TARTU ÜLIKOOL
Arvutiteaduse instituut
Informaatika õppekava

Maarika Markus

Veebirakendus funktsionaalprogrammeerimise ja -keele Idris õppimiseks

Bakalaureusetöö (9 EAP)

Juhendaja: Kalmer Apinis, PhD

Tartu 2022

Veebirakendus funktsionaalprogrammeerimise ja -keele Idris õppimiseks

Lühikokkuvõte:

Uute programmeerimiskeelte ja paradigmade õppimisel on oluline harjutamine. Kursuses „Funktsionaalprogrammeerimine“ kasutatud keele Idris õppimiseks on keeruline leida tasuta veebimaterjale, et iseseisvalt teemaga tutvuda. Bakalaureusetöö eesmärk on luua keskkond, mis võimaldab lihtsustada funktsionaalse programeerimise ja keele Idrise õppimisega alustamist.

Töös antakse ülevaade funktsionaalse programmeerimise paradigmast ja keelest Idris, tutvustatakse rakenduse loomise võimalusi, kasutatud tehnoloogiat ning valminud veebirakendust.

Võtmesõnad:

Veebirakendus, funktsionaalprogrammeerimine, Idris.

CERCS: P175 Informaatika, süsteemiteooria

Web Application to learn functional programming and language Idris

Abstract:

When learning new programming languages or paradigms practice is important. In the course „Functional programming“ programming language Idris is used for which it is difficult to find free online resources to learn by oneself. The aim of this thesis is to create an enivronment to make it easier to start learning functional programming and the language Idris.

In this thesis there is an overview of the functional programming paradigm and the language Idris, explanation of the different ways to create the environment, used technologies and the created web application.

Keywords:

Web application, functional programming, Idris.

CERCS: P175 Informatics, systems theory

Sisukord

1. Sissejuhatus

Aastate jooksul on loodud mitmeid programmeerimiskeeli, mis kuuluvad ühe või teise programmeerimise paradigma ehk stiili alla. Tartu Ülikooli Informaatika bakalaureuse õppekava kohustusliku osa hulka kuuluvad imperatiivsed keeled Python ja Java. Uudishimulikul tudengil on võimalik valikainete raames juurde võtta kursusi, mis käsitlevad teisi programmeerimise paradigmasid. Näiteks on mitmeid aastaid saanud tudengid kursuses „Programmeerimiskeeled“ tutvuda funktsionaalse programmeerimisega (FP).

Karoliine Holteri bakalaureusetöö „Funktsionaalprogrammeerimise õpetamine Idrises“ (Holter, 2021) uuris, kas Idris on sobilik keel nii klassikaliste kui ka tänapäevaste funktsionaalprogrammeerimise teemade õpetamiseks. Holter jõudis järeldusele, et Idris on sobilik. Töö tulemusena kasutatakse kursuses „Funktsionaalprogrammeerimine“ õpetatava keelena Idrist (Apinis, Holter ja Vene, 2021). Idrise õppimiseks on tasulisena saadaval keele autori Edwin Brady poolt kirjutatud õpik (Brady, 2017). Ainult teksti lugemine pole aga uue keele või paradigma õppimiseks ideaalne, vaja on ka seda harjutada. Keele dokumentatsiooni juurde kuulub küll lühiõpetus[1], kuid see nõuab kohaliku arenduskeskkonna seadistamist. Töö autor pole leidnud veebikeskkondi, mis võimaldaksid kohalikku arenduskeskkonda seadistamata Idrist harjutada.

Bakalaureusetöö eesmärk on luua nii kursuses „Funktsionaalprogrammeerimine“ kui ka eraldiseisvana kasutatav keskkond funktsionaalprogrammeerimise ja -keele Idris õppimiseks või sellega tutvumiseks. Loodava lahendusega tahetakse teha FP ja Idrisega alustamist lihtsamaks. Kui praegu tuleb harjutamiseks installida kompilaator, tekstiredaktor, pluginad jms, siis loodav keskkond võimaldaks kasutajal lisasammudeta Idrist õppima hakata. Kusjuures, töö eesmärk ei ole kohaliku arenduskeskkonna seadistamise vältimine täielikult, vaid vältimine alguses, kui huviline alles avastab keelt või paradigmat.

Töö koosneb neljast osast peatükkides kaks kuni viis. Teises peatükis antakse ülevaade kursusest „Funktsionaalprogrammeerimine” ning tutvustatakse funktsionaalse programmeerimise paradigmat ja keelt Idris. Järgmises osas tutvustatakse nõudeid rakendusele. Samas analüüsitakse töö eesmärgi täitmiseks võimalikke tarkvaralahendusi ja põhjendatakse kasutatud vahendi valikut. Neljandas peatükis antakse ülevaade valminud lahendusest ning võimalikest edasiarenduse suundadest. Lõpuks selgitatakse kasutatud tehnoloogiaid.

[1] https://idris2.readthedocs.io/en/latest/tutorial/index.html

2. Funktsionaalprogrammeerimine

Selles peatükis antakse ülevaade kursusest „Funktsionaalprogrammeerimine“ koos näidetega loodava rakenduse kasutamiseks kursusel. Samuti tutvustatakse funktsionaalse programmeerimise paradigmat koos Idrisega.

2.1 Kursus „Funktsionaalprogrammeerimine“

Kursususel „Funktsionaalprogrammeerimine“ õpetatakse FP paradigmat kasutades selleks funktsionaalset keelt Idris (Apinis, Holter ja Vene, 2021). Muuhulgas tutvustatakse kursuse käigus tüübipõhist programmeerimist, sõltuvate tüüpide teooriat ja kvantitatiivset tüübiteooriat.

Korraldusliku poole pealt kasutatakse keskkondi Courses[2] ja Moodle[3]. Courses keskkonnas on saadaval kõik loengute ja praktikumide materjalid. Iga loengu kohta on olemas loengu salvestus videona, enamasti on lisatud ka kasutatud slaidid ja näidiste lähtekood. Praktikumid on ühtlasi ka kodutööd ning edaspidi kasutatakse neid termineid samaväärsetena. Iga praktikumi kohta on eraldi alamlehekülg praktikumijuhendiga, kus on välja toodud lahendamist vajavad ülesanded ning vabatahtlik lisalugemine. Mõnikord on ülesannete juures ka teooria selgitusi. Välja on toodud ka n-ö nullis praktikum, kus on juhend kursusel osalemiseks vajaliku tarkvara paigaldamiseks oma arvutisse ning töökeskkonna tutvustus.

Moodle keskkonnas toimub kursuse hindamise osa. Iga loengu kohta on loengutest, mis on mõeldud enamasti loengu ajal lahendamiseks ja loengumaterjali kinnistamiseks. Samuti saab Moodle’is esitada oma kodutöid. Kodutöö lähtekood tuleb esitada vastava ülesande alla. Üldjuhul on iga praktikumis nõutud funktsiooni kohta olemas ka automaattestid, mille järgi võib tudeng teada saada potentsiaalse praktikumihinde. Automaatteste pole näiteks ülesannetele, kus pole täpsustatud konkreetset oodatavat väljundit. Selliste ülesannete puhul on oluline, et praktikumijuhendaja ise implementatsiooni üle vaatab. Juhendaja ülevaatamine võib mõjutada ka kodutöö lõpphinnet teiste funktsioonide puhul.

Loodav rakendus saab olla kasulik nii loengut kui ka praktikumi andes. Loengus tutvustatakse tavaliselt uut teemat, millega käib tihtipeale kaasas ka uut tüüpi ülesannete lahendamine. Rakendus võimaldaks väiksema hulga programmidega loengutes toimetada, st slaidide näitamise programmi, tekstiredaktori ja terminali võiks potentsiaalselt asendada loodava rakendusega. Praktikumides saaks rakendust kasutada analoogiliselt. Courses lehelt ülesannete jälgimise, tekstiredaktori ja terminali kasutamise asemel saaks ühes keskkonnas lugeda nii ülesannete kirjeldusi koos selgitava tekstiga kui ka kirjutada oma lahendus. Samuti ei peaks oma lahendust üles laadima keskkonda Moodle, et saada automaattestide tagasisidet.

2.2 Funktsionaalse programmeerimise paradigma ja Idris

Õpikus „Sissejuhatus funktsionaalsesse programmeerimisse“ (Nestra, 2010:8) kirjeldatakse programmeerimiskeelte liigitust selle järgi, millises paradigmas töötamiseks nad on mõeldud. Programmeerimiskeeled võivad seega olla näiteks funktsionaalsed, loogilised, protseduurorienteeritud, objektorienteeritud. Idrise looja Edwin Brady ütleb (Brady, 2017:13), et „Idris on puhas funktsionaalne programmeerimiskeel“.

Funktsionaalse programmeerimise eelisteks peetakse võimalust lahendada probleeme abstraktsemal tasemel kui seda lubavad tavalised programmeerimiskeeled (Nestra,

[2] https://courses.cs.ut.ee/
[3] https://moodle.ut.ee/

2010:14). Kui tugevalt tüübitud funktsionaalprogrammeerimiskeeltes kood kompileerub, saab üldjuhul olla ka selle korrektsuses kindlam. Näiteks, kui tüüpilises keeles peaks maatrikseid korrutades kontrollima sisendite mõõtmeid, siis Idrises saab maatriksite korrutamise funktsiooni tüübis väljendada kitsendusi sisendite mõõtmetele. Samas tuuakse puudustena esile funktsionaalses paradigmas kirjutatud programmide kulukat aja- ja mälukasutust.

Nestra (2010:8) toob välja, et funktsionaalne programmeerimine kuulub programmeerimisparadigmade hierarhias deklaratiivse programmeerimise alla. Deklaratiivne programm kirjeldab matemaatilisi objekte ja nendevahelisi seoseid. Idrist tutvustavas õpikus „Type-driven Development with Idris“ (Brady, 2017:13) kirjeldab autor, et funktsionaalses keeles koosnevad programmid funktsioonidest, kusjuures programmi töö koosneb nende funktsioonide väärtustamisest.

Lisaks sellele, et funktsionaalses programmeerimiskeeles on funktsioonid esimese taseme konstruktsioonid, toob Brady välja, et Idrises on ka tüübid esimese taseme konstruktsioonid: “Esimese taseme konstruktsioon on selline, mida käsitletakse väärtusena ilma kasutuskohtade süntaktiliste piiranguteta“ (Brady, 2017:22). Näiteks, saab kirjutada funktsioone, mis arvutavad ja seejärel tagastavad tüübi.

Keele puhtuse all peab Edwin Brady silmas, et funktsiooni tulemus sõltub ainult sisenditest. See tähendab, et funktsioonil pole kõrvalefekte nagu erindite viskamine või globaalsete muutujate ülekirjutamine (Brady, 2017:13). Kui funktsioonil puuduvad kõrvalefektid, piisab funktsiooni sisend- ja väljundtüüpide lugemisest, et mõista, mida funktsioon saab või ei saa teha (Brady, 2017:14).

3. Rakenduse loomise võimalusi

Selles peatükis antakse ülevaade loodava rakenduse nõuetest ning analüüsitakse töö realiseerimiseks uuritud võimaluste vastavust nõuetele. Tutvustatakse OpenVSCode Serveri veebipõhist IDEt. Seejärel selgitatakse Codecademy stiilis veebirakendust. Viimase uuritud vahendina tutvustatakse Moodle'i plugina loomise võimalust. Lõpuks täpsustatakse ja põhjendatakse valitud vahendit.

3.1 Loodava rakenduse nõuded

Nõuded on leitud koostöös juhendajaga lähtuvalt algaja Idrise programmeerija vajadustest. Need on järgmised:

- Mõisteid õpetava tekstiga on koos testid või ülesanded. Rakendus annab ülesannete lahendustele ka tagasisidet.
- Rakendus hoiab meeles järge, millised teemad on kasutaja läbinud.
- Koodi saab rakendusest kätte Moodle'isse esitamiseks või rakendusest otse Moodle'isse esitada.
- Suurem osa kursusest peab olema läbitav rakenduse abil.
- On avatud kõigile, st mitte ainult kursuse kuulajate jaoks.

Teisisõnu, rakendus peaks võimaldama lihtsamat ülesannete lahendamise töövoogu. Juba lahendamise ajal oleks võimalik kasutajal oma implementatsiooni korrektsust automaattestida. Rakendus annaks tagasisidet süntaksi-, tüübi- ja loogikavigade kohta.

3.2 OpenVSCode Serveri veebipõhine IDE

OpenVSCode Serveri[4] projekt võimaldab kasutada populaarset IDEt Visual Studio Code läbi veebibrauseri. IDE ehk *Integrated Development Environment* on programmeerimis-keskkond, mis koosneb lähtekoodi redaktorist, kompilaatorist ja/või interpretaatorist ning programmeerimise automatiseerimise abivahenditest[5]. Selleks kasutatakse Dockeri konteinerit, milles jookseb Visual Studio Code veebirakendus.

Projekti kasutamine antud töö kontekstis tähendaks uue Dockeri tõmmisfaili tekitamist. Docker on platvorm, mis võimaldab pakendada oma rakendusi isoleeritud keskkondadesse ehk konteineritesse[6]. Tõmmisfail on arendaja loodud keskkonna seis mingil hetkel. Selle abil saab käivitada loodud keskkonna instantse ehk konteinereid. Tõmmisfailis oleks peale OpenVSCode Serveri ka Idrise interpretaator ja Visual Studio Code'i Idrise plugin. Lisaks oleksid kaasas kõik kursuse õppematerjalid, mida tudeng iseseisvalt läbida saab. Veel võimaldaks see kasutada Visual Sudio Code'i terminali käskude jooksutamiseks. Lahenduse teeks kasutajale mugavamaks, kui kirjutada Visual Studio Code'ile plugin, mis võimaldaks õppematerjalide läbimist testida ja tudengile testide tagasisidet kuvada.

Kirjeldatud lahendus nõuaks tudengilt Dockeri paigaldamist ja oma arvutis konteineri jooksutamist. Konteineri saaks siduda mingi lokaalse kaustaga, nii et lokaalsed failid oleksid nähtavad konteineris ja konteineris loodavad failid oleksid nähtavad lokaalselt. Nii on kasutajal lihtne oma lähtekood edaspidiseks kasutamiseks salvestada.

Docker võimaldab luua sama arenduskeskkonna erinevatel masinatel, olenemata nende operatsioonisüsteemidest. Nii saaks olla kindel, et tudengitel on materjalidest ning vahenditest sama versioon. Näiteks, võib juhtuda, et kursust läbivad tudengid paigaldavad

[4] https://github.com/gitpod-io/openvscode-server
[5] Andmekaitse ja infoturbe leksikon
[6] Docker overview

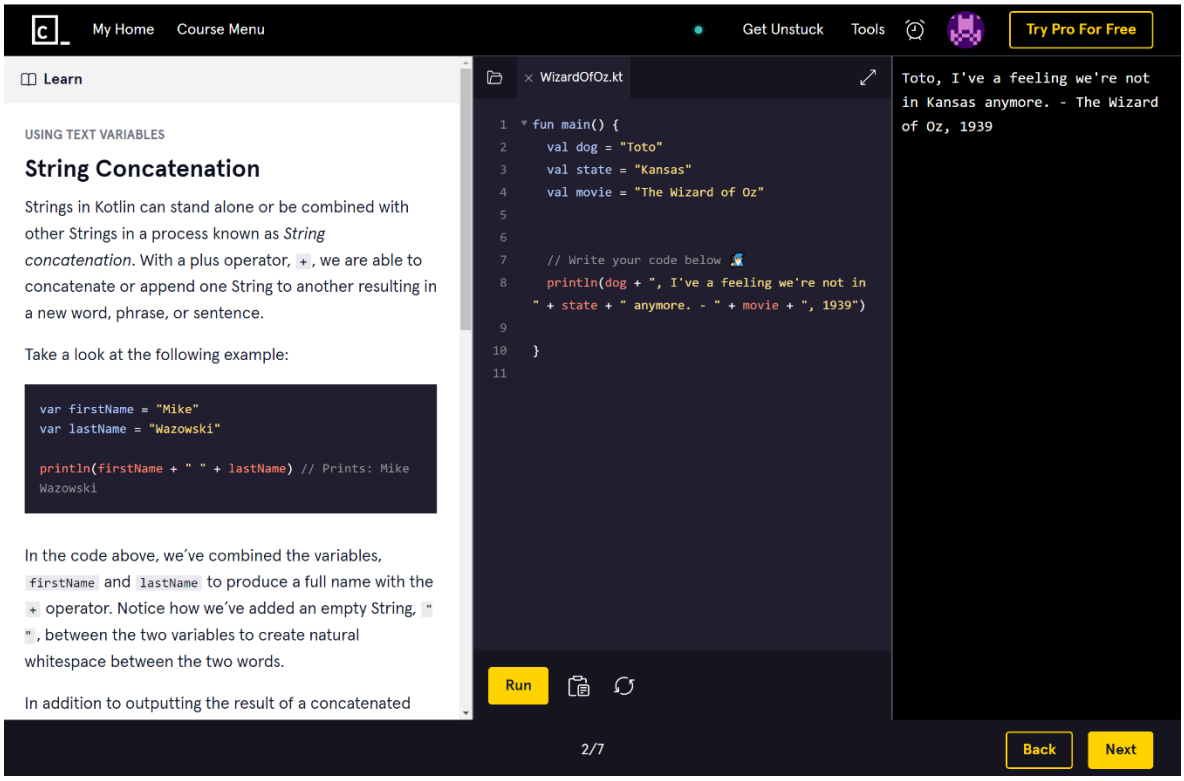
endale erineva versiooni Idrisest. See võib tekitada olukorra, kus õppematerjalides kasutatud Idrise võimalused pole kõigile tudengitele samamoodi kättesaadavad. Samas on kõigil huvilistel võimalus pääseda ligi loodud tõmmisfailile ja õppimismaterjalidele sõltumata kursusel osalemisest.

Kirjeldatud lahendus küll võimaldaks tudengitel vältida ise Idrise kompilaatori, tekstiredaktori ja erinevate pluginate installimist, kuid see asenduks Dockeri installimisega. Seega uuritud võimalus ei lahenda olukorda, kus kasutaja saaks ilma midagi installimata alustada funktsionaalprogrammeerimise ja -keele Idrisega tutvumist.

3.3 Veebipõhine õpikeskkond

Codecademy[7] on veebirakendus, mis võimaldab kasutajatel õppida erinevaid arvutiteaduse teemasid. Kursused keskenduvad nii üksikutele keeltele, näiteks JavaScript, Java, Python, kui ka raamistikele ja konkreetsetele teemadele, näiteks React, veebiarendus, andmeteadus, jne. Suurem osa keeltele orienteeritud kursustest on kättesaadavad ka tasuta versioonis, kuid spetsiifilisemate kursuste läbimiseks peab liituma tasulise versiooniga.

Codecademysse sisse logides avaneb kõigepealt kasutajaspetsiifiline avaleht, kus on näha läbimisel olevad kursused ning statistika õppimise kohta. Valides mõne juba läbimisel oleva kursuse jätkamise või uue kursusega alustamise avaneb õppimise vaade (vt joonis 1). Õppimise vaade koosneb peale päise ja jaluse n-ö sisulisest osast, mis on jaotatud kolmeks: vasakul on õppematerjal ülesannetega, keskel on tekstiredaktor ning paremal konsool.

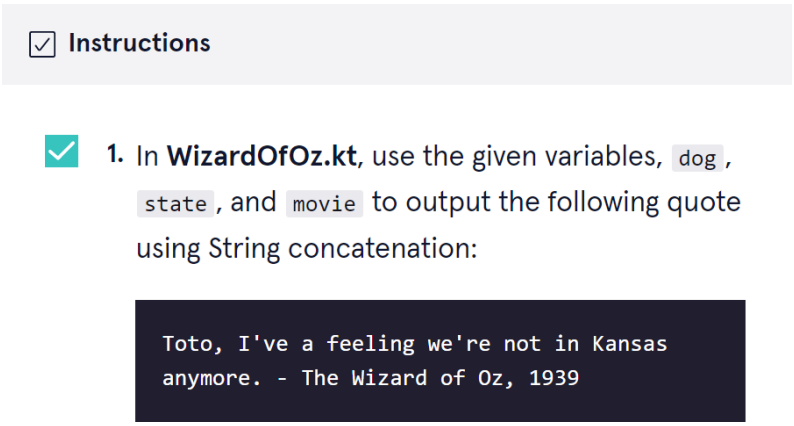


Joonis 1. Ekraanitõmmis Codecademy õppimise vaatest.

Õppematerjali lõpus on ülesannete osa (vt joonis 2), kus kõigepealt kuvatakse esimene ülesanne. Ülesande lahendus tuleb kirjutada lehe keskel olevasse tekstiredaktorisse. Kui lahendus on valmis, tuleb vajutada nupule „Run“ ning kasutaja kirjutatud programmi tulemus kuvatakse lehe paremas osas olevas konsoolis (vt joonis 1). Kui väljund vastab

[7] https://www.codecademy.com/

oodatule värvub ülesande kirjelduse ees olev kast roheliseks, mille sisse ilmub linnuke ning tekib võimalus liikuda edasi järgmise materjali juurde, ehk all paremas nurgas olev nupp „Next“ värvub kollaseks.



Joonis 2. Ülesannete osa Codecademy õppematerjali lõpus.

Sarnast õppimisvaate lahendust saaks kasutada ka käesoleva töö kontekstis. Selline lahendus võimaldaks kasutajal alustada õppimist, ilma et ta peaks ise muud tegema, kui liikuma õigele veebiaadressile. Vähendamaks takistust alustamisega, ei nõuaks loodav veebirakendus kasutajalt registreerumist, teavet kasutaja edusammudest hoitaks iga kasutaja veebibrauseri mälus.

Veebirakenduse puhul saab arendada vajalikud funktsionaalsused, et suurem osa kursusest „Funktsionaalprogrammeerimine“ oleks selle kaudu tehtav. Samuti saab lisada võimaluse jätta meelde kasutaja järge koos lähtekoodiga. Selleks saab lisada rakendusele andmebaasi või leida alternatiivne salvestusmeede. Kasutaja kirjutatud koodi kättesaamiseks saab lisada nupu, mis koodi kasutaja arvutisse alla laeb.

Codecademy sarnase veebilehe loomise jaoks tuleb esmalt uurida, milliseid tekstiredakorite teeke saaks kasutada, kuidas kasutaja poolt kirjutatud Idrise koodi kontrollida ja jooksutada ning kuidas kasutajale õppematerjale ja ülesandeid kuvada. Veel on oluline uurida veebilehe turvalisust ning erinevate kasutajate programmide isoleerituse tekitamist.

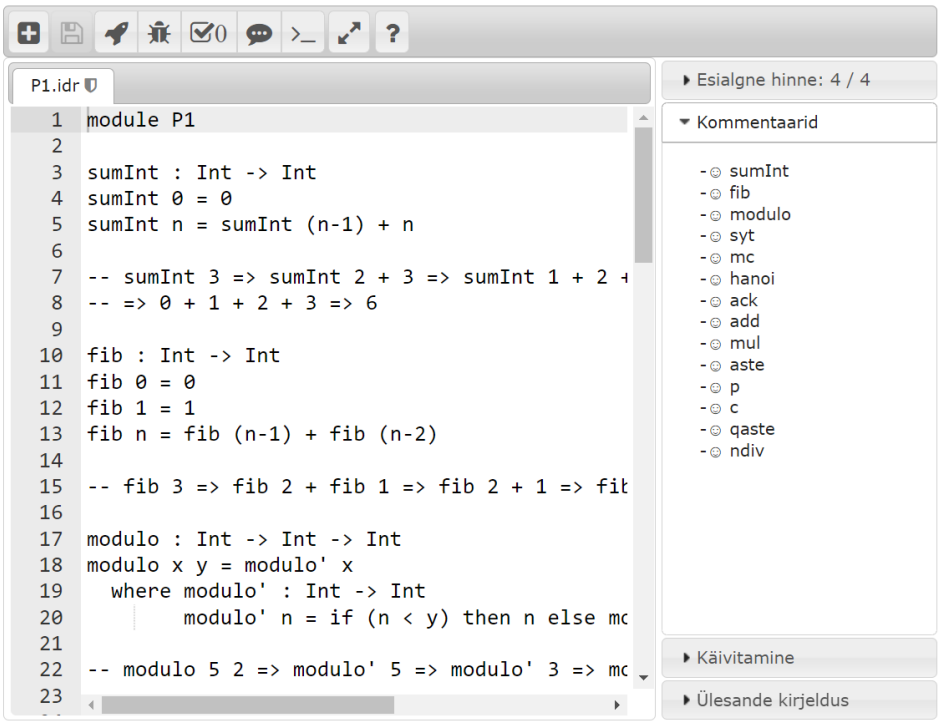
Veebirakenduse eelis on, et see on kasutajale selge eesmärgiga ning lihtne kasutama hakata.

Uuritud võimaluse puudustena võib välja tuua selle mahukuse. Ei saa täiesti kindel olla, et autor õigeks ajaks sellise rakenduse suudab valmis teha. Samas on selline projekt hariv.

3.4 Moodle’i plugin

Kursuse „Funktsionaalprogrammeerimine“ hindamise osa toimub keskkonnas Moodle, kuhu saab esitada oma erinevate kodutööde lahendusi. Hindamiseks kasutatakse automaatteste koos Moodle’i pluginaga VPL ehk *Virtual Programming Lab*[8]. Tudengil on võimalus oma kood üles laadida ülesande juurde või kasutada VPLis olevat tekstiredaktorit (vt joonis 3), kuhu oma kood kirjutada või kopeerida. Seejärel saab oma koodi kompileerida ning jooksutada automaatteste. Testide jooksmise järel kuvatakse tudengile tagasiside testide läbimise kohta (vt joonis 3).

[8] https://moodle.org/plugins/mod_vpl



Joonis 3. Ekraanitõmmis Moodle'i ülesande esitamise tekstiredaktorist koos lahenduse tagasisidega.

Kuna kursuse hindamise osa juba toimub Moodle'is, siis oleks üheks variandiks kirjutada kursuse tarbeks Moodle'i plugin, mis võimaldaks tudengil iseseisvalt materjale läbida ja ülesandeid lahendada, saades automaattestidelt tagasisidet.

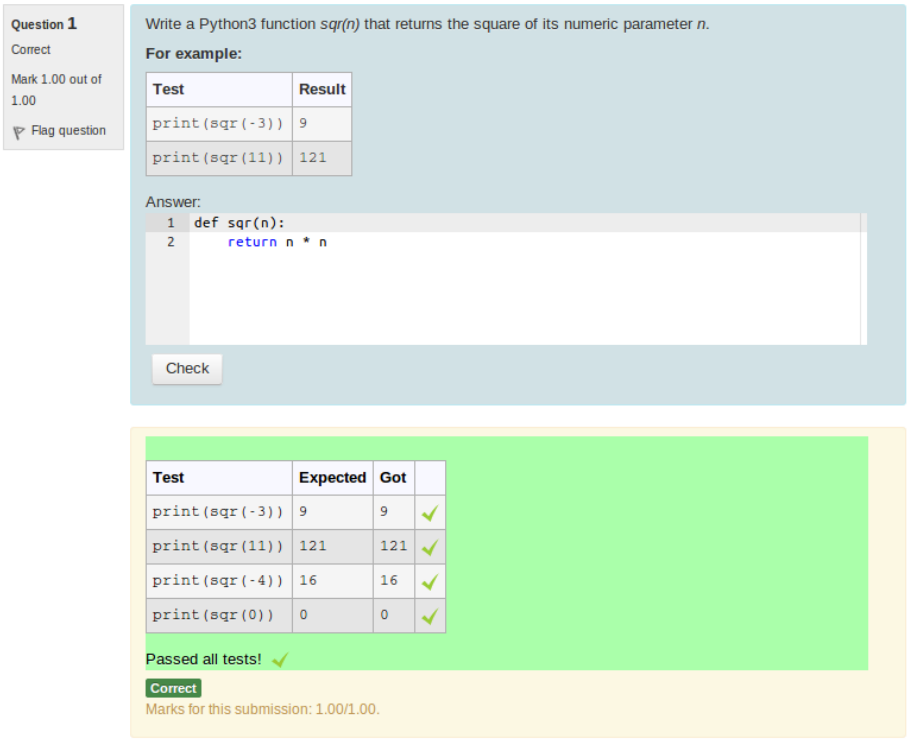
Plugin võimaldaks paigutada õppematerjali kõrvuti tekstiredaktoriga ning testida tudengi koodi funktsiooni haaval. Ülesannete juurde peaks tekkima tagasiside ülesande läbimise kohta, näiteks linnuke, kui ülesanne läbitud ja rist, kui on läbimata.

Selline lahendus võimaldaks tudengil ühes keskkonnas kursuse materjalidega tegeleda. Eelistena on Moodle'is mugav tudengite edasijõudmist meeles pidada ning Moodle on juba kogu kursuse vältel kasutusel olev keskkond.

Moodle'i plugina kirjutamist uurides avastas autor juba olemasoleva plugina CodeRunner[9] (vt joonis 4), mis teistsuguse vorminguga juba võimaldab oodatud lahendusele sarnast käitumist. Kuna CodeRunneril puudub Idrise tugi, siis tuleks see lisada.

CodeRunner võimaldab luua testi, mille vastuseid (koodilahendusi) saab kirjutada tekstiredaktorisse ning lahenduse korrektsust saab kontrollida automaattestidega. Joonisel 4 on CodeRunneri poolt toodud näide lihtsast testi ülesandest. Kasutajal tuleb kirjutada keeles Python funktsioon nimega *sqr(n)*, mis tagastab arvulise sisendi *n* ruudu. Kasutaja kirjutatud koodi testitakse nelja testjuhuga, kus tuuakse välja funktsiooni kutse koos sisendiga, oodatud tulemus ja tegelik tulemus. Selline lahendus võimaldab testide läbimise järgi automaatselt tudengite lahendusi hinnata.

[9] https://moodle.org/plugins/qtype_coderunner



Joonis 4. CodeRunneri näidis testülesandest tagasisidega[9].

Puuduseks võib tuua, et Moodle'i plugina lahenduse kasutamine on seotud Tartu Ülikoolis õpetatava kursusega, mis ei ole ülikoolivälistele huvilistele mugavalt kättesaadav. Nii juba kasutusel oleva VPL plugina kui ka CodeRunneri kasutamisele võtmise puuduseks võib tuua, et see pole ilma täiendusteta uue teema õppimiseks kuigi sobiv[10].

3.5 Valitud vahend ja põhjendus

Uuritud lahendustest sobisid nõuetega veebirakenduse ja Moodle'i plugina loomine. OpenVSCode Serveri veebipõhise IDE kasutamine ei lahendanud nõuet, et kasutajal oleks võimalikult lihtne funktsionaalprogrammeerimise ja Idrise õppimisega alustada ehk oleks endiselt nõudnud kohaliku keskkonna seadistamist.

Töö sisulise osa teostamiseks valis autor veebirakenduse loomise. Autori ambitsioon oli selle tööga arendada keerulisema veebirakenduse tegemise oskust kasutades tänapäevaseid vahendeid. Rakenduse sisu sai täpsustatud koostöös juhendajaga, mille raames uuriti ka teisi eelpool mainitud võimalusi sisu pakendamiseks.

Leiti, et veebirakenduse loomine on põnevam ja harivam ülesanne, kui Moodle'i plugina loomine. Lisaks jääb autorile võimalus rakendust edasi arendada ning huvi korral sisu juurde luua väljaspool ülikooli kursuste konteksti. Samuti võimaldab selline lahendus saada kogemuse, kuidas teha kasutatav toode algusest lõpuni.

[10] Kalmer Apinis tagasisides

4. Valminud veebirakendus

Selles peatükis tutvustatakse valminud veebirakendust, selle kasutajaliidest ja kasutaja koodi testimist, analüüsitakse arendusprotsessi ning pakutakse välja võimalikke edasiarendusi. Valminud veebirakenduse lähtekoodiga saab tutvuda GitHubi repositooriumis aadressil https://github.com/maarikamarkus/learn-idris-app.

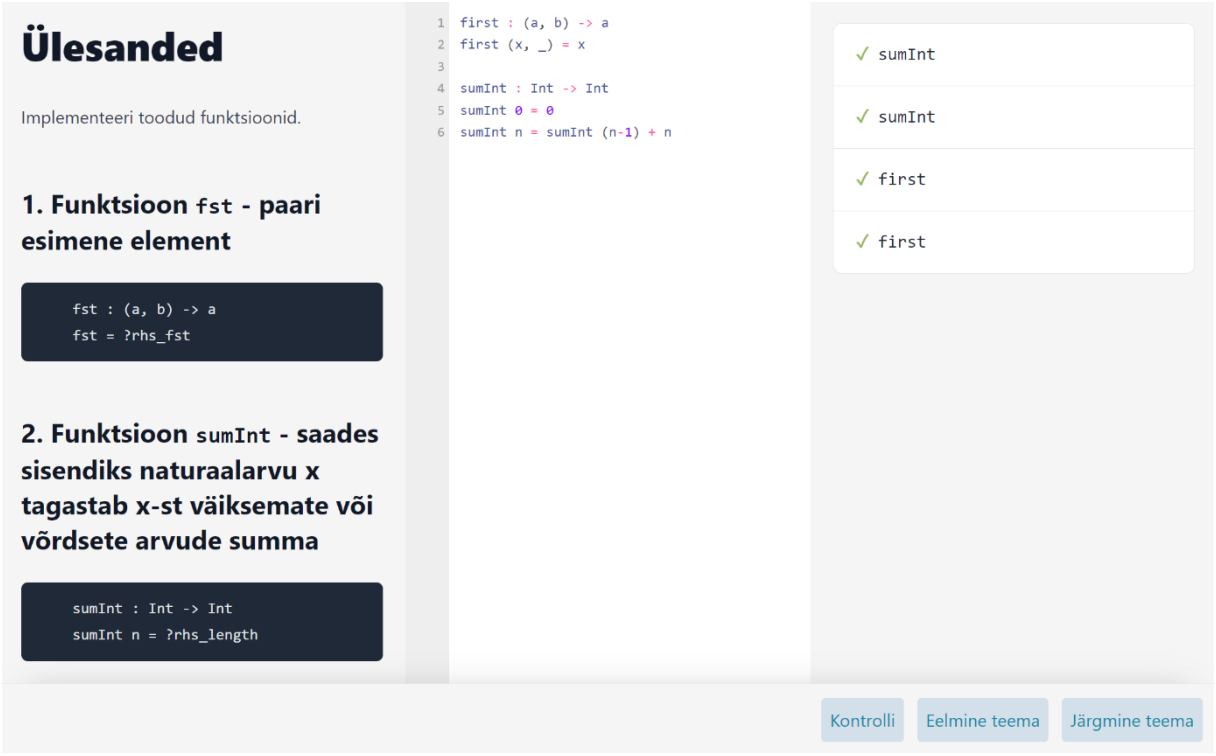
4.1 Kasutajaliides

Valminud veebirakendus koosneb kahest vaatest: avaleht koos viidetega materjalile ja ülesannetele ning ülesande vaade.

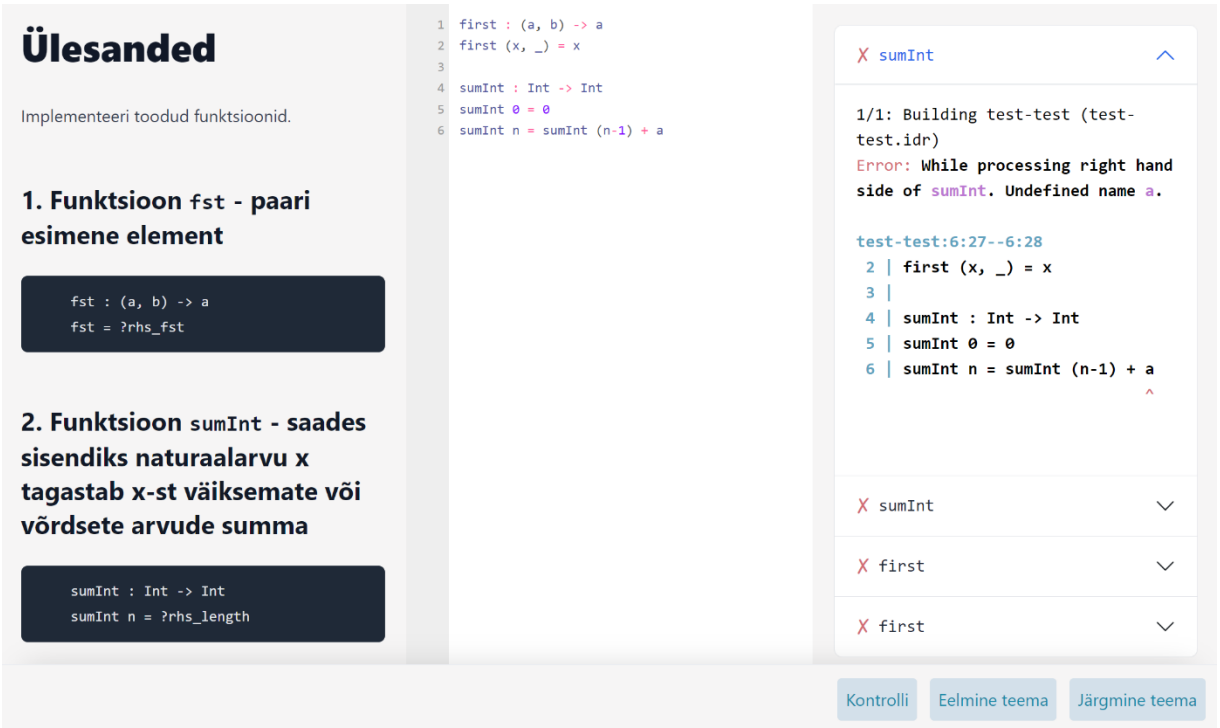
Avaleht (vt Joonis 5) koosneb n-ö ruudustikust teemadega, mis viivad vastava teema materjali ja ülesannete juurde. Samuti on iga teema kohta võimalik näha, kas vastavate ülesannete lahendused läbisid kõik testid. Kui teema on n-ö lahendatud, ehk kõik testid läbitud, kuvatakse vastava teema ette linnuke, vastasel juhul on lihtsalt teema pealkiri.

Joonis 5. Veebirakenduse avaleht.

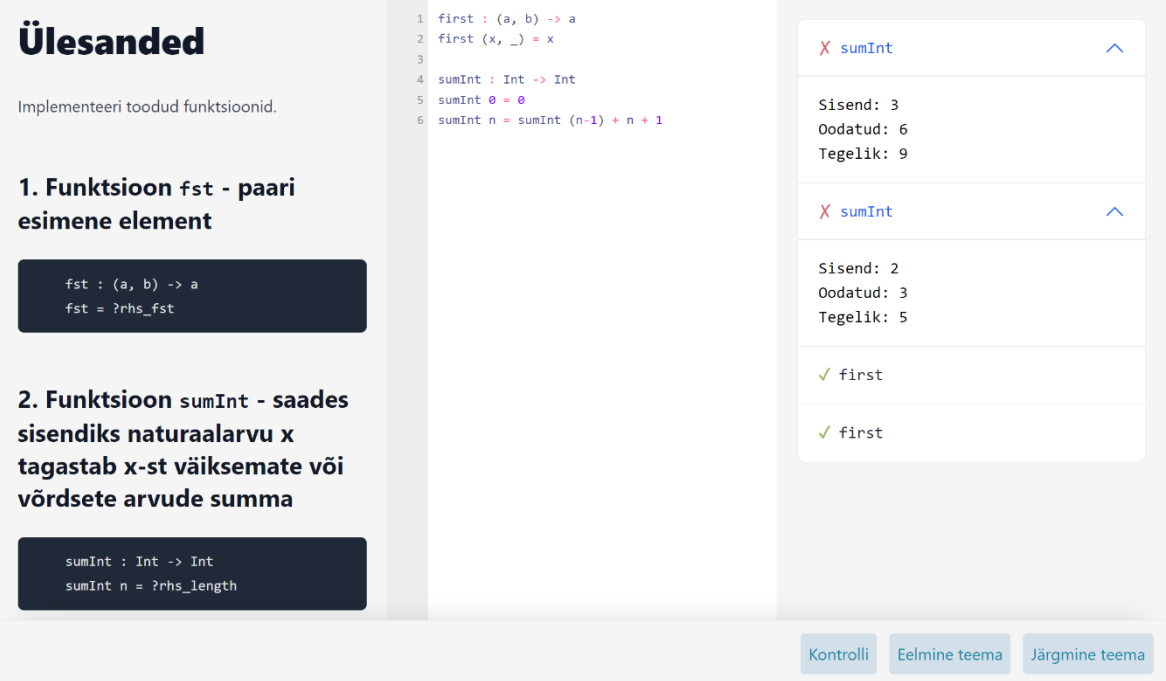
Ülesannete kuva (vt joonis 6) koosneb lihtsustatult kolmest osast: (1) materjal koos ülesannete kirjeldustega, (2) tekstiredaktor lahenduse kirjutamiseks, (3) testide tulemused ja tagasiside. Lisaks on veel jalus nuppudega „Kontrolli“, „Eelmine teema“ ja „Järgmine teema“. Nupp „Kontrolli“ võimaldab kasutajal oma lahendusega teste jooksutada ja vigade korral saada tagasisidet. Tagasisidet on kahte tüüpi vigade kohta: kompileerimisvead ja loogikavead. Kui kasutaja kood ei kompileeru, kuvatakse kompilaatori väljund vigadega (vt joonis 7). Kui kasutaja kood kompileerub, kuid sisaldab loogikavigu, kuvatakse iga läbikukkunud testjuhu ette ristike ning samas antakse tagasiside, kus täpsustatakse sisend, oodatud ja tegelik tulemus (vt joonis 8). Iga läbitud testjuhu kohta kuvatakse testjuhu ette linnuke ning täpsustavat teavet ei jagata (vt joonis 8).



Joonis 6. Ülesande kuva.



Joonis 7. Testide tagasiside kompileerimisvigade korral.



Joonis 8. Testide tagasiside loogikavigade korral.

Kasutaja saab igal korral jätkata sealt, kus eelmine kord pooleli jäi. Kirjutatud kood ja selle testimise tulemused jäetakse iga teema kohta meelde kasutaja enda brauseri *local storage*isse. *Local storage* on brauseri atribuut, mis lubab tähtajatult salvestada võti-väärtus paare[11]. Nii ei pea kasutajaid eraldi autentima ega kasutama rakenduses andmebaasi. Samas saab kasutaja igal korral jätkata sealt, kus eelmine kord pooleli jäi.

4.2 Kasutaja koodi testimine

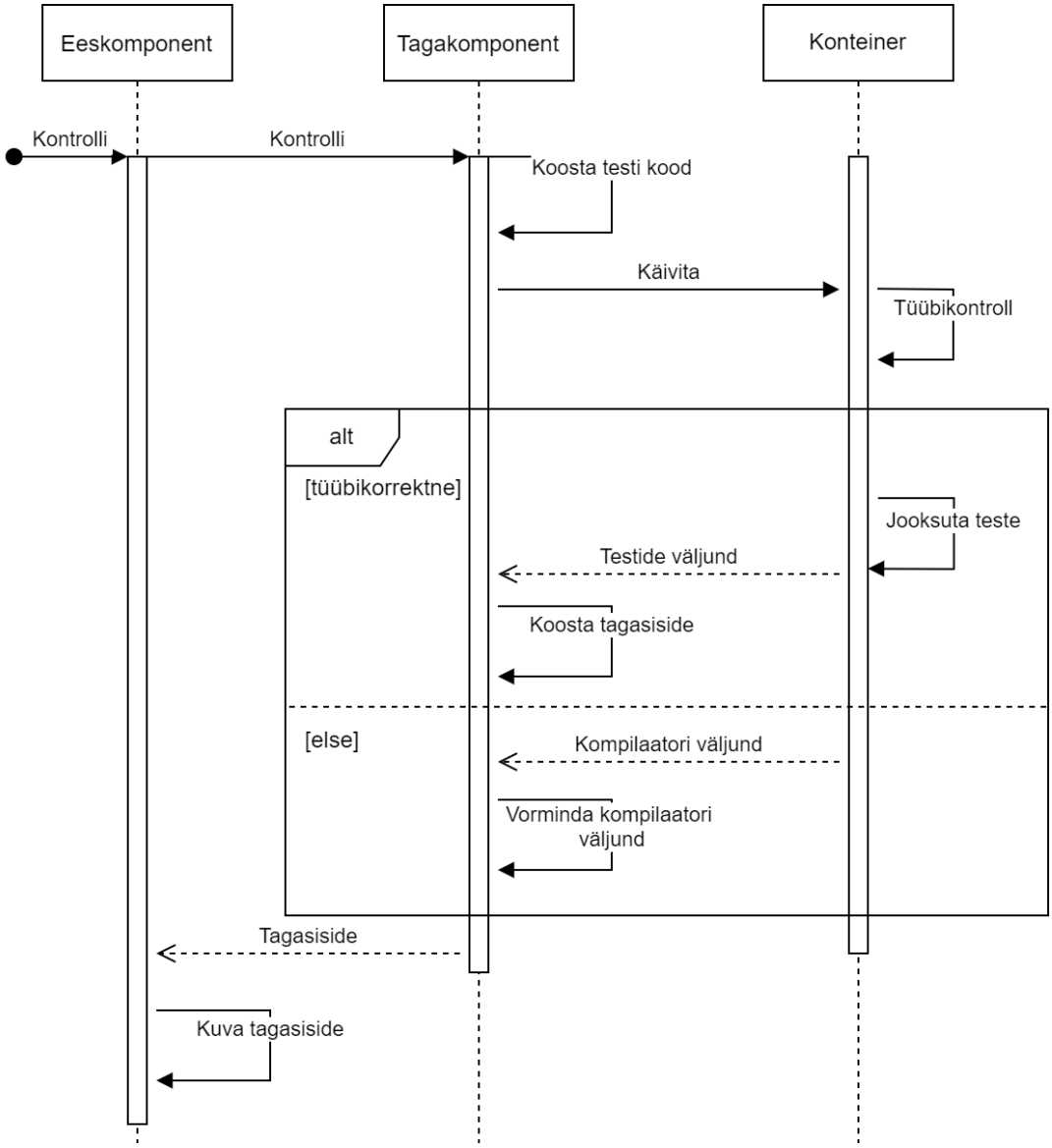
Kasutaja sisestatud koodist koostatakse testprogramm, mis kutsub testjuhtudes kirjeldatud funktsiooni ja trükib tulemuse (vt peatükk 4.3 Materjalide ja testide kirjeldamine). Testprogrammi saab tüübikontrollida ja jooksutada Idrise interpretaatori abil. Kui Idrise interpretaatorit jooksutatakse samas masinas tagakomponendiga, siis võib pahatahtlik kasutaja teha serveris ükskõik mida, näiteks hõivata palju kettaruumi või protsessori aega. Selle vältimiseks tuleks kasutaja kirjutatud koodi jooksutamist isoleerida. Isoleerimiseks otsustati kasutada Dockeri konteinerit.

Tagakomponent vastutab koodi testimise eest. See toimub järgnevate sammudena (vt joonis 9):

1. Loetakse eeskomponendist tulnud kasutaja kood.
2. Loetakse *yaml* failist testjuhud.
3. Kasutaja koodist koostatakse testprogramm, lisades *main* funktsioon. *Main* funktsioon kutsub kasutaja kirjutatud funktsiooni testjuhus antud sisenditel. Kusjuures, iga testjuhu kohta luuakse uus programm.
4. Kokkupandud programm antakse sisendina konteinerile.
5. Konteiner jooksutab testprogrammi ja tagastab väljundi.
6. Kui konteineri jooksutamine ebaõnnestus, siis esinesid programmis tüübivead. Konteineri väljundis asendatake terminalis kasutatud ANSI värvikoodid HTML märgenditega, et eeskomponent saaks tüübivigu kuvada.

[11] Local Storage

7. Kui jooksutamine õnnestus, loetakse konteineri väljundist funktsioonikutse tulemus ja võrreldakse seda oodatud väljundiga ning koostatakse testimise tulemus eeskomponendile saatmiseks.
    a. Kui funktsiooni kõikide testjuhtude väljund vastab oodatud tulemusele märgitakse see funktsioon õigesti implementeerituks.
    b. Kui funktsiooni testjuhtudes esines erisusi oodatud väljundiga, märgitakse iga sellise juhu kohta funktsiooni sisend, oodatud väljund ja tegelik väljund.
    c. Testjuhtude peale kogutud tulemused saadetakse eeskomponendile.



Joonis 9. Koodi testimise töövoog.

Testimiseks sai esmalt loodud Dockeri *image* ehk tõmmisfail, kuhu pandi Idrise interpretaator. Lisaks interpretaatorile on tõmmisfailis skript, mis:

1. Loeb sisendvoost teksti.
2. Kirjutab loetud teksti Idrise lähtekoodi failiks.
3. Tüübikontrollib Idrise programmi.
    a. Õnnestunud kontrolli korral jooksutab *main* funktsiooni, mis trükib testide tulemused.
    b. Tüübivigadega programmi korral trükib tuvastatud vead.

4. Kustutatakse eelnevalt loodud fail vältimaks liigset mälukasutust.

Testprogrammi koostamine sõltub testitava funktsiooni tüübist. Praegu on implementeeritud puhaste funktsioonide testimine ning sisend- ja väljundvoogu (edaspidi sisend-väljund) kasutavate funktsioonide lihtsama juhu ehk väljundvoo testimine. Mõlemat tüüpi funktsioonide puhul kombineeritakse kasutaja kood *main* funktsiooniga, kuhu lisatakse kõigepealt sõne „Learn Idris App test result:“ väljatrükk, et hiljem oleks lihtsam regulaaravaldisega kontrollida programmi väljundit. Seejärel lisatakse vastavalt funktsiooni tüübi järgi kasutaja kirjutatud funktsiooni väljakutsed. Puhaste funktsioonide puhul lisatakse rida, mis kutsub funktsiooni välja testjuhus kirjeldatud sisendiga ning seejärel trükib tulemuse. Sisend-väljundit kasutavate funktsioonide puhul lisatakse samuti funktsiooni väljakutse koos testjuhus kirjeldatud sisendiga, kuid tulemuse trükkimist eraldi ei lisata, sest eeldatakse, et selline funktsioon trükib ise oma töö käigus väljundit. Sisend-väljundit kasutavate funktsioonide puhul saab praegu testida ainult väljundiga funktsioone. Töö autor ei jõudnud implementeerida versiooni, kus saaks sellistele funktsioonidele ka testimise käigus sisendit juurde anda.

Kui Dockeris on test ära jooksutatud, loetakse testimise käigus tekkinud konteineri väljundit. Regulaaravaldisega kontrollitakse, kas väljundis on eelnevalt mainitud sõne väljatrükk, mis tähendab, et testprogramm kompileerus ja prooviti testi joooksutada. Kui sõne „Learn Idris App test result:“ väljatrükki ei leitud, siis järelikult tekkisid kompileerimisel vead ning need edastatakse kasutajale. Kui kasutaja kood kompileerus ja jooksutati teste, siis kontrollitakse, kas programmi väljund vastas oodatud väljundile ning vastavalt sellele, kas kuvatakse tagasiside vigase implementatsiooni kohta või märgitakse test läbituks korrektse lahenduse korral.

Kuigi konteiner piirab ressursse ühe testi jooksutamisel, pole see piisav, et piirata paralleelselt jooksvate testide arvu. Seetõttu on rakenduses kasutusel *resource pooling* lahendus: tagakomponendis on teenus, mis haldab piiratud arvu konteinereid. Kui on tarvis testi jooksutada, küsitakse teenuselt vaba konteiner. Vaba konteineri puudumisel blokeerub testi jooksutamine, kuni vabaneb mõni konteiner.

4.3 Materjalide ja testide kirjeldamine

Materjal koos ülesannetega on lähtekoodis eraldi kataloogis, *_lessons*, kirjeldatud *markdown* failides (vt joonis 10). Iga teema kohta on üks fail. Vastava teema materjal teisendatakse *markdownist* HTMLi, et seda saaks veebilehel kuvada. Kusjuures *markdown* faili metaandmetes olevat atribuuti *title* kasutatakse avalehel nuppude sisuna.

```
---
title: 'Test materjal'
---

# Ülesanded

Implementeeri toodud funktsioonid.

## 1. Funktsioon `fst` - paari esimene element

        fst : (a, b) -> a
        fst = ?rhs_fst


## 2. Funktsioon ` sumInt ` - saades sisendiks naturaalarvu x tagastab x-st
väiksemate või võrdsete arvude summa

        sumInt : Int -> Int
        sumInt n = ?rhs_sumInt
```

Joonis 10. Näidismaterjali ja -ülesannete kirjeldus.

Samuti on olemas kataloogid lahendamist vajavate ülesannete põhjadele ja ülesannete testjuhtudele, vastavalt *_code* ja *_tests*. Ülesannete põhi võimaldab näiteks ette anda funktsioonide signatuurid ja osaliselt realiseeritud implementatsioonid. Kui ülesannete põhja pole vastavale teemale antud, kuvatakse tühi tekstiredaktor.

Testjuhud on kirjeldatud *yaml* formaadis (vt joonis 11). Iga oodatud funktsiooni kohta on toodud mõned testjuhud, kus on märgitud iga sisendi kohta vastav oodatud väljund. Teste jooksutatakse Dockeri abil. Testide jooksutamiseks kombineeritakse kasutaja kood koos testjuhtudega, nii et iga testjuhu kohta tekib üks testprogramm, mis edastatakse konteinerile. Dockeri abil kõigepealt kontrollitakse kasutaja koodis tüüpe ja sõltuvalt tüübikontrolli tulemusest jooksutatakse kogu koodi koos testiga. See tähendab, kui kasutaja kood ei läbi tüübikontrolli, siis ei proovita ka testi jooksutada. Kui kasutaja koodis on tüübivead, loetakse väljundist tüübivead ja kuvatakse kasutajale. Kui tüübikontroll läbiti ja jooksutati teste, loetakse samuti väljundist tulemused ja töödeldakse seda viisil, mis võimaldab kasutajale anda alampeatükis „Kasutajaliides“ kirjeldatud tagasisidet.

```
groups:
  - function: sumInt
    type: pure
    test:
      - parameters: 3
        output: 6
      - parameters: 2
        output: 3

  - function: first
    test:
      - parameters: (1, 2)
        output: 1
      - parameters: ("tere", "head aega")
        output: '"tere"'
```

Joonis 11. Näidismaterjali ülesannete testjuhud.

Testidel on funktsiooni grupis parameeter *type* (vt joonis 11), millega saab määrata, mis tüüpi funktsiooniga on tegemist. Puhaste funktsioonide puhul oleks selleks väärtuseks *pure*, kuid kuna puhtad funktsioonid on n-ö tavalised, siis nende puhul võib selle parameetri ka

ära jätta. Teiseks variandiks on määrata parameetri väärtuseks *IO*, mis märgib, et tegemist on sisend-väljundit kasutava funktsiooniga. Parameetri *type* järgi kontrollitakse funktsiooni tüüpi, et puhta ja sisend-väljundit kasutava funktsiooni puhul erinevalt testprogramm kokku panna.

4.4 Arendusprotsess

Veebirakendus loodi lihtsustatult kahes etapis. Esimesena mõeldi välja struktuur: millised kuvad võiksid rakenduses olla, mis komponentidest need koosnevad ning kuidas nende vahel navigeerida. Teiseks suuremaks etapiks oli kasutaja koodi testimise lahendus.

Kui põhistruktuur oli loodud, sai järgmisena paika materjali ja ülesannete kuvamine. See läks lihtsamini, kui alguses tundus. Abiks oli peamiselt Next.js õpetus[12] väiksema projekti näol, mis sisaldas valminud rakendusele sarnast osa, kus tuli *markdown* failist tekitada HTML sisu. Järgmisena lisati tekstiredaktor kasutades teeki CodeFlask[13]. Tekstiredaktori lisamine nõudis rohkem vaeva, sest kasutatud tehnoloogiad olid rohkem võõrad ning vajasid lisauurimist. Seejärel lisati ülesande vaatele jalus koos nuppudega „Kontrolli“, „Eelmine teema“ ja „Järgmine teema“, kuid funktsionaalsus lisati nendest ainult esimesele.

Kõige keerulisemaks osaks kogu arenduse juures võib pidada testimislahenduse loomist. Kuna töö autoril puudus varasem praktiline kogemus Dockeriga, siis tuli selle kohta juurde uurida. Samuti põhjustas raskusi esialgne Dockeri integreerimise lahenduse valik. Alguses oli soov kasutada Dockeriga suhtlemiseks mõeldud teeki Dockerode[14], kuid selle kaudu oli Dockeri sisendvoogu kirjutamine liiga vaevaline. Seetõttu sai asemele võetud Node.js standardteegi moodul Child process[15], et Dockeri käsureaklienti käivitada ja konteinerile sisend saata.

Viimaks sai lisatud funktsionaalsus ülejäänud jaluse nuppudele. Kusjuures, kui nupule „Eelmine teema“ vajutades pole enam ühtegi n-ö eelmist teemat valida, suunatakse kasutaja tagasi avalehele. Analoogiliselt toimib nupp „Järgmine teema“. Samuti tekitati koodi testimise ajaks kasutajale indikaator laadimisratta näol. Laadimisratas täitis ka funktsiooni takistada kasutajal testimise ajal muid nuppe kasutamast, et vältida korduvate päringute saatmist. Lõpuks küljendati ka avaleht, mis esialgu tähelepanuta jäi.

4.5 Võimalikud edasiarendused

Võimalikke edasiarenduse variante on mitmeid. Kõige olulisem neist on kasutaja koodi testimise optimeerimine. Praegu on testide jooksmine võrdlemisi aeglane. Näidismaterjalis on kaks funktsiooni ning mõlemale funktsioonike on kirjeldatud kaks ehk kokku neli testjuhtu. Õige või loogikaveaga lahenduse testimine nelja testjuhuga võtab aega ligikaudu 10 sekundit. Tüübiveaga lahendus võtab aega ligikaudu neli sekundit ehk natuke rohkem kui poole vähem, sest testide jooksutamise aeg jääb olemata. Ühe testjuhu tüübikontroll võtab ligikaudu 0,8 sekundit ning seejärel testi jooksutamine ligikaudu 1,2 sekundit. Seega kulub korrektse funktsiooni puhul ühe testjuhu peale ligikaudu 2 sekundit ehk nelja juhu peale kokku ligi 8 sekundit. Arendusprotsessi käigus prooviti testimise osa kiiremaks saada käivitades Dockeri konteinerid esimese testjuhu ajaks ning ülejäänud juhtude jaoks taaskasutati alguses loodud konteinereid. Nii sai testimise aja umbes 3 sekundit lühemaks.

[12] https://nextjs.org/learn/basics/create-nextjs-app/setup
[13] https://github.com/kazzkiq/CodeFlask
[14] https://github.com/apocas/dockerode
[15] https://nodejs.org/api/child_process.html

Testimise optimeerimiseks võiks uurida näiteks Idrise *Language Server*i kasutuselevõttu. *Language Server* on server, mis pakub IDEle keele funktsioone nagu tekstikursori all oleva muutuja definitsioonini liikumine või tüübi kuvamine[16]. *Language Server*i ja IDE vaheliseks suhtluseks kasutatakse standardiseeritud protokolli *Language Server Protocol* (LSP). Kui keelele on programmeeritud *Language Server*, saavad kõik IDEd, mis kasutavad LSP-d vastava keele toe. Idrise *Language Server*it kasutatakse näiteks Visual Studio Code pluginas tüübikontrolliks. Selle asemel, et Idrise kompilaatorit käivitada, saaks tüübivigu küsida *Language Server*i käest, mis on optimeeritud madalale latentsusele.

Praegu on testimise lahendus piiratud võimalustega. Saab testida puhtaid funktsioone ja lihtsamaid väljundvoogu kasutavaid funktsioone. Seda saaks paremini teha. Näiteks luua struktureeritud konteineri sisendformaat, kus saaks lisaks testjuhtude parameetritele kirjeldada sisendvoo. Testjuhu *yaml* struktuuri tuleks samuti lisada parameeter sisendvoo kirjeldamiseks. Täiendus võimaldaks testida funktsioone, mis kasutavad sisendvoogu.

Materjalide ja ülesannete lisamist rakendusse saaks muuta lihtsamaks. Selle asemel, et neid lisada lähtekoodi, võiks näiteks luua selleks eraldi n-ö administreerimiskuva. Paraku veebirakenduse praegusesse versiooni ei jõutud seda lisada.

Veebirakendust saab edasi arendada ka teiste sarnaste valikkursuste jaoks. Selleks tuleks nendel kursustel kasutatavate keelte jaoks luua uus Docker'i tõmmisfail interpretaatori või kompilaatoriga ning lisada keelespetsiifiline kasutaja koodist testprogrammide koostamine.

[16] https://microsoft.github.io/language-server-protocol/

5. Kasutatud tehnoloogiad

Viiendas peatükis antakse ülevaade valminud veebirakenduses kasutatud tehnoloogiatest.

5.1 React

React on Facebookis (praeguses Metas) loodud avatud lähtekoodiga JavaScripti teek arendamaks komponendipõhiseid kasutajaliideseid[17]. React võimaldab luua keerulisi kasutajaliideseid kombineerides väiksemaid isoleeritud komponente. Näiteks võib rakenduse avaleht koosneda komponentidest nagu päis, jalus, sisu. Need omakorda võivad sisaldada ühiseid komponente nuppude jms jaoks.

Reactis kasutatakse deklaratiivset programmeerimist. Reacti programmeerija vastutab komponendi oleku tekitamise ja muutmise eest. Oleku kuvamiseks kirjutab programmeerija funktsiooni, mis teisendab oleku veebilehe esituseks. See vastandub varasematele imperatiivsetele raamistikele, mille puhul pidi arendaja ise olekut uuendades muutma ka vastavat olemasolevat veebilehel esitust. Teisisõnu, Reactis on äriloogika eraldatud kasutajaliidese kuvamisest.

5.2 Next.js

Next.js on paindlik Reacti raamistik, mis teeb Reactiga veebirakenduste loomise kergemaks[18]. Näiteks on Reactis kohmakas alamlehtede vahetamine. Tüüpiline veebileht koosneb mitmest alamlehest, mida saab luua Reacti komponentidena. Next.js tegeleb alamlehtede vahel navigeerimisel komponentide vahetamisega, mida Reactis tuleks ise programmeerida.

Raamistik võimaldab lisaks eeskomponendile programmeerida ka tagakomponent. Seejuures lihtsustab näiteks päringute marsruutimist. Kui tavaliselt tuleb tagakomponenti programmeerides defineerida, milline funktsioon vastab millisele otspunktile, siis Next.js võimaldab otspunkte defineerida konkreetse failipuu struktuuriga, kus kõik alamkausta „api" alla loodud failid kujutavad erinevaid rakenduse URI teid, näiteks fail nimega „run.js" vastab teele /api/run. Tee on osa veebiaadressist, mis määrab otsitava ressursi[19].

5.3 Docker

Docker on platvorm, mis võimaldab pakendada oma rakendusi isoleeritud keskkondadesse ehk konteineritesse[20]. Näiteks, konteineris jooksvat programmi saab piirata, nii et see ei pääse ligi konteineri käivitanud operatsioonisüsteemi failisüsteemile või võrgule. Kuna konteinerisse on pakendatud justkui eraldiseisev operatsioonisüsteem, ei sõltu konteineris käivitatud programm kasutaja operatsioonisüsteemist. See tähendab, et arendades saab olla kindel, et tarnitud rakendus töötab samamoodi nagu ta töötas arenduskeskkonnas.

5.4 Express.js

Express.js on Node'i raamistik tagakomponentidele, mis võimaldab arendada serverit päringute haldamiseks[21]. Loodud rakenduses võeti see kasutusele asendamaks Next.js poolset serveri lahendust, et luua konteinerite haldamise süsteem koos testide

[17] React.js
[18] What is Next.js
[19] e-Teatmik: IT ja sidetehnika seletav sõnaraamat
[20] Docker overview
[21] Express/Node introduction

jooksutamisega. Kuna Next.js tagakomponent on olekuta ehk käivitatakse iga päringu korral uuesti, siis ei saanud sellega kasutada *resource pooling*ut.

5.5 Tailwind CSS

Tailwind CSS on Adam Wathani loodud raamistik veebirakenduste küljendamiseks[22]. Tailwind koosneb paljudest kasulikest eeldefineeritud CSS klassidest. CSS reeglite kirjutamise asemel lisab arendaja rakenduse küljendamiseks HTML märgenditele Tailwindi klasse. Näiteks valge teksti saamiseks tuleb lisada klass *text-white*.

Veel võimaldab raamistik arendajal korduvad Tailwindi eeldefineeritud stiilid ühendada enda defineeritud CSS klassi, mida saab samamoodi HTML märgendites kasutada. Näiteks, ümarate nurkadega, sinise tausta ja valge tekstiga nuppude küljenduseks kasutatud Tailwindi klassid (*text-white, bg-blue, rounded*) saab korduste vältimiseks viia ühise nimetaja *button* alla.

Kõikide Tailwindi klasside stiil kokku oleks suure mahuga. Seetõttu kaasneb Tailwindiga programm, mis otsib koodist päriselt kasutatud klasse ning koostab nende klasside stiilidest kompaktse CSS faili.

[22] Tailwind CSS

6. Kokkuvõte

Saavutati bakalauresuetöö eesmärk luua keskkond, mis võimaldaks kasutajal tutvuda funktsionaalse programmeerimisega ja keelega Idris. Loodud keskkond võimaldab alustada õppimisega ilma kohalikku arenduskeskkonda üles seadmata. Rakendus hoiab kasutaja brauseris meeles läbitud teemad koos lahendustega.

Valminud lahenduse õppimisvaates näidatakse *markdown* failist HTMLi teisendatud ülesannete selgitusi, mille kõrval on tekstiredaktor, kuhu kasutaja saab oma lahenduse kirjutada. Vajutades nupul „Kontrolli“ automaattestitakse kasutaja lahendust, kusjuures kasutajale antakse tagasisidet vigase implementatsiooni korral ning näidatakse tüübivigu.

Töö esimeses osas tutvustati kursust „Funktsionaalprogrammeerimine“ koos valminud lahenduse võimalike kasutuskohtadega ja anti ülevaade funktsionaalse programmeerimise paradigmast ning keelest Idris. Teises osas analüüsiti rakenduse loomise võimalusi ning põhjendati valitud lahendust. Valminud rakendust koos soovitustega edasiarendusteks tutvustatakse eelviimases osas ning seejärel viimases osas on kirjeldatud rakenduse loomisel kasutatud tehnoloogiad.

Võimalusi edasiarendusteks on mitmeid. Olulisim edasiarendus on kasutaja koodi testimise lahenduse optimeerimine. Samuti saab edasi arendada sisend-väljundit kasutavate funktsioonide testimist. Veel on võimalik luua juurde funktsionaalsust nagu koodi allalaadimine või lahenduse esitamiseks integratsioon Moodle’iga.

Viidatud kirjandus

Apinis, K., Holter, K., Vene, V. (2021). Tartu Ülikooli kursus Funktsionaalprogrammeerimine, sügis 2021. https://courses.cs.ut.ee/2021/fp/fall/Main/Syllabus (01.04.2022)

Apinis, K., Saan S. (2020). Tartu Ülikooli kursus Programmeerimiskeeled, sügis 2020. https://courses.cs.ut.ee/2020/PK/fall/Main/Syllabus (01.04.2022)

Brady, Edwin (2017). Type-driven Development with Idris. *Manning Publications*.

Cybernetica AS. Andmekaitse ja infoturbe leksikon. https://akit.cyber.ee/

Docker overview. https://docs.docker.com/get-started/overview/ (18.04.2022)

e-Teatmik: IT ja sidetehnika seletav sõnaraamat. http://ww.vallaste.ee

Express/Node introduction. https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs/Introduction (01.05.2022)

Holter, K. (2021). Funktsionaalprogrammeerimise õpetamine Idrises. *TÜ arvutiteaduse instituudi bakalaureusetöö*.

Language Server Protocol. https://microsoft.github.io/language-server-protocol/ (03.05.2022)

Nestra, Härmel (2010). Sissejuhatus funktsionaalsesse programmeerimisse. Tartu Ülikooli kirjastus.

React.js kodulehekülg. https://reactjs.org/ (18.04.2022)

Tailwind CSS kodulehekülg. https://tailwindcss.com/ (18.04.2022)

What is Next.js?. https://nextjs.org/learn/foundations/about-nextjs/what-is-nextjs (18.04.2022)

Window.localStorage. https://developer.mozilla.org/en-US/docs/Web/API/Window/localStorage (18.04.2022)

Lisad

I. Paigaldamise juhend

1. Laadige alla ja installige Node.js (versioon 16.14.0) ning Docker (alates versioonist 20.10.13).
2. Laadige alla rakenduse lähtekood:
    a. `git clone https://github.com/maarikamarkus/learn-idris-app.git`
3. Avage konsooliaken kaustas /docker ja sisestage järgmised käsud:
    a. Käivita Docker, näiteks Ubuntus: `sudo systemctl start docker`
    b. `docker build . -t idris` – Ehitab tõmmisfaili.
4. Liikuge konsoolis tagasi projekti juurkausta ja sisestage järgmised käsud:
    a. `npm install` – Laeb alla kõik vajalikud paketid.
    b. `npm run build` – Ehitab serveritarkvara.
    c. `npm start` – Käivitab serveritarkvara.
5. Brauseris navigeerige aadressile http://localhost:3000 ja avage mõni teema.

II. Rakenduse lähtekood

Veebirakenduse lähtekood on saadaval GitHubi repositooriumis https://github.com/maari-kamarkus/learn-idris-app.

III. Litsents

Lihtlitsents lõputöö reprodutseerimiseks ja üldsusele kättesaadavaks tegemiseks

Mina, **Maarika Markus**,

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

   Veebirakendus funktsionaalprogrammeerimise ja -keele Idris õppimiseks,

   mille juhendaja on Kalmer Apinis,

   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.

Maarika Markus

09.05.2022