TALLINNA TEHNIKAÜLIKOOL

Infotehnoloogia teaduskond

Sten Kalev 185800IADB

# IntelliJ platvormil põhinev pistikprogramm programmeerimisülesannete lahendamiseks

Bakalaureusetöö

Juhendaja: Bahdan Yanovich
BSc

Tallinn 2024

# Autorideklaratsioon

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

Autor: Sten Kalev

04.01.2024

# Annotatsioon

Käesoleva töö eesmärgiks on lahendada probleem Charoni programmeerimisülesannete lahenduste esitamisega, mis on hetkel liiga pikk ja seetõttu ka ebamugav ning esma-kasutajatele liiga raske.

Töös uuritakse, mis integreeritud tarkvaraarenduskeskkonda või tekstiredaktorit tudengid Charoni ülesannete lahendamise jaoks kasutavad ning kuidas on sarnane probleem lahendatud mujal. Planeeritakse ette pistikprogrammi kujundus ning arhitektuur. Olenevalt uurimistulemustest planeeritakse pistikprogrammi arendus, mis ka töö käigus teostatakse. Pistikprogrammi arenduse käigus luuakse enim kasutatud integreeritud tarkvaraarenduskeskkonna või tekstiredaktori kasutajaliidesesse mitu akent ja teostatakse integreerimise eesmärgil muudatusi Charoni taga-rakenduses.

Lõpuks ühendatakse käesoleva töö käigus valmiv pistikprogramm Charoniga ning teostatakse tulemuste analüüs.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 48 leheküljel, 6 peatükki, 26 joonist, 0 tabelit.

# Abstract

IntelliJ Platform Based Plugin for Solving Programming Exercises

The aim of this thesis is to solve the problem with the process of submitting Charon programming exercises. The process is currently too long and therefore inconvenient, and too difficult for newcomers.

The thesis studies which integrated development environment or text editor is used the most for solving Charon exercises and how similar problems are solved elsewhere. Depending on the results of the study, the development of a plugin will be planned out and eventually accomplished. During the development of the plugin, several windows will be created for the user interface of the integrated development environment or text editor used the most for solving Charon programming exercises. Also, several changes will be made to the back-end application of the Charon plugin for integration purposes.

Finally, the plugin developed during the thesis will be integrated with the Charon plugin and result analysis will be performed.

This thesis is written in Estonian and is 48 pages long, including 6 chapters, 26 figures and 0 tables.

# Lühendite ja mõistete sõnastik

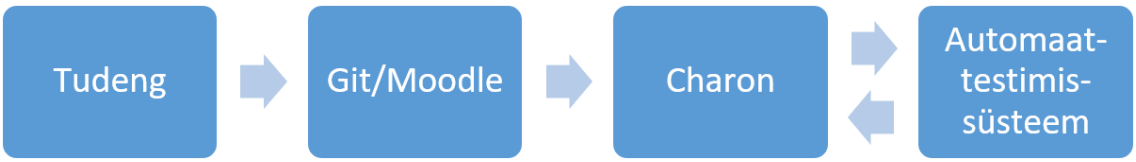
| | |
|---|---|
| API | Rakendusliides |
| *Endpoint* | Digitaalne asukoht, kus rakendusliides võtab vastu päringuid selle serveris olevate spetsiifiliste ressursside kohta [1] |
| Git | Versioonihaldustarkvara |
| HTML | *HyperText Markup Language*, märgistuskeel |
| HTTP | *HyperText Transfer Protocol*, veebiprotokoll |
| IDE | Integreeritud programmiarenduskeskkond |
| Java | Programmeerimiskeel |
| Kotlin | Programmeerimiskeel |
| MOOC | Veebipõhine massikursus |
| Moodle | Veebipõhine õpihaldussüsteem |
| SDK | Tarkvaraarenduskomplekt |
| ZIP | Kokku pakitud failivorming |

# Sisukord

# Jooniste loetelu

# 1 Sissejuhatus

Charon on Moodle'i pistikprogramm, mis võimaldab luua automaattestidega kontrollitavaid programmeerimisülesandeid [2]. Charoni ülesande lahendamiseks laeb õpilane alla malli(d) nende olemasolul kas Giti ülesande hoidlast või kopeerib Charoni ülesande lehelt. Seejärel teostab tarkvaraarenduskeskkonnas (edaspidi IDE) või tekstiredaktoris (*text editor*) lahendused, mille saadab Giti abil automaattestimissüsteemi ja tagasiside tulemuste kohta saadetakse e-postile ning on vaadeldav ka Charoni ülesande lehel (Joonis 1). Lisaks saab ülesande lahendust esitada veel läbi Charoni ülesande lehe, mis lahendab käsitsi Giti abil saatmise vajaduse, kuid tekitab juurde vajaduse lahendust edasi-tagasi kopeerida ja kleepida (Joonis 1). Tulemusena on protsess ebamugav ning keeruline uutele kasutajatele, kes ei oska Giti kasutada.




Joonis 1. Charoni ülesande testimise voog.

## 1.1 Aktuaalsus

Charoni pistikprogrammi kasutatakse Tallinna Tehnikaülikoolis kooliõpilastele suunatud MOOCil ning Tarkvarateaduse Instituudis õpetavates ainetes. Nendeks on ITI0214 Programmeerimise erikursus, ITI0204 Algoritmid ja andmestruktuurid, ITI0202 Programmeerimise põhikursus, ITI0102 Programmeerimise algkursus, ITI0322 Loogiline programmeerimine ning ITT8060 Programmeerimise erikursus. Ainuüksi ITI0102 Programmeerimise algkursust deklareerib igal aastal üle 500 tudengi. Neid aineid õpetatakse järgmistel erialadel: IAIB Informaatika, IADB IT süsteemide arendus, IAAB IT süsteemide administreerimine, IACM Riistvara arendus ja programmeerimine, IAPM Informaatika, IVSM Tarkvaratehnika. Charoni pistikprogrammis on õpilastele tagatud eestikeelne kasutajaliides.

## 1.2 Eesmärk ja skoop

Käesoleva töö eesmärk on arendada ja avaldada algversioon pistikprogrammist IntelliJ platvormile lihtsustamaks õpilastele Charoni ülesannete lahenduste esitamist. Käesoleva töö pistikprogrammi kasutamist tudengite poolt nõuab autentimist, mis vajaks palju suhtlemist ja kokkuleppeid Moodle'it haldavad meeskonnaga, milleni ei pruugi lihtne jõuda. Seega otsustati skoopi piirata ainult kooliõpilastele. Sellisel juhul autentimine ei ole nii kriitiline, kuna MOOCi eesmärk on pigem ülesandeid lahendada ja punktiseisu salvestamine ei ole nii oluline. Lisaks annab see võimaluse pistikprogrammi varasemaks väljalaskmiseks ning esmase tagasiside saamiseks.

## 1.3 Metoodika

Lõputöös kasutatakse tarkvaraarendusmetoodikat:

- veebipõhise küsitluse koostamine („*The Mom Test*“ ja „*Problem Validation Script*“ abil) probleemi kinnitamiseks [3], [4];
- küsitluse tulemuste analüüsimine probleemi olemasolu kinnitamiseks ja enim kasutatava programmeerimiskeskkonna leidmiseks;
- sarnast probleemi lahendavate infotehnoloogia lahenduste uurimine (eesmärgiga õppida nende arendusloo pealt);
- pistikprogrammi arendamise võimalikkuse uurimine (olenevalt küsitluse analüüsist);
- funktsionaalsete ja mittefunktsionaalsete nõuete koostamine;
- põhjalik pistikprogrammi kasutajaliidese ja arhitektuuri planeerimine;
- pistikprogrammi arendamine, Charoniga ühendamine;
- pistikprogrammi testimine;
- pistikprogrammi esmaversiooni avaldamine (tagasiside saamise eesmärgil).

# 2 Analüüs

Käesolevas peatükis analüüsitakse Charoni kasutajatele suunatud küsitluste tulemusi ning uuritakse, kuidas on juba lahendatud sarnast probleemi.

## 2.1 Kasutajate uuring

Selles alapeatükis tuvastatakse Charoni kasutajamugavuse puudused, mida saaks pistikprogrammis likvideerida. Kuna IDE-sid ja tekstiredaktoreid on kasutusel mitmeid, siis oli tarvis veel tuvastada tudengite seas populaarseim IDE või tekstiredaktor, millele lisavõimalusi looma hakata.

Informatsiooni hankimine käis kahe võrguküsimustiku abil. Kahele küsimustikule vastas kokku 109 tudengit, ühele 102 ja teisele 7. Märkimisväärselt suur vahe vastajate numbris tulenes sellest, et üks küsitlus oli lühike ja teine oli pikem ja põhjalikum, mille tõttu võib arvata, et tudengid ei olnud väga motiveeritud küsitlusele vastama. Pikema ja põhjalikuma küsitluse vastustest oli üks sisuliselt tühi, mille tõttu analüüsis töö autor ainult ülejäänud kuut vastust. Küsimustikud olid nii eesti kui ka inglise keelsed.

### 2.1.1 Populaarseima arenduskeskkonna küsitluse analüüs

Küsitluse eesmärk oli tuvastada enim kasutatud IDE või koodiredaktor tudengite seas. Küsitleti infotehnoloogia tudengeid, sest tudengitel on rohkem Charoni kasutamise kogemust ehk nende tagasiside on relevantsem ning pärast töö eesmärgi saavutamist on soov lisada võimalus pistikprogrammi kasutama hakata ka tudengite poolt ehk integreerida Moodle’i autentimisega. Täiendavalt lootis töö autor, et pistikprogrammi kasutavad kooliõpilased hakkavad ka tulevikus tudengitena Charoni ülesandeid lahendama, mille tõttu on tudengite kogemus siin sobilik. Lisaks eelnevale on tudengiteni lihtsam jõuda, sest kooliõpilaste küsitlemine võib nõuda nende vanemate nõusolekut, mis teeks küsitlemise keeruliseks.

Küsimustikku jagati Charonit kasutavate kursuste tudengitele. Populaarseimaks IDE-ks või koodiredaktoriks osutus IntelliJ platvorm (JetBrains Suite), mida kasutas programmeerimisülesannete lahendamiseks 87 tudengit 102-st ehk ligikaudu 85 protsenti vastanutest. Järgmiseks enim kasutatud valikuks osutus Visual Studio Code, mida kasutas

13 tudengit ehk ligikaudu 13 protsenti vastanutest. Vähem populaarsemad valikud olid Eclipse, Thonny, Sublime Text, Neovim ja Geany.

### 2.1.2 Charoni kasutajamugavuse küsitluse analüüs

Järgmise küsitluse peamine eesmärk oli tutvuda täpsemalt lahenduse esitamise protsessiga Charoni ülesannete lahendajate seas. Lisaks sellele kasutati küsitlust ära veel teiste kasutajaid häirivate Charoni kasutajamugavuste puudustega tutvumiseks, mis tulevad kasuks edasiarenduses. Idee lahenduse esitamise ebamugavusest tuli Charoni arendustiimi juhtkonnast, kuid autor tahtis veel kindlaks teha kõikvõimalikud muud kasutajamugavuse puudulikkused.

Küsitluse vastustest tuleb välja, et malli kopeeritakse ning kleebitakse Charoni ülesande leheküljelt IDE-sse ning 100% vastanutest esitab lahendusi IDE-s Giti kaudu. See kinnitab käesoleva töö probleemi, et ülesande lahenduse esitamise protsess on pikk ja seega ka ebamugav.

Kasutajamugavust hindasid kõik vastajad viie palli süsteemis kolm või enam. Peamisteks ebamugavusteks olid liigne testitulemuste ooteaeg ning ebaselge tagasiside esituste kohta. Lisaks eelnevalt mainitutele nimetati ka ebamugavuseks veakoodi 429, mis tähendab kasutajale ajutist keeldu päringute tegemisest, sest teatud aja jooksul on maksimaalne päringute arv ületatud. Veakood 429 olevat täpsemalt põhjustatud ülesande esituste liigsest küsimisest testitulemusi teada soovides. Vastajatelt Charoni kasutajamugavuse tõstmise ideesid küsides olid varasemalt mainitud kolme ebamugavuse kaotamine välja toodud, mis said ka kirja pandud Charoni arendustiimi tegemata tööde nimekirja ning mõned said ka lahendatud. Lisaks eelnevalt mainitutele oli veel soov teha testid kohalikult kättesaadavaks.

## 2.2 Olemasolevad lahendused

Enne uue pistikprogrammi arendamisega peale hakkamist uuris autor mõningaid olemasolevaid lahendusi, et vältida võimalikku „ratta uuesti leiutamist“ ning leida võimalikke mugavusi ning ebamugavusi, mida pistikprogrammile lisada või lisamata jätta.

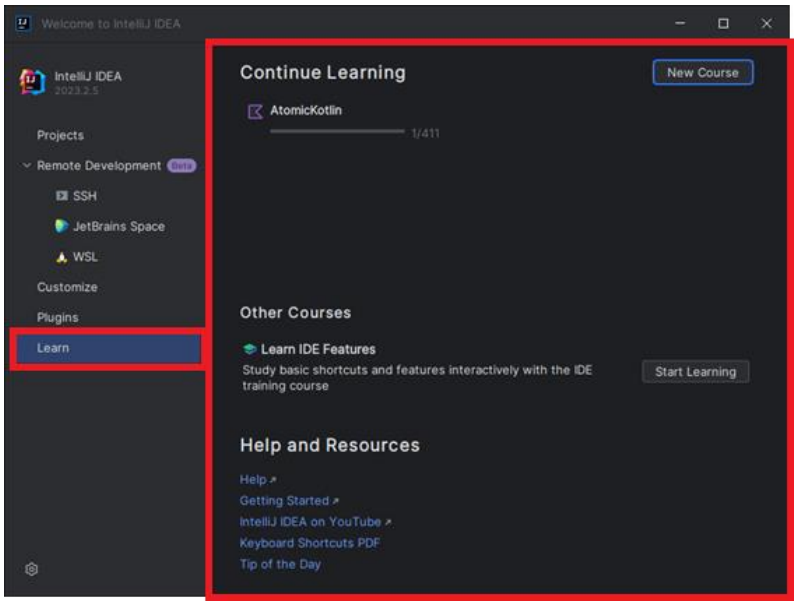
### 2.2.1 Koodi esitamine Charoni ülesande veebilehel

Esimesena tuleks ära mainida lahendus, mida käesolev töö üritab lihtsustada ja mugavdada. Moodle'is Charoni ülesande lehel on olemas lahenduse esitamise võimalusega koodiredaktor, mis on algväärtustatud malliga (selle olemasolu korral). Nimetatud koodiredaktor tagab mõningad mugavused, nagu näiteks programmikoodi esiletõstmine (*code highlighting*) ning sõna lõpuleviimine (*word completion*). Paraku ei ole nimetatud mugavused piisavad, et peatada tudengeid kopeerimast malli IDE-sse ning alles seal ülesande lahendamisega alustada, nagu kasutajamugavuse küsitluse analüüsi käigus kinnitati.

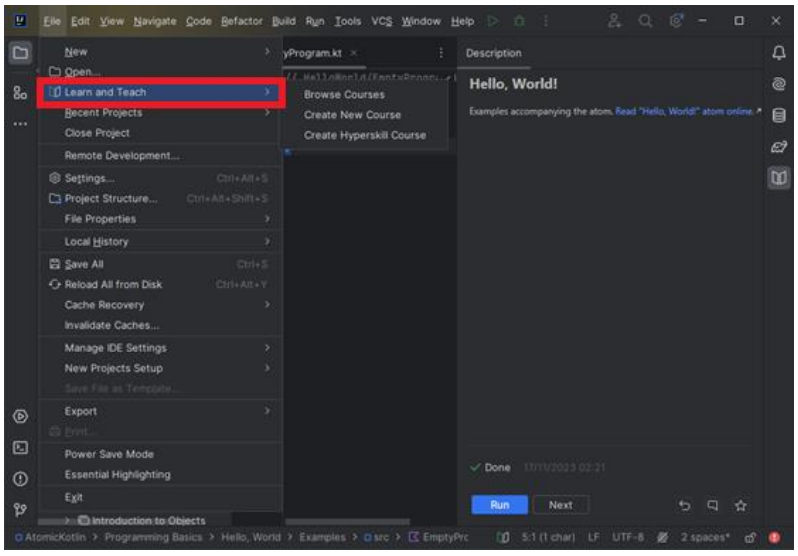
### 2.2.2 JetBrains Academy

Järgmisena võttis töö autor analüüsimisele JetBrainsi poolt välja töötatud JetBrains Academy, mis on pistikprogramm IntelliJ platvormi IDE-dele. Pistikprogrammi abil oli mugav nii programmeerimise kursust üles seada kui ka programmeerimise kursust alustada ning seejärel programmeerimisülesandeid lahendada [5]. Kursuseid oli võimalik võtta lisaks JetBrainsi enda turult (*marketplace*) erinevatest e-õppesüsteemidest, nagu näiteks Coursera ja CheckiO, kuid puudus integreerimise võimalus nii Moodle'i keskkonna kui ka Charoni automaattestimissüsteemiga, mille tõttu analüüsis autor rohkem õpilase kogemust õpetaja kogemuse asemel. Mõned e-õppesüsteemid nõudsid kursuste kättesaamiseks võrguseansi alustamist.

Programmeerimiskursust sai alustada IDE tervituse vaatest (*welcome screen*) kui ka läbi koodiredaktorivaate (*code editor view*) tööriba (Joonis 2) (Joonis 3). IDE tervituse vaates oli lisaks võimalusele alustada uut kursust veel avada juba varasemalt alustatud kursuseid (Joonis 2).

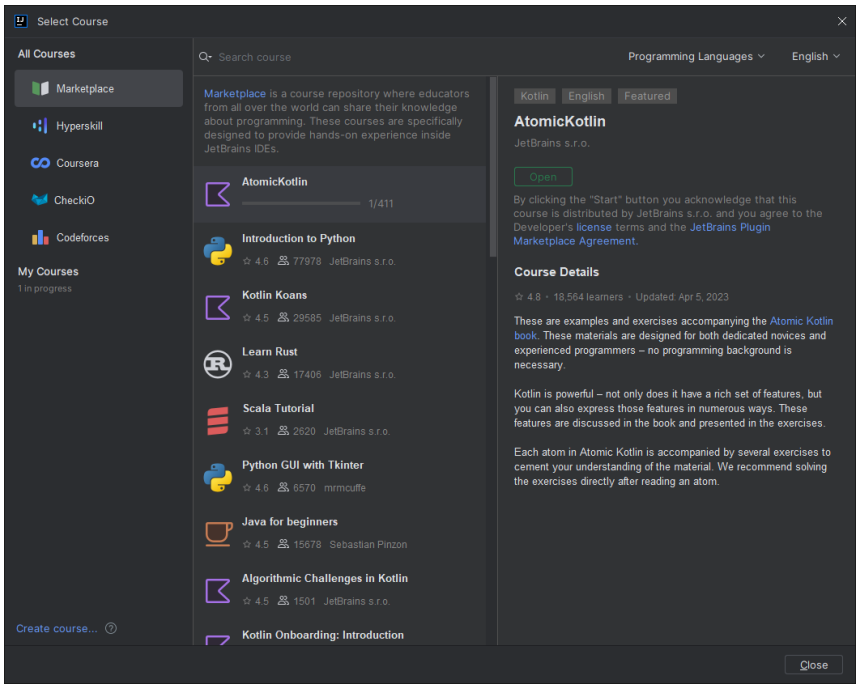



Joonis 2. IntelliJ platvormi IDE. Vasakpoolsest menüüst on avatud JetBrains Academy poolt lisatud aken. Alustatud on ühte kursust.




Joonis 3. IntelliJ platvormi IDE. Aktiivne on koodiredaktorivaade ning avatud on tööriba, millel on esile tõstetud uue ülesande alustamise (või loomise) valik.

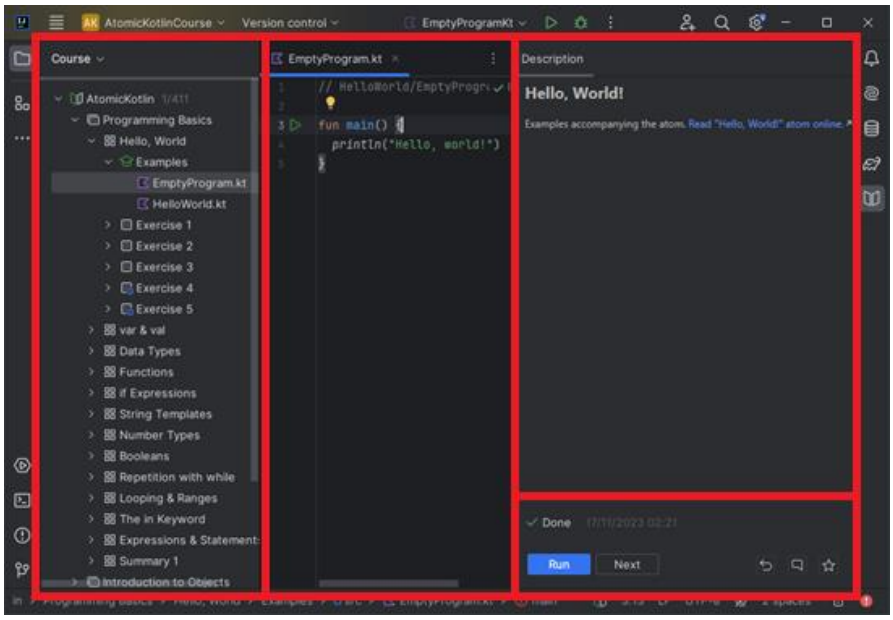
Üritades alustada uut kursust kas läbi tervituse vaate või läbi IDE tööriba avanes kursuse valimise aken, mis oli vertikaalselt jagatud omakorda kolmeks erinevaks aknaks (Joonis 2) (Joonis 3) (Joonis 4). Vasakpoolne aken oli e-õppesüsteemi valimiseks, mille tulemusena ilmusid keskmisse aknasse valitud süsteemi kursused (Joonis 4). Kursuste peale vajutades ilmus parempoolsesse aknasse kursuse kirjeldus (Joonis 4). Kursuseid oli võimalik ka filtreerida (Joonis 4).




Joonis 4. JetBrains Academy kursuse valimise aken.

Kursuse valimise aknast kui ka IDE tervituse vaatesse pistikprogrammi poolt lisatud aknast oli veel võimalik lokaalselt olemasolevaid kursuseid nii avada kui ka kustutada (Joonis 3) (Joonis 4). Lokaalselt olemasolevad kursused olid avatavad ka tervituse vaate projektide aknas.

Õpilaspoolne vaade ülesannet lahendades koosnes kolmest osast: kursuse aken, koodiredaktor ning kirjelduse aken (Joonis 5). Kursuse aken käitus sarnaselt näiteks IntelliJ IDEA projektiaknaga – puu kursuse sees olevatest kaustadest ning failidest (Joonis 5). Kirjelduse aknas oli võimalik kuvada erinevaid tekstifaile (nagu näiteks ülesande kirjeldust ja automaattestide tulemusi) ning selle juures oli ka nupp järgmise ülesande lahendamisega alustamise jaoks (Joonis 5). Kirjelduse all oli erinevaid nuppe testide jooksutamiseks, malli esialgseks taastamiseks ning pistikprogrammile kommentaari jätmiseks (Joonis 5). Testide tulemus kuvati üldiselt testide jooksutamise nupu kõrvale, kuid selle jaoks oli vahepeal ka eraldi vaade IDE alumisel pool sarnaselt näiteks IDE käsureale.



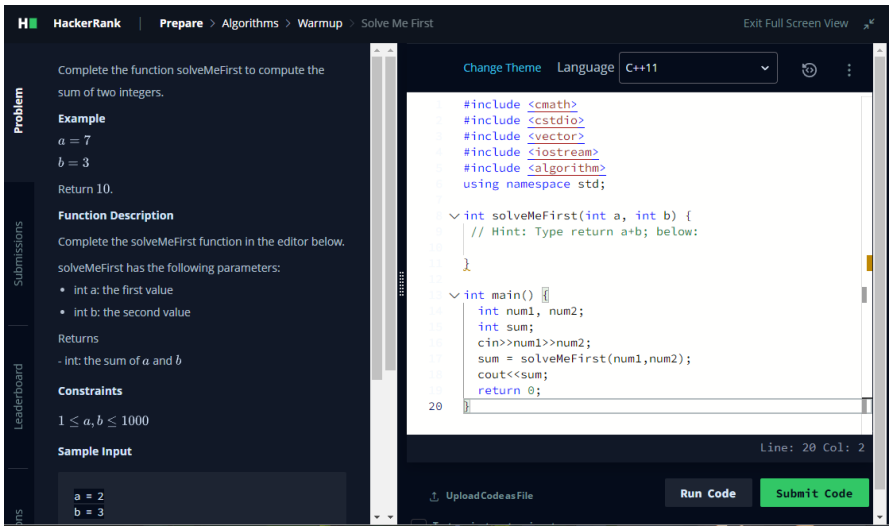

Joonis 5. IntelliJ platvormi IDE. Aktiivne on koodiredaktorivaade. Vasakul on kuvatud kursuse aken, keskel koodiredaktor ning paremal ülesande kirjelduse aken.

### 2.2.3 HackerRank ja sarnased platvormid

Järgmisena uuris töö autor, kuidas lahendab sarnast probleemi HackerRank, mis on tarkvaraarendajate oskusi hindav värbamisplatvorm [6]. Veebilehel on võimalik programmeerimisvõistlustel osaleda, programmeerimisvõistluste jaoks treenida või siis niisama oma oskusi tõestada. Töö autor uuris, kuidas käib veebilehel programmeerimisvõistluste jaoks treenimine.

Treenida saab veebilehel kas teemade kaupa, nagu näiteks algoritmika, tehisintellekt või Java, kui ka „ettevalmistuskomplekti“ (*preparation kit*) abil, mis on rohkem suunatud oskuste lihvimiseks ning tööintervjuude jaoks ettevalmistamiseks. Töö autor uuris, kuidas näeb näiteks ette teema kaupa treenimine.

Teemat valides kuvas veebileht valitud teemaga seotud ülesandeid, mida sai neile vajutades kohe lahendama hakata. Lahendama asudes kuvati aken probleemi kirjeldusega, mis kattis vasaku osa veebilehitseja aknast (Joonis 6). Parempoolne osa sisaldas sisseehitatud koodiredaktorit, mis juba sisaldas ülesande malli, mida sai omakorda taastada (Joonis 6).




Joonis 6. HackerRank veebilehel ülesande lahendamise vaade.

Koodiredaktor oli konfigureeritav mitut moodi: muudetavad olid koodiredaktori kujundusteema, programmeerimiskeel, redaktori töörežiim (kas vaikimisi tavaline, Emacs või Vim, mis on mõlemad tekstiredaktorid), tabeldusmärgi suurus tühikutes ja veel programmikoodi automaatne lõpetamine. Ülesande lahendust sai niisama jooksutada nii vaikimisi sisendiga kui ka enda kohandatud sisendiga ja lõpuks esitada. Lahendust oli veel võimalik failina üles laadida.

Probleemi kirjelduse aknas oli veel võimalik kuvada enda eelnevalt esitatud lahendused, ülesande edetabel, diskussioon ülesande kohta või siis lahenduse spikker erinevates programmeerimiskeeltes.

Lisaks uuris töö autor veel teisi HackerRank-ile sarnaseid lahendusi, nagu Codeforces ja Online Judge. Kuna nimetatud lahendused olid üksteisega väga sarnased, siis jättis töö autor nende jaoks eraldi alapeatükid tegemata.

### 2.2.4 Olemasolevate lahenduste kokkuvõte

Kõik antud peatükis uuritud lahendused lubasid programmeerimisülesannet valida ja lahendada, kuvasid ülesande kirjeldust, võimaldasid esitada ehk testida ülesande lahendust ning salvestasid esituste tulemusi. Suurim vahe HackerRankil (ja sarnastel) teiste uuritud lahendustega oli see, et esimesed olid programmeerimisvõistluste jaoks ettevalmistamiseks suunatud platvormid ning teiste eesmärk oli pigem programmeerimise õppimine. See tähendab seda, et programmeerimise õppimise eesmärgil antakse

lahendajale rohkem tagasisidet kui programmeerimisvõistluseks ette valmistuvale, kes harilikult saab tagasisideks teada vaid seda, kas ta lahendus oli õige või vale. Lisaks ei pakkunud Codeforces ja Online Judge ülesande lahendamiseks mingeid malle, mille tõttu ei saanud ka malle taastada ning Codeforces ei võimaldanud samal ajal kuvada nii kirjeldust kui ka koodiredaktorit, mille tõttu võib järeldada, et veebileht eeldab, et ülesande lahendus koostatakse kuskil veebilehest eemal.

## 2.3 Funktsionaalsed ja mittefunktsionaalsed nõuded

Analüüsi tulemusena on koostatud funktsionaalsete ja mittefunktsionaalsete nõuete nimekiri. Funktsionaalsed nõuded on esitatud kasutajalugudena:

1. Õpilasena tahan programmeerimiskursust valida.
2. Õpilasena tahan programmeerimisülesannet valida.
3. Õpilasena tahan programmeerimisülesande lahendust esitada.
4. Õpilasena tahan näha testitulemusi.
5. Õpilasena tahan ülesannet lahendades samaaegselt kirjeldust näha.
6. Õpilasena tahan peale ülesande valimist, et mall oleks esialgu automaatselt koodiredaktorisse laetud.
7. Õpilasena tahan malli taastada.
8. Õpilasena tahan näha enda varasemalt esitatud lahenduste tulemusi.

Mittefunktsionaalseteks nõueteks on:

- Lahendus töötab IntelliJ platvormi IDE-l;
- Lahendus saab automaattestrilt tagasiside kahe minuti jooksul;
- Lahendus toetab IntelliJ IDE versioone alates 2023. aasta versioonidest;
- Lahendus on liidestatud Charoni taga-rakendusega;

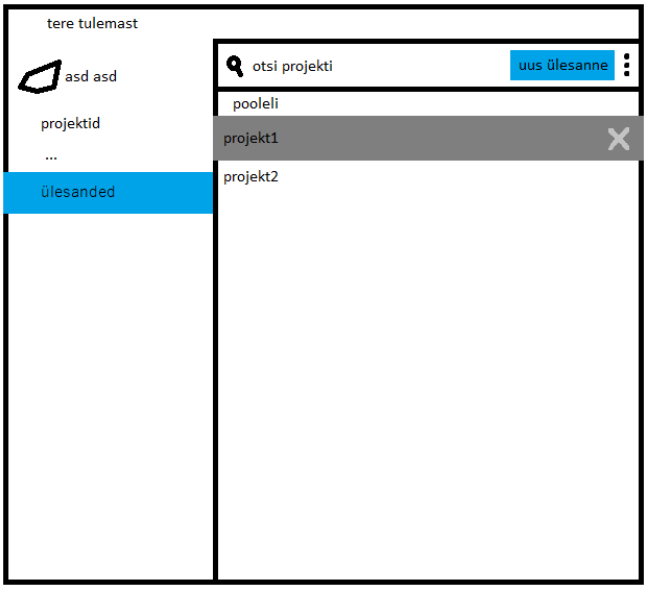
- Lahendus on piisavalt abstraktne, et võimaldab lahendada ka muid (Charonist erinevaid) programmeerimisülesandeid.

Töö autor otsustas, et funktsionaalsed nõuded 2 kuni 6 ja esimesed 4 mittefunktsionaalset nõuet on võrreldes teistega olulisemad ning need peaks olema täidetud pistikprogrammi algversioonis enne selle avaldamist.

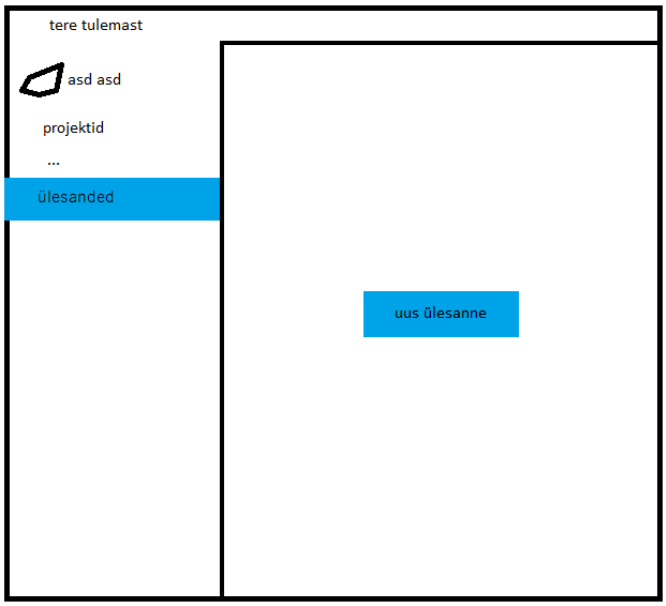
# 3 Pistikprogrammi kujundamine

Eesmärgiga uutele kasutajatele programmeerimisülesannete lahendamist mugavaks ning lihtsaks teha nõudis kasutajaliidese kujundamisel kasutajamugavuse tagamist. Kasutajaliidese kujundamisega polnud autoril kuigi palju kogemust. Kuna iga IntelliJ platvormi IDE kasutajaliidese struktuur on sarnane nii üksteisele kui ka teistele töö käigus analüüsitud lahendustele, siis tulemusena võttis töö autor rohkesti eeskuju JetBrains Academy kasutajaliidesest.

Esiteks on vaja, et kasutaja saaks IDE-d esmalt avades valida tervituse vaatest uue programmeerimisülesande. Sealne vasakpoolne menüü kuvab pistikprogrammi valiku, mida valides ilmub menüü paremale poole pistikprogrammi üks lisatud akendest. Aken sisaldab võimalust alustada uut ülesannet (Joonis 7) (Joonis 8). Aken kuvab veel alustatud ülesandeid nende olemasolu korral ning annab võimaluse neid soovi korral otsida, jätkata või lõpetada (Joonis 7). Ülesande lõpetamine eemaldaks ülesande ainult alustatud ülesannete nimekirjast ning täielikuks eemaldamiseks tuleks ülesanne kustutada kasutades operatsioonisüsteemi failisüsteemi (sarnaselt IntelliJ platvormi IDE-des projektide haldamisega nende tervituse vaates). Juhul, kui ei ole alustatud ühtegi ülesannet, siis pakub aken ainult võimalust alustada uut ülesannet (Joonis 8).

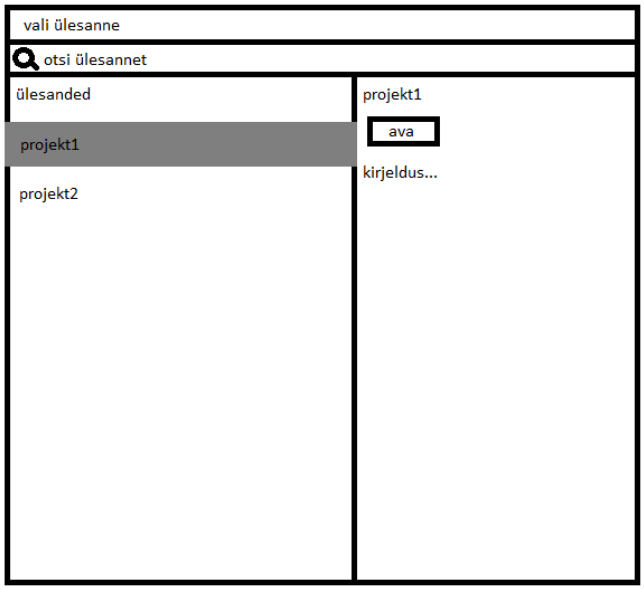



Joonis 7. Tervituse vaate illustratsioon. Ülesandeid on alustatud.



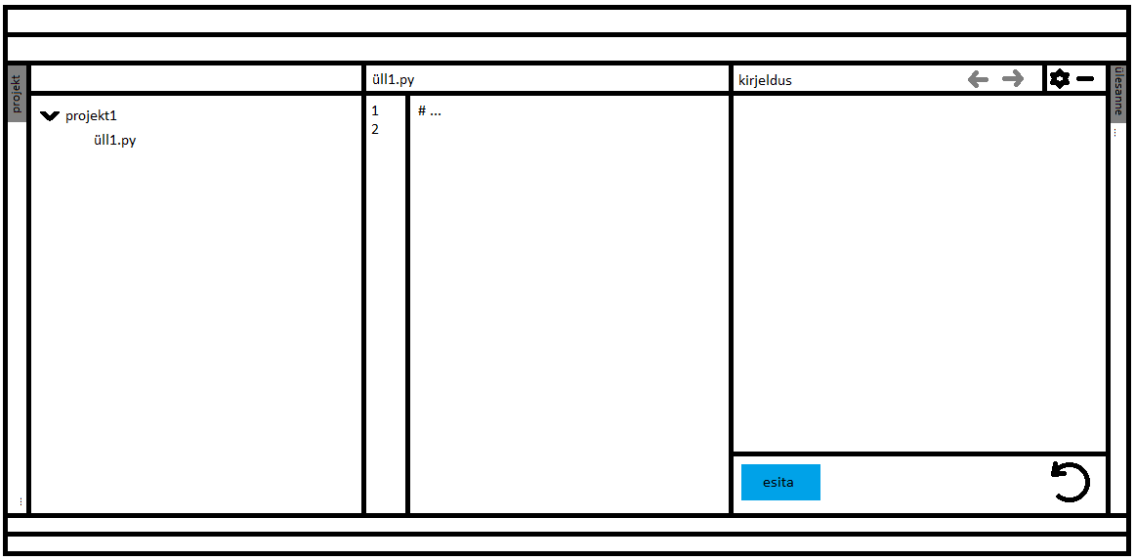

Joonis 8. Tervituse vaate illustratsioon. Ülesandeid ei ole alustatud.

Järgmisena on tarvis akent, mis avaneb siis, kui üritada alustada tervituse vaatest uut ülesannet. Vastav aken peab kuvama nimekirja kõikidest avalikest ülesannetest ning ühte neist valides kuvatakse ülesande juures selle detailid ja kirjeldus (Joonis 9). Nimekirja ning detailide tõttu on tarvis aken kuidagi kaheks jagada – näiteks vertikaalselt (Joonis 9). Ülesannet avades laaditakse see IDE-sse.




Joonis 9. Ülesande valimise akna illustratsioon. Valitud on ülesanne "projekt1".

Kui ülesanne on valitud ja laaditud, siis peab projektivaates saadaval olema aken, mis kuvaks lahendajale ülesande kirjeldust (Joonis 10). Kuna tagasiside saamiseks on lahendajal vaja esitus testimisele saata, siis on mõistlik seda akent ära kasutada veel testimise nupu kuvamiseks (Joonis 10). Testimise tagasiside on nähtav nupu juures. Lisaks on veel võimalus lahendajal ülesande esialgne mall taastada, mis on kuvatud samuti testimise nupu juures (Joonis 10).




Joonis 10. Projektivaate illustratsioon.

# 4 Lahenduse arhitektuur

Antud peatükk üritab seletada pistikprogrammi arhitektuuri ning käitub sissejuhatusena järgmisele peatükile.

## 4.1 IntelliJ platvormi pistikprogrammi tarkvaraarenduskomplekt

Kuna küsitluse käigus osutus kõige populaarsemaks IDE-ks JetBrains Suite ehk mingi IntelliJ platvormil põhinev IDE, siis tutvus töö autor IntelliJ platvormi pistikprogrammi tarkvaraarenduskomplektiga (edaspidi SDK). IntelliJ platvormi pistikprogrammi SDK on suurejooneline, mille tõttu kirjeldatakse antud alapeatükis vaid töö jooksu kasutatud SDK põhiosi:

- Teenus – sarnaselt teiste raamistikega on ka IntelliJ platvormi pistikprogrammi SDK-s teenused (*service*), mis kapseldab äriloogikat ning pakub funktsionaalsuse taaskasutatavust;

- Tegu – tegude (*action*) süsteem võimaldab tarkvaraarendajatel lisada IDE menüüdesse erinevaid valikuid ning neid valides nendega seotud tegusid töödelda;

- Püsivuse mudel – püsivuse mudel (*persistence model*) lubab pistikprogrammil salvestada erinevaid meta-andmeid (näiteks projektiga seotud ülesande infot ja kirjeldust);

- Virtuaalne failisüsteem – virtuaalne failisüsteem (*virtual file system*) on universaalne IntelliJ platvormi pistikprogrammi SDK komponent, mis võimaldab faile hallata (näiteks neid luua, nende sisu vaadata, neis muudatuse jälgida) sõltumata failide asukohas (kettal, veebiserveris või kuskil mujal);

- Projekt – projektiga (*project*) seotud info, näiteks milliseid teeke kasutatakse, kuidas projekti ehitada ja käima panna ja millist SDK-d selleks kasutatakse. Projekt võib koosneda ühest või mitmest moodulist. Projekti rakendusliides annab võimaluse selle andmetega tegeleda.

- Laiendus – laiendused (*extension*) võimaldavad IntelliJ platvormi funktsionaalsust oluliselt täiendada kas näiteks kasutajaliidesesse akent lisades või täiendavaid pistikprogrammi sätteid IDE sätete aknas kuvades;

- Pistikprogrammi konfiguratsioonifail – enamus ülaltoodud funktsionaalsusest konfigureeritakse pistikprogrammi konfiguratsioonifailis `plugin.xml`. Seal näiteks registreeritakse laiendusi, tegusid, teenuseid ning see sisaldab üldist informatsiooni pistikprogrammi kohta (näiteks nimi, versioon, kirjeldus ning sõltuvused).

## 4.2 Pistikprogrammi loomise võimalused ja kasutatud teegid

Antud alapeatükk üritab selgeks teha erinevad pistikprogrammiga alustamise variandid ning põhjendab töö autori valikut. Lisaks tutvustatakse erinevaid töös kasutatud teeke, mis ei olnud otseselt IntelliJ platvorm SDK-ga seotud.

### 4.2.1 Pistikprogrammi loomise võimalused

Dokumentatsiooni järgi oli pistikprogrammi arendamise jaoks vajalik olemasolev kogemus järgmistega: Java virtuaalmasina rogrammeerimiskeel; Gradle või sarnane süsteemi ehitamise süsteem; Java Swing; *Java Concurrency Model*; IntelliJ platvormil põhinev IDE [7].

Uue pistikprogrammi loomiseks andis dokumentatsioon kaks valikut, milleks olid uue pistikprogrammi projekti genereerimine uue projekti viisardiga (*New Project Wizard*) IntelliJ platvormi IDE-s või „harutada“ (*fork*) IntelliJ platvormi pistikprogrammi mallist uus projekt [8], [9]. Viimase eesmärk on uue pistikprogrammi projekti alustamist kiirendada jättes vahele vajaduse uut projekti ja pidevat integratsiooni (*continuous integration*) üles seada ja sätestada. Kuna malli hoiustati GitHubis, mis on arendusprojektide hoiustamisplatvorm, siis nõudis malli harutamine ka pistikprogrammi hoiustamist seal.

Mall kasutas võimalikult palju Kotlini programmeerimiskeelt. Dokumentatsioon soovitas kasutada ehitussüsteemina Gradle’it, mille tõttu oli ka Gradle’il üles ehitatud projekti mall [10]. Malliga tuli veel kaasa näidisprogrammikood, funktsionaalseid teste sisaldavad programmifailid, Gradle Changelog Plugin, mis lihtsustas projekti muudatuste logimist,

Gradle Kover Plugin, mis andis ülevaate programmikoodi katvusest (*code coverage*), Gradle Qodana Plugin, mis jälgis programmikoodi kvaliteeti ehk sarnaselt IntelliJ platvormi IDE-ga soovitas programmikoodis võimalusel erinevaid optimaalsusi või hoiatas halbade tavade eest nende olemasolu korral, näidisfail kasutajaliidese testimise jaoks IntelliJ UI Test Robot abil, predefineeritud pistikprogrammi jooksutamis- ning silumiskonfiguratsioonid [11], [12], [13], [14]. Lisaks neile oli üles seatud veel pidev integratsioon, mis sõltus GitHub Action-itest [15]. Töövooge oli defineeritud mitmeid: ehitamine (*build*), väljalaskmine (*release*), kasutajaliidese testide jooksutamine (*run UI tests*). Projekti sõltuvuste haldamine oli tehtud automaatseks kasutades GitHub Dependabot-i.

IntelliJ platvormi pistikprogrammi väljalaskmise lihtsustamiseks on projekti mallis konfigureeritud veel pistikprogrammi „alla kirjutamine“ (*plugin signing*) ja pistik-programmi „avaldamine“ (*plugin publishing*). Pistikprogrammi alla kirjutamine on kavandatud kinnitamaks, et pistikprogramme ei muudeta nende avaldamise ja „üle-andmise“ (*delivery*) käigus ning kontrollib, et avalik osa genereeritud võtmest on avaldamiseks esitatud, ning et pistikprogrammi signatuur on õige [16]. Pistikprogrammi alla kirjutamine ning avaldamine teostatakse läbi GitHub-i väljalaskmise töövoo.

Töö autor otsustas nende hulgast pistikprogrammi malli kasuks, sest mallis oli suurem osa pistikprogrammi sätestamisest juba tehtud ning ta oli huvitatud uue programmeerimiskeele õppimisest. Eesmärgiga vältida muudatuse tegemist projekti põhiharus (*main branch*) loodi järgmist väljalaskmist ette kujutav haru, mida hakati uuendama edasiste muudatustega. Malli kasutama alustamise viimistluseks oli vaja manuaalselt kohandada muutujaid Gradle’i omaduste failis ning näidisfailide kustutamise asemel liigutati need taaskasutamise eesmärgil vaikimisi paketist ümber.

### 4.2.2 Kotlin, Gradle, Korutiinid ja Retrofit

Lisaks malliga kaasa tulnud teekidele kasutas töö autor veel muid teeke, mida ta sai ise valida. Käesoleva töö pistikprogrammi suhtlemiseks Charoni taga-rakendusega oli töö autoril vaja kasutada mingisugust asünkroonse programmeerimise tehnikat ning HTTP funktsionaalüksust (*HTTP client*). Kotlini välja pakutud asünkroonse programmeerimise tehnikatest otsustas töö autor käesolevas töös kasutada Kotlini „ko-rutiinide“ (*coroutines*) lähenemist, mis tundus valikutest kõige mugavam [17], [18]. Enne Kotlini korutiine

kasutama hakkamist oli töö autoril vaja talle uue asünkroonse programmeerimise tehnikaga tutvuda.

Üks korutiin on isend peatatavast arvutusest ning on põhimõtteliselt sarnane lõimega (*thread*), kuid erinevalt lõimest võib üks korutiin alustada ühes lõimes ning jätkata teises lõimes [19]. Korutiinid järgivad „struktureeritud kokkulangevuse“ (*structured concurrency*) printsiipi ehk uusi korutiine saab algatada ainult spetsiifilise „korutiini skoobi“ (*coroutine scope*) sees, mis piirab korutiini eluaega [19]. Struktureeritud kokkulangevus kindlustab, et korutiinid ei kao ära ega leki – välimine skoop ei saa enda tööd lõpetada enne, kui kõik tema laps-korutiinid on enda töö lõpetanud [19]. „Korutiini skoopide ehitajaid“ või lihtsalt „korutiinide ehitajaid“ (*coroutine builders*) on mitmeid, millest üks on näiteks *launch*, mis algatab uue korutiini samaaegselt ülejäänud programmikoodiga, jätkab iseseisvalt tööd ning tagastab *Job* objekti, mille kaudu saab algatatud korutiini käsitleda [19].

Korutiinid käivituvad alati mingis kontekstis, mis esindab *CoroutineContext* tüübi väärtust [20]. „Korutiini kontekst“ (*coroutine context*) on hulk erinevaid elemente, millest peamised on korutiini *Job* ja „korutiini lähetaja“ (*coroutine dispatcher*), mis otsustab, millist lõime või lõimesid vastav korutiin enda teostamiseks kasutab [20]. Kasutamaks Kotlini korutiine programmikoodis klassisiseselt laiendas töö autor vastavat klassi *interface*’iga `CoroutineScope`, mis pakkus abstraktsiooni, hõlmates endas käsitsi kontekstide ja *Job* objektide juhtimist, et siduda vastava klassi elutsüklid ja selle korutiinid (Joonis 11) [20]. Klassi laiendamine `CoroutineScope` *interface*’iga nõudis veel üle kirjutada klassisisene `coroutineContext` muutuja (Joonis 11).

```
class BrowseAssignmentsDialog : DialogWrapper(true), CoroutineScope {

    private val job = Job()
    override val coroutineContext: CoroutineContext
        get() = job + Dispatchers.Swing
    private val dialogPanel: JPanel = JPanel(BorderLayout())

    ...
}
```

Joonis 11. Ülesande valimise akent realiseeriv klass `CoroutineScope` *interface*'i laiendusega.

Käesoleva töö HTTP funktsionaalüksusena võeti kasutusele tüübikindel Square. Inc. Retrofit, millega sai töö autor tuttavaks läbi Kotlini korutiinide dokumentatsiooni [21],

[22]. HTTP funktsionaalüksuse initsialiseerimiseks lõi töö autor programmi rakendustasemelise (*application level*) teenuse sisse funktsiooni `getCharonService()`, kus seda veel konfigureeriti – näiteks päringu lugemise ajalõpp (*read timeout*) seati kahele minutile, kuna ülesande esituse testimine võib võtta aega kauem, kui vaikimisi väärtus (10 sekundit) ning lisati „client“ päis, et päringu tuvastaks ära Charoni vahevara (Joonis 12) [21].

```
fun getCharonService(): CharonService {
    val httpClient = OkHttpClient.Builder()
        .readTimeout(120, TimeUnit.SECONDS)
        .addInterceptor { chain ->
            val original = chain.request()
            val builder = original.newBuilder()
                .header("Accept", "application/json")
                .header("client", MyBundle.message("name"))
            val request = builder.build()
            chain.proceed(request)
        }.build()

    val contentType = "application/json".toMediaType()
    val json = Json { ignoreUnknownKeys = true }
    val retrofit = Retrofit.Builder()
        .baseUrl("https://moodle.taltech.ee/mod/charon/api/")
        .addConverterFactory(json.asConverterFactory(contentType))
        .client(httpClient)
        .build()

    return retrofit.create(CharonService::class.java)
}
```

Joonis 12. HTTP klienti initsialiseeriv ning tagastav funktsioon.

Päringute jaoks lõi töö autor eraldi *interface*'i `CharonService`, milles iga funktsioon esindas üht päringut (Joonis 13) [21]. Päringu meetodit ning veebiaadressi sai täpsustada vastava päringu meetodi annotatsiooniga, mille ainsaks parameetriks sõne veebiaadressi täpsustusest (Joonis 13) [21]. Kotlini korutiinid nõudis siinkohal veel funktsiooni ette märksõna *suspend* märkimaks ära asünkroonset funktsiooni, mida saab vajadusel peatada ning hiljem jätkata (Joonis 13) [17].

```
@GET("charons/{charon}?with=templates")
suspend fun getCharon(
    @Path("charon") charon: Int,
): Response<Assignment>
```

Joonis 13. Charoni ülesande kättesaadavat päringut esindav funktsioon.

# 5 Pistikprogrammi arendus ja Charoniga ühendamine

Edasi kirjeldatakse ja põhjendatakse pistikprogrammi üles ehitamist ning seda Charoniga ühendamist.

## 5.1 Kasutajaliidese loomine

Esialgu pidas töö autor vajalikuks luua kõik aknad, mida võiks vaja minna pistikprogrammi kasutamise käigus. Nende loomisel üritati lähtuda kolmandas peatükis mainitud visanditest.

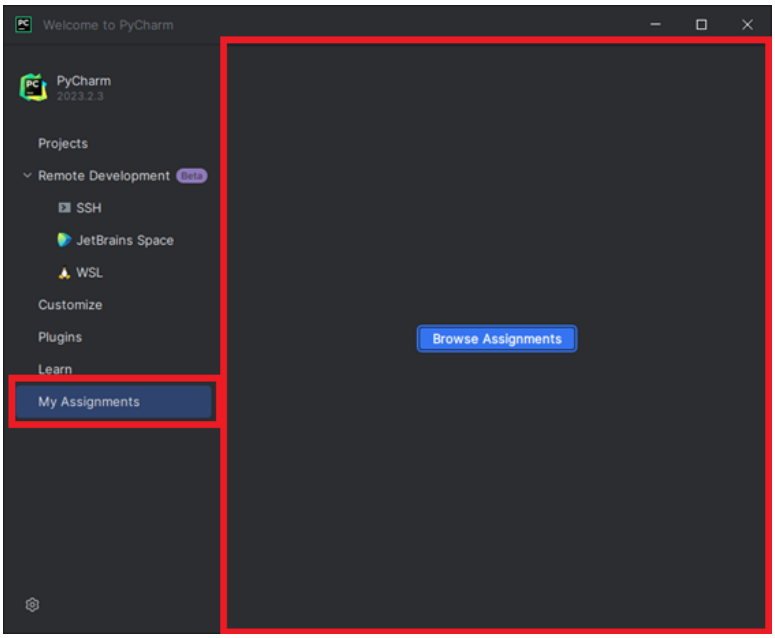
### 5.1.1 Tervituse vaate aken

Esimest akent tervituse vaatesse lisamise jaoks lisas töö autor pistikprogrammile tervituse vaate laienduse, mille ta leidis IntelliJ platvormi pistikprogrammi SDK laienduspunktide (*extension point*) ja kuulajate (*listener*) loendist [23]. Selle jaoks deklareeris töö autor laienduse projekti pistikprogrammi konfiguratsioonifailis. Deklareerimine nõudis laienduse realisatsiooni atribuudiks määrata laiendust realiseeriva klassi (Joonis 14). Deklareerimine pakkus veel võimalust täpsustada ära, kuhu tervituse vaate menüüosiste loetelus uus valik täpsemalt paigutatakse (Joonis 14). Töö autor määras projektipoolse valiku vastavas loetelus viimaseks ehk alumisimaks (Joonis 14).

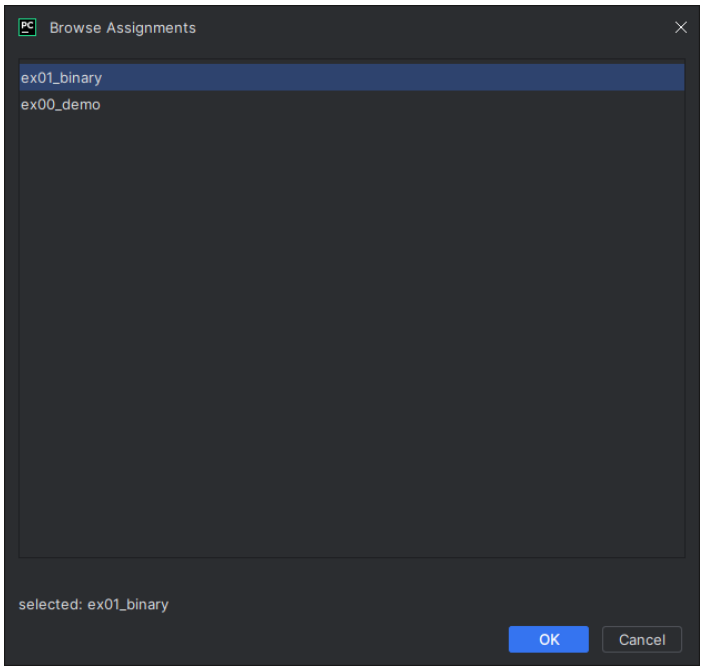
Aru saamaks kuidas laiendust korrektselt realiseerida, suundus töö autor „IntelliJ platvormi uurijasse" (*IntelliJ Platform Explorer*), mis oli otsimistööriist uurimaks laienduspunkte ja kuulajaid [24]. Otsimistööriista abil oli võimalik tuvastada kõik avatud lähtekoodiga pistikprogrammid, mis kasutasid otsimise tulemusena valitud laiendust või kuulajat. Lisaks tagas see ka hüperlingi laiendust realiseeriva faili juurde. Üks nendest pistikprogrammidest oli varem nimetatud JetBrains Academy ning töö autor kasutas selle lähtekoodi õppimaks realiseerima käesoleva töö tervituse vaate akent (Joonis 14) (Joonis 15) [25]. Akna keskele lõi töö autor nupu, millele vajutades avaneks ülesande valimise aken (Joonis 15) (Joonis 16).

```
class MyWelcomeTabFactory : WelcomeTabFactory {
    override fun createWelcomeTabs(
        ws: WelcomeScreen,
        parentDisposable: Disposable
    ): MutableList<WelcomeScreenTab> {
        return mutableListOf(
            object : DefaultWelcomeScreenTab(
                MyBundle.message("myWelcomeTabTitle")
            ) {
                override fun buildComponent(): JComponent {
                    return MyWelcomeTabPane()
                }
            }
        )
    }
}
```

Joonis 14. Tervituse vaate akna laiendust realiseeriv klass.




Joonis 15. Tervituse vaade. Vasakpoolsest menüüst on vajutatud käesoleva töö pistikprogrammi valikule ning avatud on ka vastav aken. Ülesandeid ei ole alustatud.




Joonis 16. Käesoleva töö pistikprogrammi ülesande valimise aken.

### 5.1.2 Ülesande valimise aken

Järgmisena oli vaja luua aken, mis kuvab kõiki Charoni avalikult kättesaadavaid programmeerimisülesandeid (Joonis 16). Selle realiseerimiseks kasutas töö autor juhiseid ning näidet IntelliJ platvormi pistikprogrammi SDK-st [26]. Tarvis oli luua klass, mis laiendaks *interface*'i `DialogWrapper` ning kutsuks initsialiseerides välja selle konstruktorit (Joonis 17). Lisaks oli vaja üle kirjutada *interface*'i meetod `createCenterPanel()` (Joonis 17).

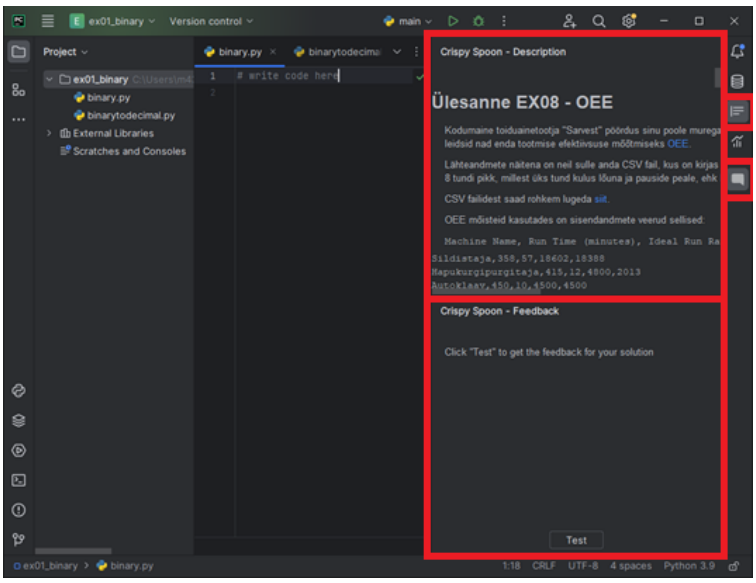
```
class BrowseAssignmentsDialog : DialogWrapper(true) {

    private val dialogPanel: JPanel = JPanel(BorderLayout())

    init {
        title = MyBundle.message("browseAssignmentsDialogTitle")
        dialogPanel.preferredSize = Dimension(600, 500)
        init()
    }

    @Nullable
    override fun createCenterPanel(): JComponent {
        loadAssignments()
        return dialogPanel
    }
}
```

Joonis 17. Ülesande valimise akent realiseeriv klass.

Nupud kirjadega „*OK*“ ja „*Cancel*“ tulid `DialogWrapper` *interface*’ist vaikimisi (Joonis 16). Ülesannete kuvamiseks lõi töö autor akna sisse nimekirjalaadse komponendi ning valitud ülesande selgemaks kuvamiseks märgise nimekirja alla (Joonis 16).

### 5.1.3 Ülesande kirjelduse ning esituse tagasiside aknad

Järgmisena oli vaja luua projektivaatesse aken kirjelduse kuvamiseks ning testimise võimaluse tagamiseks. Nende jaoks otsustas töö autor erinevalt illustratsioonist luua kaks iseseisvat akent – üks eraldi kirjelduse kuvamiseks ning teine testimiseks ning tagasiside kuvamiseks (Joonis 10) (Joonis 18). Sarnaselt ülesande valimise akna loomisega leidis töö autor nende kahe „tööriistaakna“ (*tool window*) kohta abi IntelliJ platvormi pistikprogrammi SDK-st ning näidis oli veel eraldi olemas IntelliJ platvormi pistikprogrammi SDK koodinäidiste hoidlas [27], [28]. Tööriistaakende üles seadmiseks oli töö autoril tarvis sarnaselt tervituse vaatesse akna lisamisega pistikprogrammi konfiguratsioonifailis need deklareerida. Täpsustuseks oli vaja kirja panna aknaid realiseerivad klassid ning nende identifikaatorid, mille abil sai akendele programmikoodis mugavalt ligi. Lisas 2 on välja toodud kirjelduse akent realiseeriv programmikood.




Joonis 18. PyCharm IDEs projektina avatud Charoni ülesanne. Paremal pool on esile toodud kirjelduse aken (ülemine), testimise aken (alumine) ning mõlema ikoonid.

Kuna uuemad IntelliJ IDE versioonid nõuavad, et igal tööriistaaknal oleks olemas mingisugune ikoon, mille abil neid ilma tekstita kuvada, siis oli töö autor neid samuti

sunnitud akendele lisama (Joonis 18). Selle jaoks taaskasutas töö autor mõningaid IDE-s juba olemasolevaid (mitte tööakende identifitseerimiseks kasutatud) ikoone [29]. Mõlemasse tööriistaaknasse lisas töö autor keritavad komponendid ning ainult testimise aknasse testimise nupu (Joonis 18).

### 5.1.4 Ülesande valimise akna kuvamine

Töö autor eeldas, et parema kasutajamugavuse tagamiseks peaks uue ülesande valimine olema võimalik veel läbi IDE tööriistariba, mille tõttu ülesande valimise akna kuvamiseks otsustas töö autor kasutada IntelliJ platvormi tegu [30]. Sarnaselt laiendustele oli tegevus samuti vaja deklareerida pistikprogrammi konfiguratsioonifailis, kus kirjeldati ära veel tegevuse koht tööriistaribal – töö autor määras selleks „fail“ (*File*) grupi ja asukohaks kohe peale muu projekti avamise tegevust (*Open…*). Tegevuse klass pidi laiendama `AnAction` klassi ning selle tõttu veel üle kirjutama meetodi `actionPerformed(event)`, mis aktiveeris siis, kui tegevus käivitada (näiteks vajutada sellele tööriistaribal) (Joonis 19). Töö autor kasutas sama meetodit ära ülesande valimise akna kuvamiseks (Joonis 19).

```
class BrowseAssignmentsAction : AnAction() {

    override fun actionPerformed(@NotNull e: AnActionEvent) {
        val dialog = BrowseAssignmentsDialog()
        dialog.show()
    }
}
```

Joonis 19. Ülesande valimise akna avamise tegevust realiseeriv klass.

## 5.2 Muudatused Charonis

Kuna pistikprogramm ei saanud täielikult töötada ilma mingite muudatusteta Charonis, siis pidi töö autor sellele lisama vastava liidese.

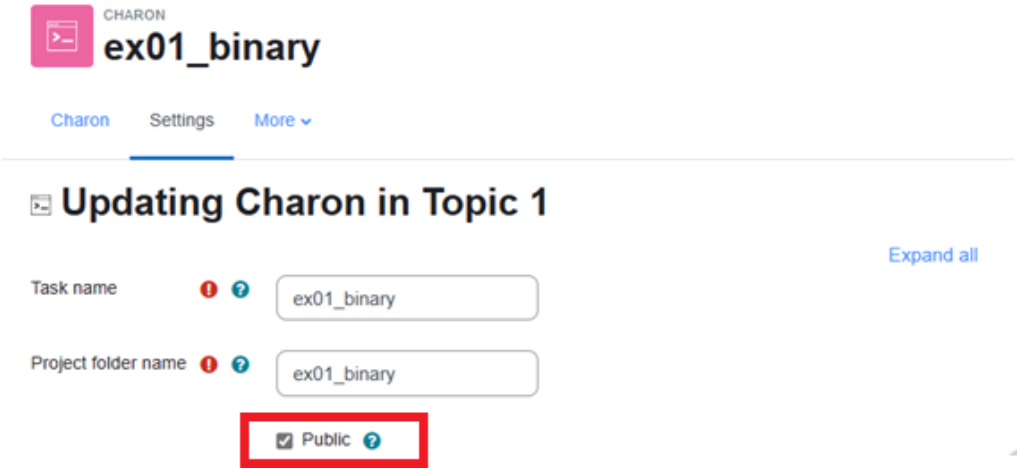
### 5.2.1 Andmebaasi väli

Vaikimisi peavad Moodle’i kasutajad Charoni ülesannete lahendamiseks võrguseanssi alustama. Käesolev töö nõudis võimalust ülesandeid kuidagi avalikeks määrata. Üheks variandiks oli ära kasutada selle jaoks Moodle’i tegevuse (*activity*) silte (*tag*), mis on lisatavad Moodle’i tegevuse sätete leheküljel.

Charoni ülesande avalikkuse määramiseks otsustas töö autor siiski Moodle'i tegevuse siltide kasutamise asemel lisada Charoni ülesande andmebaasi tabelisse eraldi välja, sest sildi lisamine tundus talle ebaintuitiivne – koht Moodle'i tegevuse sätete leheküljel oli siltide jaoks vaikimisi lehekülje lõpus ning puudus variant võimalust kuidagi eraldi esitleda ja kirjeldada. Lisatud andmebaasi välja nimeks sai `is_public`, mis hoiab täisarvu kombel ülesande avalikkuse tõeväärtust [31].

### 5.2.2 Kasutajaliides ja andmebaasi välja salvestamine

Peale välja lisamist andmebaasi tabelisse oli töö autoril võimalik hakata seda salvestama. Töö autor lisas märkeruudu avalikkuse atribuuti muutmiseks Charoni ülesande sätete leheküljele ülesande põhisätete juurde koos lühikese vihjega (Joonis 20). Vihje kuvatakse vihjemullina siis, kui kursor asetada atribuudi paremal pool asuva küsimärgi peale.




Joonis 20. Charoni ülesande sätete lehekülg. Põhisätted. Esile on tõstetud käesoleva töö käigus lisatud märkeruut.

Atribuudi salvestamiseks tuli töö autoril lisada lehekülje väljale kuulaja ning taga-rakenduses kirjutada välja nimi modifitseeritavate väljade massiivi muudatuste lubamiseks.

### 5.2.3 Rakendusliides

Charoni rakendusliidesele lisati üks *endpoint* kõikide avalike ülesannete kättesaamiseks. *Endpoint* tagastab ülesanded koos identifikaatori, nime ning kirjeldusega. Isegi kui esialgses ülesande valimise aknas pole ülesannete kirjeldused kuidagi kättesaadavad, on plaan need seal tulevikus nähtavaks teha.

Järgmiseks muutis töö autor kahte olemasolevat *endpoint*i – üks ühe ainsa ülesande kättesaamiseks ning teine ülesande esituse testimiseks läbi automaattesteri. Olemasolev *endpoint* ainsa ülesande kättesaamiseks oleks olnud piisav, kui see oleks valikuliselt koos ülesandega tagastanud ülesande malli(d). Niisiis lisas töö autor *endpoint*ile võimaluse küsida ülesanne koos malli(de)ga kasutades päringu parameetrit.

Esituse testimise *endpoint*i oli vaja rohkem muuta, sest see salvestas lisaks testimisele veel testitulemused andmebaasi ja sidus testitulemused reaalse Moodle'i kasutajaga. *Endpoint*i taaskasutamiseks käesoleva töö pistikprogrammi poolt loodi Moodle'i kasutaja asemel imitatsioon kasutajast, kellest edaspidi olenes, kas kontrollida e-postiaadressi, kas salvestada esituse tulemused ning kui palju infot tagastada. Pistikprogrammile tagastatakse ainult automaattesteri väljund, mis on HTML tekst.

Viimaks oli pistikprogrammi poolt pöördutavatele *endpoint*idele lisada vaja vahevara (*middleware*) päringute valideerimiseks ja võimalike rünnakute vähendamiseks. Lisati kaks erinevat vahevara klassi – üks kontrollimaks, et päritud ülesanne oleks avalik ning teine kontrollimaks, kas päringul on enda päises klienti identifitseeriv väli „client" õigesti väärtustatud ehk väärtuseks on käesoleva töö projekti nimi („Crispy Spoon"). Eristamaks päringuid Moodle'ist ning käesoleva töö pistikprogrammist lisati erinevalt väärtustatud päis veel Moodle'i poolt tehtavatele päringutele.

## 5.3 Charoni ja pistikprogrammi integreerimine

Kui olid olemas nii pistikprogrammi tühjad aknad kui ka võimalus läbi Charoni taga-rakenduse ülesandeid akendes kuvama hakata, siis sai töö autor lõpuks Charoni ning pistikprogrammi integreerimisega alustada.

### 5.3.1 Ülesannete kuvamine ülesande valimise aknas

Et ülesandeid kuvada ülesande valimise aknas, siis defineeris töö autor esialgu ülesande valimise akent esindava klassi skoobina uutele korutiinidele, mis jõustus lisades klassi laienduseks *interface*'i `CoroutineScope` (Joonis 11). Avalike ülesannete kättesaamiseks loodi vastavasse klassi eraldi funktsioon `loadAssignments()`, mis algatab *launch* funktsiooniga korutiini Charoni taga-rakendusega suhtlemiseks (Joonis 21). Ligipääsu HTTP funktsionaalüksusele teostas IntelliJ platvormi pistikprogrammi SDK „komponentide haldur" (*component manager*), millega sai erinevatel tasemetel

(rakendus-, projekti- ja mooduli-tasemeline) teenustele ligi, nagu seda on tehtud näiteks Joonis 21 kuvatud programmikoodi teisel real [32].

```
private fun loadAssignments() {
    val service = service<MyApplicationService>()
        .getCharonService()

    launch {
        val assignments = service
            .getCharons()
            .body() ?: emptyList()

        updateAssignments(assignments)
    }
}
```

Joonis 21. Ülesande valimise aknas ülesandeid päriv funktsioon.

Kui kõik kättesaadavad programmeerimisülesanded olid päritud ja vastu võetud, saadeti need edasi `updateAssignments()` funktsiooni, mis loodi samuti töö autori poolt ning mille eesmärgiks sai uuendada kasutajaliidest vastava informatsiooniga (Joonis 21) (Joonis 22). Vastavas funktsioonis lisatakse ülesanded IntelliJ platvormi poolt visualiseeritavasse massiivi, luuakse silt informeerimaks kasutajat valitud ülesandest, pannakse massiivile külge kuulaja, mis muudab valitud ülesannet (ja seehulgas ka lisatud silti) ning lõpuks akna paikapidavust kontrollitakse (Joonis 16) (Joonis 22).

```
private fun updateAssignments(data: List<AssignmentInit>) {
    val assignmentList = JBList(data)
    assignmentList.selectionMode = ListSelectionModel.SINGLE_SELECTION

    val selectedLabel = MyBundle.message("selectedLabel")
    val lbl = JBLabel("$selectedLabel: " + MyBundle.message("selectedNone"))

    val mouseListener = object: MouseAdapter() {
        override fun mouseClicked(e: MouseEvent) {
            lbl.text = "$selectedLabel: ${assignmentList.selectedValue}"
            assignment = assignmentList.selectedValue
        }
    }
    assignmentList.addMouseListener(mouseListener)
    val assignmentScroller = JBScrollPane(assignmentList)

    dialogPanel.add(assignmentScroller, BorderLayout.NORTH)
    dialogPanel.add(lbl, BorderLayout.SOUTH)
    dialogPanel.revalidate()
}
```

Joonis 22. Ülesande valimise aknas ülesandeid kuvav funktsioon.

### 5.3.2 Ülesande avamine IntelliJ projektina

Ülesande avamiseks IntelliJ projektina kasutas töö autor ära ülesande kuvamise akna „OK“ nuppu (Joonis 16). Ülesande valimise akna klassi loodi `assignment` muutuja, mida peale „OK“-nupule vajutamist kontrollida – kui on valitud, siis avada ülesanne, kui mitte, siis lihtsalt suleb akna (funktsionaalsuselt võrdne vajutamisega nupu „Cancel“ peale) (Joonis 16) (Joonis 11). Lisas 3 on välja toodud modifitseeritud teisend Joonis 19 kuvatud funktsioonist, mis lisaks ülesande valimise akna kuvamisele kontrollib sama akna sulgemise viisi (funktsiooniga `dialog.showAndGet()`), kontrollib, et ülesanne on valitud, genereerib sellest IntelliJ projekti, salvestab ülesande kirjelduse ning muutuja eristamaks käesoleva töö projekte teistest IntelliJ projektidest, pärib valitud ülesande Charoni taga-rakenduselt uuesti (nüüd koos mallidega), loob mallide jaoks IDE kausta failid, kopeerib mallid vastavatesse failidesse ning annab korralduse tööriistaakendel endid kuvada [26]. Lisas 3 välja toodud programmikoodi tulemusena avaneb projektivaade ning kuvatud on veel ülesande kirjelduse aken ning ülesande esitamise aken (Joonis 18).

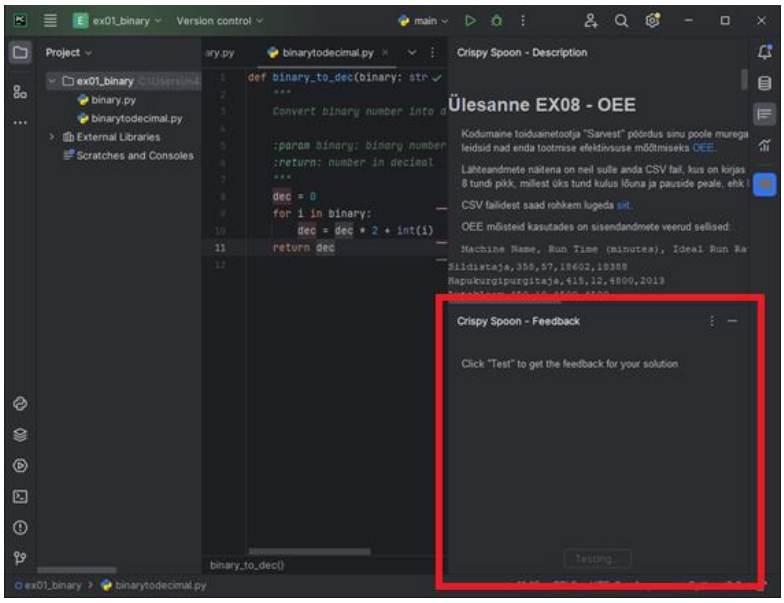
### 5.3.3 Ülesande esituse testimine ning tagasiside kuvamine

Ülesande testimiseks lisati tagasiside akna nupule kuulaja, mis algataks selle peale vajutades ülesande testimise protsessi (Joonis 18). Ülesande testimise protsess algab projekti kausta leidmisest ja `.idea` kausta leidmisest ning seejärel esimese rekursiivsest läbikäimisest (Joonis 23). Lahenduse failide hulka lisatakse kõik sealsed failid, mis ei ole omakorda `.idea` kausta sees, mis on IDE poolt loodud kaust enda sätete salvestamiseks (Joonis 23). Kaustasid ja faile käsitletakse käesolevas töös enamasti läbi virtuaalfailisüsteemi (*virtual file system*), mis IntelliJ platvormil abstraheerib ning *cache*’ib failisüsteemi [33] [34]. Kindlustamaks lahenduste failides kõik IDE-s teostatud muudatused oli töö autoril tarvis enne faili sisu lugemist fail virtuaalfailisüsteemi kaudu värskendada (Joonis 23). Töö autor otsustas projekti kausta rekursiivse läbi käimise üle lihtsa failinimede salvestamise asemel, et lahendajal oleks võimalus enda lahendusi kuidagi koondada, näiteks lisada projekti kausta mingi uus kaust, mille ainsaks elemendiks on mingisugune mitme esituse faili poolt kasutatud programmikood (Joonis 23).

```
val templates = mutableListOf<Template>()

val lfs = LocalFileSystem.getInstance()
val projectFolder = lfs.findFileByPath("${project.basePath}")
    ?: error("project folder not found")
val ideaFolder = projectFolder.findChild(".idea")

for (child in VfsUtil.collectChildrenRecursively(projectFolder)) {

    val ideaFile = ideaFolder == null
            || child.path.startsWith(ideaFolder.path)
    if (!child.isDirectory && !ideaFile) {
        val template = lfs.findFileByPath(child.path)
            ?: error("file with path ${child.path} not found")
        template.refresh(false, false)
        val text = VfsUtil.loadText(template)
        templates.add(Template(template.name, text))
    }
}
```

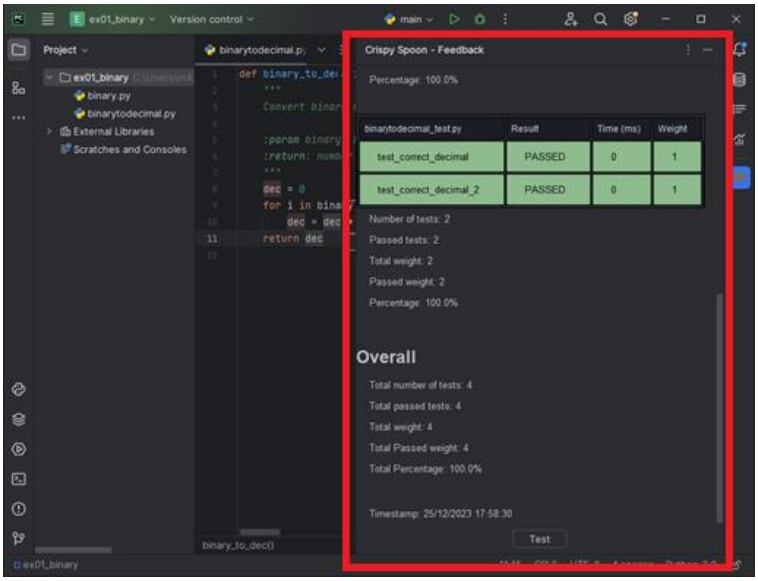
Joonis 23. Ülesande lahendusfaile koondav programmikood.

Lisas 4 on välja toodud ülesande lahendusfailide koondamisele järgnev programmikood. Algatatakse uus korutiin, mille peamiseks eesmärgiks on saata Charoni taga-rakendusele kasutaja poolt muudetud lahendusi sisaldavad failid. Teada saamaks programmeerimis-ülesande identifikaatorit pöördutakse pistikprogrammi „seisundi“ (*state*) poole, kuhu see projekti genereerides koos ülesande kirjeldusega salvestati. Peale testimise nupu peale vajutamist ning enne Charoni taga-rakenduselt vastuse saamist tühistatakse võimalus nupu peale uuesti vajutada ning kasutaja informeerimiseks vahetatakse nupu teksti (Joonis 24).

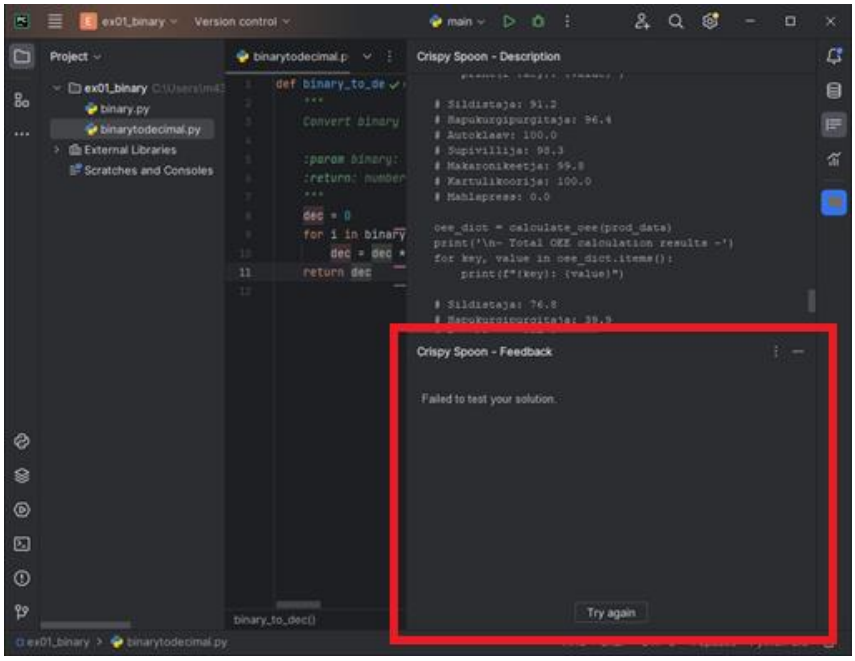



Joonis 24. PyCharm IDE-s projektina avatud ülesanne. Testimise protsess on käimasolev.

Kui kasutaja lahenduse testimine õnnestub, siis kuvatakse automaattesteri väljund tagasiside tööriistaaknas testimise nupu üleval ning testimise nupp taastatakse algsesse olekusse (Joonis 25). Kui aga kasutaja lahenduse testimine mingil põhjusel ebaõnnestus, siis informeeritakse sellest samuti kasutajale tagasiside tööriistaaknas (Joonis 26).




Joonis 25. PyCharm IDE-s projektina avatud ülesanne. Kasutaja lahendus on testitud ning tagasiside on kuvatud. Tagasiside paremaks ülevaateks on ülesande kirjelduse tööriistaaken peidetud.




Joonis 26. PyCharm IDE-s projektina avatud ülesanne. Kasutaja lahenduse testimine ebaõnnestus ning vastav tagasiside on kuvatud.

# 6 Kokkuvõte ja edasiarendus

Valmis saadud funktsionaalsus sai autori poolt ka manuaalselt testitud. Funktsionaalsetes nõuetes 2 kuni 6 kirjeldatud funktsionaalsused töötavad ootuspäraselt. On võimalik programmeerimisülesannet valida, programmeerimisülesannet lahendada, programmeerimisülesande lahendust esitada ja näha testitulemusi. Peale ülesande valimist mall on automaatselt koodiredaktorisse laetud ning kuvatud on ka ülesande kirjeldus.

Algversiooniks plaanitud mittefunktsionaalsed nõuded said ka täidetud: arendatud pistikprogramm töötab IntelliJ platvormi IDE-l (IntelliJ IDEA ja PyCharm) alates 2023. aasta versioonidest, tagasiside automaattestimissüsteemist jõuab kahe minuti jooksul ning pistikprogramm sai Charoni taga-rakendusega liidestatud.

Võib öelda, et töö eesmärk sai enamasti saavutatud. Ainuke samm, mis jäi tegemata on pistikprogrammi avaldamine. Kuna käesoleva töö muudatused on Charonis planeeritud jõustuma alles järgmise väljalaskmisega, siis oleks ennekõike vaja ära oodata vastavad muudatused ning seejärel kindlustada pistikprogrammi suhtlus reaalse Charoni taga-rakendusega.

Üldisemad väljakutsed, mis käesoleva töö jooksul eesmärgini jõudmist takistasid olid töö autorile uued tehnoloogiad, nagu näiteks IntelliJ platvormi pistikprogrammi SDK ning API, Kotlini programmeerimiskeel ning selle korutiinid ja Java Swing, mida IntelliJ platvormi SDK enda jaoks kohandanud oli.

Töö autor plaanib pistikprogrammi ka edasi arendada. Pärast pistikprogrammi esialgset avaldamist tuleks küsida kasutajatelt tagasisidet kinnitamaks käesoleva töö probleemi lahendamist. Kuna algselt on plaanis avaldada pistikprogramm zip-failina, mida peaks kasutaja IDE-sse paigaldama masina kettalt, siis järgmisena tuleks pistikprogramm avaldada JetBrainsi turul (*JetBrains Marketplace*). Turul avalikustatuna saaks pistikprogrammi läbi IDE mugavamalt alla laadida ning paigaldada. Lõpuks tuleks pistikprogrammiga jõuda nii kaugele, nagu esialgne skoop ette nägi ehk kasutaja autentimiseks integreerida pistikprogramm ka Moodle’iga, et esitusi saaks salvestada ning käesoleva töö lahendusest saaks kasu ka tudengid.

Lisaks eelnevalt mainitud edasiarenduse plaanidele on ka muid, kuid nende tähtsuse määrab töö autor teisejärguliseks. Mõned näited nendest plaanidest on pistikprogrammi

kasutajaliidese tõlkimine eesti keelde, tööriistaakende kuvamine ainult siis, kui avatud on pistikprogrammi projekt, originaalsed ikoonid tööriistaakendele, tööriistaakendes hüperlinkide avatavaks muutmine, kirjelduse kuvamine kirjelduse algusest ja kirjelduse kerimise salvestamine, puhas kood (*clean code*) ja ühiktestid.

# Kasutatud kirjandus


[1] J. Juliver, „What Is an API Endpoint? (And Why Are They So Important?),“ HubSpot, Inc., 14 aug. 2023. [Võrgumaterjal]. Available: https://blog.hubspot.com/website/api-endpoint. [Kasutatud 3 jaan. 2024].

[2] J. F. M. Alviste, „Programmeerimisülesannete automaattestimissüsteemi liidestus Moodle'i keskkonnaga ja mugav kasutajaliides tudengite hindamiseks,“ bakalaureusetöö, Infotehnoloogia teaduskond, TalTech, 7 juuni 2017. [Võrgumaterjal]. Available: https://digikogu.taltech.ee/et/Item/b628d504-57e3-4d90-9c60-4bcd6bea3d61. [Kasutatud 2 jaan. 2024].

[3] R. Fitz, „The Mom Test,“ UCL Enterprise, 18 nov. 2014. [Võrgumaterjal]. Available: https://www.youtube.com/watch?v=Hla1jzhan78. [Kasutatud 24 veebr. 2022].

[4] „Problem validation script,“ Board of Innovation, [Võrgumaterjal]. Available: https://www.boardofinnovation.com/tools/problem-validation-script/. [Kasutatud 24 veeb. 2022].

[5] „JetBrains Academy,“ (2023.11-2023.3-421). JetBrains s.r.o., [Võrgumaterjal]. Available: https://plugins.jetbrains.com/plugin/10081-jetbrains-academy. [Kasutatud 7 dets. 2023].

[6] „About Us,“ HackerRank, [Võrgumaterjal]. Available: https://www.hackerrank.com/about-us/. [Kasutatud 30 dets. 2023].

[7] „Required Experience,“ JetBrains s.r.o., [Võrgumaterjal]. Available: https://plugins.jetbrains.com/docs/intellij/plugin-required-experience.html 07. [Kasutatud 7 dets. 2023].

[8] „Creating a Plugin Gradle Project,“ JetBrains s.r.o., [Võrgumaterjal]. Available: https://plugins.jetbrains.com/docs/intellij/creating-plugin-project.html. [Kasutatud 7 dets. 2023].

[9] JetBrains s.r.o., „Template repository for creating plugins for IntelliJ Platform,“ GitHub, Inc., [Võrgumaterjal]. Available: https://github.com/JetBrains/intellij-platform-plugin-template. [Kasutatud 07 dets. 2023].

[10] „Developing a Plugin,“ JetBrains s.r.o., [Võrgumaterjal]. Available: https://plugins.jetbrains.com/docs/intellij/developing-plugins.html. [Kasutatud 18 dets. 2023].

[11] JetBrains s.r.o., „Plugin for parsing and managing the Changelog in a "keep a changelog" style,“ GitHub, Inc., [Võrgumaterjal]. Available: https://github.com/JetBrains/gradle-changelog-plugin. [Kasutatud 7 dets. 2023].

[12] JetBrains s.r.o., „⚙ Scan your Go, Java, Kotlin, PHP, Python, JavaScript, TypeScript, .NET projects at GitHub with Qodana. This repository contains Qodana for Azure, GitHub, CircleCI and Gradle,“ GitHub, Inc., [Võrgumaterjal]. Available: https://github.com/jetbrains/qodana-action. [Kasutatud 7 dets. 2023].

[13] JetBrains s.r.o., „Kotlin/kotlinx-kover,“ GitHub, Inc., [Võrgumaterjal]. Available: https://github.com/Kotlin/kotlinx-kover. [Kasutatud 18 dets. 2023].

[14] JetBrains s.r.o., „The library allows you to write and execute UI tests among IntelliJ IDEA. You can test your Plugin,“ GitHub, Inc., [Võrgumaterjal]. Available: https://github.com/JetBrains/intellij-ui-test-robot. [Kasutatud 7 dets. 2023].

[15] „GitHub Actions documentation,“ Github, Inc., [Võrgumaterjal]. Available: https://github.com/JetBrains/intellij-ui-test-robot. [Kasutatud 7 dets. 2023].

[16] „Plugin Signing,“ JetBrains s.r.o., [Võrgumaterjal]. Available: https://plugins.jetbrains.com/docs/intellij/plugin-signing.html. [Kasutatud 31 dets. 2023].

[17] „Asynchronous programming techniques,“ JetBrains s.r.o., [Võrgumaterjal]. Available: https://kotlinlang.org/docs/async-programming.html. [Kasutatud 7 dets. 2023].

[18] „Coroutines,“ JetBrains s.r.o., [Võrgumaterjal]. Available: https://kotlinlang.org/docs/coroutines-overview.html. [Kasutatud 7 dets. 2023].

[19] „Coroutines basics,“ JetBrains s.r.o., [Võrgumaterjal]. Available: https://kotlinlang.org/docs/coroutines-basics.html. [Kasutatud 18 dets. 2023].

[20] „Coroutine context and dispatchers,“ JetBrains s.r.o., [Võrgumaterjal]. Available: https://kotlinlang.org/docs/coroutine-context-and-dispatchers.html. [Kasutatud 19 dets. 2023].

[21] „Retrofit,“ Square, Inc., [Võrgumaterjal]. Available: https://square.github.io/retrofit/. [Kasutatud 20 dets. 2023].

[22] „Coroutines and channels − tutorial,“ JetBrains s.r.o., [Võrgumaterjal]. Available: https://kotlinlang.org/docs/coroutines-and-channels.html. [Kasutatud 19 dets. 2023].

[23] „Extension Point and Listener List,“ JetBrains s.r.o., [Võrgumaterjal]. Available: https://plugins.jetbrains.com/docs/intellij/extension-point-list.html. [Kasutatud 7 dets. 2023].

[24] „IntelliJ Platform Explorer,“ JetBrains s.r.o., [Võrgumaterjal]. Available: https://plugins.jetbrains.com/intellij-platform-explorer/extensions. [Kasutatud 7 dets. 2023].

[25] JetBrains s.r.o., „Educational plugin to learn and teach programming languages such as Kotlin, Java, Python, JavaScript, and others right inside of JetBrains IntelliJ Platform based IDES,“ GitHub, Inc., [Võrgumaterjal]. Available: https://github.com/JetBrains/educational-plugin. [Kasutatud 7 dets. 2023].

[26] „Dialogs,“ JetBrains s.r.o., [Võrgumaterjal]. Available: https://plugins.jetbrains.com/docs/intellij/dialog-wrapper.html. [Kasutatud 7 dets. 2023].

[27] „Tool Windows,“ JetBrains s.r.o., [Võrgumaterjal]. Available: https://plugins.jetbrains.com/docs/intellij/tool-windows.html. [Kasutatud 7 dets. 2023].

[28] JetBrains s.r.o., „Mirror of the IntelliJ SDK Docs Code Samples,“ GitHub, Inc., [Võrgumaterjal]. Available: https://github.com/JetBrains/intellij-sdk-code-samples. [Kasutatud 7 dets. 2023].

[29] „Icons list,“ JetBrains s.r.o., [Võrgumaterjal]. Available: https://jetbrains.design/intellij/resources/icons_list/. [Kasutatud 7 dets. 2023].

[30] „Actions,“ JetBrains s.r.o., [Võrgumaterjal]. Available: https://plugins.jetbrains.com/docs/intellij/basic-action-system.html. [Kasutatud 7 dets. 2023].

[31] „Moodle APIs: xmldb_field Class Reference,“ Doxygen, 14 nov. 2020. [Võrgumaterjal]. Available:

https://wimski.org/api/3.8/d2/d26/classxmldb__field.html. [Kasutatud 7 dets. 2023].

[32] „Services,“ JetBrains s.r.o., [Võrgumaterjal]. Available: https://plugins.jetbrains.com/docs/intellij/plugin-services.html. [Kasutatud 22 dets. 2023].

[33] „Key Topics,“ JetBrains s.r.o., [Võrgumaterjal]. Available: https://plugins.jetbrains.com/docs/intellij/key-topics.html. [Kasutatud 24 dets. 2023].

[34] „Virtual File System,“ JetBrains s.r.o., [Võrgumaterjal]. Available: https://plugins.jetbrains.com/docs/intellij/virtual-file-system.html. [Kasutatud 24 dets. 2023].

# Lisa 1 – Lihtlitsents lõputöö reprodutseerimiseks ja lõputöö üldsusele kättesaadavaks tegemiseks[1]

Mina, Sten Kalev

1. Annan Tallinna Tehnikaülikoolile tasuta loa (lihtlitsentsi) enda loodud teose „IntelliJ platvormil põhinev pistikprogramm programmeerimisülesannete lahendamiseks“, mille juhendaja on Bahdan Yanovich
   1.1. reprodutseerimiseks lõputöö säilitamise ja elektroonse avaldamise eesmärgil, sh Tallinna Tehnikaülikooli raamatukogu digikogusse lisamise eesmärgil kuni autoriõiguse kehtivuse tähtaja lõppemiseni;
   1.2. üldsusele kättesaadavaks tegemiseks Tallinna Tehnikaülikooli veebikeskkonna kaudu, sealhulgas Tallinna Tehnikaülikooli raamatukogu digikogu kaudu kuni autoriõiguse kehtivuse tähtaja lõppemiseni.
2. Olen teadlik, et käesoleva lihtlitsentsi punktis 1 nimetatud õigused jäävad alles ka autorile.
3. Kinnitan, et lihtlitsentsi andmisega ei rikuta teiste isikute intellektuaalomandi ega isikuandmete kaitse seadusest ning muudest õigusaktidest tulenevaid õigusi. 04.01.2024

1 Lihtlitsents ei kehti juurdepääsupiirangu kehtivuse ajal vastavalt üliõpilase taotlusele lõputööle juurdepääsupiirangu kehtestamiseks, mis on allkirjastatud teaduskonna dekaani poolt, välja arvatud ülikooli õigus lõputööd reprodutseerida üksnes säilitamise eesmärgil. Kui lõputöö on loonud kaks või enam isikut oma ühise loomingulise tegevusega ning lõputöö kaas- või ühisautor(id) ei ole andnud lõputööd kaitsvale üliõpilasele kindlaksmääratud tähtajaks nõusolekut lõputöö reprodutseerimiseks ja avalikustamiseks vastavalt lihtlitsentsi punktidele 1.1. ja 1.2, siis lihtlitsents nimetatud tähtaja jooksul ei kehti.

# Lisa 2 – Arenduskeskkonna projektivaates pistikprogrammi kirjelduse akent realiseeriv programmikood

```
class DescriptionToolWindowFactory : ToolWindowFactory {

    override fun createToolWindowContent(
        @NotNull project: Project,
        @NotNull toolWindow: ToolWindow
    ) {
        toolWindow.stripeTitle =
            MyBundle.message("descriptionToolWindowTitle")
        toolWindow.setIcon(AllIcons.Gutter.JavadocRead)
        val myToolWindow = DescriptionToolWindow(project)
        val content = ContentFactory
            .getInstance()
            .createContent(myToolWindow.getContent(), null, false)
        toolWindow.contentManager.addContent(content)
    }

    class DescriptionToolWindow(project: Project) {

        private val contentPanel = JPanel(BorderLayout())

        init {
            contentPanel.add(
                JBScrollPane(JTextPane()),
                BorderLayout.CENTER
            )
        }

        fun getContent(): JComponent = contentPanel
    }
}
```

# Lisa 3 – Ülesande valimise akent realiseeriv ning ülesandest IntelliJ projekti genereeriv programmikood

```
override fun actionPerformed(@NotNull e: AnActionEvent) {
    val dialog = BrowseAssignmentsDialog()
    if (dialog.showAndGet() && dialog.assignment != null) {
        val assignment = dialog.assignment
        val generator = ProjectGenerator(assignment!!)
        val project = generator.createAssignmentProject(
            "${ProjectUtil.getBaseDir()}/${assignment.name}"
        )

        if (project != null) {
            val state = MyPersistentState.getInstance(project).state
            state.trustedProject = true
            state.assignmentDescription = assignment.description

            val applicationService =
                project.getService(MyApplicationService::class.java)

            launch {
                val response = applicationService
                    .getCharonService()
                    .getCharon(assignment.id)
                    .body()

                if (response != null) {
                    state.assignmentId = response.id
                    state.assignmentName = response.name

                    val templates = response.templates
                    for (template in templates) {
                        val templateFile =
                            File("${project.basePath}/${template.path}")
                        FileUtil.createIfDoesntExist(templateFile)
                        FileUtil.writeToFile(templateFile, template.content)
                    }
                }
            }

            val twm = ToolWindowManager.getInstance(project)
            val descTw = twm.getToolWindow(DescriptionToolWindowFactory.ID)
                ?: error("description tool window not found")
            val fbTw = twm.getToolWindow(FeedbackToolWindowFactory.ID)
                ?: error("feedback tool window not found")
            descTw.show()
            fbTw.show()

        } else {
            error("could not create project")
        }
    }
}
```

# Lisa 4 – Ülesande lahendust esitav programmikood

```
    val appService = ApplicationManager
        .getApplication()
        .getService(MyApplicationService::class.java)
    val state = MyPersistentState.getInstance(project).state

    launch {

        button.isEnabled = false
        button.text = "${MyBundle.message("testingLabel")}..."

        val submission = Submission(templates)
        var windowText: String?
        var buttonText = MyBundle.message("testLabel")

        try {
            val response = appService
                .getCharonService()
                .testSubmission(state.assignmentId!!, submission)
            windowText = response.body()?.output
        } catch (e: Exception) {
            windowText =
                "<p>${MyBundle.message("failedToTestSolution")}.</p>"
            buttonText = MyBundle.message("tryAgainLabel")
            e.printStackTrace()
        }

        feedbackTextPane.text = windowText
        button.isEnabled = true
        button.text = buttonText
    }
}
```