TALLINNA TEHNIKAÜLIKOOL

Infotehnoloogia teaduskond

Kirill Denisov 164208IAPB

KOODI KIRJUTAMISE HARJUTAMISPLATVORMI ARENDAMINE

Bakalaureusetöö

Juhendaja: Ago Luberg,

tarkvarateaduse instituut, MSc

Tallinn 2019

Autorideklaratsioon

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

Autor: Kirill Denisov

21.05.2019

Annotatsioon

Enamik programmeerimise aineid algab kõigil tudengitel sarnaselt. Enne ülesannete lahendamist peavad nad töökeskkonna seadistama, paigaldades vajalikku tarkvara. Kahjuks tekitab see paljudele tudengitele segadust. Lisaks soovib selliste ainete õppejõud jagada tudengitele täiendavaid ülesandeid harjutamiseks, kuid õppejõu jaoks on mugavam neid tavalistest ülesannetest eraldi hoida.

Käesoleva töö esimeseks eesmärgiks on luua koodi kirjutamise harjutamisplatvorm, mida saavad tudengid kasutada, spetsiifilist tarkvara kasutamata. Teiseks eesmärgiks on teha see platvorm kõigile TalTech'i tudengitele ja töötajatele kättesaadavaks. See platvorm lihtsustab programmeerimise ainete korraldamisega seotud aspekte ning lahendab eelnevalt väljatoodud probleeme.

Platvorm on arendatud klient-server arhitektuuri järgi, kasutades Spring Boot ja Vue.js raamistikke, PostgreSQL andmebaasi ning Dockerit. Uute versioonide serverisse paigaldamise protsess on automatiseeritud Gitlabi tööriistade kasutusega.

Töö tulemuseks on koodi kirjutamise platvorm, mis on kättesaadav aadressil codera.cs.ttu.ee. Sisselogimiseks on vaja TalTech'i UNI-ID'd.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 33 leheküljel, 6 peatükki ja 21 joonist.

Abstract

Development of a Code Writing Practice Platform

Most programming courses start similarly for every student. In order to start doing exercises, they have to set up a development environment by downloading and installing necessary software. Many students find such practices confusing; it makes them feel helpless. Furthermore, professors want to provide students extracurricular exercises to practice, however, they find it convenient to keep them separately from mandatory work.

First goal of the thesis is to create a code writing practice platform which can be used by students without the need to install additional software on their computers. Second goal is to make the platform available for all students and employees of TalTech. This platform is expected to simplify considerable aspects of programming courses organization and solve problems mentioned above.

The platform is designed corresponding to Client/Server architecture and uses Spring Boot and Vue.js frameworks, PostgreSQL database and Docker. The process of new versions deployment is automated using Gitlab tools.

The platform is now available on the website codera.cs.ttu.ee for all students and employees of TalTech. To log in, TalTech UNI-ID is required.

The thesis is in Estonian and contains 33 pages of text, 6 chapters and 21 figures.

Sisukord

Jooniste loetelu

Lühendite ja mõistete sõnastik

API	*Application programming interface.* Protokoll rakenduste omavaheliseks suhtlemiseks.
CSS	*Cascading Style Sheets.* Keel veebilehtede kujunduse kirjeldamiseks.
ES	*ECMAScript.* JavaScripti standard.
Git	Versioonihaldustarkvara, mis lihtsustab koodi muudatuste haldamist.
HTML	*Hypertext Markup Language.* Keel veebilehtede märgendamiseks.
HTTP	*Hypertext Transfer Protocol.* Veebiprotokoll andmete edastamiseks üle võrgu.
IDE	*Integrated development environment.* Keskkond tarkvara arendamiseks.
Java EE	*Java Enterprise Edition.* Laiendatud JDK versioon veebirakenduste loomiseks.
JDBC	*Java Database Connectivity.* API andmebaasiga suhtlemise lihtsustamiseks.
JDK	*Java Development Kit.* Tööriistad Java rakenduste arendamiseks.
JPA	*Java Persistence API.* Java rakenduse ja andmebaasi vahelise liidese kirjeldus.
JPQL	*Java Persistence Query Language.* Keel olemitega manipuleerimiseks, mida teisendatakse SQL päringuks.
JSON	*JavaScript Object Notation.* Andmete esitamise formaat.
JWT	*JSON Web Token.* Autentimise lubade standard. Sellised load sisaldavad kasutaja andmeid JSON kujul.

REST	*Representational state transfer*. Veebiteenuste arhitektuuri stiil.
SQL	*Structured Query Language*. Päringukeel relatsiooniliste andmebaasidega suhtlemiseks.
SRP	*Single Responsibility Principle*. Printsiip, mille järgi peab klass tegema vaid ühte asja.
TalTech	Tallinna Tehnikaülikooli nime ametlik lühend.
trafarett-kood	(*boilerplate code*). Kood, millel ei ole reaalset tähendust, aga mida tuleb korduvalt kirjutada.
UNI-ID	TalTech'i tudengi või töötaja unikaalne identifikaator.
URL	*Uniform Resource Locator*. Aadress, mis viitab ressursile veebis.
XML	*Extensible Markup Language*. Andmete esitamise formaat.
YAML	Andmete esitamise formaat.

1 Sissejuhatus

Iga programmeerimise aine algab sellega, et enne ülesannete lahendamist peavad tudengid kõigepealt töökeskkonna endale seadistama, tõmmates alla ning paigaldades oma arvutisse vajalikke tööriistu.

See on näiteks suureks probleemiks Programmeerimise algkursuse (ITI0101) aines, mida õpetatakse Informaatika esimese kursuse tudengitele. Vaatamata sellele, et tudengitele antakse töökeskkonna seadistamise juhend, tekib neil ikkagi probleeme. See ehmatab nad ära ning tekitab tunde, et nad juba aine alguses ei saa hakkama. Kahjuks ei ole võimalik seda protsessi lükata hilisemaks, kuna tudengitele antakse ülesandeid lahendada juba alates semestri esimesest nädalast. Lisaks on selle kursuse õppejõul soov jagada tudengitele täiendavaid ülesandeid harjutamiseks, kuid nende ülesannete hoidmine kursusel kasutatavas Moodle'i keskkonnas on ebamugav nii õppejõule kui ka tudengitele. Samuti on hea, kui on võimalik kohustuslikke ja lisaülesandeid hoida lahus.

Käesoleva töö üheks eesmärgiks on luua koodi kirjutamise platvorm, mida saavad tudengid programmeerimisainetes kasutada ülesannete lahendamiseks ilma, et nad peavad spetsiifilist tarkvara oma arvutisse paigaldama. See platvorm ei asenda olemasolevat Moodle'i keskkonda, vaid hakkab sellega paralleelselt eksisteerima, andes nii õppejõule kui ka tudengitele võimaluse kursust rahulikult alustada, kulutamata liigset aega töökeskkonna seadistamisele. Lisaks saavad tudengid hiljem seda platvormi oma oskuste harjutamiseks kasutada.

Selle töö teiseks eesmärgiks on teha see platvorm kättesaadavaks kõigile TalTech'i tudengitele ning töötajatele, et seda saaks kasutama hakata juba järgmisel õppeaastal toimuvatel kursustel. Antud töö struktuur koosneb järgmistest osadest: kõigepealt tehakse taustauuring, kus tutvutakse maailmas eksisteerivate lahendustega ettekujutuse loomiseks. Seejärel tutvustatakse nõudeid platvormile ja peale seda kirjeldatakse platvormi arhitektuuri ja arendamisel kasutatud tehnoloogiaid. Viimasena räägitakse rakenduse publitseerimisest: milliseid tehnoloogiaid selleks kasutatakse ja kuidas muudatused toodangusse jõuavad.

2 Taustauuring

Selles peatükis võrreldakse erinevaid koodi kirjutamise platvorme ja tuuakse välja nende positiivsed ja negatiivsed küljed.

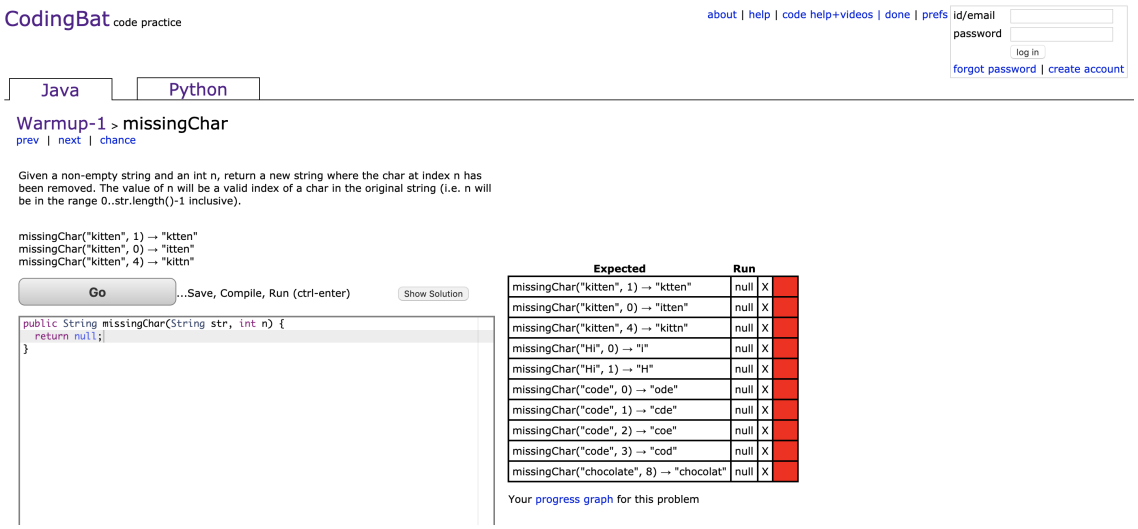
2.1 CodingBat

Varem mainitud Programeerimise algkursusel pakutakse tudengitele CodingBat [1] harjutamisplatvormi. Järgmisena on välja toodud selle platvormi eelised:

- Lihtne ja arusaadav disain.
- Ülesanded on jagatud kategooriateks.
- Võimalus luua endale konto, et hiljem oma varasemaid esitusi vaadata.
- Hea tagasiside tudengitele.

CodingBati puudus on see, et sinna ülesandeid enam juurde ei lisata.

Joonis 1 näitab CodingBati kasutajaliidest.

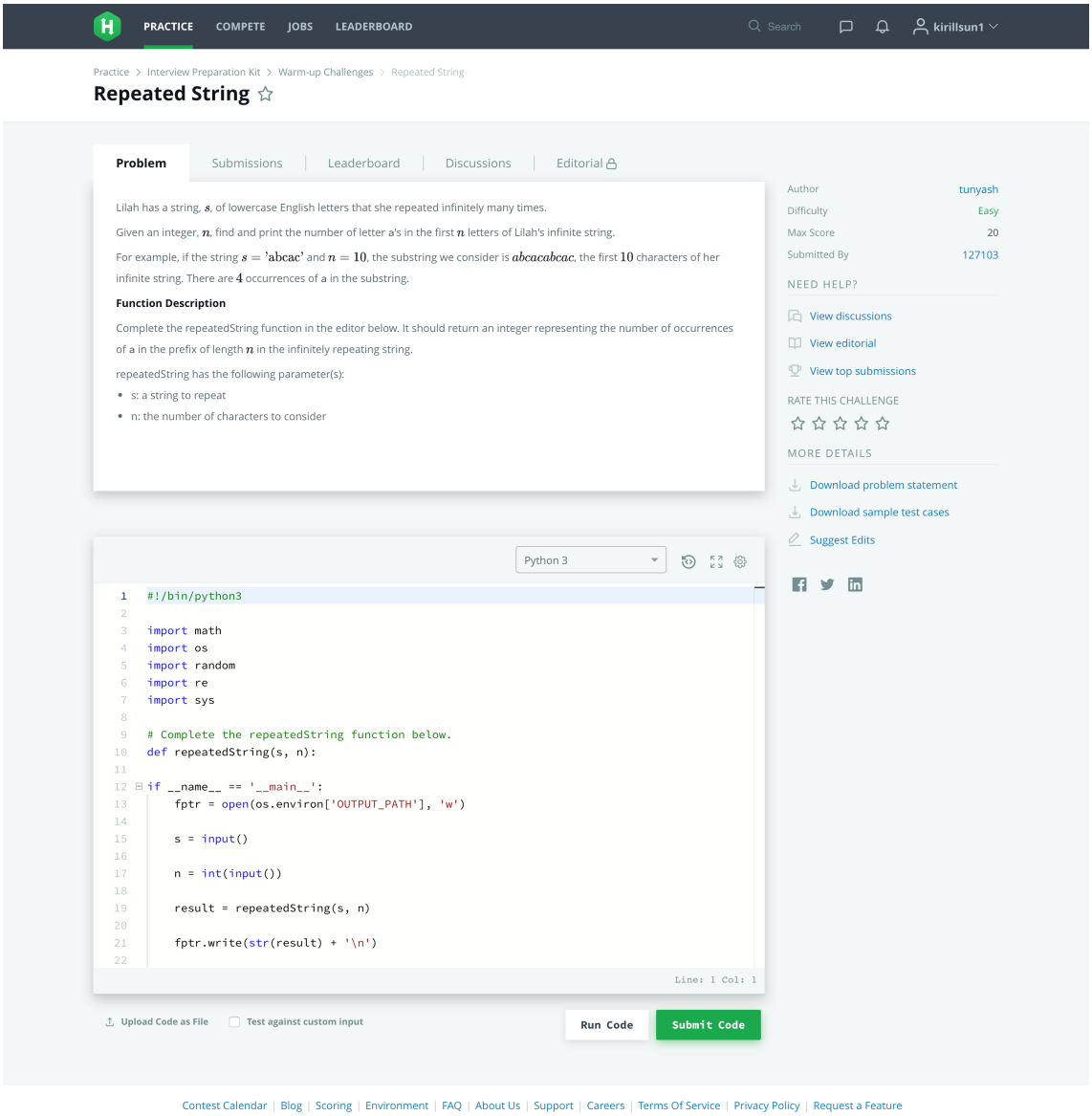


Joonis 1: CodingBati kasutajaliides

[1] https://codingbat.com

2.2 HackerRank

HackerRank [1] on CodingBatile sarnane koodi kirjutamise platvorm. See likvideerib Codingbati puuduse: seal on rohkem ülesandeid ning neid lisatakse aja jooksul juurde. HackerRanki ülesanded on jagatud kategooriatesse ja igas kategoorias on kolme keerukustasemega ülesandeid: lihtsad, keskmised ja rasked. HackerRank võimaldab samamoodi luua endale konto, mis annab võimaluse koodi salvestada ja hiljem vaadata. HackerRanki kasutajad saavad omavahel võistelda ülesannete lahendamises, kus hinnatakse nende lahenduste keerukusi. Joonis 2 näitab HackerRanki kasutajaliidest.



Joonis 2: HackerRanki kasutajaliides

[1] https://hackerrank.com

2.3 TalTech'i vilistlase panus

2018. aastal tegi Deniss Potapenko oma bakalaureusetöö raames sarnase platvormi. [11] Tema lahenduses on realiseeritud ülesannete andmebaasist lugemine ning koodi testimine TalTech'i testri kasutamisega. Kuid tema lahenduses on üks tehniline puudus: osa äriloogikast on realiseeritud klientrakenduses. Tema lahendus oli pigem prototüüp, seepärast tehti otsus keskenduda uue platvormi arendamisele, aga selle käigus kindlasti võetakse arvesse Deniss Potapenko töö tulemust ja kogemust.

2.4 Muud platvormid

Koodi kirjutamise platvorme on tänapäeval päris palju ja igaühel on oma huvitavad aspektid. Järgmisena on välja toodud mõned teised koodi kirjutamise platvormid:

- Codecademy [1]
  - Õpetab programmeerimise aluseid. Visualiseerib koodi.
- Hackr.io [2]
  - Ülesanded edasijõudnutele. Õpetab ka keelte raamistikke.
- Exercism [3]
  - Palju huvitavaid algoritmilisi ülesandeid, mida saab lahendada soovitud keeles.
- Codingame [4]
  - Mänguline lähenemine: kasutaja koodist sõltuvad tegevused mängus. Lai keelte valik.

2.5 Kokkuvõte

Vaatamata mõnede platvormide funktsionaalsusele ja suurepärasele kvaliteedile, on nendel TalTech'i perspektiivist üks väga suur puudus: TalTech'i õppejõud ei saa

[1] https://codecademy.com
[2] https://hackr.io
[3] https://exercism.io
[4] https://codingame.com

sinna ülesandeid lisada, tudengite koodi vaadata ega hinnata. Seepärast võeti selle töö eesmärgiks luua TalTech'ile platvorm, mis lahendab eespool mainitud probleemi. Järgmisena on välja toodud mainitud platvormide eelised, mida oleks hea uues platvormis realiseerida:

- Jagada ülesandeid kategooriateks ja raskustaseme järgi.
- Võimalus UNI-ID-ga sisse logida, et ülesande lahendused salvestuks süsteemi.
- Võimalus oma varasemaid lahendusi vaadata.

3 Analüüs

Selles peatükis räägitakse süsteemi funktsionaalsetest ja mittefunktsionaalsetest nõuetest. Funktsionaalsed nõuded peavad selgelt kirjeldama, mida süsteem peab oskama ehk milliseid funktsioone peab antud süsteem realiseerima. Mittefunktsionaalsed nõuded kirjeldavad hoopis seda, kuidas süsteem peab oma funktsioone täitma. [5]

3.1 Rollid süsteemis

Antud arendusetapis piisab, et süsteemis oleks kolm rolli: külastaja, kasutaja ning administraator. Oleks ka hea, kui süsteemis oleks õppejõu roll, kuid seda antud töö raames ei realiseerita. Valitud kolm rolli peavad olema hierarhias, kus igal kõrgemal rollil on olemas kõigi madalamate rollide õigused. Ehk kasutaja saab teha kõike seda, mida saab teha külastaja ning administraator saab teha kõike, mida saavad teha külastaja ning kasutaja.

3.2 Funktsionaalsed nõuded

Järgmisena on välja toodud funktsionaalsed nõuded.

- Süsteemis saab vaadata ülesandeid, mis sisaldavad nime, kirjeldust ning malli, milles saab olla vähemalt üks lähtekoodi fail.
- Süsteemis saab vaadata ülesannete kategooriaid ning nendes sisalduvaid ülesandeid.
- Üks ülesanne saab kuuluda ühte või mitmesse kategooriasse.
- Administraator saab ülesandeid luua.
- Administraator saab kategooriaid luua.
- Administraator saab ülesandeid kategooriasse määrata.
- Administraator saab kategooriaid kasutajate ning külastajate eest ära peita.
- Administraator saab kategooriates olevatele ülesannetele silte (*tag*) määrata.
- Külastaja saab avalikele kategooriatele ja nendes olevatele ülesannetele ligi.
- Kasutaja saab ülesandeid lahendada.

- Kasutaja saab ülesande lahendamisel automaatset tagasisidet.
- Kasutaja saab näha oma varasemaid esitusi.
- Administraator saab kõikide kasutajate lahendusi vaadata.

Lisaks on mõned nõuete ideed, mis ei ole lõpuni analüüsitud, kuna antud töö neid ei kata. Need nõuded puudutavad õppejõu rolli, mida antud töös ei realiseerita.

- Ülesannet saab jagada sõltuvateks osadeks.
- Ülesannet saab lahendada mitmes programmeerimiskeeles.
- Administraator saab kategooriale õppejõudu määrata.
- Õppejõud saab hallata talle määratud kategooriaid (muuta nime ja kirjeldust, lisada, muuta ja kustutada ülesandeid ning vaadata tudengite lahendusi selle kategooria piires).
- Õppejõud saab piirata ligipääsu kategooriale vaid mõnedele kasutajatele.

3.3 Mittefunktsionaalsed nõuded

Järgmisena on välja toodud ja lahti seletatud mittefunktsionaalsed nõuded arendatavale süsteemile:

- Kättesaadavus: platvorm peab olema suurem osa ajast kõigile kättesaadav. Lühikesed katkestused süsteemi töös on lubatud.
- Turvalisus: kasutajad ei tohi näha teiste kasutajate andmeid (v.a. administrator, kes saab vaadata kõigi kasutajate andmeid).
- Süsteem peab võimalikult palju kasutama eksisteerivaid teeke, raamistikke ja tehnoloogiaid, et lihsustada arendamist.
- Platvorm peab olema realiseeritud veebirakendusena, et kasutaja ei peaks lisatarkvara oma arvutisse paigaldama.
- Veebirakendus peab toetama ning samahästi töötama kaasaegsetes populaarsetes brauserites (Google Chrome, Mozilla Firefox, Microsoft Edge, Safari).

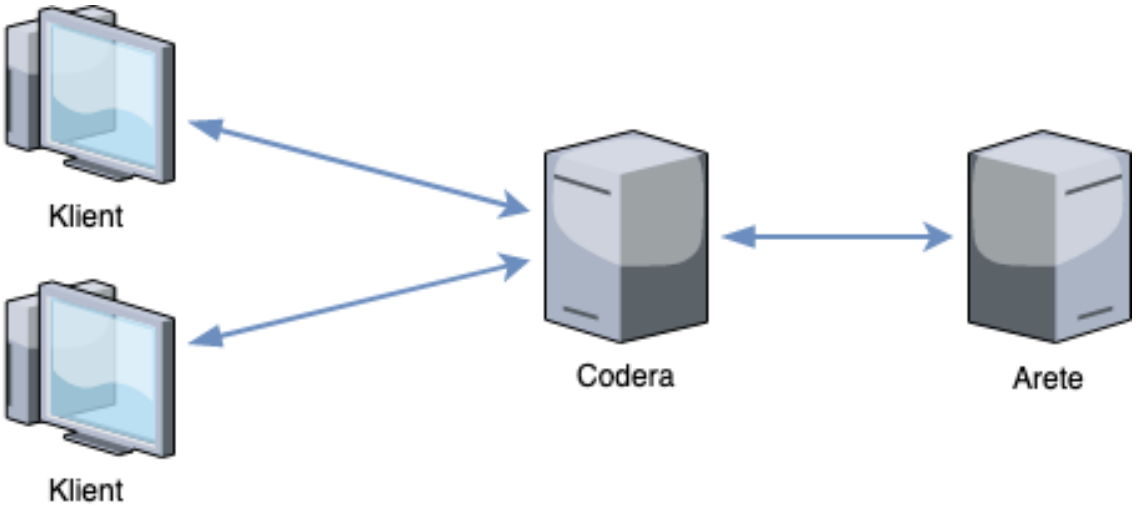
4 Arendus

Selles peatükis kirjeldatakse arendatava platvormi (edaspidi Codera) arhitektuuri ning kasutatud tehnoloogiaid.

4.1 Arhitektuur

Süsteemi arhitektuur annab kõrgtasemelise ettekujutuse, millistest komponentidest süsteem koosneb ning kuidas nad on omavahel seotud. Codera on realiseeritud klient-server arhitektuurimustri järgi, mis tähendab, et on olemas tsentraalne server, mis teenindab kliente vastavalt nende päringutele. See on standardne arhitektuurimuster veebirakenduste jaoks. [10] Kliendiks saab lugeda kasutaja brauseris jooksutatavat rakendust, mis suhtleb serveriga üle võrgu.

Codera arhitektuuri kuulub ka TalTech'i poolt arendatud tester Arete, mida kasutatakse koodi testimiseks. Codera saab kasutada Arete poolt pakutavat REST API-t oma funktsionaalsuse realiseerimiseks. Joonis 3 ilustreerib Codera arhitektuuri.



Joonis 3: Codera arhitektuur

4.2 Serverirakendus

Codera server kujutab endast veebiteenust, mille eesmärk on töödelda, hoida ning näidata kliendile andmeid. Erinevalt Deniss Potapenko prototüübist, kus server oli

ainult ülesannete andmete hoidmiseks, on kogu Codera äriloogika (kaasa arvatud Arete'ga suhtlemine) realiseeritud serveri peal. Andmete vahetamiseks pakub Codera server REST API liidest, mida tarbib klientrakendus. Järgmine alampeatükk annab ülevaate REST arhitektuuri stiilist.

4.2.1 Representational State Transfer

REST (*Representational State Transfer*) on viimastel aastatel populaarseks saanud arhitektuuri stiil, mis kirjeldab reegleid veebiteenuste arendamiseks. REST arhitektuuril põhinevaid veebiteenuseid nimetatakse RESTful veebiteenusteks. RESTful veebiteenuse keskel asub ressurss ja kõiki tegevusi saab kirjeldada ressursside manipuleerimisena. Sellise veebiteenusega saab suhelda HTTP protokolli kaudu, selleks on vaja teada ressursi URL-i ehk lõpp-punkti. Tegevus sõltub päringu meetodist. Kasutatakse 4 meetodit ressursside manipuleerimiseks: `GET` (ressursi saamiseks), `POST` (ressursi lisamiseks), `PUT` (ressursi uuendamiseks) ja `DELETE` (ressursi kustutamiseks). Vastuseks saab klient HTTP koodi (nt. 200, kui päring õnnestus) ja objekti, mis asub vastuse kehas. Objekt võib sõltuvalt operatsioonist puududa. [12]

Näiteks Coderas on olemas ressurss „Ülesanne" (*Exercise*) ja ülesannete käsitlemiseks on olemas järgmised lõpp-punktid:

- `GET /exercises`
  - Tagastab kõik ülesanded.
- `GET /exercises/EX02String`
  - Leiab üles ja tagastab ülesande, mille ID on „EX02String". Kui ülesannet ei leidu, tagastab vastuse 404 (*Not found*).
- `POST /exercises`
  - Lisab uue ülesande. Ülesanne peab olema kirjeldatud JSON formaadis päringu kehas.
- `PUT /exercise/EX02String`
  - Muudab ülesannet ID-ga „EX02String". Ülesande väljad (v.a. ID) peavad olema kirjeldatud JSON formaadis päringu kehas. Kui ülesannet ei leidu, siis see luuakse ja tagastatakse vastus 201 (*Created*).

- `DELETE /exercise/EX02String`
  - Kustutab ülesande, mille ID on „EX02String“. Kui ülesannet ei leidu, tagastab vastuse 404.

REST arhitektuuri baasil arendatud rakendused ei kasuta sessioone ehk ei jäta kunagi kasutaja seisundit meelde. [12] Kliendi seisundi hoidmise eest vastutavad kliendid ise ning nad peavad iga päringuga saatma vajalikke seisundi andmeid kaasa.

4.3 Serverirakenduse tehnoloogiad

4.3.1 Java

Java [1] on Sun Microsystems firma poolt arendatud programmeerimiskeel, mis on TIOBE [2] edetabeli järgi kõige populaarsem keel 2019. aasta seisuga. Seda keelt iseloomustavad objektorienteeritud koodi kirjutamine, staatilised tüübid, automaatne mäluhaldus ning platvormist sõltumatus. Java koodi kirjutatakse `.java` laiendiga failidesse, kompileeritakse Java kompilaatoriga baitkoodiks ning jooksutatakse Java virtuaalmasinas. Tänapaeval arendab Java keelt Oracle Corporation. Arendajate jaoks antakse välja JDK (*Java Development Kit*), mis sisaldab tööriistu rakenduste loomiseks. Täna päeva seisuga on viimane versioon 12, mis anti välja märtsis 2019. Codera kasutab versiooni 11, mida toetatakse 2023. aasta septembrini. [3]

4.3.2 Gradle

Gradle [3] on avatud lähtekoodiga kooste automatiseerimise vahend. Gradle'i ülesanded on alla tõmmata koodis kasutatavaid teeke, jooksutada teste ning koostada lähtekoodist programmi jooksutavad failid. Teeke tõmbab Gradle alla Maven repositooriumist projektis oleva konfiguratsioonifaili järgi. Gradle'i konfiguratsioonifailide kirjeldamiseks kasutatakse Groovy keelt, mis on üks keeltest, mida kompileeritakse Java baitkoodiks ning jooksutatakse Java

[1] `https://www.java.com/en/`
[2] `https://www.tiobe.com/tiobe-index/`
[3] `https://docs.gradle.org/current/userguide/what_is_gradle.html`

virtuaalmasinas. Gradle'i kõige populaarsemaks konkurendiks on Apache Maven [1]. Kuigi ülesanded on mõlemal samad, on nende töötamise viisid erinevad. Gradle'i protsess koosneb väikestest ülesannetest (*task*). Tänu Gradle'i paindlikkusele saab kasutaja soovi korral oma ülesandeid defineerida. [16] Joonis 4 illustreerib näidet Gradle'i ülesandest.

```
task updateNodeSassBindings(type: Exec) {
    commandLine "npm", "rebuild", "node-sass"
}
```

Joonis 4: Gradle ülesanne

Maveni protsess omakord koosneb fikseeritud faasidest (*phase*). Faas Mavenis täidab sama funktsiooni nagu ülesanne Gradle'is, kuid kasutaja ei saa oma faase defineerida. Laiendamiseks saab luua nn eesmärke (*goals*), kuid see on keerulisem, kuna tuleb eraldi pistikprogramm arendada. [4]

4.3.3 Spring Framework ja Spring Boot

Spring Framework [2] on avatud lähtekoodiga Java raamistik, mis pakub palju tööriistu rakenduste arendamise lihtsustamiseks ja kiirendamiseks, vähendades trafarett-koodi hulka. Spring Framework on ehitatud Java EE platvormi põhjal ja sobib veebirakenduste arendamiseks. Vajadusel saab funktsionaalsuse laiendamiseks lisada lisamooduleid. [9]

Spring raamistiku vundamendiks on sõltuvuse sisestamine (*Dependency Injection*). [9] Sõltuvuse sisestamine on disainimuster, mis nõuab, et kui klassi A loogika realiseerimiseks tuleb klassi B loogikat kasutada, siis klass A peab loogika duplitseerimise asemel kasutama klassi B objekti enda sees, mida süstitakse klassi A objekti sisse. [8] See teeb koodi taaskasutatavaks ja lihtsustab testimist. Aga see nõuab, et klassid oleks võimalikult väikesed ja jälgiks SRP printsiipi. Spring raamistiku üheks osaks on IoC konteiner, kuhu Spring loob ja paneb nn ube (*beans*). Springi uba on sisuliselt Java objekt, aga seda haldab Spring raamistik ise. Selle objekti külge pannakse metaandmed, mis objekti olekut täpsemalt kirjeldavad.

[1] https://maven.apache.org/
[2] https://spring.io/projects/spring-framework

Lisaks pakub Spring Framework keskkonna profiilide funktsionaalsust, mida kasutatakse Coderas. Profiilid on vajalikud, et toetada erinevaid konfiguratsioone erinevate keskkondade jaoks. Vaikimisi kasutab Spring Framework vaikeprofiili (*default profile*). Coderas on vaikeprofiil seadistatud lokaalse arendamise jaoks. Toodangus aga kasutatakse „prod“ profiili.

Coderas ei kasutata puhast Spring raamistikku, vaid Spring Booti. Spring Boot on laiendus Spring raamistikule, mis sisaldab lisakonfiguratsioone, et arendajal oleks võimalus kohe rakendust arendama hakata, mõtlemata konfigureerimisele. Spring Boot arendajad on loonud mitu nn *starter*'it, mis on vajalike raamistike kogumikud. Näiteks, `spring-boot-starter-web` on *starter*, mis sisaldab endas Tomcat serverit, Jackson raamistikku JSON-iga opereerimiseks ning Spring Web ja Spring WebMvc raamistikke, mille sees on vajalikud Java klassid ja liidesed veebikihi arendamiseks.

4.3.4 Andmebaasisüsteemid

Codera serverirakendus kasutab andmete hoidmiseks andmebaasi. Lisaks ilmsetele nõuetele nagu „võimalus hoida andmeid“, peab valitud andmebaasisüsteem vastama järgmistele nõuetele:

- Peab toetama mitut samaaegset kirjutamist.
- Saab määrata andmetele kitsendusi.
- Ei tohi andmeid kaotada.
- On tasuta.

Coderas kasutatakse andmete hoidmiseks kahte andmebaasisüsteemi. Esimene on PostgreSQL [1], mida kasutatakse toodangus. PostgreSQL on avatud lähtekoodiga relatsiooniline andmebaas, mis suures mahus realiseerib SQL:2011 standardit, kaasa arvatud `CHECK` ja `UNIQUE KEY` kitsendusi, indekseid ja transaktsioone, mida kasutatakse Coderas. Kuid arendaja arvutis on keeruline PostgreSQL andmebaasisüsteemi käima panna, seepärast kasutatakse arendamiseks hoopis H2 [2] andmebaasisüsteemi. H2 andmebaasisüsteem on lisatud sõltuvusena Codera

[1] `https://www.postgresql.org/`
[2] `https://www.h2database.com/html/main.html`

serverirakenduses, mis tähendab, et ei ole vaja midagi täiendavat arvutisse paigaldada. Lisaks on võimalik kasutada H2 andmebaasisüsteemi mälupõhiselt. Siis ei tekita andmebaasisüsteem arvutisse faile, vaid hoiab andmeid arvuti muutmälus. Peale rakenduse kinnipanemist kustutatakse andmed muutmälust ära. Coderas kasutatakse seda varianti integratsioonitestides, kus on vaja andmebaasi kasutada.

4.3.5 Spring Data JPA

Spring Data JPA [1] on teek, mis abstraheerib andmebaasiga suhtlemist. Tavaliselt kasutatakse andmebaasiga suhtlemisel nn repositooriumi mustrit (*Repository Pattern*), mille järgi pannakse kõik andmebaasi poole pöördumise loogika eraldi klassi. Ilma Spring Data JPA teegita peab arendaja ise realiseerima repositooriumi meetodeid, mis võib olla tüütü töö, kuna tuleb kirjutada palju trafarett-koodi. Loomulikult saab kasutada mõnda tehnoloogiat, mis teeb andmebaasiga suhtlemist lihtsamaks (nt. JDBC), kuid Spring Data JPA teeb seda veelgi lihtsamaks. [6] Nüüd peab arendaja vaid ühe liidese looma ja panema ta ühte Spring Data JPA repositooriumitest laiendama. Üks sellistest repositooriumitest on JpaRepository. Joonis 5 illustreerib näidet JpaRepository kasutamisest Coderas.

```
@Repository
public interface UserRepository extends JpaRepository<User, String> {

}
```

Joonis 5: JpaRepository kasutamine

Selle liidese implementatsiooni loob Spring Framework ise programmi käivitamise ajal. JpaRepository'is on defineeritud meetodid olemi (*entity*) salvestamiseks, lugemiseks ning kustutamiseks. Kui on vaja mõnda muud meetodit kasutada, siis saab arendaja lisada sellise meetodi repositooriumi liidesesse. Lihtsamatel juhtudel oskab Spring Data JPA koostada päringu meetodi nime järgi. Näiteks kui tekib soov lisada UserRepository'isse meetod kasutajate e-posti järgi otsimiseks, siis piisab sellise rea lisamisest nagu on näidatud joonisel 6.

[1] https://spring.io/projects/spring-data-jpa

```
@Repository
public interface UserRepository extends JpaRepository<User, String> {

    Optional<User> findByEmail(String email);

}
```

Joonis 6: Oma meetodi loomine JpaRepository'is

Keerukamate meetodite puhul ei saa Spring Data JPA nime järgi päringut koostada. Päring tuleb arendajal kirjutada iseseisvalt JPQL keeles. JPQL keel omab SQL-iga sarnast süntaksit, kuid JPQL-is opereeritakse Java olemitega mitte andmebaasi tabelitega. Päring tuleb kirjutada meetodile määratud @Query annotatsiooni parameetrina. Joonis 7 illustreerib, kuidas keerukama päringuga meetoteid luua Spring Data JPA abil.

```
@Query("SELECT NEW ee.taltech.codera.submissionresult.projections.
    StatisticsSubmissionResultProjection(" +
           "result.id, result.overallPercentage, result.submission.
              exerciseCategory.id, " +
           "result.submission.exerciseCategory.name, " +
           "result.submission.exercise.id, result.submission.exercise.name," +
           "result.timeTested, result.submission.user.email) " +
           "FROM SubmissionResult result " +
           "WHERE EXISTS (SELECT exercise FROM ExerciseCategory category " +
           "INNER JOIN category.exercises exercise " +
           "WHERE category.id = result.submission.exerciseCategory.id AND " +
           "exercise.id = result.submission.exercise.id) " +
           "ORDER BY result.id DESC")
    List<StatisticsSubmissionResultProjection>
       findLastSubmissionResultsInDescOrderById(Pageable page);
```

Joonis 7: Keerukam päring JPQL keeles

4.3.6 Liquibase

Liquibase [1] on avatud lähtekoodiga lahendus andmebaasi skeemi versiooni kontrollimiseks (*version control*). Coderas kasutatakse Liquibase'i skeemi defineerimiseks ning andmebaasi migratsioonide läbiviimiseks. Liquibase'i üheks suureks eeliseks on paljude andmebaasisüsteemide tugi. Liquibase toetab nii PostgresSQL'i (kasutatakse Codera toodangus) kui ka H2 (kasutatakse lokaalseks

[1] https://www.liquibase.org/

arendamiseks). Liquibase kasutamiseks tuleb kirjutada nn ChangeSet'e. ChangeSet kujutab endast andmebaasi muudatuse kirjeldust mingis kindlas formaadis. Liquibase toetab järgmiseid standardseid formaate: XML, JSON, YAML ja SQL. Kolme esimese puhul tuleb jälgida dokumentatsioonis kirjutatud reegleid muudatuse kirjeldamiseks, siis Liquibase koostab SQL laused kasutatava andmebaasisüsteemi jaoks. Viimase formaadi puhul tuleb arvestada, et SQL laused võivad erinevates andmebaasisüsteemides erineda. [13] Coderas paneb Liquibase'i käima Spring Boot iga rakenduse jooksutamise ajal. Kui mõni migratsioon ebaõnnestub, siis rakendus ei lähe käima ja kuvatakse veateade. ChangeSeti näitega saab tutvuda joonisel 8

```
---
databaseChangeLog:
  - changeSet:
      id: add-hidden-column-to-exercise-category
      author: kirill.denisov
      changes:
        - addColumn:
            columns:
              - column:
                  name: hidden
                  type: BOOLEAN
            tableName: exercise_category
        - addNotNullConstraint:
            columnDataType: BOOLEAN
            columnName: hidden
            tableName: exercise_category
            defaultNullValue: FALSE
```

Joonis 8: Liquibase'i migratsiooni näide YAML formaadis

4.3.7 Spring Security

Spring Security [1] on raamistik, mis lihtsustab autentimise ja autoriseerimise realiseerimist. Spring Security lisamiseks Spring Boot rakendusesse tuleb lisada sõltuvus
`org.springframework.boot:spring-boot-starter-security`.

Kui sõltuvus on lisatud, siis kohe hakkavad kehtima konfiguratsioonid, mis nõuavad autentimist lõpp-punktide kasutamiseks. Seda konfiguratsiooni saab üle

[1] `https://spring.io/projects/spring-security`

kirjutada, luues uue klassi, mis laiendab Spring Security `WebSecurityConfigurerAdapter` klassi, nagu on näidatud joonisel 9

```
@Configuration
@EnableWebSecurity
@RequiredArgsConstructor
@EnableGlobalMethodSecurity(prePostEnabled = true)
public class SecurityConfiguration extends WebSecurityConfigurerAdapter {

    private final AzureActiveDirectoryAuthenticationFilter
        azureActiveDirectoryAuthenticationFilter;

    @Override
    public void configure(HttpSecurity http) throws Exception {
        http
                .sessionManagement().sessionCreationPolicy(
                    SessionCreationPolicy.STATELESS)
                .and()
                .csrf().disable()
                .addFilterBefore(azureActiveDirectoryAuthenticationFilter,
                    UsernamePasswordAuthenticationFilter.class)
                .authorizeRequests()
                .antMatchers(HttpMethod.OPTIONS, "/**").permitAll()
                .antMatchers("/actuator/**").anonymous()
                .antMatchers(HttpMethod.GET,
                        ExerciseCategoryController.EXERCISE_CATEGORIES_ENDPOINT
                            + "/**").permitAll()
                .anyRequest().authenticated()
                .and()
                .exceptionHandling()
                .authenticationEntryPoint(new HttpStatusEntryPoint(HttpStatus.
                    UNAUTHORIZED));
    }

}
```

Joonis 9: Spring Security konfiguratsiooni ülekirjutamine

Spring Security raamistiku kasutamisega kaasnevad mugavad annotatsioonid õiguste kontrollimiseks. Üks sellistest on `@PreAuthorize`. Joonisel 10 on toodud näide Codera lähtekoodist, kus kasutatakse `@PreAuthorize` annotatsiooni Administraatori rolli olemasolu kontrollimiseks.

```
@PostMapping
@PreAuthorize("hasRole('ADMINISTRATOR')")
@ResponseStatus(CREATED)
public Exercise addExercise(@Valid @RequestBody Exercise exercise) {
    return exerciseService.saveEntity(exercise);
}
```

Joonis 10: `@PreAuthorize` kasutamine

4.3.8 Testimisraamistikud

Codera serverirakendus on kaetud nii üksus- kui ka integratsioonitestidega. Kõik mooduli testid asuvad test alammoodulis ja on paigutatud samadesse pakettidesse, kus asuvad testitatavad klassid. Testide jooksutamiseks kasutatakse JUnit 5 [1] raamistikku. Testid ise kujutavad endast `@Test` annotatsiooniga märgitud meetodeid, mis kutsuvad toodangu koodi meetodeid välja ja kontrollivad, et konkreetse sisendi puhul nad annaksid õige väljundi. Coderas jälgitakse ühtset malli testide kirjutamiseks:

- Testi nimi kirjeldab, mida antud testis kontrollitakse.
- Test on jagatud kolmeks osaks:
  - *Given*: sisendi defineerimine. Vajalike muutujate defineerimine, objektide loomine.
  - *When*: testitatava meetodi väljakutse.
  - *Then*: tulemuse kontroll.

Üksustestides kasutatakse Mockito raamistikku, et jäljendada neid objekte, millest antud klass sõltub. Joonisel 11 on toodud Codera üksustesti näide.

[1] `https://junit.org/junit5/`

```
@Test
void shouldReturnAddedExerciseWhenAddingExercise() {
    // given
    Exercise exercise = Exercise.builder().build();
    when(exerciseRepository.save(exercise)).thenReturn(exercise);

    // when
    Exercise addedExercise = exerciseService.saveEntity(exercise);

    // then
    assertThat(addedExercise, is(equalTo(exercise)));
}
```

Joonis 11: Üksustesti näide

Integratsioonitestid testivad enamasti REST kihti ja nendes enam ei jäljendata sõltuvusi, vaid kasutatakse reaalseid objekte ning andmebaasi. Integratsioonitestide kirjutamiseks kasutatakse REST Assured [1] teeki, mis oskab REST päringuid teha ning teeb testi loetavamaks. Joonisel 12 on illustreeritud Codera integratsioonitesti näide REST Assured teegi kasutusega:

```
@Test
void shouldCreateHiddenExerciseCategory() {
    // given
    given()
            .port(localServerPort)
            .auth().oauth2(CoderaMockUser.ADMINISTRATOR_USER.getToken())
            .contentType(ContentType.JSON)
            .body(newModifyCategoryRequestBuilder("categoryName")
                    .description("categoryShortDescription")
                    .hidden(true).build())

            // when
            .when()
            .post(ExerciseCategoryController.EXERCISE_CATEGORIES_ENDPOINT)

            // then
            .then()
            .statusCode(201)
            .body("id", is(notNullValue()))
            .body("name", is(equalTo("categoryName")))
            .body("description", is(equalTo("categoryShortDescription")))
            .body("hidden", is(equalTo(true)));
}
```

Joonis 12: Integratsioonitesti näide Coderast REST Assured kasutusega

[1] http://rest-assured.io/

4.4 Klientrakendus

Klientrakendus on programm, mille kaudu suhtleb lõppkasutaja serveriga. Tänu klientrakendusele ei pea kasutaja teadma, mis on reeglid serveriga suhtlemiseks. Kasutajale kuvatakse andmeid serverist kasutajasõbralikul kujul ning ta saab tehteid teostada kasutades kasutajaliidese elemente (nt. nuppe).

Codera klientrakendus kujutab endast JavaScriptis kirjutatud rakendust. Kui kasutaja läheb Codera veebilehele, tõmmatakse tema brauserisse alla väike JavaScript rakendus, mis genereerib Codera veebilehe kasutaja brauseris. Seda nimetatakse kliendipoolseks renderdamiseks (*Client-side rendering*). Teine variant oleks võinud olla serveripoolne renderdamine, mis tähendab, et kasutajale näidetavaid veebilehti genereerib server. Kuid serveripoolsel renderdamisel on oma puudused:

- Rohkem päringuid serverile,
- Uuele veebilehele minek nõuab kogu lehe uuesti genereerimist,
- Sobib rohkem staatilistele lehtedele.

Kliendipoolse renderdamise miinused:

- Esmakordne lehe avamine võtab rohkem aega,
- Arendaja peab oskama rohkem tehnoloogiaid. [17]

Esimese miinuse vastu saab öelda, et leht ikkagi genereeritakse suhteliselt kiiresti. Probleeme võib tekkida vaid väga aeglase interneti kiiruse juhul.

Andmete vahetamiseks serveriga kasutab klientrakendus serverirakenduse REST rakendusliidest.

4.5 Klientrakenduse tehnoloogiad

4.5.1 Vue.js

Vue.js [1] on avatud lähtekoodiga JavaScripti raamistik kasutajaliideste ja üheleheliste rakenduste (*Single Page Application*) loomiseks. Vue.js olulisemaks võimaluseks on reaktiivne renderdamine. Kui muuta mõne muutuja väärtust, siis Vue.js kohe uuendab need kohad veebilehel, kus selle muutuja väärtust kasutatakse.

Vue rakendus koosneb taaskasutatavatest komponentidest. Vue komponent koosneb kolmest osast:

- Mall (*template*) - komponendi visuaali kirjeldamine HTML-is.
- Skript (*script*) - komponendi loogika kood.
- Stiilid (*styles*) - komponendi CSS stiilid.

Vue.js komponendi mall on näidatud joonisel 13.

```
<template>
  ...
</template>

<script>
...
</script>

<style>
...
</style>
```

Joonis 13: Vue komponendi faili mall

Kuigi vaikimisi kasutab Vue.js JavaScripti, siis Coderas kasutatakse Vue komponentides TypeScripti. TypeScript on programmeerimiskeel, mis laiendab JavaScripti ja teeb koodi rohkem objektorienteerituks ja loetavamaks.

TypeScripti kasutamise eelised:

- Võimalus määrata tüüpe (aitab vigu vältida).

[1] https://cli.vuejs.org/

- Võimalus kasutada uuemate ES standardite võimalusi.
- Lakoonilisem kood. [7]

TypeScripti jaoks on loodud hulk kasulikke dekoraatoreid, mis on abiks TypeScripti klasside Vue komponentideks konverteerimisel.

Joonisel 14 saab võrrelda ühte ja sama komponendi koodi JavaScriptis ja TypeScriptis.

```
@Component
export default class ResultTag extends
     Vue {

    @Prop() private result: number;

    get resultTagClass(): string {
        if (this.result === 100) {
            return 'is-success';
        }
        if (this.result >= 50) {
            return 'is-warning';
        }
        return 'is-danger';
    }

}
```

(a)

```
Vue.extend({
    props: {
        result: {
            type: Number
        }
    },
    computed: {
        resultTagClass() {
            if (this.result === 100) {
                return 'is-success';
            }
            if (this.result >= 50) {
                return 'is-warning';
            }
            return 'is-danger';
        }
    }
});
```

(b)

Joonis 14: Vue komponendi kood TypeScriptis (a) ja JavaScriptis (b)

4.5.2 Sass

Sass [1] on stiililehe keel veebilehtede kujundamiseks, mida kompileeritakse CSS keeleks. Sass pakub selliseid võimalusi nagu muutujad, *mixin*'id ja funktsioonid. Coderas kasutatakse Sassi koos Vue'ga. Sass vähendab stiililehe koodi hulka ning võimaldab koodi mooduliteks jagada. Lisaks eelistavad paljud CSS raamistikud just Sassi, et anda kasutajatele võimalus väärtusi üle kirjutada. Joonisel 15 on toodud Sassi kasutamise näide Coderas ülesande siltide (*tag*) värviklasside genereerimiseks:

[1] https://sass-lang.com/

```
$exercise-tags-colors: (
        #F44336,
        #E91E63,
        #9C27B0,
);

@mixin category-tag-colors {
  @for $i from 1 through length($exercise-tags-colors) {
    .exercise-tag--color-#{$i} {
      background-color: nth($exercise-tags-colors, $i) !important;
      color: findColorInvert(nth($exercise-tags-colors, $i)) !important;
    }
  }
}

@include category-tag-colors;
```

Joonis 15: Sassi kasutamine CSS klasside genereerimiseks

Sassi koodi konverteerib CSS koodiks preprotsessor sass-loader, mis on lisatud Vue rakendusse sõltuvusena.

Joonisel 16 on illustreeritud joonise 15 kood, mis on konverteeritud CSS keeleks.

```
.exercise-tag--color-1 {
  background-color: #F44336 !important;
  color: findColorInvert(#F44336) !important;
}

.exercise-tag--color-2 {
  background-color: #E91E63 !important;
  color: findColorInvert(#E91E63) !important;
}

.exercise-tag--color-3 {
  background-color: #9C27B0 !important;
  color: findColorInvert(#9C27B0) !important;
}
```

Joonis 16: Eelmise joonise näide konverteeritud CSS-iks

4.5.3 Bulma ja Buefy

Codera kujundamiseks kasutatakse Bulma [1] CSS raamistikku. Bulmat eristab teistest CSS raamistikest see, et see on modulaarne, atraktiivne, kohandatav ning teeb rakendust mugavalt kasutatavaks nii lauaarvutis kui ka mobiilis.

[1] https://bulma.io/

Lisaks Bulma klassidele kasutatakse Coderas Buefy [1] raamistikku. Buefy raamistik on Bulmal baseeruvate komponentide kogumik Vue jaoks.

4.5.4 Monaco Editor

Codera on loodud koodi kirjutamiseks. Seega peab Coderas olema mugav redaktor koodi kirjutamiseks. Coderas kasutatakse Microsofti poolt arendatud tekstiredaktorit - Monaco Editor [2]. Sama koodiredaktorit kasutab Microsoft oma Visual Studio Code IDE tootes. Järgmisena on välja toodud mõned Monaco Editor koodiredaktori huvipakkuvad võimalused:

- Süntaksi värvimine vastavalt keelele.
- Teksti otsimine ja asendamine.
- Multikursor.
- *Autocompletion.*

4.6 Autentimine

TalTech'is on igal tudengil olemas unikaalne UNI-ID, millega ta saab TalTech'i süsteemidesse sisse logida. TalTech kasutab Microsofti Office365 pilveteenust, mis omakorda kasutab Azure AD (*Azure Active Directory*) autentimise teenust kasutajate haldamiseks. Azure AD autentimise teenust kasutatakse näiteks ained.ttu.ee keskkonda sisselogimiseks. Seda võimalust otsustati kasutada ka Coderas, et TalTech'i tudengid ja töötajad ei peaks endale eraldi kontosid Coderas looma.

Azure AD kasutamiseks tuleb vastavas portaalis rakendus registreerida. Registreerimisel tuleb määrata rakenduse nimi ning veebiaadress. Veebiaadressi määramine on vajalik, sest Azure AD saadab kasutajate andmed vaid sellele aadressile. Kahjuks rakenduse registreerimise õigus on määratud ainult TalTech'i töötajatele. Codera rakenduse haldamisega Azure AD portaalis tegeleb antud töö juhendaja, Ago Luberg. Azure AD pakub ka lisavõimalusi, nt õiguste määramine rakenduse piirides, kuid neid võimalusi Coderas ei kasutata.

[1] `https://buefy.org/`
[2] `https://microsoft.github.io/monaco-editor/index.html`

4.6.1 JSON Web Token

Kasutajate andmete saatmiseks kasutatakse JWT luba (*token*). JWT luba on base64 kodeeritud kasutaja andmeid sisaldav sõne. See koosneb kolmest osast: päis, andmed ja signatuur. Päises on JSON kujul kirjeldatud, millist räsialgoritmi on vaja kasutada signatuuri valideerimiseks. Andmete osa on kasutaja andmetega JSON, mis sisaldab selliseid välju nagu kasutaja e-meil, nimi, perekonnanimi ja profiilipildi veebiaadress. Signatuur on kokkupandud päise ja andmete räsi. Joonisel 17 on illustreeritud dekodeeritud JWT loa päis ja andmed.


```
{
   "typ":"JWT",
   "alg":"RS256",
   "x5t":"HBxl9mAe6gxavCkcoOU2THsDNa0",
   "kid":"HBxl9mAe6gxavCkcoOU2THsDNa0"
}
{
   "iat":1557655294,
   "nbf":1557655294,
   "exp":1557659194,
   "amr":[
      "pwd"
   ],
   "family_name":"Denisov",
   "given_name":"Kirill",
   "name":"Kirill Denisov",
   "unique_name":"kideni@ttu.ee",
   "upn":"kideni@ttu.ee",
   "ver":"1.0"
}
```

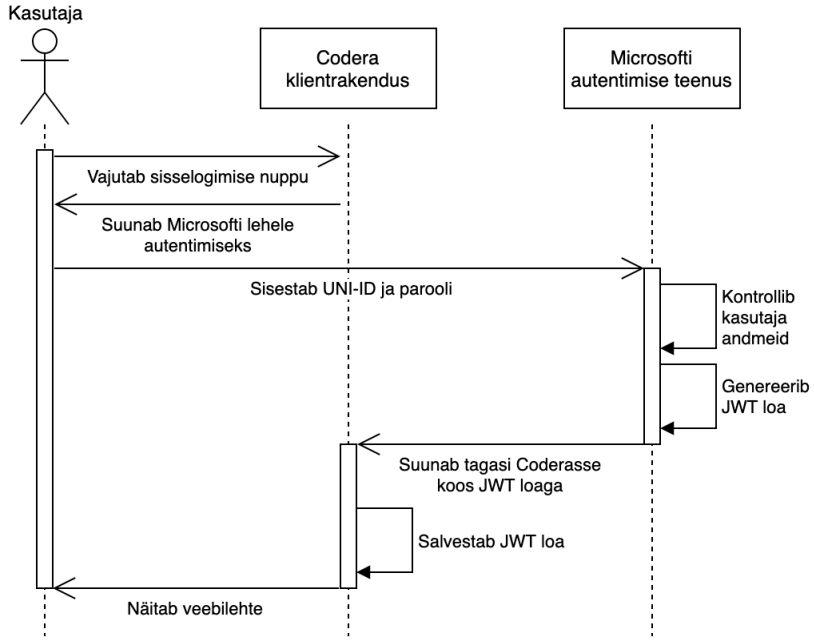

Joonis 17: Dekodeeritud JWT loa päis ja andmed (mõned väljad on ära kustutatud)

Serverirakenduse poole pöördumisel peab klientrakendus päringu päises saatma kasutaja JWT loa, et serverirakendus saaks kasutajat identifitseerida. Loomulikult peab serverirakendus seda luba enne päringu täitmist valideerima.

JWT luba aegub mõne aja pärast ja kasutaja peab uuesti sisse logima, et uus luba saada. [15]

4.6.2 Autentimise voog

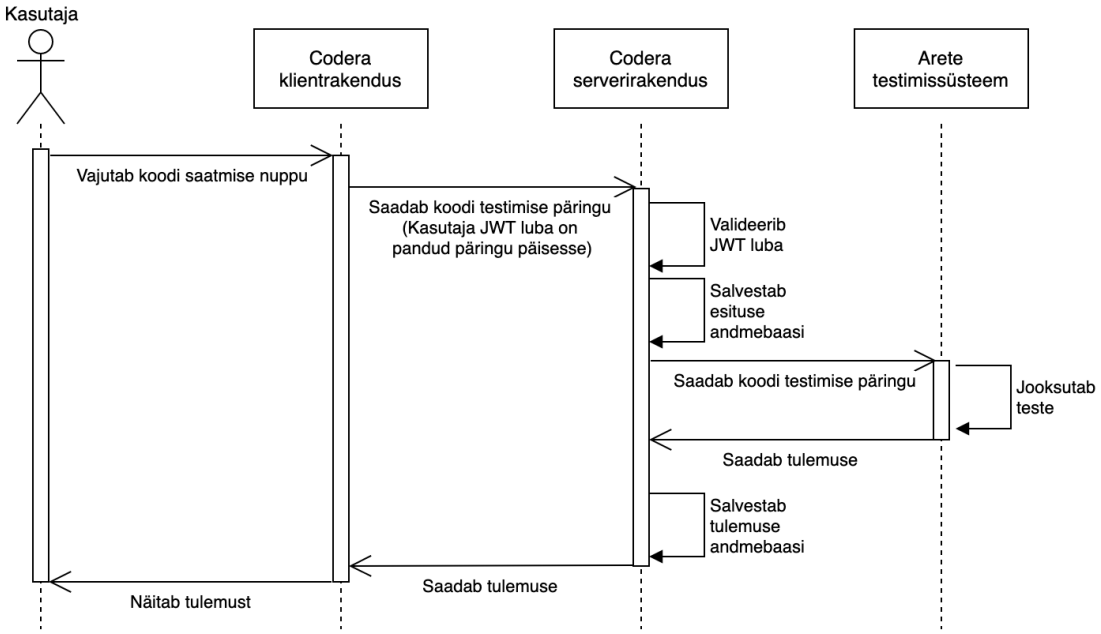
Autentimise voog kirjeldab JWT loa saamise protsessi. Codera autentimise voog on realiseeritud vastavalt OAuth2 protokollile. [1] Joonisel 18 on illustreeritud Codera autentimise voog.



Joonis 18: Autentimise voog

4.7 Ülesande lahendamise voog

Ülesande lahendamise voog on Codera põhivoog, milles on kasutatud nii Arete testimissüsteemi kui ka Azure AD autentimise teenust. Ülesande lahendamise voog algab ülesande valimisega ja kujutab endast tsüklit, kus kasutaja kirjutab koodi ja saadab seda testimiseks niikaua, kuni ta saab talle sobiva tulemuse. Pärast testimist näidatakse kasutajale testimissüsteemi tagasisidet, mis sisaldab detailset informatsiooni jooksutatud testidest. (Lisa 3) Joonisel 19 on illustreeritud koodi testimise voog.



Joonis 19: Koodi testimise voog

5 Rakenduse publitseerimine

Käesoleva töö üheks eesmärgiks oli teha tulemusena saadud platvorm TalTech'i tudengitele ja töötajatele kättesaadavaks. Codera jaoks eraldati TalTech'is server ja domeen codera.cs.ttu.ee. Selles peatükis kirjeldatakse, kuidas jõuavad koodimuudatused serverisse ning kuidas rakendust serveeritakse kasutajatele.

5.1 Gitlab

Projekti haldamiseks kasutatakse TalTech'i Gitlabi, mis on kättesaadav aadressil gitlab.cs.ttu.ee. Gitlab on veebipõhine tööriist Git salvede haldamiseks. Lisaks pakub Gitlab järgnevaid võimalusi:

- Ülesannete (*issue*) haldamine, mida kasutatakse ülesannete või probleemide kirjeldamiseks ja töö planeerimiseks.
- Õiguste määramine erinevatele kasutajatele salve piirides.
- Wiki lehtede loomise võimalus.

5.2 Arenduse protsess

Iga uue funktsionaalsuse arendamine algab sellega, et tehakse Gitlabi uus ülesanne, kus on uuendus lühidalt kirjeldatud. Ülesande põhjal luuakse *Merge Request*, mis loob kaasa ka uue haru, kuhu muudatused pannakse. Kui arendus on lõpetatud, siis muudatused mestitakse `master` harusse. Uue versiooni publitseerimiseks tuleb teha soovitud kehtestusele (*commit*) silt. Siis ehitatakse uus versioon ja paigaldatakse see toodangu serverisse.

5.3 Gitlab Continuous Integration

Igakord kui salves tehakse muudatusi, on vaja veenduda, et projektis ei tehtud midagi katki, mis tähendab, et kood endiselt kompileerub ja kõik testid õnnestuvad. See protsess on Coderas automatiseeritud tänu Gitlab CI (*Gitlab Continuous Integration*) tööriistale. Gitlab CI seadistamiseks tuleb koostada

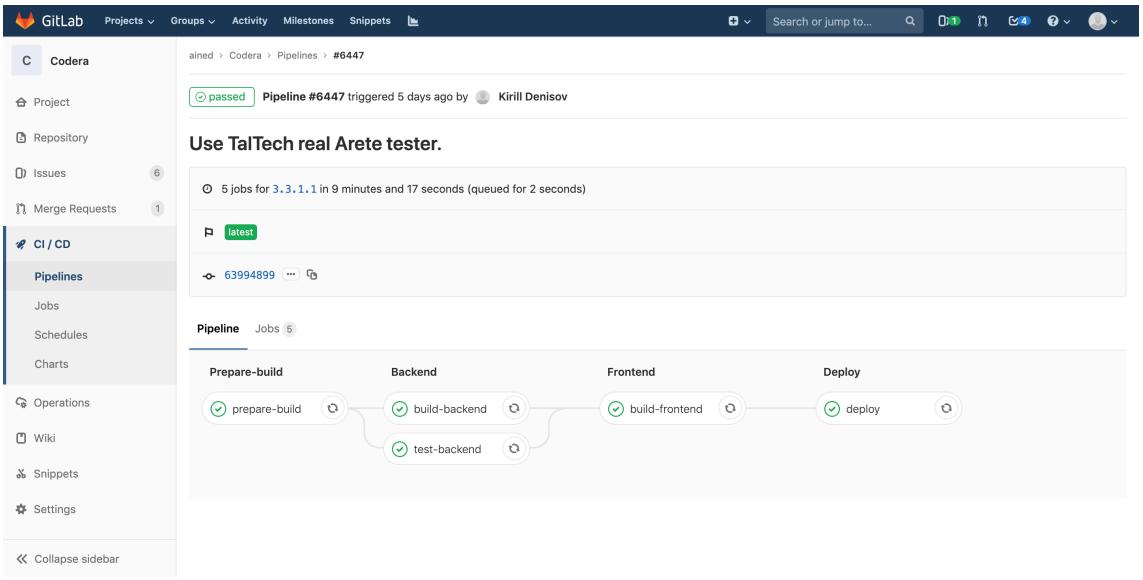
konfiguratsioonifail `.gitlab-ci.yml`, panna see projekti juurkausta ning seadistada masin, mis hakkab konfiguratsioonifaili järgi projekti ehitama. Konfiguratsioonifailis on kirjeldatud etapid (*stages*), etappidesse kuuluvad tööd (*jobs*) ja töös jooksutatavad käsud. Konfiguratsioonifaili näide on illustreeritud joonisel 20.

```
stages:
  - prepare-build
  - backend
  - frontend
  - deploy

prepare-build:
  stage: prepare-build
  script:
      - chmod +x codera/gradlew
      - codera/gradlew -p codera clean

build-backend:
    stage: backend
    script:
      - codera/gradlew -p codera assemble
    artifacts:
      paths:
        - codera/app/build/libs/*.jar
      expire_in: 2 days

test-backend:
    stage: backend
    script:
      - codera/gradlew -p codera check

build-frontend:
    stage: frontend
    script:
      - codera/gradlew -p codera npm_install updateNodeSassBindings
          buildFrontend
    artifacts:
      paths:
        - codera/frontend/dist/*
      expire_in: 2 days

deploy:
    stage: deploy
    script:
      - docker-compose down
      - docker-compose build
      - docker-compose up -d
    only:
      refs:
        - /^\d+.\d+.\d+(.\d+)?(.+)?$/
    retry: 2
```

Joonis 20: Coderas kasutatava Gitlab CI konfiguratsioonifaili sisu

Vaikimisi iga haru muutmisel käivitatakse kõik etapid, kuid konfiguratsioonifailis saab kehtestada piiranguid. Coderas näiteks ei käivitata *deploy* etappi, kui kehtestustele ei lisatud uut silti.

Ehitamise protsess ehk konveier (*pipeline*) loetakse õnnestunuks, kui kõik tööd on lõpetatud ja Gradle'i protsess tagastas lõpetamise koodi 0. Kui mõni test kukub läbi või kompileerimine ebaõnnestub, siis tagastab Gradle muu lõpetamise koodi ja töö loetakse ebaõnnestunuks. Samuti on võimalik vaadata logi töö kohta. Joonisel 21 on illustreeritud edukalt lõpetatud konveier.



Joonis 21: Edukalt lõpetatud konveier Gitlab CI keskkonnas

5.4 Docker

Docker [1] on tasuta tarkvara, mis laseb rakendusi käivitada isoleeritud keskkonnas, mida nimetatakse konteineriks. Docker konteinerit ei saa kutsuda virtuaalmasinaks, kuna see ei sisalda täielikku operatsioonisüsteemi, vaid kasutab selle operatsioonisüsteemi tuuma, kuhu Docker tarkvara on paigaldatud. Konteineri sisse pannakse rakendus, rakenduse töö jaoks vajalikud teegid ja muud failid. [2]

Coderas kasutatakse Dockerit rakenduste serveerimiseks. On olemas kolm konteinerit. Esimeses konteineris asub PostgreSQL'i andmebaasisüsteem. Selle

[1] https://www.docker.com/

konteineri ametliku tõmmise (*image*) saab leida Docker Hub repositooriumist. Teises konteineris käivitatakse serverirakendus ja selleks on konteineris olemas JDK 11. Kolmandas konteineris asub Nginx'i server, mis serveerib klientrakendust kasutajatele.

Docker tõmmiseid luuakse projektis olevate Docker failide järgi. Sellise faili näide on näidatud joonisel 22.

```
FROM openjdk:11-jdk-oracle
VOLUME /tmp
ADD /build/libs/app-*.jar codera-app.jar
ENTRYPOINT ["java","-Djava.security.egd=files:/dev/./urandom","-Dspring.
    profiles.active=prod","-jar","/codera-app.jar" ]
```

Joonis 22: Docker faili näide

5.4.1 Docker Compose

Docker Compose [1] lihtsustab mitme konteineri haldamist. Kuna Coderas kasutatakse kolme konteinerit, oleks tüütu igaühte eraldi hallata. Sarnaselt Gitlab CI-ga tuleb luua YAML formaadis konfiguratsioonifail, kus on määratud vajalikud konteinerid ja nende käivitamise jaoks vajalikud parameetrid. Docker Compose tarkvaraga koondub konteinerite püstipanemine kolme lihtsama käsu käivitamiseks.

- `docker-compose down`
  - Paneb vanad konteinerid kinni.
- `docker-compose build`
  - Ehitab uued konteinerid Docker failide järgi.
- `docker-compose up -d`
  - Jooksutab uusi konteinereid, võttes parameetreid Docker Compose konfiguratsioonifailist. Parameeter `-d` paneb konteinerid käima taustaprotsessina.

[1] `https://docs.docker.com/compose/`

5.5 HTTP server

Codera vajab HTTP veebiserverit klientrakenduse serveerimiseks. Järgmisena on toodud nõuded serverile:

- Staatiliste failide serveerimise võimalus.
- HTTPS tugi, kuna Coderal on olemas TalTech'i poolt antud sertifikaadid turvalise ühenduse jaoks.
- Reverse Proxy võimalus.

Variante on kaks: Nginx [1] ja Apache [2] ja mõlemad täidavad ülaltoodud nõudeid. Kuid Shreyasi Shrivastav on ühes oma artiklis tõestanud, et Nginx suudab vähemalt 2.5 korda kiiremini serveerida staatilisi faile ja on turvalisem kui Apache. Kuid Apache'i kohta öeldakse, et see on parem, kui on vaja spetsiifilisi konfiguratsioone. [14] Codera puhul selliseid ei ole, seepärast valiti Nginx klientrakenduse serveerimiseks.

5.5.1 Reverse proxy

Codera serveris on turvalisuse huvides avatud vaid kaks porti: 80 ja 443. Mõlemat porti kasutab klientrakendus. Probleem on selles, et klientrakendus käivitatakse kasutaja arvutis, aga see peab suhtlema serverirakendusega, mille jaoks porti ei jatku. Lahendus on kasutada Reverse Proxy võimalust.

Idee seisneb selles, et pordi 443 kaudu on kättesaadavad mõlemad rakendused. Kokkupõrgete vältimiseks kasutatakse erinevaid URL-e. Coderas on näiteks URL `/api` eraldatud just serverirakendusele. Kui kasutaja läheb sellele URL-ile, siis Nginx suunab tema päringu serverirakendusele, kui aga kasutaja läheb muule aadressile, siis tema päring antakse täitmiseks klientrakendusele.

[1] `https://www.nginx.com/`
[2] `https://httpd.apache.org/`

6 Kokkuvõte

Töö esimeseks eesmärgiks oli luua koodi kirjutamise harjutamisplatvorm, kus tudengid saavad ülesandeid lahendada ilma, et nad peavad spetsiifilist tarkvara enda arvutisse paigaldama. Teiseks eesmärgiks oli teha see platvorm TalTech'i tudengitele ja töötajatele kättesaadavaks.

Esimese eesmärgi saavutamiseks viidi läbi analüüs, mille käigus uuriti eksisteerivaid platvorme ja nende funktsionaalsust ning toodi välja funktsionaalsed ja mittefunktsionaalsed nõuded uuele platvormile. Järgmine etapp oli tehnoloogiate valimine ja platvormi arendamine. Saadud platvorm on realiseeritud klient-server arhitektuuri järgi, kus serverirakendus on loodud kasutades Spring Boot raamistikku ning PostgreSQL andmebaasisüsteemi. Klientrakendus kujutab endast Vue.js raamistikuga tehtud rakendust, mis käivitatakse kasutaja brauseris.

Teise eesmärgi saavutamiseks käivitati see platvorm TalTech'i serveris, kasutades selliseid kaasaegseid tehnoloogiaid nagu pidevkooste ja Docker konteinerid. Lisaks on Gitlab CI tarkvara kasutusega välja töötatud protsess, mis lihtsustab muudatuste toodangusse lisamist.

Codera on kättesaadav aadressil codera.cs.ttu.ee. Sisselogimiseks tuleb kasutada TalTech'i UNI-ID'd, mis on automaatselt olemas kõikidel ülikooli tudengitel ja töötajatel.

6.1 Edasine arendus

Codera realiseerib vaid minimaalset vajalikku funktsionaalsust. Potentsiaalsed ideed, mida võiks realiseerida, on kirjeldatud peatüki 3.2 lõpus. Kuid nende hulka võib lisada Codera integreerimise olemasolevasse Moodle'i keskkonda, et tulemused Coderast kajastuks automaatselt Moodle'isse. Lisaks sellele oleks hea tõlkida Codera eesti keelde ning anda kasutajale võimalus valida, mis keeles ta tahab Coderat näha. Selleks võib kasutada Vue I18n pistikprogrammi, mida kasutatakse Vue rakenduste internatsionaliseerimiseks.

Kasutatud kirjandus

[1] Authorize access to azure active directory web applications using the oauth 2.0 code grant flow. 2019. URL https://docs.microsoft.com/en-us/azure/active-directory/develop/v1-protocols-oauth-code.

[2] A beginner's guide to docker - how to create your first docker application. 2019. URL https://medium.freecodecamp.org/a-beginners-guide-to-docker-how-to-create-your-first-docker-application-cc03de9b639f.

[3] Oracle java se support roadmap. 2019. URL https://www.oracle.com/technetwork/java/java-se-support-roadmap.html.

[4] Maven goals and phases. 2019. URL https://www.baeldung.com/maven-goals-phases.

[5] U. Eriksson. The difference between functional and non-functional requirements. 2015. URL https://reqtest.com/requirements-blog/understanding-the-difference-between-functional-and-non-functional-requirements/.

[6] R. Fadatare. What is the difference between hibernate and spring data jpa? 2018.

[7] R. Feng. The benefits of migrating from javascript to typescript. 2015. URL https://www.appdynamics.com/blog/engineering/the-benefits-of-migrating-from-javascript-to-typescript/.

[8] T. Janssen. Design patterns explained – dependency injection with code examples. 2018.

[9] W. Krzywiec. Why spring framework is so cool. 2018. URL https://medium.com/@wkrzywiec/why-spring-framework-is-so-cool-8472ceabaab1.

[10] V. Mallawaarachchi. 10 common software architectural patterns in a nutshell. 2017.

[11] D. Potapenko. Ühise programmeerimisülesannete kogu ning koodi kirjutamise harjutamisplatvormi arendamine., 2018.

[12] M. Rouse. Rest (representational state transfer). 2017.

[13] S.Šhmeltzer. Introduction to liquibase and managing your database source code. 2017.

[14] S.Šhrivastav. Apache vs nginx – which is the best web server for you? 2019. URL https://serverguy.com/comparison/apache-vs-nginx/.

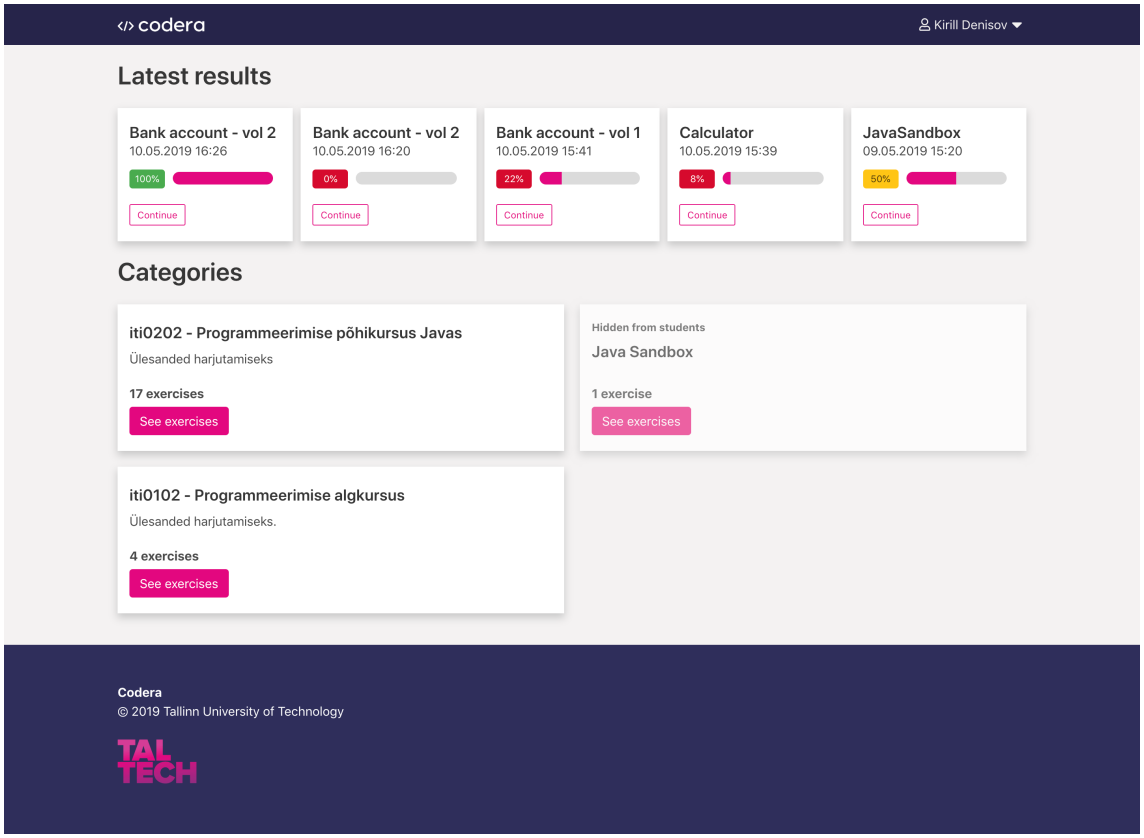
[15] M.Štecky-Efantis. 5 easy steps to understanding json web tokens (jwt). 2016. URL https://medium.com/vandium-software/5-easy-steps-to-understanding-json-web-tokens-jwt-1164c0adfcec.

[16] A.Štringfellow. Gradle vs. maven. 2017. URL https://dzone.com/articles/gradle-vs-maven.

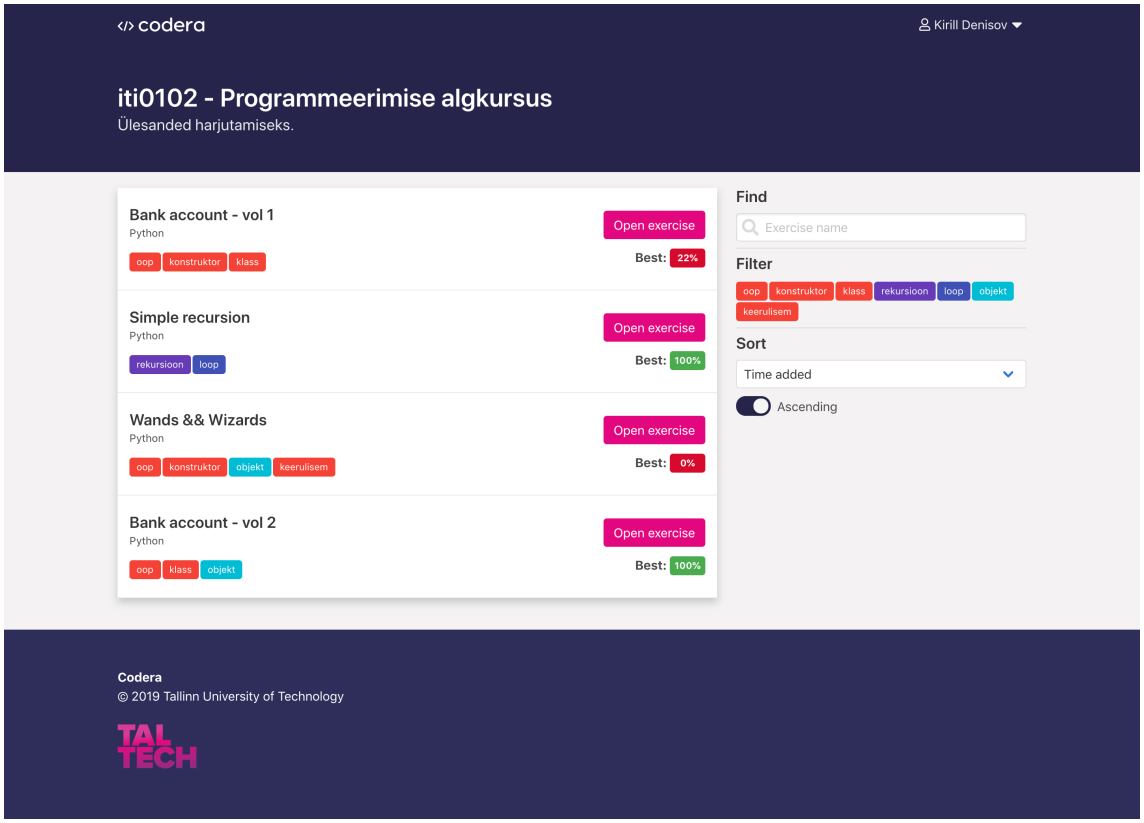
[17] J. Vega. Client-side vs. server-side rendering: why it's not all black and white. 2017.

Lisad

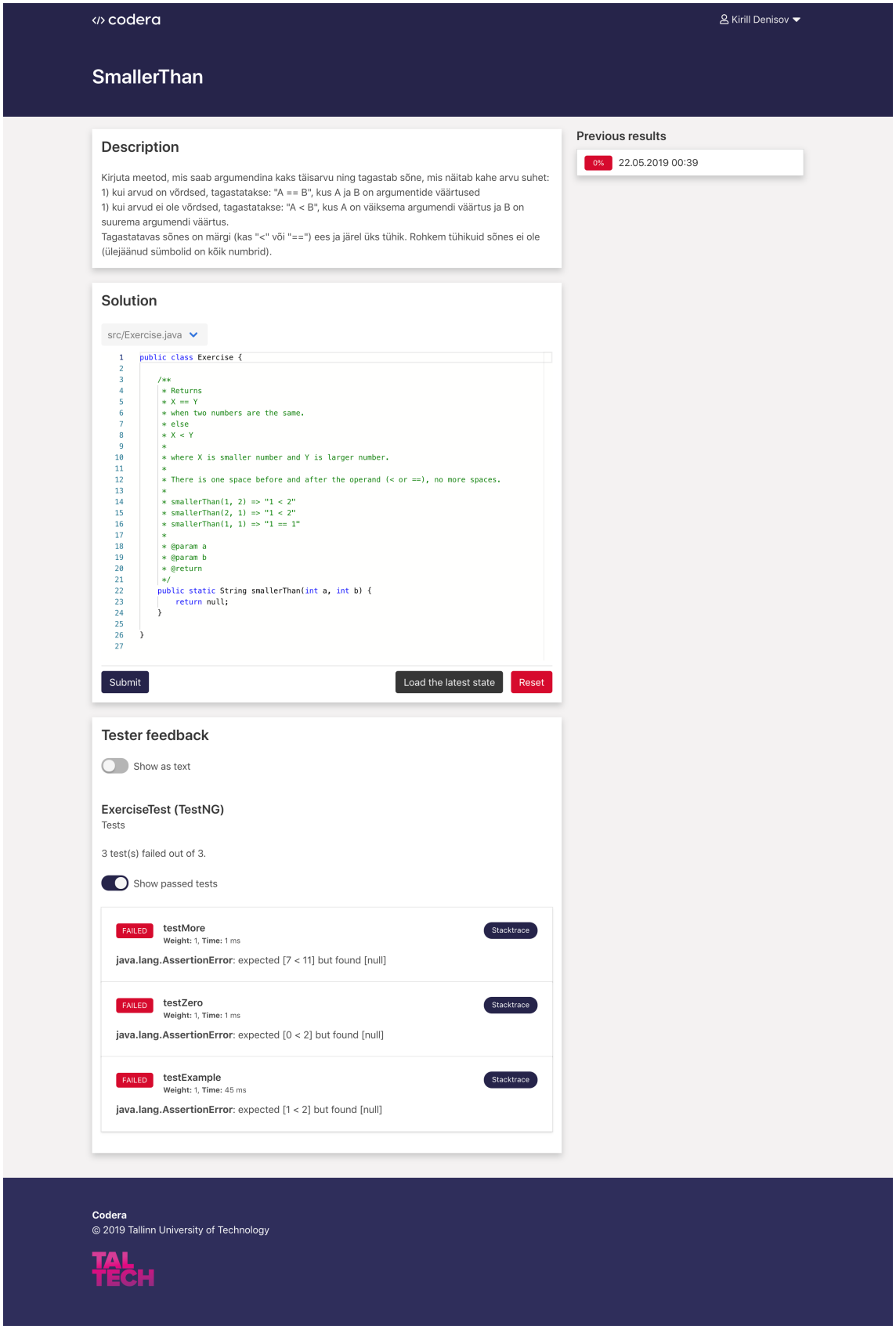
Lisa 1. Codera pealeht



Lisa 2. Codera kategooria vaade



Lisa 3. Codera ülesande vaade



Lisa 4. Codera tudengi esituse vaade