, x=0.0, y=1416.0, text=Previous, input-type=0, ime-target=false, has-links=false} is not aligned top with view view.getId() is <2131230935> Expected: is <true> Got: was <false>

Kontrolli, kas nupp "Previous" on ikka all vasakus nurgas.

PortraitTests	nextButtonTestsP	1.228s	Failure
			50.0%

Joonis 9. Hindamistulemuste lehel olev tabel koos hindamistulemustega. Ühe ebaeduka testi veateade koos vihjega on avatud.

Loodud automaatne kodutööde hindamissüsteem saab tudengile anda tagasisidet sekundites kuni paaris minutis, olenevalt, kui palju töid hetkel süsteemis hindamisjärgus on. Praegu, kui tudengil on küsimus oma lahenduse kohta, siis tal on võimalus küsida seda loengu või praktikumi lõpus, või saata e-kiri aine õppejõule. Hindamissüsteemi on tudengil võimalus oma kodutööd esitada mitu korda enne kodutöö esitamise tähtaega ning saada kiiret tagasisidet iga tema esituse kohta. Kui süsteem töötab ka öösel, siis on kohest tagasisidet võimalik saada ka tudengitel, kes loovad kodutöid öösiti. Tagasisidega saab tudeng veenduda, et tema kodutöö funktsionaalsus on selline, nagu palutud. Kui ei ole, siis ta näeb, missugune

kontroll polnud edukas ning saab vihje ja veateate abiga välja mõelda, kuidas oma koodi vastavalt siluda. Kuna hindamissüsteem ei sisalda endas staatilise koodianalüüsi elemente, siis tudeng peab veenduma oma koodi struktuuri puhtuses enda materjalidega.

Kui töö autor MIoT ainet läbis 2020/21 sügissemestril, siis tema vaatluse kogemusel jõudis esitatud kodutöö tagasiside tema e-posti aadressile 1 kuni 15 tööpäevaga. Androidi kodutöid esitatakse MIoT aines keskmiselt iga nädala tagant. Siin võib tekkida olukord, kus tagasiside eelmise kodutöö kohta saabub hiljem, kui nädala pärast, ning tudeng ei saa tagasisidest saadud teadmisi rakendada järgmisesse kodutöösse ning temalt võidakse sama vea eest uuesti punkte maha võtta. Mida kiiremini tagasiside tudengini jõuab, seda rohkem on tudengil aega oma veast õppida.

4.2 Hindaja kasutamine õppejõu vaatepunktist

Praegu MIoT aines kontrollivad õppejõud tudengite lahendusi täiesti käsitsi. Kodutöö esitamisel peavad tudengid esitama oma kodutöö kokkupakitud lähtekoodi. Õppejõud peab seejärel lähtekoodi oma arvutis tööle saama ning kodutöö oma Androidi seadmes käivitama. Eduka käivitamise järel kontrollib õppejõud, kas kodutöö funktsionaalsus vastab kirjeldatud nõuetele. Õppejõud viib kodutöö peal läbi ka staatilist koodianalüüsi. Tagasisides annab õppejõud tudengile infot nii rakenduse funktsionaalsuse ja rakenduse elutsükli kui ka Kotlini koodi ja vaadete XML-koodi kohta.

Hindamissüsteem loodi mõttega, et õppejõud peamiselt kasutavad süsteemi andmebaasi ning loovad kodutöödele teste.

Andmebaasi saavad õppejõud lisada testide vihjeid, mida hiljem hindamissüsteem saab sisse lugeda ja tagasisidele kaasa anda. Vihjeid saab andmebaasi lisada JSON-failina. Täpset vihje struktuuri saab õppejõud vaadata andmebaasis juba olemas olevate vihjete pealt.

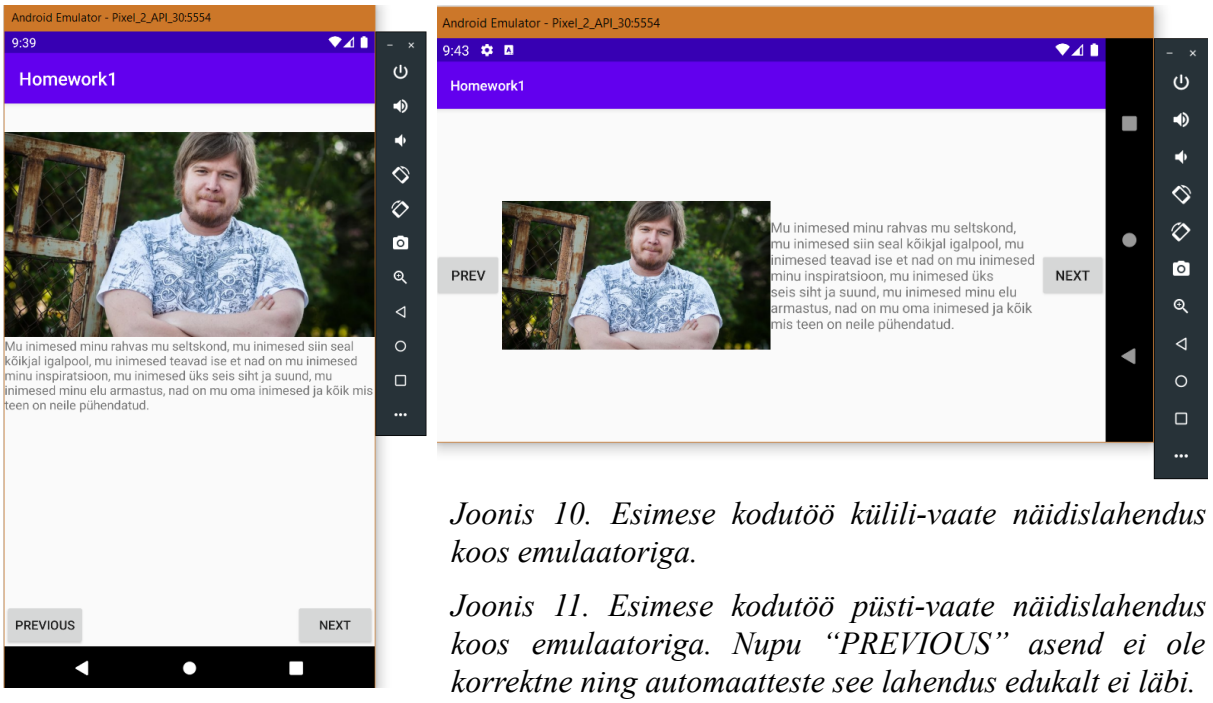
Automaathindamisel tekkinud logisid saavad õppejõud lugeda samuti andmebaasist. Seal on nendel on ülevaade tudengite esitatud kodutööde tulemustest. Kuna automaattestid hindavad kodutöö kasutajaliidese korrektsus ja funktsionaalsust, siis see säästab õppejõude käivitamast kodutöid nende endi Androidi seadmetel. Lisaks tulemusele on igas logis kirjas ka ajad, millal päring serverisse saabus ning millal töö päringu kallal algas ja lõppes. Kui on näha, et tööle kulunud aeg kaldub normaalist välja, siis saab õppejõud uurida selle põhjust tudengi lähtekoodist seda temalt eelnevalt küsides. Vajadusel saab õppejõud ka ise hindamissüsteemi veebirakendust kasutada, et veenduda tudengi kodutöö hindamistulemuste õigsuses.

Hindamissüsteemi näidisteste loodi 2020/21 õppeaasta sügissemestri MIoT aine esimesele ja teisele kodutööle.

Esimeses kodutöös on palutud luua inimese biograafia vaatamis-rakenduse kasutajaliides. Kodutöö eesmärk on õpetada, kuidas luua vaateid ning kuidas nendele elemente lisada. Kodutöö vaate elemendid on pilt, tekstikast ning kaks nuppu. Funktsionaalsust nuppudele lisatud ei ole, aga automaatselt testida on võimalik kõikide elementide asetust. Käesoleva töö autori esimese kodutöö näidislahendusi on kujutatud joonistel 10 ja 11.

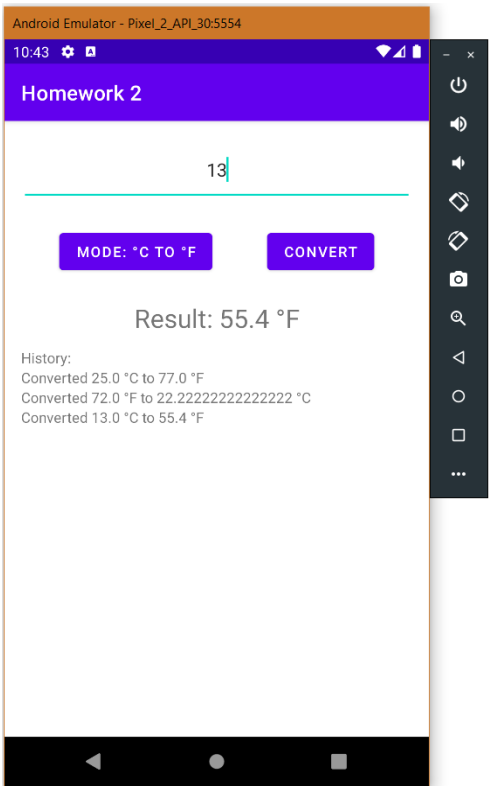
Teises kodutöös on eesmärgiks luua temperatuuride teisendamise rakendus. UI väljanägemine anti tudengile endale otsustada, põhirõhk sellel kodutööl on UI elementidega suhtlemisel. Selle kodutöö toimingu failis programmeeritakse juurde, kuidas UI komponendid omavahel suhtlema peavad. Vaatel peab olema tekstilahter, kuhu kasutaja saab sisestada arvu. Nupu “Convert” vajutamisel teisendatakse sisestatud temperatuuri väärtus vastavalt Celsiusest

Fahrenheiti või vastupidi olenevalt sellest, mis režiimis režiimivaliku nupp on. Režiimivaliku nupule vajutades peab režiim vahetuma. Käesoleva töö autori teise kodutöö näidislahendust on kujutatud joonisel 12.



Joonis 10. Esimese kodutöö külili-vaate näidislahendus koos emulaatoriga.

Joonis 11. Esimese kodutöö püsti-vaate näidislahendus koos emulaatoriga. Nupu "PREVIOUS" asend ei ole korrektne ning automaatteste see lahendus edukalt ei läbi.



Joonis 12. Teise kodutöö näidislahendus koos emulaatoriga.

Loodud hindamissüsteem ei kontrolli testide õigsust, millega kodutööde hindamist läbi viiakse. Androidi seadmes, kus testimine läbi viiakse, on eelnevalt testide APK-fail paigaldatud. Süsteem ainult käitab skripti, millega kodutööde hindamist testidega alustada. Seega peavad aine õppejõud testide korrektsuse eest ise hoolt kandma.

Tudengitele antava tagasiside kvaliteet sõltub loodud testide kvaliteedist. Esimese ja teise kodutöö näidistestide loomise käigus selgus, et kõike, mida on palutud kodutöö juhendis täita, ei ole võimalik täpselt kontrollida. Näiteks, mõlemas kodutöös on palutud vaadetele komponente lisada kasutades kitsendusi (*constraints*). Kitsendused määravad komponendil ära tema kauguse teiste valitud komponentide suhtes. Teegiga Espresso, mis on spetsialiseerunud rakenduse vaadete testimisele, on aga võimalik kontrollida ainult komponentide absoluutset asetust vaatel või joondatust teiste komponentidega. Selleks, et testimisteekide pakutud tööriistadega kodutöid võimalikult hästi kontrollida, peab edaspidi kodutööde disainimise ajal silmas pidama, mida on võimalik automaattestidega hinnata.

4.3 Hindaja kohandamine MIoT ainele

Loodud hindamissüsteem töötab ja täidab edukalt oma eesmärki anda kohest tagasisidet. Selleks, et aga süsteemi MIoT aines täielikult kasutada saaks, siis oleks süsteemi vaja täiendada mitmete uuenduse ja lisanditega.

Üks suurimaid arhitektuurilisi probleeme loodud süsteemi juures on see, et mitmed kasutatavad kataloogirajad on tagaliidesesse püsiprogrammeeritud. Kui süsteem MIoT aines kasutusele võtta, siis süsteemi lõplik server hakkab olema virtuaalmasin, mille operatsioonisüsteem on Linux. Linuxi kataloogirajad on Windowsist erineva struktuuriga ning seega oleks vaja rajad ümberkohandada. Lisaks on on hetkel kasutusel olevad kataloogirajad absoluutsed, mitte relatiivsed (näide joonisel 4), mis lubaksid süsteemi tagaliisese koos skriptidega implementeerida puhtamal kujul.

Samuti on püsiprogrammeeritud ka valideerimisprotsessis aktsepteeritavad kodutööd. Hetkel, kui õppejõud tahab lisada juurde kodutöid, mida aktsepteerida, siis ta peab avama süsteemi lähtekoodi ja aktsepteeritavate kodutööde massiivi lisama juurde vastava kodutöö koos paketi nimega.

Teine mõjukam probleem kasutajaga suhtlemine. Igale kodutöö hindamispäringule saadetakse tagasi üks vastus. Hetkel, kui kasutaja esitab oma kodutöö hindamiseks, siis kasutaja jäetakse ootele, kuni tema päring lõpuni töödeldakse. Töötlemise ajal kasutaja veebileht jääb värskenduse ootele. Kui kasutaja lehelt ära läheb või ise manuaalselt lehte värskendab, siis süsteem ei oska hindamistulemusi kasutajale edasi anda. Samuti jääb see ilma sihita päring süsteemi jooksma edasi, kuni kodutöö hindamistulemused on käes ning teised päringud peavad sisutu töö pärast ootama. Moderne lahendus oleks nt kohe kasutajale saata vastus hindamistulemuste lehega, mis aeg-ajalt pärib tagaliiseselt, kas hindamistulemused on olemas ning seejärel esitab need tabelkujul ilma lehte värskendamata. Sellise lahendusega saaks lehele juurde lisada ka kirje, mille sisu vahetub vastavalt sellele, mis staadiumis parasjagu on kasutaja kodutöö.

Kuna Flask mikroraamistik nõuab lehe näitamiseks kasutajale eelnevalt kogu infot, mida lehel kuvada, siis on keeruline implementeerida viisi, kuidas lehel olevat infot jooksvalt värskendada. Flaskiga on võimalik JavaScripti komponente kaasa anda, aga nii loodav lahendus ei oleks puhas ega loetav. Sel juhul on vaja kogu ees- ja tagaliides, mis on hetkel implementeeritud Flaski ja Pythoniga, vaja ümber tõsta nüüdisaegsematele JavaScripti

raamistikele. Kuna loodud hindamissüsteem on olemuselt veebirakendus, siis ümbertõstmine ei peaks olema ülimalt keeruline. Raamistikega saab veebilehtedele JavaScripti elemente lisada puhtal ja loetaval kujul. Populaarsed raamistikud, mis võivad sobida eesliidese implementeerimiseks, on nt Nuxt.js või React. Tagaliidese raamistikeks võivad sobida nt Express.js või Next.js.

Probleemikohaks on ka päringute töötlemise skaleeruvus. Praegune päringute järjekorrasüsteem sai loodud, sest Androidi emulaatoris sai korraga testida ühte kodutööd korraga. Kui emulaatoris töö testimine lõppes, siis saab alles järgmist kodutööd hindama hakata. Kui emulaatoris töö testimist oleks võimalik paralleliseerida, siis kuluks mitme kodutöö korraga hindamisele kokkuvõttes vähem aega. See tuleb kasuks siis, kui MIoT aine kodutöö esitamise aeg hakkab otsa saama ning süsteemi koormus on suurem kui teistel aegadel.

Skaleeruvust saaks lahedada kahel viisil. Esimene ja lihtsam viis on serveris korraga jooksutada ühe Androidi emulaatori asemel mitu identset emulaatorit. Mitme emulaatori vahel saaks sissetulevad päringud ära jaotada koormuse hajutamise eesmärgiga. Igale emulaatorile peaks anda ka eraldi järjekorra. Sellise lahenduse hea pool on see, et seda oleks võrdlemisi lihtne implementeerida. Halb on aga see, et emulaatorid hakkavad omavahel konkureerima serveri ressurssidele. Iga emulaator nõuab palju andmekandja mälu ning töötamiseks ka muutmälu. Emulaatorite arvu saaks kohendada vastavalt nõudlusele. Kodutöö esitamise tähtaja lähenemisel saaks süsteemi heaks tööle panna mitu emulaatorit. Muul ajal võiks hinnanguliselt piisata ühest töötavast emulaatorist, sest eeldatavalt on nõudlus väiksem. Kui serveril on piisavalt ressursse, mida emulaatoritele anda, siis on selline lahendus sobilik.

Teine viis, kuidas skaleeruvuse probleemi lahendada, on kasutada tarkvara konteineriseerimist. Konteinereid hindamissüsteemi loomiseks kasutasid ka Bruzual et al. [11]; nemad panid kogu oma süsteemi jooksma ühte konteinerisse. Lahenduse idee on see, et iga kord, kui hindamissüsteemi uus päring saabub, pannakse serveris käima uus konteiner, mis tegeleb saabunud kodutöö testimisega. Seejärel antakse konteinerile sisendiks vastav kodutöö ning väljundiks saadakse töötlemata hindamistulemused. Väljundi saabumisel hindamissüsteemi konteiner lõpetab oma töö ja see kustutatakse serverist. Sellise lahendusega saaks võimalusel ka päringute järjekorra ära kaotada, sest päringut hakatakse töötlema kohe, kui see hindamissüsteemi jõuab, mitte ei pea ootama, kuni emulaator oma töö lõpetab. Sellise lahenduse juures on aga ebakindel see, kui kaua just loodud konteiner kulutab aega testimisele. Konteineris vaja külmkäivitada Androidi emulaator enne, kui see on võimeline teste läbi viima. Töö jooksul kasutatud serveris (riistvara kirjeldus peatükis 3.4) on Androidi emulaatori külmkäivitamise peale kulunud keskmiselt 34±2 sekundit. Sellise üldajakulu juurde lisamisel muutub hindamine aga liiga aeglaseks ning süsteem kaotab oma eesmärgi, anda kasutajale kiiret tagasisidet kodutöö kohta.

Viimaseks probleemiks, mis takistab hindamissüsteemi täismahus kasutusele võtmist, on süsteemi mitmed turvaprobleemid. Praeguses süsteemis lastakse kasutajal otse sisestada oma matrikkel tekstilahtrisse. Miski ei takista kavalal tudengil sooritamast akadeemilist petturlust ning sisestada sinna midagi muud, kui oma matrikkel. Plagiaati oleks võimalik nt sooritada nii, et üks tudeng palub oma kaastudengil üles laadida oma töö esimese tudengi matrikli alt. Samuti tuvastati kaks võimalikku turvaauku. Esimeseks turvaauguks võib kujuneda asjaolu, et süsteem ei soorita sisestatud matrikli kontrolli ning kirje loetakse andmebaasi logisse otse sisse. Teiseks turvaauguks on see, et APK-failile ei tehta eelnevat turvakontrolli, vaid paigaldatakse otse Androidi emulaatorisse. See teeb võimalikuks juhu, kus pahatahtlik tudeng

võib APK-faili laadungina kaasa anda viiruse, mis Androidi emulaatori potentsiaalselt kasutuskõlbmatuks teeb.

Nimetatud turvaauke saaks teatud määral leevendada sellega, et enne, kui päringut hindamiseks süsteemile saata saab, peab kasutaja oma isikut tõendama. Selleks võiks kasutaja läbida autentimise, kus ta oma Tartu Ülikooli kasutajatunnuse ja salasõnaga sisse logib. Nüüd, kui pahatahtlik tudeng midagi ebasihilikku teeb, siis sellest jääb andmebaasi jälg maha.

4.4 Arutelu ja kommentaarid

Loodud alapeatükk sisaldab endas mõtteid ja lõike, mida töö autor otsustas mitte varem välja tuua eelnenud lõputöö alapeatükkides või soovis jätta lugejale lõppkõlaks.

Töö autor peab loodud hindamissüsteemi edukaks näiteks analoogse süsteemi kasutamisest MIoT aines. Süsteem ei olnud loodud eesmärgiga, et seda oleks võimalik otse ja kohe peale loomist MIoT aines kasutama hakata. Süsteem loodi MIoT ainele kontseptsiooni tõestuseks prototüübina.

Loodud süsteemis oli kõige suurem hindamisele kuluv ajakulu just testide jooksutamine Androidi seadmel. Töö autor ei leidnud viisi, kuidas testimisprotsessi seadmel kiiremini läbi viia. Iga kodutöö testimise ajad võivad minna minutiteni. Sarnast ajakulu probleemi kirjeldasid ka Bruzual et al. [11], kus ka nemad nentisid, et testide jooksutamise kiirust kodutöö kohta ei saadud alla paari minuti. Käesoleva töö autor ei suutnud välja selgitada, kas tegemist on süstemaatilise veaga, või aspektiga, millega tuleb lihtsalt leppida, kui analoogseid süsteeme luua.

Loodud hindamissüsteemi lähtekood on avalik ning saadaval versioonihaldussüsteemi GitHub repositooriumis lingil https://github.com/SanRoosalu/Android-homework-grader.

5. Kokkuvõte

Viimaste aastate jooksul on Android tõusnud maailma kõige kasutatumaks operatsioonisüsteemiks. See asjaolu annab mõista, et nõudlus üha kvaliteetsemalt programmeeritud telefonirakenduste vastu aina kasvab. Ka TÜ ATI on hakanud pakkuma ainet, mis õpetab rakendusi looma platvormile Android. Käesoleva bakalaureusetöö eesmärgiks oli luua TÜ mobiilirakenduste ainele automaatne Androidi kodutööde hindamissüsteem, kuhu kasutaja saaks oma Androidi kodutöö üles laadida ning saada kohest tagasisidet oma üleslaetud tööle.

Töö teoreetilises ülevaates tegi autor kokkuvõtte, mida kujutavad endast automaatne hindamine ning seda kasutavad süsteemid. Uuriti, kuidas kodutööde hindajaid on kasutatud Tartu Ülikooli Arvutiteaduse instituudis ning teistes maailma ülikoolides. ATI ainetes on automaathindajad ja -kontrollid kasutusel, kuid mitte ainetes, mis keskenduvad mobiilirakenduste arendamisele. Välisülikoolidest pärinevate teadustööde autorid on kirjeldanud, kuidas nemad on lähenenud ja millele keskendunud mobiilirakenduste automaatse hindaja arendamisele. Saadud informatsiooni analüüsiti ning kasutati ära autori enda loodud hindamissüsteemi arendamisel. Tehti ka lühiülevaated Androidi rakenduste arendusprotsessist ja nende testimisest.

Autori loodud automaatne kodutööde hindaja loodi peamiselt programmeerimiskeeles Python. Süsteemi veebirakenduse eesliidesena kasutati Pythoni mikroraamistikku Flask. Automaatteste viidi läbi virtuaalse Android emulaator-seadme peal. Kodutöö hindamise käigus kasutatavaid vajalikke andmeid loeti ja kirjutati MongoDB andmebaasi. Lõpplahenduse veebirakendus koosnes kahest veebilehest. Esimesel lehel sai kasutaja esitada oma kodutööd hindamiseks, teisel sai kasutaja oma kodutöö hindamistulemusi vaadata.

Arutleti, kuidas loodud süsteemi ainele kohandada, et see oleks paremini aine siseselt kasutatav. Peamised murekohad, mida loodud süsteemil parandada ja lahendada, on püsiprogrammeeritud koodiread, päringute töötlemise skaleeruvus, kasutajaga suhtlus ning turvaprobleemid. Süsteem loodi ainele kontseptsiooni tõestuseks prototüübina, mida autor hindab üldiselt edukaks.

Üht tüüpi edasised uurimisteemad saavad endas hõlmata analoogse süsteemi täismahus väljaarendamist ja ainele kohandamist ning vaadelda väljaarendatud süsteemi kasutust mobiiliarenduse aine siseselt. Teist tüüpi edasised uurimisteemad saavad endas hõlmata süvenenumat uurimist, mida täpselt on hetkel aktiivse ATI mobiiliarenduse aine kodutöödes võimalik kontrollida Androidi rakenduste testimisele spetsialiseerunud teekidega.

Viidatud kirjandus

[1] GlobalStats statcounter. Operating System Market Share Worldwide, Jan 2012 - Mar 2022. https://gs.statcounter.com/os-market-share#monthly-201201-202203 (13.03.2022)

[2] Statista. Number of available applications in the Google Play Store from December 2009 to March 2022. https://www.statista.com/statistics/266210/number-of-available-applications-in-the-google-play-store/ (13.03.2022)

[3] Sinaga AM, Wibowo PA, Silalahi A, Yolanda N. Performance of automation testing tools for android applications. In2018 10th International Conference on Information Technology and Electrical Engineering (ICITEE) 2018 Jul 24 (pp. 534-539). IEEE. https://ieeexplore-ieee-org.ezproxy.utlib.ut.ee/stamp/stamp.jsp?tp=&arnumber=8534756 (13.03.2022)

[4] Tartu Ülikooli õppeinfosüsteem. Mobiiliarvutus ja asjade internet LTAT.06.009. 2020/2021 sügis. https://ois2.ut.ee/#/courses/LTAT.06.009/version/ac1fa759-7dfc-fe62-f11a-8bed381c0f10/details (13.03.2022)

[5] Paiva JC, Leal JP, Figueira Á. Automated Assessment in Computer Science Education: A State-of-the-Art Review. ACM Transactions on Computing Education (TOCE). 2022. https://dl.acm.org/doi/pdf/10.1145/3513140 (27.03.2022)

[6] Zhang S, Saff D, Bu Y, Ernst MD. Combined static and dynamic automated test generation. InProceedings of the 2011 international symposium on software testing and analysis 2011 Jul 17 (pp. 353-363). https://dl.acm.org/doi/pdf/10.1145/2001420.2001463 (17.04.2022)

[7] SpotBugs. https://spotbugs.github.io/ (08.05.2022)

[8] Hamilton T. What is Dynamic Testing? Types, Techniques & Example. Guru99. https://www.guru99.com/dynamic-testing.html (17.04.2022)

[9] Õunmaa M. Automaattestide loomine sessioonõppe ainele „Sissejuhatus andmebaasidesse“. https://comserv.cs.ut.ee/ati_thesis/datasheet.php?id=66472 (08.05.2022)

[10] Plangi S. Automaatne programmeerimisülesannete kontrollija Tartu Ülikooli kursuse “Algoritmid ja andmestruktuurid” jaoks. https://comserv.cs.ut.ee/ati_thesis/datasheet.php?id=40821 (08.05.2022)

[11] Bruzual D, Montoya Freire ML, Di Francesco M. Automated assessment of Android exercises with cloud-native technologies. InProceedings of the 2020 ACM Conference on Innovation and Technology in Computer Science Education 2020 Jun 15 (pp. 40-46). https://dl.acm.org/doi/pdf/10.1145/3341525.3387430 (27.03.2022)

[12] Modesti P. A script-based approach for teaching and assessing Android application development. ACM Transactions on Computing Education (TOCE). 2021 Jan 22;21(1):1-24. https://dl.acm.org/doi/pdf/10.1145/3427593 (27.03.2022)

[13] Keuning H, Jeuring J, Heeren B. A systematic literature review of automated feedback generation for programming exercises. ACM Transactions on Computing Education (TOCE). 2018 Sep 28;19(1):1-43. https://dl.acm.org/doi/pdf/10.1145/3231711 (01.05.2022)

[14] Android Developers. Fundamentals of testing Android apps. https://developer.android.com/training/testing/fundamentals, (17.04.2022)

[15] Android Developers. What to test in Android. https://developer.android.com/training/testing/fundamentals/what-to-test (17.04.2022)

[16] Android Developers. Build local unit tests. https://developer.android.com/training/testing/local-tests, (17.04.2022)

[17] Android Developers. Build instrumented tests. https://developer.android.com/training/testing/instrumented-tests, (17.04.2022)

[18] Android Developers. Configure your build. https://developer.android.com/studio/build (17.04.2022)

[19] Android Developers. Android Debug Bridge (adb). https://developer.android.com/studio/command-line/adb (17.04.2022)

[20] schroepf. Android XmlRunListener. https://github.com/schroepf/TestLab/tree/master/android/android-xml-run-listener (08.05.2022)

[21] Flask. Welcome to Flask. https://flask.palletsprojects.com/en/2.1.x/ (10.05.2022)

[22] Python. Welcome to Python.org. https://www.python.org/ (10.05.2022)

[23] JetBrains. PyCharm: the Python IDE for Professional Developers by JetBrains. https://www.jetbrains.com/pycharm/ (10.05.2022)

[24] Android Developers. Android Studio. https://developer.android.com/studio (10.05.2022)

[25] MongoDB: The Application Data Platform. https://www.mongodb.com/ (10.05.2022)

Lisad

I. Terminid

Binaarfail Arvutile loetav fail, mis on salvestatud kahendvormingusse.	Binary file A file, which is readable only to a machine, saved in a binary format.
Eesliides Kliendipoolne vaade, mille abil klient saab suhelda tagaliisedega.	Front end A part of a program, which helps the user interact with the back end.
Lähtekood Programmeerimiskeeles kirjutatud programm selle lähtekujul.	Source code Program written in a programming language in its original form
Mikroraamistik Raamistik, mis hoiab oma struktuuri lihtsana ja lisadega laiendatavana.	Micro framework A framework, which keeps its core simple and extensible with additions.
Pukseerima Kasutajaliidese objekti hiirega teisaldama.	Drag and drop To move an UI component to a view using a mouse.
Püsiprogrammeeritud Programmi sisse paigutatud nii, et lõppkasutaja seda muuta ei saa.	Hard coded Embedded to a program, immutable by the end user.
Siluma Programmi vigu kõrvaldama.	To debug To remove errors from a program.
Skoop Programmi osa, milles deklaratsioon, väärtus, identifikaator vm kehtib.	Scope A part of a program, where a declaration, a value or a variable is applicable.
Tagaliides Programmi osa, mis vastutab loogika õigesti toimimise eest.	Back end A part of a program, which handles the logic operations for the user.
Trafarett-kood Koodilõik või -struktuur, mida kasutatakse korduvalt mitmes kohas väikeste variatsioonidega.	Boilerplate code Sections of code that are repeated in multiple places with little to no alternation.

II. Litsents

Lihtlitsents lõputöö reprodutseerimiseks ja üldsusele kättesaadavaks tegemiseks

Mina, **Sander Roosalu**,

1. annan Tartu Ülikoolile tasuta loa (lihtlitsentsi) minu loodud teose

   Automaattestidel põhinev lahendus Androidi kodutööde automaatseks hindamiseks õppetöös,

   mille juhendaja on **Jakob Mass**,

   reprodutseerimiseks eesmärgiga seda säilitada, sealhulgas lisada digitaalarhiivi DSpace kuni autoriõiguse kehtivuse lõppemiseni.

2. Annan Tartu Ülikoolile loa teha punktis 1 nimetatud teos üldsusele kättesaadavaks Tartu Ülikooli veebikeskkonna, sealhulgas digitaalarhiivi DSpace kaudu Creative Commonsi litsentsiga CC BY NC ND 3.0, mis lubab autorile viidates teost reprodutseerida, levitada ja üldsusele suunata ning keelab luua tuletatud teost ja kasutada teost ärieesmärgil, kuni autoriõiguse kehtivuse lõppemiseni.

3. Olen teadlik, et punktides 1 ja 2 nimetatud õigused jäävad alles ka autorile.

4. Kinnitan, et lihtlitsentsi andmisega ei riku ma teiste isikute intellektuaalomandi ega isikuandmete kaitse õigusaktidest tulenevaid õigusi.

Sander Roosalu

10.05.2022