TARTU ÜLIKOOL

Arvutiteaduse instituut

Informaatika õppekava

Cardo Tisler

Veebilehe loomine DeepMOOC platvormile SvelteKit raamistikuga

Bakalaureusetöö (9EAP)

Juhendajad: Ahti Põder, PhD

Tõnis Hendrik Hlebnikov

Tartu 2023

Veebilehe loomine DeepMOOC platvormile SvelteKit raamistikuga

Lühikokkuvõte:

DeepMOOC on arendusfaasis olev platvorm, mille eesmärk on luua keskkond nii õppejõududele kui ka tudengitele programmeerimisülesannetes esitatud koodi automaattestimiseks ning hindamiseks. Antud platvormi loomise vajadus seisneb asjaolus, et hetkel Tartu Ülikoolis olev lahendus, Virtual Programming Lab, on piiratud programmeerimiskeelte toega ning ei paku funktsionaalsusi, mis oleksid kasulikud tudengitele õppimiseks ning õppejõududele teadmiste edastamiseks. Uue platvormi eesmärk on eeltoodud piirangud kaotada ning pakkuda erinevaid tööriistu programmeerimisainete efektiivseks läbiviimiseks. Bakalaureusetöö eesmärk on luua eesrakendus DeepMOOC platvormile, mis tulevikus hakkab suhtlema platvormi tagarakendusega.

Võtmesõnad: DEEPMOOC, veebileht, veebirakendus, SvelteKit

CERCS: P175 Informaatika, süsteemiteooria

Developing a front-end application for the DeepMOOC platform with SvelteKit

Abstract:

DeepMOOC is a platform that is currently in the development stage. The aim of the platform is to become an environment for automatically testing code and grading the code written for assignments. It is meant for both students and lecturers. The need for a platform like this stems from the fact that the current solution, Virtual Programming Lab, used in University of Tartu, has limited support for different programming languages and is missing functionalities that would benefit both students in learning new material and lecturers in their teaching. The goal for the new platform is to eliminate the aforementioned limitations and provide different tools to enable programming courses to be carried out effectively. The objective of this thesis is to create a front-end application for the platform, which will communicate with back-end in the future.

Keywords: DEEPMOOC, web page, web application, SvelteKit

CERCS: P175 Informatics, systems theory

Sissejuhatus

DeepMOOC arenduse eesmärk on luua platvorm tudengitele ning õppejõududele programmeerimisalaste teadmiste omandamiseks ning edasiandmiseks. Platvormi idee on muuta ülesannete sooritamine tudengite jaoks huvitavamaks tudengitevaheliste võistluste tekitamise abil ning parendada õppejõudude võimalusi uute ülesannete loomisel ja esitatud lahenduste hindamisel.

Uue platvormi vajadus on tingitud asjaolust, et Tartu Ülikoolis hetkel kasutusolev lahendus, VPL ehk Virtual Programming Lab, on piiratud nii keeletoe kui ka funktsionaalsuste poolest. Uue platvormiga antud piirangud eemaldatakse, tänu millele on võimalik tõsta õpetamise ja õppimise kvaliteeti.

Käesoleva bakalaureusetöö eesmärk on luua eelmainitud platvormile eesrakendus, mis on skaleeruv ja edasiarendatav ning järgib arenduse ja disaini häid tavasid. Valminud tarkvaraprojekti kvaliteeti valideeritakse läbi automaattestide ja Google poolt loodud avatud lähtekoodiga veebilaienduse, mis annab hinnangu veebilehe kvaliteedile üle 75 erineva kriteeriumi alusel. Antud töö sooritatakse koostöös disaineri lõputööga, kellega autor töötab välja funktsionaalnõuded ning disaineri ülesandeks jääb luua eesrakenduse disain vastavalt funktsionaalnõuetele. Autori töö on vastavalt disainile implementeerida eesrakendus ja vajadusel iseseisvalt täiendusi sisse viia.

Käesolev töö koosneb viiest peatükist: esimeses peatükis antakse teoreetiline ülevaade valitud tehnoloogiatest, põhjendatakse neid valikuid ning selgitatakse tähtsamaid mõisteid; teises peatükis selgitatakse tausta ning varasemaid töid antud platvormiga; kolmandas peatükis kirjeldatakse funktsionaal- ning mittefunktsionaalnõudeid; neljandas peatükis arutatakse tööprotsessi üle ja kirjeldatakse tähtsamaid implementatsioone; viiendas peatükis tehakse ülevaade validatsioonist ning valminud rakendusest.

1 Teoreetiline ülevaade

Järgnevalt kirjeldatakse tähtsamaid mõisteid, kirjeldatakse tehnoloogiliseid valikuid ning antakse põhjendusi nende valimisele. Samuti arutletakse erinevate testimismetoodikate eripärasustest ning kasudest. Viimaseks, antakse ülevaade juurdepääsetavuse vajadusest ning tähtsamatest funktsionaalsustest, mida on soovitatud, et tagada veebirakenduse kasutajasõbralikkus.

1.1 Mõisted

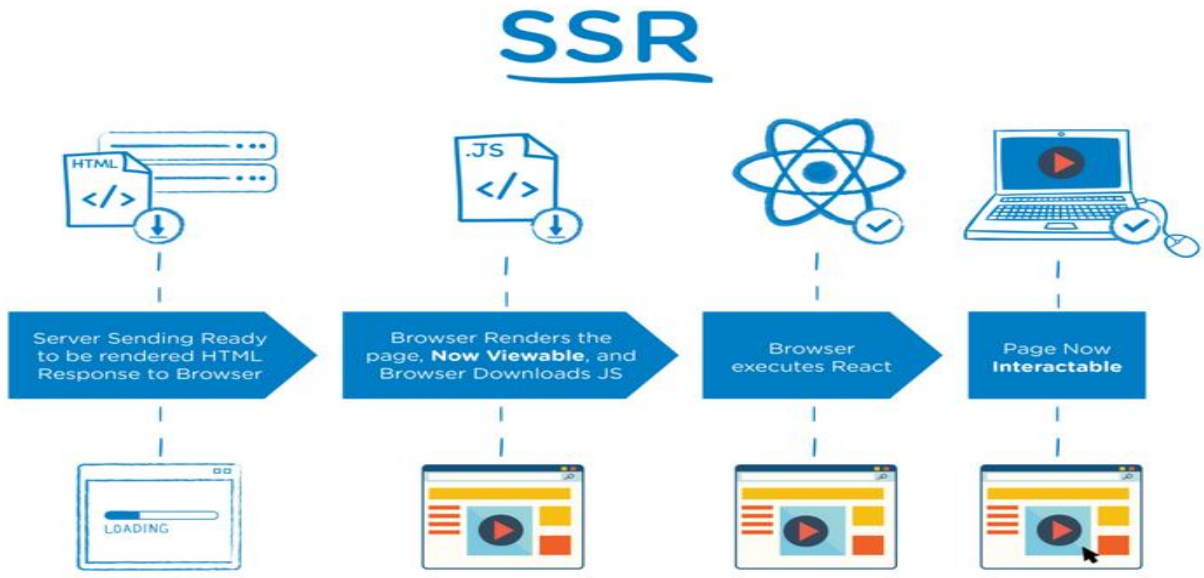
Mõistete peatükis kirjeldatakse olulisemaid mõisteid, et anda lugejale vajalik kontekst hiljem välja toodud tehnoloogiliste valikute mõistmiseks.

1.1.1 SSR

SSR ehk *server-side rendering* võimaldab veebilehe kasutajale saata kuvamiseks valmisolev leht. Teine variant on CSR ehk *client-side rendering*, mille puhul saadetakse kõik vajaminevad failid kasutaja brauserisse, kus JavaScripti abil sooritatakse vajalikud protsessid ning seejärel kuvatakse brauseris leht. Esimese variandi puhul teeb suure osa tööst ära server, mis on eriti kasulik, kui veebilehe külastaja kasutab vanemat või madalama jõudlusega seadet. CSR puhul teeb väga suure osa tööst ära kasutaja enda seade, mistõttu madalama jõudlusega seadmete puhul võib kasutajakogemus olla halb tänu sisu aeglasele laadimisele ja töötlemisele.

SSR-i puhul teeb kasutaja brauser päringu serverile, et küsida kuvamiseks veebilehte, siis server genereerib HTML-i ja CSS-i ning edastab selle brauserisse koos JavaScript failidega, peale mida brauser saab JavaScripti abil järgmiseid tegevusi sooritada. Antud funktsionaalsuse abil vähendatakse oluliselt kasutajapoolset töömahtu.

Järgneval joonisel illustreeritakse kuidas SSR puhul liiguvad andmed veebilehe ja kasutaja vahel.



Joonis 1: Illustratsioon SSR tööpõhimõttele [2]

Heavy.ai veebilehel olevas artiklis [1] on välja toodud SSR-i positiivsed ja negatiivsed aspektid.

Positiivne:

1. Veebilehe esialgse avamise kiirus: kuna server saadab kasutajale kuvamiseks valmis lehe, siis lehe avamine on oluliselt kiirem kui CSR arhitektuuriga veebilehe avamine.

2. Parem SEO: SEO ehk *Search Engine Optimization* on SSR lähenemisega oluliselt parem kui CSR arhitektuuriga lehtedel, sest kasutajale saadetava lehe sisu on koheselt kättesaadav ning tänu sellele saavad otsingumootorid lehed paremini indekseerida. Tänu sellele saab veebileht otsingumootoris parema hinnangu, mis tähendab, et leht on kergemini ülesleitav.

3. Kasutajasõbralikkus: abistavaid tööriistu kasutavate kasutajate jaoks on SSR arhitektuuriga veebilehed paremad, sest vastavad tööriistad saavad lehele terviklikult ligi, mis võimaldab vastavatel tööriistadel oma tööd paremini teostada. Sedasorti abistavad tööriistad on kriitilised inimestele, kes on puuetega ning vajavad abi veebilehel navigeerimisega.

4. Turvalisus: SSR-i abil on võimalik teha veebileht turvalisemaks. Kasutaja sisendi sanitiseerimise abil on võimalik vältida levinud ründemeetodeid nagu XSS ehk *cross-site scripting*.

Negatiivne:

1. Suurendatud koormus serverile: Delegeerides suure osa tööst serverile, suurendame töökoormust, millega server peab hakkama saama. Suur töökoormus võib muutuda problemaatiliseks kui tekivad perioodid kus on ootamatult palju kasutajaid, kes teevad nõudlikke päringuid või kui veebilehe jaoks valitud server on liiga nõrk.

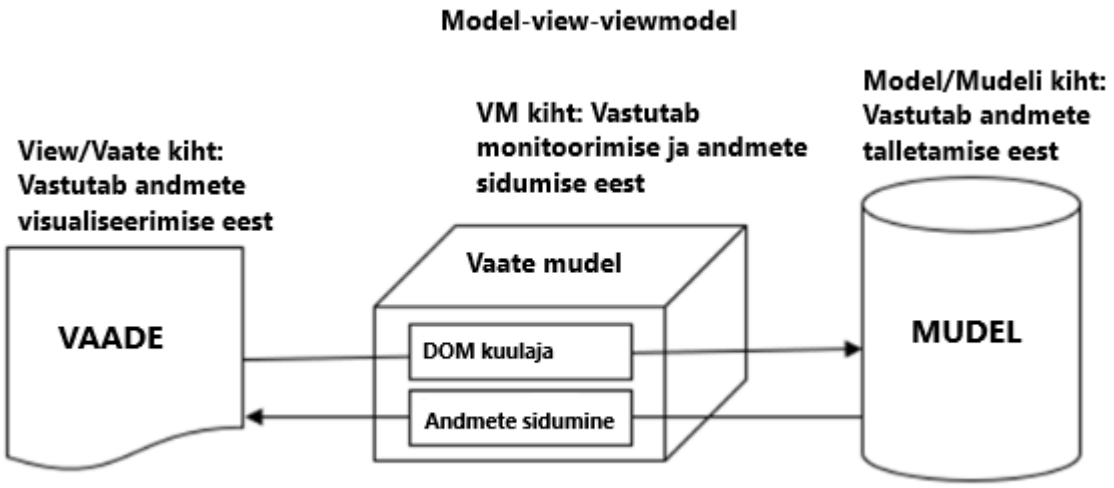
2. Keerukus: Kui veebilehe arendajatel puudub eelnev kogemus SSR-iga, siis võib see suurendada arendusperioode ning põhjustada probleeme veebilehe funktsionaalsuses. See probleem kehtib nii SSR, CSR kui ka iga teise metoodika, raamistiku ning programmeerimiskeele puhul.

1.1.2 Model-view-viewmodel

MVVM ehk *model-view-viewmodel* on mudel, mille põhjal on Svelte ehitatud. Mudeli tööpõhimõte on selgitatud Nian Li ning Bo Zhangi artikli alusel. MVVM mudelit kasutatakse andmete sünkroniseerimiseks veebilehe ning eesrakenduse vahel. Selle protsessi jaoks kasutatakse päästikuid ning andmete sidumist. Veebilehel toimuvate muudatuste korral kasutatakse pääs-

tikuid, et teavitada eesrakendust muudatustest ning seeläbi eesrakenduses olevaid andmeid uuendada. Teistpidi, kui eesrakenduses toimuvad muudatused, nt kui sooritatakse rakendusliidese päring, siis andmete sidumise abil tehakse vastavad muudatused veebilehel ehk uuendatakse HTML-i, mida kasutajale parasjagu kuvatakse [3].

MVVM mudeli tööpõhimõte on illustreeritud järgneval joonisel.



Joonis 2: Model-view-viewmodel mudeli visualiseerimine [3]

1.2 Tehnoloogilised valikud

Peatükis tuuakse välja töö sooritamiseks valitud tehnoloogiad, kirjeldatakse nende tööpõhimõtteid ja tähtsamaid omadusi ning põhjendatakse tehtud valikute sobivust antud töö jaoks.

1.2.1 SvelteKit

Veebirakenduse loomiseks valiti SvelteKit raamistik. SvelteKit dokumentatsiooni [13] kohaselt on tegu raamistikuga, mis on ehitatud omakorda Svelte raamistiku peale. Nii Svelte kui ka SvelteKit on ehitatud JavaScript programmeerimiskeeles. Svelte abil saab luua kasutajaliidese komponente, mis muudetakse kompilaatori abil JavaScriptiks, mis omakorda on suuteline kuvama veebilehele HTML-i ning kuvatavat HTML-i CSS-i abil kujundada. Svelte on loodud MVVM mudeli põhimõtteid järgides. Antud funktsionaalsus ei ole piisav, et ainult selle tööriistaga luua täisväärtuslik veebirakendus ning selle eesmärgi jaoks loodi SvelteKit. Selle abil saame lisaks eelnimetatud funktsionaalsustele ka ruutimise ning SSR ehk *Server Side Rendering* võimalused.

SvelteKit-i põhilised tunnused on:

1. Efektiivsus ja kiirus: SvelteKit kompilaator optimeerib arendaja poolt kirjutatud koodi, seeläbi koodi sisaldavate failide suurus väheneb. Tulemus on kiirem lehe laadimise aeg ning parem kasutajakogemus.

2. Eesrakenduse arendajate poolt kõrgelt hinnatud: 2022 aasta *State of JavaScript* iga-aastase küsitluse tulemuste alusel on Svelte 2. kohal, skooriga 90%. Esimesel kohal on raamistik Solid, tulemusega 91%. Antud protsentuaalne tulemus on kombinatsioon erinevatest faktoritest nagu arendajate huvi, raamistiku kasutamise sagedus, teadmiste hulk tööriista kohta jmt. [4]

3. Komponendipõhine arhitektuur: Tänu sellele saab terve rakenduse eraldada väiksemateks, taaskasutatavateks osadeks. Selle abil on kood modulaarne, kergemini hallatav ning suurendab arendusprotsessi kiirust.

4. Skaleeritavus: Tänu SSR-ile on SvelteKit rakendused kergesti skaleeritavad. Kuna SSR rakenduste puhul on leheküljel toimuvate muudatuste töö enamjaolt delegeeritud serverile, siis rakenduse skaleerimiseks piisab, kui valida võimsam server või lisada juurde uusi servereid.

5. Aktiivne kommuun: Kuna tegu on populaarse raamistikuga, siis aktiivseid kasutajaid on palju ning tänu sellele on võimalik lihtsamini leida vajaminevat informatsiooni ning saada oma küsimustele kiiremini vastuseid. See on väga väärtuslik omadus inimestele, kes alustavad antud raamistiku õppimisega, sest informatsiooni olemasolu ning kerge leitavus tagavad, et õppimisprotsess ei ole pärsitud.

Antud projekti käigus sai SvelteKit valituks tänu aktiivsele kommuunile ning skaleeritavusele. Aktiivse kommuuni abil on lihtne leida informatsiooni vastava raamistiku kohta, mis lihtsustab arendustööd selle rakenduse edasiarendajatele ning pakub head võimalust õppida uusi teadmisi populaarse eesrakenduse raamistiku kohta. SvelteKiti skaleeritavus tagab, et on lihtsam garanteerida veebirakenduse üleval püsimist aegadel, kus on tavapärasest rohkem veebiliiklust, näiteks eksamite sooritamise ajal.

1.2.2 TailwindCSS

TailwindCSS on CSS raamistik, mis pakub kogumi klasse, mis on eelnevalt valmis loodud. CSS klass on muutujanimetus, millele on vastavusse kaardistatud erinevad CSS reeglid. Järgnev lõik on kirjutatud TailwindCSS dokumentatsiooni [5] põhjal. Raamistiku tööpõhimõte on skännida kõiki HTML ning JavaScript faile, üles leida vastavad klassinimetused ning nende alusel genereerida CSS, mis kirjutatakse staatilisse faili. Selline lähenemine on kiire, taaska-

sutatav ning kergesti laiendatav, tänu millele on arendusprotsess oluliselt kiirendatud võrreldes tavapärase lahendusega, kus CSS kirjutatakse käsitsi.

Eelnimetatud raamistikul on palju arendajasõbralikke hüvesid:

1. Lühem arendusprotsess: Raamistik pakub väga põhjalikku kogumit erinevatest klassidest, millega saab koheselt alustada erinevate keerukate ning hea välimusega veebirakenduste loomist.

2. Ühtlane disain: Kasutades raamistiku poolt pakutud klassinimetusi üle terve rakenduse, on tagatud ühtlane disain, sest ühele klassile vastab alati sama hulk CSS reegleid. See vähendab riski, et rakenduses tekivad visuaalselt ebaühtlased komponendid. Taaskasutatavate muutujate rakendamine koodibaasis on üldlevinud standard, et ühtlustada funktsionaalsusi ning visuaale.

3. Reaktiivne disain: TailwindCSS abil on lihtne luua disaine, mis on reaktiivsed ekraanisuuruste muudatustele. Raamistik pakub erinevaid klasse, mis võimaldavad muuta veebilehel olevate elementide paigutust ning kujundust vastavalt ekraani mõõtmetele. Selle abil on lihtne tagada, et veebileht näeb visuaalselt hea välja olenemata seadmest, millel veebilehte kuvatakse.

4. Kohandatavus: Raamistik pakub lihtsasti rakendatavaid meetodeid klasside muutmiseks ning kohandamiseks vastava töö vajaduste järgi. Olemasolevaid klasse saab ümber nimetada, neile vastavaid reegleid ümber kirjutada ning uusi klasse juurde lisada konfiguratsioonifaili muutmisega.

5. Minimaalse mahuga failid: Skännides HTML ning JavaScript faile, TailwindCSS genereerib CSS-i ainult neile klassidele, mis on päriselt rakenduses kasutuses. See tähendab, et kui kasutaja pärib veebilehe, siis talle saadetakse staatiline CSS fail, mis sisaldab ainult neid reegleid, mida tal vaja on. Tänu sellele hoitakse failide suurust minimaalsena, mis vähendab serveri töökoormust ning kiirendab veebilehe kuvamist kasutajale.

TailwindCSS valiti projekti põhiliseks CSS raamistikuks tänu selle lihtsusele ning võimekusele. Antud raamistik ei nõua eelteadmisi, et seda kasutada, mistõttu on ta hea valik projektile, millel ajapikku vahetuvad arendajad, kel on erinevad teadmised ja oskused. Samuti on Tailwindi eelis paljude muude CSS raamistike puhul tema minimaalse mahuga failid, kuna lõppkasutajale saadetakse CSS fail, mis sisaldab ainult klasse, mida lõppkasutajal vaja, siis see vähendab serveri- ning võrgukoormust ja töötab hästi koostöös SvelteKiti *server-side-rendering* tööpõhimõttega.

1.2.3 Vitest

Vitest on ühiktestimise raamistik, mille eesmärk on võimaldada arendajatel kirjutada teste JavaScript koodi testimiseks. Samuti on tegu ka SvelteKiti loojate poolt soovitatud testimisraamistikuga, mis oli põhiline motivaator antud lahenduse valimisel. Antud lõik on kirjutatud Vitesti dokumentatsiooni alusel [6]. Üks põhilisi featuure Vitesti puhul on HMR ehk *hot module replacement*. HMR abil on võimalik koodi muudatuste korral kiiresti testid uuesti kompileerida ning jooksutada, seeläbi toetada arendusprotsessi ning tagada, et arenduse käigus tehtud funktsionaalsed vead on kiiresti üles leitavad. Tänu sellele on koodi testimise protsess efektiivne, sest testikomplektide jooksutamise tagasiside on kiiresti kättesaadav.

Vitest pakub palju erinevaid funktsionaalsusi:

1. Tugi erinevatele *assertion* raamistikele, mis võimaldavad kitsendada testide skoopi, et iga individuaalne test testiks võimalikult konkreetset osa koodist. See on üleüldine hea tava testimise juures, et testkomplekti tulemuse loetavus oleks üheselt arusaadav ning probleemide korral oleks probleemne koht võimalikult kiiresti leitav.

2. Testide filtreerimine ning grupeerimine. Võimaldab arendajal valida komplekt teste, mida arendusprotsessi ajal jooksutada. Vähendades üleliigsete testide kompileerimist ning jooksutamist, on testimisprotsess kiirem ning antud testide poolt antud tagasiside jõuab arendajani kiiremini, seeläbi muutes arendusprotsess efektiivsemaks.

3. Asünkroonse koodi testimine. Keerukamates projektides on vajadus kirjutada asünkroonset koodi, näiteks API päringute sooritamine. Projekti kvaliteedi tagamiseks on vajalik ka seda koodi testida.

4. Lihtsus. Arendaja kasutab Vitesti läbi lihtsa ning intuitiivse API, mis lihtsustab testide kirjutamist ning konfigureerimist. Vitest integreerub Vite-ga väga hästi, mistõttu on ta väga hea valik projektis, kus Vite on juba kasutusel. Antud bakalaureusetöös kirjutatud projekt kasutab Vitet.

1.2.4 Playwright

Nagu ka ühiktestimise raamistik Vitest, on Playwright soovitatud SvelteKiti loojate poolt. Tänu selle mugavale ühildusele SvelteKit projektiga, sai see valitud antud töö *end-to-end* testide raamistikuks. Järgnev lõik on kirjutatud Playwright raamistiku ametliku dokumentatsiooni alusel. [7]. Playwright on avaliku lähtekoodiga *end-to-end* testide kirjutamise raamistik. Raamistiku abil on võimalik automatiseerida tegevusi veebibrauseris, nagu lehel ringi liikumine, nuppude vajutamine, lahtritesse sisendite kirjutamine ning lehel leiduva informatsiooni valideerimine. Antud tegevused on automatiseeritavad mitmes erinevas brauseris - Google Chrome, Mozilla Firefox, Edge ning Safari. Raamistik võimaldab jooksutada teste paralleel-

selt, mis oluliselt vähendab testide jooksmise aega ning kiirendab arendusprotsessi, andes arendajale kiiret tagasisidet testide õnnestumise või läbikukkumise kohta.

1.2.5 Vercel

Antud bakalaureusetöö vältel kirjutatud kood on üles sätitud Verceli serverisse, tänu millele on see ligipääsetav kõigile. Antud sektsioon on kirjutatud Verceli ametliku dokumentatsiooni alusel. [8]. Vercel on platvorm eesrakenduse arendajatele, et kiiresti ja töökindlalt oma koodi kasutusele võtta ning teha see lõppkasutajatele kättesaadavaks. Vercel pakub null-konfiguratsiooni tuge üle 35 erineva eesrakenduse raamistikule, kaasaarvatud antud bakalaureusetöö jooksul kasutatud SvelteKit-ile. Null-konfiguratsiooni tugi tähendab, et rakenduse saab minimaalse vaevaga Verceli serverisse üles laadida, sest Vercel suudab vastavalt kasutatud raamistikule ise vajaliku konfiguratsiooni projektile peale panna.

1.3 Testimine

Enimlevinud meetod tarkvararakenduste kvaliteedi tagamiseks on automatiseeritud testide kirjutamine. Hästi üles ehitatud testid annavad kindluse, et testitud osad funktsioneerivad korrektselt ning võimaldavad tekkinud vigu varakult tuvastada ning parandada. Töö käigus kirjutati kahte tüüpi teste, ühikteste ning *end-to-end* teste. Järgnevalt selgitatakse valitud meetodeid ning nende omadusi.

1.3.1 Ühiktestimine

SmartBear artikli [9] alusel on ühiktestimine protsess, mille käigus kirjutatakse koodi, et testida individuaalseid koodiplokke, enamjaolt meetodeid või funktsioone. Antud protsessi käigus tagatakse, et valitud koodiplokk töötab nii nagu ette nähtud - õigete sisendite korral saadakse õiged vastused ning valede sisendite korral tegutseb koodiplokk õigesti. Ühiktestimise puhul on tähtis hoida koodiplokid võimalikult väikesed, mistõttu jooksutatakse testid isolatsioonis muust koodist ning on tavaks igasugused *dependency-d* ehk sõltuvused välja vahetada. Testi käigus ebatähtsa koodi välja vahetamist nimetatakse koodi *mock*-imiseks. See tagab, et test on võimalikult väike ning võimalikult väheselt mõjutatud muu koodi poolt, seeläbi muutes testi lühemaks, arusaadavamaks ning vigade leidmise kiiremaks.

Põhjalikult testitud kood muudab koodi vastupidavamaks muutuste suhtes, kuna ootamatult tekkinud vead on võimalik testidega üles leida, samuti toimivad testid ka omamoodi dokumentatsioonina, mis on väga tähtis koodi hallatavuse ja edasiarenduse perspektiivist.

1.3.2 *End-to-end* testimine

Kataloni artiklis [10] kirjeldatakse *End-to-end* ehk e2e testimist protsessina, kus testitakse tervet rakendust tervikuna, et tagada rakenduse töökindlus ning vastavus nõuetele. E2e testimise puhul seotakse kogu lähtekood ning kõik *dependency*-d üheks tervikuks. Antud protsess on kõige ligilähedasem päriskasutaja kogemusele, sest testid jooksutatakse brauseris, automatiseeritakse nuppude vajutused, lehel ringi liikumine, informatsiooni lugemine ja valideerimine, info sisestamine jpm.

Tavapäraselt on e2e testimine viimane samm testimisprotsessis, peale seda kui erinevad koodijupid on isolatsioonis testitud. Kui isoleeritud testid ehk ühiktestid on edukalt läbitud, siis jätkatakse keerukamate testidega, et tagada ühilduvus ja koormustaluvus.

1.4 Juurdepääsetavus

Juurdepääsetavus viitab rakenduste disainile ning arendusele, mis on kasutatav võimalikult paljudele inimestele, ka puuetega inimestele. Veebileht, mis järgib põhilisi juurdepääsetavuse põhimõtteid, on oluliselt mugavam kasutada, parandab SEO-d ning suurendab potentsiaalsete kasutajate hulka. Mozilla artiklis [11] on välja toodud potentsiaalsed lahendused või leevendused, mida arendajad ja disainerid saavad jälgida ning implementeerida, et luua võimalikult kasutajasõbralik veebileht:

1. Võimalus suurendada veebilehe fonti või veebilehte iseennast. Tänapäeval on enamus veebibrauseritel sisseehitatud lahendus selle jaoks, tänu millele piisab kasutajale lühiõpetuse andmisest, kuidas antud brauseri funktsionaalsust kasutada.

2. HTML-i semantiline kirjeldamine, et lihtsustada erinevate ekraanilugejate tööd. Antud lahendus aitab veebilehel ringi liikuda inimestel, kel on raskusi nägemisega. Ekraanilugeja töö on skaneerida kuvatavat HTML-i, ning kasutajale öelda, mis tähtsaid elemente ta sealt leidis, nagu näiteks nupud, lingid jne.

3. Tekstilised alternatiivid video sisule. Kui veebilehel kuvada videosid, siis peab videodel leiduma ka subtiitreid. Ka piltidele on mõistlik lisada tekstilisi alternatiive, et toetada kasutajaid, kel on aeglasem internetiühendus.

4. Klaviatuuripõhine liikumine veebilehel. Kuna ei saa eeldada, et igal kasutajal on olemas arvutihiir, või võimekus seda kasutada, siis on olemas erinevad HTML funktsionaalsused, et võimaldada veebilehel fokusseerida erinevaid elemente näiteks TAB nupu abil. Arendaja ning disaineri töö on paika panna loogiline järjekord elementide fokuseerimiseks.

2. Taust

Aastal 2022 loodi kolm lõputööd seoses DeepMOOC platvormiga. Tööde eesmärk oli luua erinevaid tagarakenduse lahendusi: Andre Anijärve “DeepMOOC platvormile tarkvarakonveieri arendamine” [14]; Kaarel Kangro “Koondhinnete ning hinnetetabelite moodustamise keel DeepMOOC platvormile” [15]; Joosep Näksi “DeepMOOC platvormile tagarakenduse dispetšeri arendamine” [16].

Andre Anijärve lõputöö raames valmis tarkvara, mis võimaldab vastu võtta tudengi poolt kirjutatud koodi, seda jooksutada ning veenduda koodi funktsionaalsuses. See on kriitiline osa, et antud platvormi saaks kasutada automaattestimiseks, mis on hetkel platvormi põhieesmärk. Töö vajalikkus on tingitud asjaolust, et hetkel Tartu Ülikoolis kasutatav lahendus, VPL (Virtual Programming Lab), on piiratud võimalustega. Anijärv on oma lõputöös kirjeldanud VPL-i potentsiaalseid turvanõrkusi, mida DeepMOOC platvormiga püütakse vältida. Põhiliseks välja toodud probleemiks osutus VPL-i virtualiseerimise puudumine – testitava koodi jooksutamine toimub füüsilise serveri peal, mistõttu on olemas oht, et pahatahtlik kood saab ära kasutada serveril olevaid turvanõrkusi.

Kaarel Kangro lõputöö tulemusena valmis lahendus koondhinnete ning hinnetetabelite moodustamiseks. Töös on välja toodud, et antud lahendus võimaldab kirjutada loetavamaid valemeid, mis tähendab, et nad on pikas perspektiivis jätkusuutlikumad ning arusaadavamad. Kangro töö võimaldab anda kiiret tagasisidet tudengite lahendustele, mis on uuringute [17, 18] sõnul hea viis parandada õppetulemusi ning suurendada motivatsiooni.

Joosep Näksi poolt tehtud töö lõi dispetšeri, mille ülesandeks oli suhtlus ees- ja tagarakenduse vahel. Dispetšer võtab vastu sõnumeid eesrakenduselt, käivitab tagarakenduses sõnumi sisule vastava tegevuse ja tagastab küsitud andmed. Töö valmimise hetkeks ei suudetud tehtud töid integreerida, mistõttu on Joosep Näksi sõnul antud töös puudu edetabelite kuvamise ja lahenduste hindamise funktsionaalsus.

3. Nõuded veebirakendusele

Töö algas funktsionaal- ja mittefunktsionaal nõuete defineerimisega. Järgnevalt tuuakse välja nõuded, mille põhjal antud töö valmis.

3.1 Funktsionaalsed nõuded

Veebirakenduses on maandumisleht

- Maandumislehel on sisselogimisnupp
- Maandumislehel on võimalus juurdepääsetavuse sätteid muuta
    - Lehte on võimalik suuremaks ja väiksemaks teha
    - Lehel on võimalik kasutada kõrge kontrastiga värviskeemi

Veebirakenduses on tudengi vaade

- Avalehel peab olema kursuste loetelu ja lühikirjeldus
    - Loetelus olev kursus peab näitama järgmist ülesannet
    - Loetelus olev kursus peab näitama edetabelit

Veebirakenduses peab olema kursuse leht

- Kursuse lehel peab olema loetelu ülesannetest
- Kursuse lehel peab olema loetelu edetabelitest
- Kursuse lehel peab olema tudengi punktide seis antud kursusel
- Kursuse lehel peab olema edetabeli eelvaade
    - Edetabeli eelvaade peab näitama top 3 tudengit, nimede asemel pseudonüümid
    - Edetabeli eelvaade peab näitama sisselogitud tudengi asukohta edetabelis
    - Edetabeli eelvaade peab eraldama tudengite pseudonüüme kolme vertikaalse punktiga

Veebirakenduses peab olema ülesande vaade

- Ülesande vaates peab olema ülesande nimi
- Ülesande vaates peab olema ülesande kirjeldus
- Ülesande vaates peab olema ülesande esitamise kast nähtaval

3.2 Mittefunktsionaalsed nõuded

- Tähtsam HTML peab sisaldama defineeritud aria-label’eid ehk HTML-i semantilisi kirjeldusi, et ekraanilugejad saaksid korrektselt oma ülesannet täita
- Leht peab olema SSR, et oleks kergesti skaleeritav
- Eesrakendus peab olema lihtne ning sisaldama võimalikult vähe äriloogikat
- Kood peab olema testitav
- Koodil peab olema üle 80% kaetus testidega
- Sisselogimine peab olema DeepMOOC tagarakenduse autentimisteenusega seotud

4. Tööprotsess

Antud bakalaureusetöö ajal tegeles DeepMOOC projektiga 7 inimest. 2 inimest, käesoleva töö autor ja disainer, olid seotud eesrakendusega ning ülejäänud 5 tegelesid tagarakendusega.

Eesrakenduse puhul oli tööjaotus jagatud kahe inimese vahel kaheks lõputööks, antud töö autor arendas veebirakendust enamjaolt vastavalt disainile, mis valmis teise lõputöö jooksul, erinevused tekkisid projekti lõpus, kui autor lisas juurdepääsetavuse funktsionaalsused, mida disainis ei leidunud.

Olenemata asjaolust, et töö skoobi alusel piirdus eesrakenduse ja tagarakenduse suhtlus ainult sisselogimisega, siis eesrakendus loodi dünaamilisena ning tehakse päringuid andmete küsimiseks, kuid päringu vastused on talletatud eesrakenduses. Tänu sellele on tulevikus võimalik tehtud töö lihtsasti ühendada tagarakendusega - kui tagarakendus suudab anda andmeid defineeritud kujul, siis tuleb eesrakenduses ainult päringu URL-id ära vahetada.

4.1 Lahendusteni jõudmine

Antud töö sooritamiseks valiti põhiliseks raamistikuks SvelteKit. Enne tööga alustamist puudus autoril eelnev kogemus väljavalitud raamistikuga, kuid oli olemas kogemus muude komponendipõhiste raamistikega, nagu React ja Vue. Tänu sellele piisas autoril dokumentatsiooni lugemisest, et tööga alustada. Täiendavad teadmised saadi töö käigus erinevate probleemide lahendamisega ning seejärel muudatuste tegemisega.

Arenduse jaoks kulus umbes 2 kuud, alustades SvelteKiti õppimisega ning lõpetades viimaste muudatustega koodibaasis.

Arenduse lõpuni viimiseks piisas enamjaolt SvelteKiti elementaarsetest teadmistest – SvelteKit muutub keerulisemaks, kui tekib reaalset loogikat serveri poolele, kuid kuna antud töö ühendati ainult sisselogimisteenusega, siis serveri poolel ei ole eriti keerulist loogikat, tänu millele oli arendusprotsess oluliselt lihtsam. Kuna keerukamaid funktsionaalsusi ei implementeeritud, otsustas autor keskenduda rohkem eesrakenduse kvaliteedi ning juurdepääsetavuse aspektidele, mis saavutati läbi erinevate testikomplektide üles sättimise ning tähtsamatele funktsionaalsustele tähelepanu pööramise.

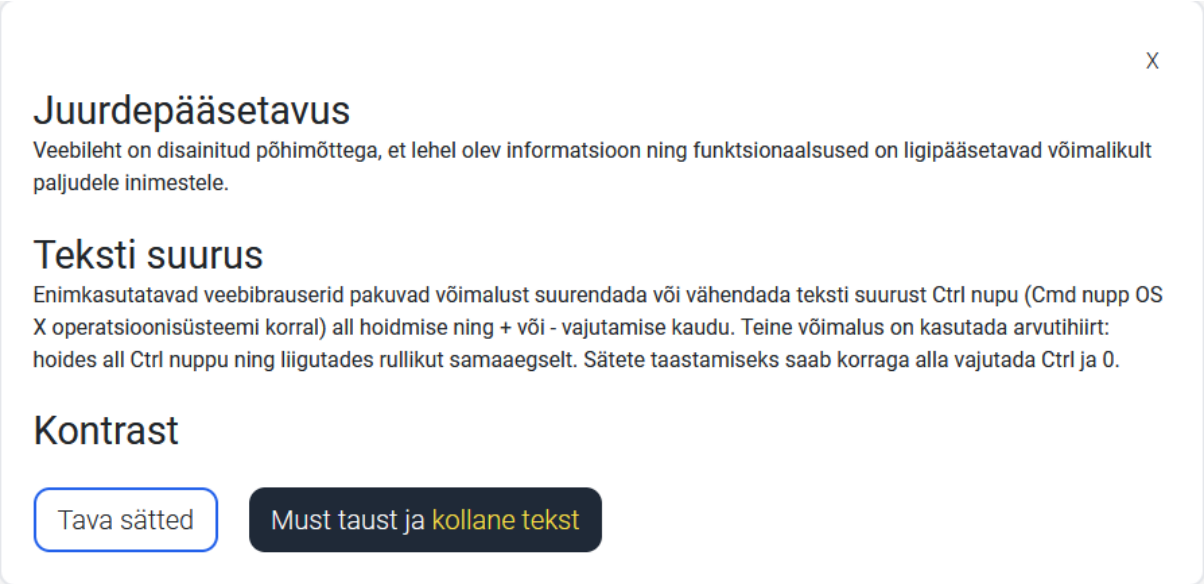
Sisselogimisteenuse toimimiseks on vajalik tagarakenduse projekti lokaalselt jooksutamine, mistõttu autor võimaldas veebilehele ligipääsu ka ilma tagarakenduse suhtluseta. Antud juhul veebirakendus tagarakendusega ei suhtle ning sisselogimise funktsionaalsus tegelikkuses ei toimi. See loogika on kirjutatud ainult lõputöö demonstreerimise lihtsustamise eesmärgil ning eemaldatakse peale töö kaitsmist, peale mida saab veebilehele ligi ainult eduka sisselogimise järel.

4.2 Juurdepääsetavuse implementeerimine

N Wedasinghe jt. uuringu [19] esimese faasi kokkuvõttes ja järeldustes on välja toodud, et kõige tähtsamad juurdepääsetavuse aspektid, mida veebilehtedel jälgida on klaviatuuri ligipääsetavus, semantiline HTML, piltidele alternatiiv teksti lisamine, korrektsete päiste kasutamine veebilehe struktuuri luues, võimalikult juurdepääsetav disain, tabelite kasutamine ainult tabelikujul andmete esitamiseks, võimalus lehe sisu suurust muuta ning vältida automaatseid navigeerimisi.

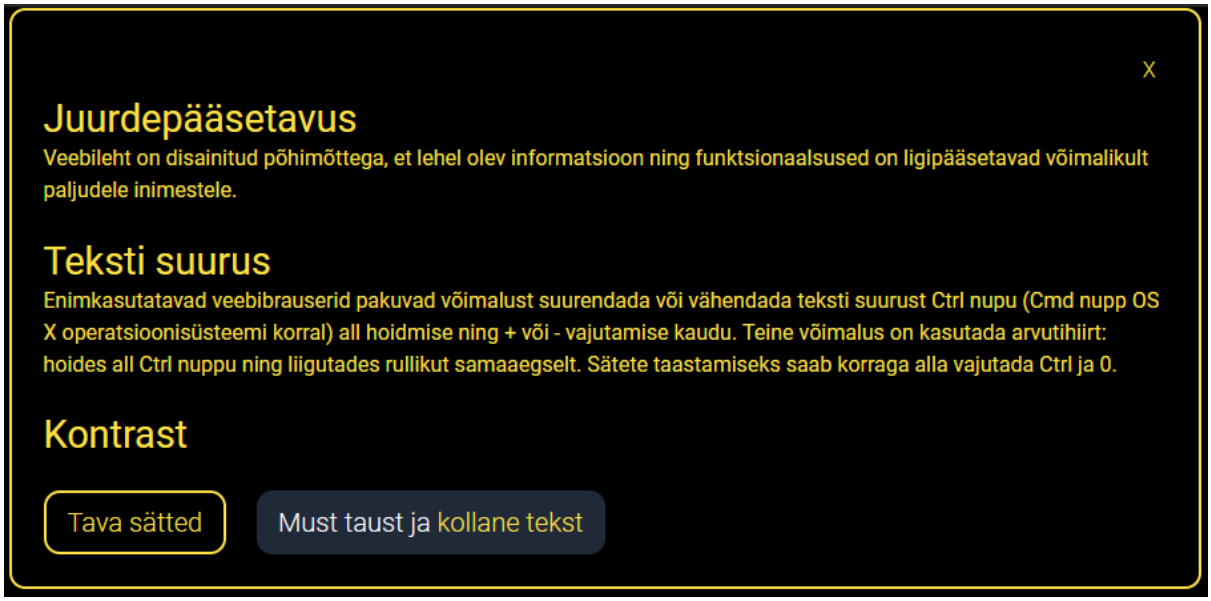
Antud töö raames pandi rõhku põhiliselt neljale juurdepääsetavuse funktsionaalsusele: kasutaja võimekus fonti ja rakendust suurendada või väiksemaks teha, valikuline kõrge kontrastiga värviskeem, semantiline HTML ekraanilugejate jaoks ning võimekus veebirakenduses hakkama saada ilma hiireta. Valikulised võimalused on kõik kasutaja poolt sätitavad enne sisselogimist maandumislehel.

Kaasaegsetel veebibrauseritel on sisseehitatud funktsionaalsus veebilehe suurendamiseks ja väiksemaks tegemiseks. Seetõttu autor ei loonud ise mingit lahendust veebilehe suuruse muutmiseks, vaid lõi juurdepääsetavuse sätete jaoks komponendi, inspireeritud sarnase lahenduse poolt www.emta.ee lehel, ning kirjutas arusaadava juhendi, kuidas antud brauseri funktsionaalsust kasutada, nähtav antud joonisel 3.



Joonis 3: Bakalaureusetöö käigus loodud juurdepääsetavuse komponent

Inimestel, kellel on raske näha, võib olla kasu teistsugusest värviskeemist, mis eemaldab ebavajalikud disaini elemendid ning teeb info leidmise lihtsamaks. Antud probleemi silmas pidades implementeeris autor kõrge kontrastiga värviskeemi, mida on kasutajal võimalik paigaldada maandumislehel juurdepääsetavuse komponendis, nähtav joonis 3-l. Järgnev joonis 4 illustreerib vastava funktsionaalsuse aktiveerimise tulemust:



Joonis 4: Bakalaureusetöö käigus loodud juurdepääsetavuse komponent, kõrge kontrasti värviskeemiga.

Semantiline HTML saavutati läbi HTML-i aria-label atribuudi kasutamist. Ekraanilugejad otsivad antud atribuudi väärtust, et seda lõppkasutajale ette lugeda, mistõttu on tähtis, et väärtused oleksid loogilised, informatiivsed ning korrektsed. Töö käigus lisati kõikidele nuppudele, linkidele ning tähtsamatele komponentidele aria-label väärtused, et ekraanilugejad need üles leiaksid ning kasutajat nende olemasolust teavitaksid.

Veebilehel on võimalik liikuda hiirt kasutamata ükskõik kust ükskõik kuhu, mistõttu autori hinnangul on antud nõue täidetud.

5. Tulemus

Peatükis kirjeldatakse töö tulemuse valideerimiseks kasutatud meetodeid ning esitletakse valminud veebirakendust.

5.1 Validatsioon

Töö tulemuse kvaliteedi valideerimiseks kasutati kolme meetodit - ühiktestimist, *end-to-end* testimist ning Lighthouse laiendust. Järgnevalt kirjeldatakse loodud testkomplektide abil saavutatud tulemusi ning kasutatud laienduse tagasisidet.

5.1.1 Automaattestid

Ühiktestimiseraamistiku abil mõõdetud tulemuste alusel saavutati töö käigus ühiktestide abil 100% koodiridade, 81.81% funktsioonide, 90.32% harude ning 100% avaldiste katvus, testide tulemused on esitatud joonisel 5.

All files components

100% Statements 394/394 **90.32%** Branches 28/31 **81.81%** Functions 9/11 **100%** Lines 394/394

Press *n* or *j* to go to the next uncovered block, *b*, *p* or *k* for the previous block.

Filter:

File	Statements		Branches		Functions		Lines	
AccessibilityModal.svelte	100%	61/61	100%	2/2	100%	0/0	100%	61/61
AssignmentOverview.svelte	100%	60/60	100%	5/5	100%	0/0	100%	60/60
CourseOverview.svelte	100%	46/46	100%	6/6	100%	0/0	100%	46/46
CourseTopic.svelte	100%	24/24	100%	0/0	100%	0/0	100%	24/24
FileInput.svelte	100%	69/69	100%	0/0	100%	0/0	100%	69/69
Leaderboard.svelte	100%	37/37	0%	0/3	100%	0/0	100%	37/37
Placeholder.svelte	100%	14/14	100%	0/0	100%	0/0	100%	14/14
Timeline.svelte	100%	17/17	100%	2/2	100%	0/0	100%	17/17
TimelineEvent.svelte	100%	55/55	100%	4/4	100%	0/0	100%	55/55
index.ts	100%	11/11	100%	9/9	81.81%	9/11	100%	11/11

Joonis 5: Vitest raamistiku koodikaetavuse raport

Atlassiani lehel Sten Pitteti poolt avaldatud artiklis [12] on kirjutatud, et laialdaselt levinud ja aktsepteeritud koodi katvuse protsent on 80% ning antud töö käigus ületati see eesmärk. Tänu kõrgele koodikatvusele on projekti edasiarendus oluliselt lihtsam, sest implementeeritud kood

on väga suures ulatuses testidega kaetud ehk edasiarenduse käigus tekkinud muudatused saavad testide poolt kinni püütud. Leitud muudatusi saab testide tagasiside abil analüüsida, seejärel otsustada kas uuendada teste või parandada tekkinud vead.

Töö käigus implementeeriti ka *end-to-end* testid. Nende testide eesmärk on tagada, et erinevad komponendid töötavad ühise tervikuna ning veebirakendus on töökorras. Antud testidega tagati, et kasutaja saab liikuda igalt lehelt igale muule lehele, seeläbi kindlustades, et veebirakenduses olevad komponendid töötavad omavahel korrektselt.

5.1.2 Lighthouse

Geekflare artikkel [20] kirjeldab Lighthouse-i kui Google poolt loodud avatud lähtekoodiga veebibrauseri laiendit, mille tööpõhimõte on jooksutada palju erinevaid teste veebilehe vastu. Laiendus testib üle 75 erineva kategooria ning tagastab rapordi, millel on tulemused kokku agregeeritud põhilistesse kategooriatesse: jõudlus ehk *performance*; juurdepääsetavus ehk *accessibility*; parimad praktikad ehk *best practices*; otsingumootori optimisatsioon ehk SEO. Üldjoontes ei tasu üritada kõikide kategooriate skoore maksimiseerida vaid raporti abil valida välja suurimad murekohad ning need parandada.

Veebirakenduse validatsiooni käigus mõõdeti iga loodud lehe tulemusi lokaalselt ja tavasätetega ning saavutati järgmised tulemused, mis on esitatud tabelis 1:

	Jõudlus	Juurdepääsetavus	Parimad praktikad	SEO
Maandumisleht	100	96	100	100
Avaleht	100	96	100	100
Kursuse leht	100	96	100	100
Ülesande leht	100	100	100	100

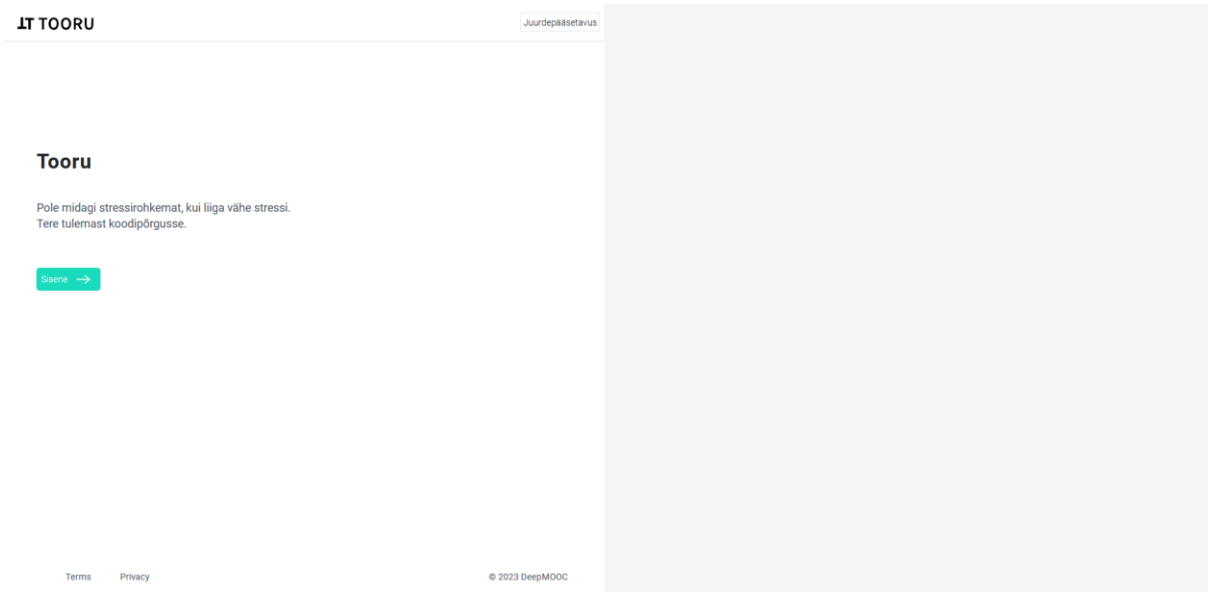
Tabel 1: Google Lighthouse laienduse raportite kokkuvõte

Lighthouse raportite järgi on näha, et jõudlus, parimad praktikad ja SEO on väga hästi implementeeritud, laiendusel ei ole nende kategooriate jaoks ei soovitusi ega parandusi. Juurdepääsetavuse ainus murekoht, laienduse sõnul, on vähene kontrast elementide vahel. Antud soovituse sai implementeerida kahel viisil, muutes veebirakenduse disaini või andes kasutajale võimaluse kasutada kõrge-kontrastiga värviskeemi. Kuna autor implementeeris veebirakenduse disaini teise bakalaureusetöö alusel, siis lahenduseks sai kõrge-kontrastiga värviskeemi võimaldamine kasutajale. Autor samuti veendus, et antud värviskeemi kasutamise korral andis Lighthouse laiendus juurdepääsetavuse skooriks 100/100.

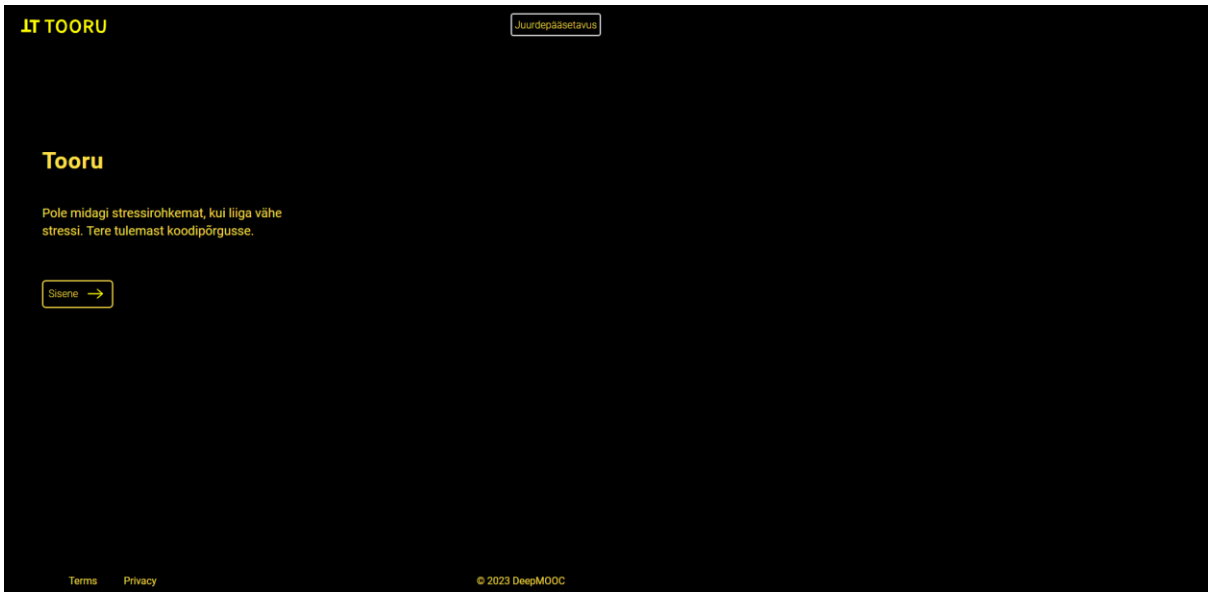
5.2 Veebirakenduse ülevaade

Peatükis näidatakse valminud tööd, kirjeldatakse implementeeritud komponente ning arutletakse potentsiaalseid edasiarendusi.

5.2.1 Maandumisleht



Joonis 6: Veebirakenduse maandumisleht



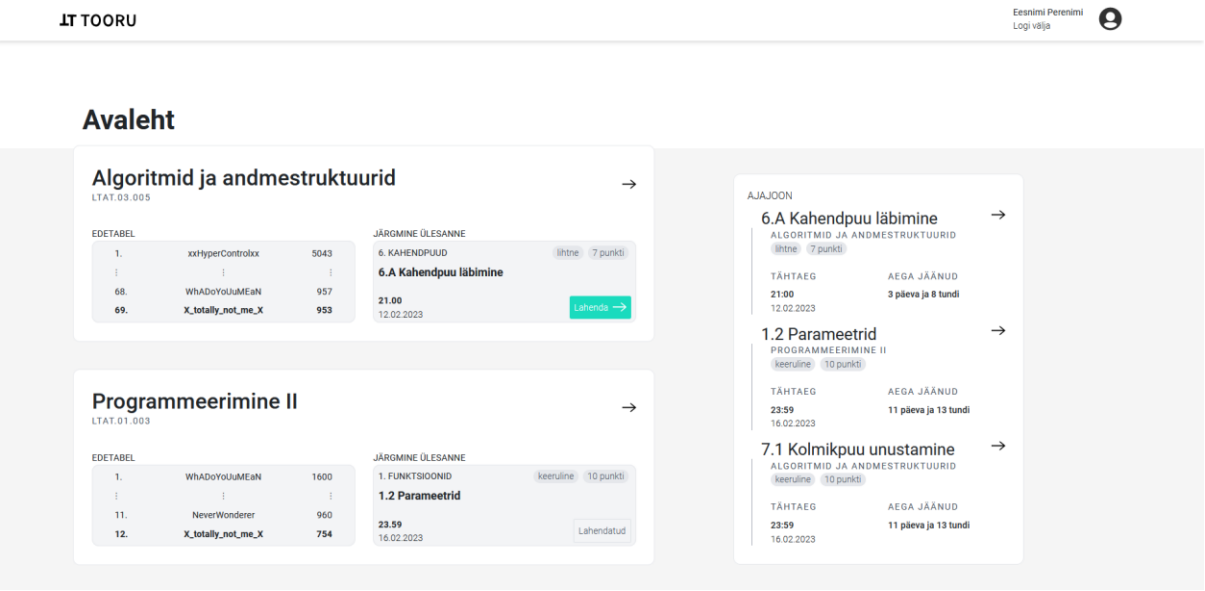
Joonis 7: Veebirakenduse maandumisleht kõrge-kontrastiga värviskeemis

Maandumisleht sisaldab põhiliselt sisselogimisnuppu, mille vajutamise korral tehakse päring tagarakendusele, ja juurdepääsetavuse featuuride kontrollimise komponenti. Tagarakendus kontrollib, kas kasutaja on sisse loginud ning vajadusel suunab edasi lehele, kus toimub sisselogimise protsess. Kasutajanime ja parooli eduka sisestamise korral antakse kasutajale vajali-

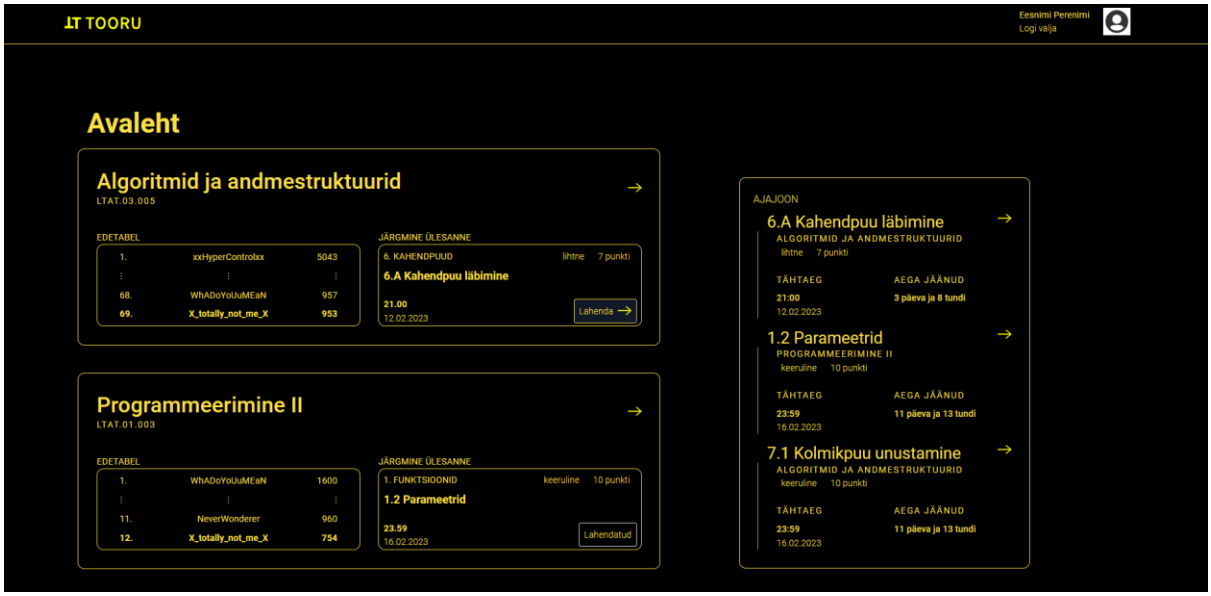
ku informatsiooniga küpsis ning suunatakse tagasi lehele, kust kasutaja esialgu tuli. Antud juhul tähendab see, et kasutaja suunatakse maandumislehele tagasi. Maandumislehel kontrollitakse vastavat küpsist ning kasutaja suunatakse edasi veebirakendusesse.

Juurdepääsetavuse komponenti näeb täpsemalt joonis 3-el.

5.2.2 Avaleht



Joonis 8: Veebirakenduse avaleht tudengivaates

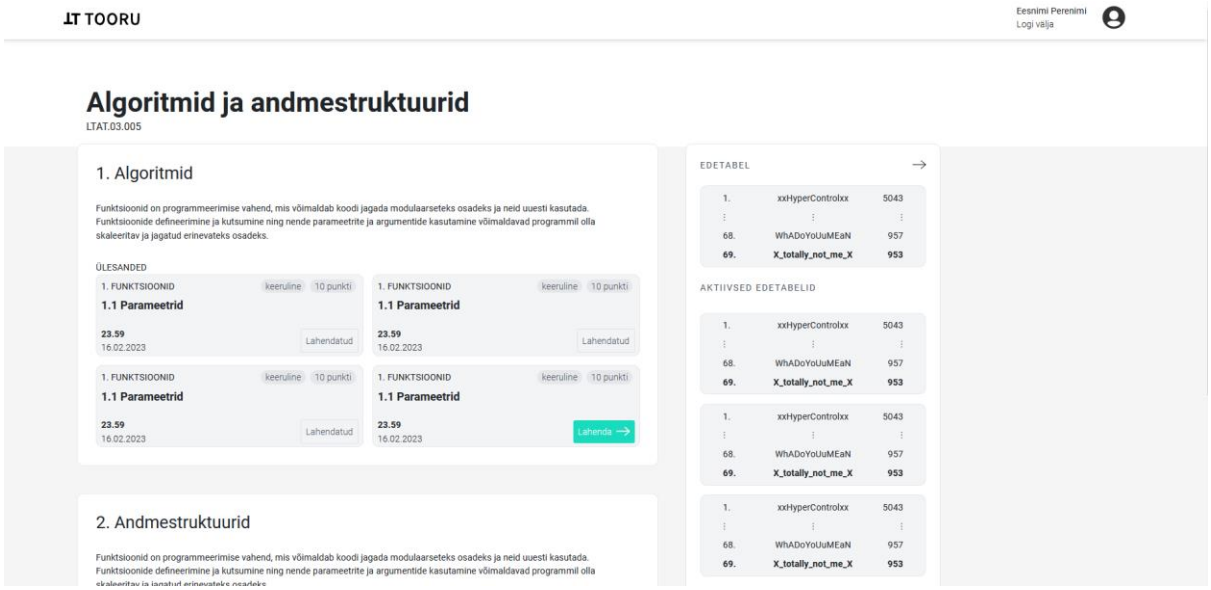


Joonis 9: Veebirakenduse avaleht tudengivaates kõrge-kontrastiga värviskeemis

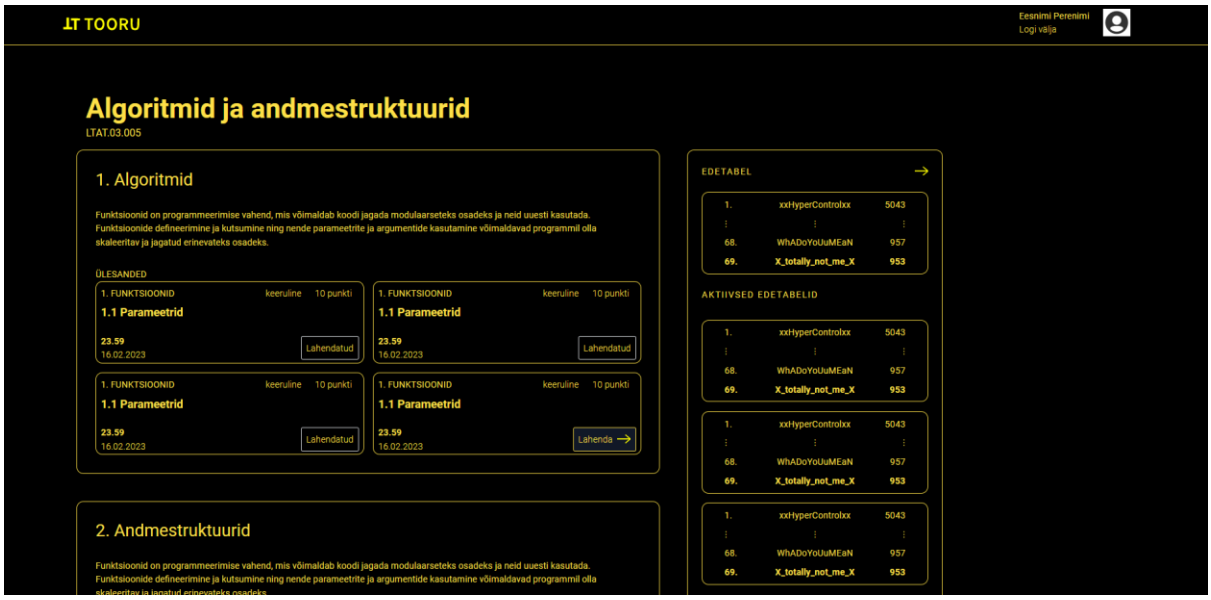
Avalehel kuvatakse tudengi registreeritud aineid ning ajajoont, mis sisaldab ülesandeid. Ülesanded peaksid olema järjestatud kasvavas järjekorras tähtaja alusel. Hetkel kuvatav info on osa

eeldefineeritud andmestikust eesrakenduses. Tulevikus tuleks avalehele jõudes teha päring tagarakendusesse, millele vastatakse sisselogitud tudengi registreeritud kursustega.

5.2.3 Kursuse vaade



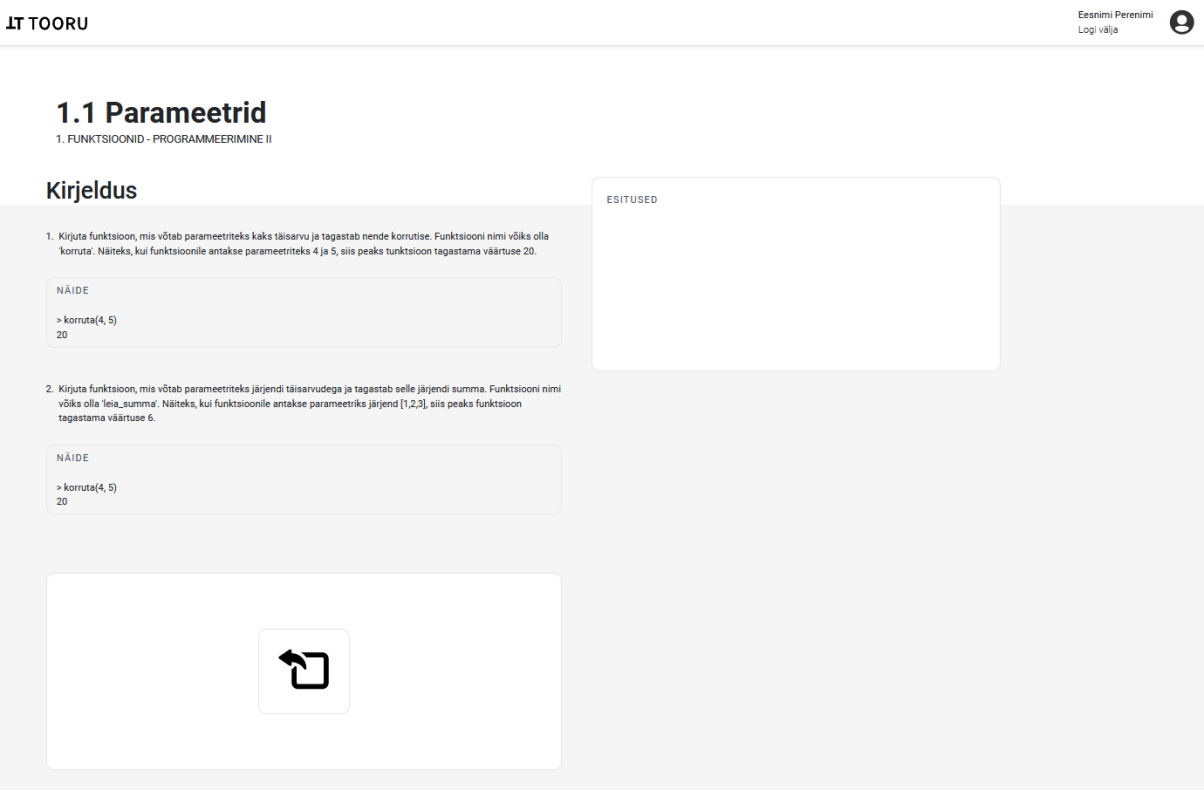
Joonis 10: Veebirakenduse kursuse leht tudengivaates



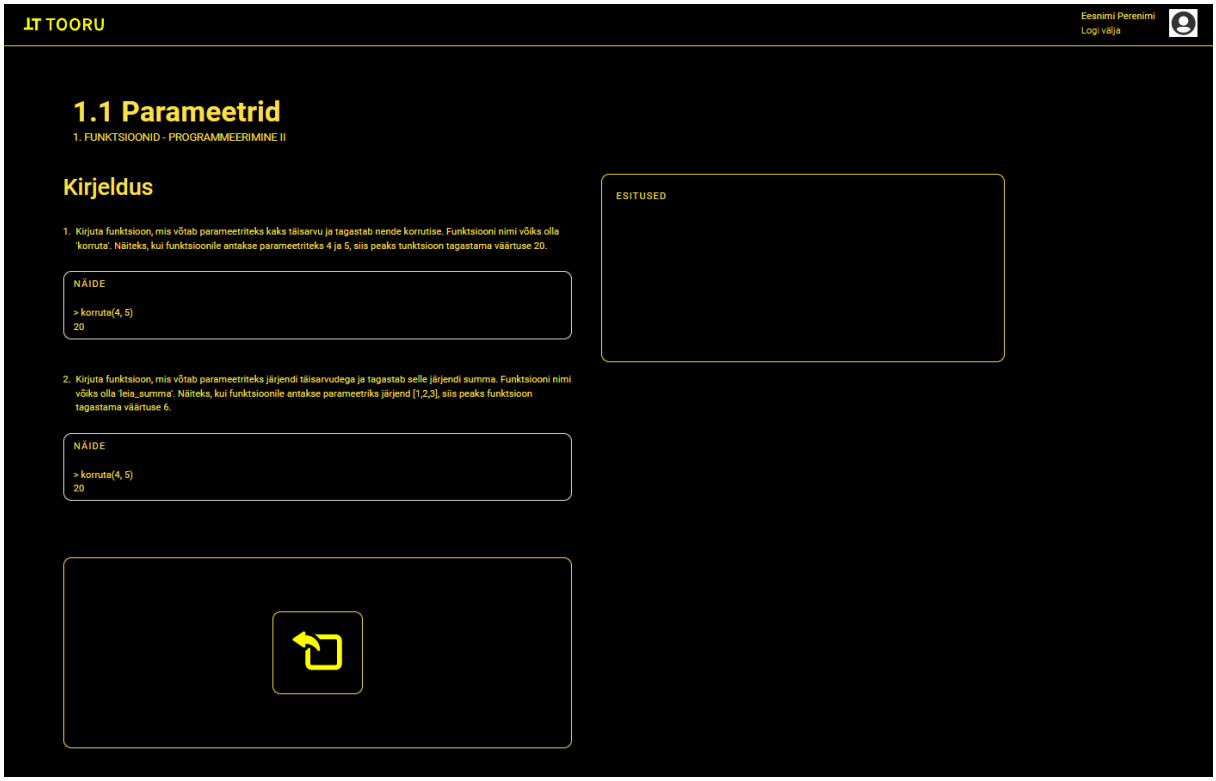
Joonis 11: Veebirakenduse kursuse leht tudengivaates kõrge-kontrastiga värviskeemis

Avades kindla kursuse, antud näites Algoritmid ja andmestruktuurid, avaneb detailsem vaade kursusele. Sellel lehel on järjestatud kursuse jooksul läbitavad teemad ning iga teema sisaldab ülesandeid, mida tudeng peab sooritama. Antud vaates näeb tudeng valitud kursuse edetabeleid ning enda positsiooni neis. Edasiarenduse korral tuleks ka siin teha päring tagarakendusesse, mille vastuseks on kursuse teemad, teemade jooksul sooritatavad ülesanded ning edetabeli informatsioon.

5.2.4 Ülesande vaade



Joonis 12: Veebirakenduse ülesande leht tudengivaates



Joonis 12: Veebirakenduse ülesande leht tudengivaates kõrge kontrastiga värviskeemis

Ülesande vaates on tudengi jaoks lahti seletatud ülesande kirjeldus ning näited oodatavatest tulemustest. Lehe põhjas on tudengil võimalik oma lahenduse fail üles laadida. Hetkel saab faili serverisse üles laadida, kuid antud failiga ei tehta midagi ning seda ei saadeta kuhugi edasi. Tulevikus on ka siia vaja lisada päring tagarakendusesse, mis vastab ülesande detailidega, tudengi eelnevate esitustega ning täiendada faili üleslaadimise funktsionaalsust, et lisatud fail jõuaks õigesse kohta tagarakenduses.

Kokkuvõte

Bakalaureusetöö eesmärk oli luua eesrakendus DeepMOOC platvormile. Töö tulemusena valmis SvelteKit ja TailwindCSS raamistike abil implementeeritud eesrakendus mitme erineva tudengivaatega. Töö kvaliteet tagati Vitest ja Playwright testimisraamistikega, millega kaeti vähemalt 93% kirjutatud koodist. Defineeritud funktsionaal- ning mittefunktsionaalnõuded täideti täies mahus.

Edasise integratsiooni lihtsustamise eesmärgil kasutab eesrakendus eeldefineeritud andmestikku, et kuvada erinevaid komponente, kuid antud andmestik on lihtsasti asendatav tagarakenduse suhtlusega. Implementeeriti ka algeline suhtlus platvormi tagarakendusega sisselogimise funktsionaalsuse võimaldamiseks.

Veebirakendust luues peeti silmas ka juurdepääsetavust, mille jaoks implementeeriti võimalus kasutada kõrge kontrastiga värviskeemi, et aidata nägemisraskustega kasutajaid. Samuti lisati ka vajalik loogika, et ekraanilugejad saaksid oma tööd korrektselt teha ning seeläbi aidata kasutajaid, kel vastavat abi vaja.

Töö esimeses peatükis anti ülevaade olulisematest mõistetest, põhjendati veebirakenduse loomise käigus kasutatud tehnoloogiate valikuid, kirjeldati kahte erinevat automaat testimise meetodit ning selgitati juurdepääsetavuse tähtsust ja selle implementeerimiseks potentsiaalseid lahendusi. Teises peatükis toodi välja kolm lõputööd, mis on varasemalt antud platvormi valmimisse panustanud. Kolmandas peatükis kirjeldati veebirakenduse valmimisel silmas peetud funktsionaal- ning mittefunktsionaal nõudeid. Neljandas peatükis kirjutati ülevaade lahendusteni jõudmise protsessist ning juurdepääsetavuse implementeerimisest. Viimaseks, viiendas peatükis näidati validatsiooni tulemusi, mis tagati automaat testide abil ning anti ülevaade valminud veebirakendusest.

Kasutatud kirjandus

[1] Heavy.ai artikkel server-side-rendering kohta. Kasutatud 16.03.2023, https://www.heavy.ai/technical-glossary/server-side-rendering

[2] Rockcontent blogist võetud illustratsioon server-side-rendering kirjeldamiseks. Kasutatud 16.03.2023, https://rockcontent.com/blog/client-side-rendering-vs-server-side-rendering

[3] Nian Li ja Bo Zhang. 2021. The Research on Single Page Application Front-end development Based on Vue, Journal of Physics Conference Series, doi: 10.1088/1742-6596/1883/1/012030

[4] 2022. aasta State of JavaScript küsitlus. Kasutatud 16.03.2023,

https://2022.stateofjs.com/en-US/libraries/front-end-frameworks/

[5] TailwindCSS ametlik dokumentatsioon. Kasutatud 16.03.2023,

https://tailwindcss.com/docs

[6] Vitesti ametlik dokumentatsioon. Kasutatud 25.03.2023,

https://vitest.dev

[7] Playwright raamistiku ametlik dokumentatsioon. Kasutatud 16.04.2023,

https://playwright.dev/

[8] Vercel platvormi ametlik dokumentatsioon. Kasutatud 16.04.2023,

https://vercel.com/docs

[9] SmartBear veebilehel olev artikkel ühiktestimise kohta. Kasutatud 16.04.2023,

https://smartbear.com/learn/automated-testing/what-is-unit-testing/

[10] Katalon lehel olev artikkel *end-to-end* testimise kohta. Kasutatud 16.04.2023,

https://katalon.com/resources-center/blog/end-to-end-e2e-testing

[11] Mozilla artikkel juurdepääsetavuse põhimõtete kohta. Kasutatud 16.04.2023,

https://developer.mozilla.org/en-US/docs/Learn/Accessibility/What_is_accessibility

[12] Sten Pitteti artikkel “What is code coverage?”. Kasutatud 27.04.2023,

https://www.atlassian.com/continuous-delivery/software-testing/code-coverage

[13] SvelteKit dokumentatsioon. Kasutatud 16.04.2023,

https://kit.svelte.dev/docs/introduction

[14] Andre Anijärv, DeepMOOC platvormile tarkvarakonveieri arendamine, 2022

https://comserv.cs.ut.ee/ati_thesis/datasheet.php?id=74578

[15] Kaarel Kangro, Koondhinnete ning hinnetetabelite moodustamise keel DeepMOOC platvormile, 2022

https://comserv.cs.ut.ee/ati_thesis/datasheet.php?id=75354

[16] Joosep Näks, DeepMOOC platvormile tagarakenduse dispetšeri arendamine, 2022

https://comserv.cs.ut.ee/ati_thesis/datasheet.php?id=75274

[17] Cheng, L.-C., Li W., Tsend, J. C. R. Effects of an automated programming assessment system on the learning performances of experienced and novice learners. Interactive Learning Environments, 2021. https://www.tandfonline.com/doi/full/10.1080/10494820.2021.2006237 (08.08.2022)

[18] Sabag, N, Kosolapov, S. Using instant feedback system and micro exams to enhance active learning. American Journal of Engineering Education (AJEE), 2012, nr 3(2), lk 115-122.

[19] N Wedasinghe, NT Sirisoma, APR Wickramarachchi, A Design Guideline to Overcome Web Accessibility Issues Challenged by Visually Impaired Community in Sri Lanka, https://doi.org/10.48550/arXiv.2304.06924

[20] Geekflare artikkel “How to test your Site with Google Lighthouse?”. Kasutatud 03.05.2023, https://geekflare.com/google-lighthouse/

Lisad

1. Litsents

Lihtlitsents lõputöö reprodutseerimiseks ja üldsusele kättesaadavaks tegemiseks

Mina, **Cardo Tisler**,

1. Annan Tartu Ülikoolile tasuta loa (lihtlitsentsi) minu loodud teose **Veebilehe loomine DeepMOOC platvormile SvelteKit raamistikuga**,

   mille juhendajad on Ahti Põder ja Tõnis Hendrik Hlebnikov

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

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

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

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

Cardo Tisler

01.05.2023

2. GitHub

Töö käigus valminud kood on leitav GitHub-is: https://github.com/CardoTisler/deepmooc-fe