TALLINNA TEHNIKAÜLIKOOL

Infotehnoloogia teaduskond

Kristjan Vool 192368IAPM

Serveri- ja kliendipoolne veebiraamistik C++ keeles koos integreeritud mallimootoriga

Magistritöö

Juhendaja Gert Kanter

Doktorikraad

Tallinn 2021

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: Kristjan Vool

01.01.2022

Annotatsioon

Käesoleva töö eesmärgiks on luua serveri- ja kliendipoolne veebiraamistik, kuhu on integreeritud mallimootor, mida kasutades oleks veebiarendajatel võimalik luua kvaliteetseid veebilehti võimalikult kiiresti, kuna raamistik pakub konkreetset struktuuri ning liidesekomponente, mida oleks võimalik kasutada veebilehe loomisel.

Veebilehtede arendamiseks on võimalik valida väga paljude erinevate veebiraamistike vahel, kuid uue veebilehe loomisel tuleb raamistikke omavahel sobitada ning kulutada aega standardfunktsionaalsuse lisamisele. Antud töös pakutakse välja lahendus, mille abil on võimalik luua uusi veebilehti mugavamalt ning väiksema töömahuga.

Käesoleva töö esimeses pooles keskendutakse olemasolevate veebiraamistike uurimisele ning raamistiku arendamiseks vajaminevate nõuete püstitamisele. Töö teises pooles kirjeldatakse ja kasutatakse loodud raamistikku olemasoleva kliendi veebilehe migreerimiseks uuele süsteemile ning uuritakse, kas loodud raamistik pakub piisavalt funktsionaalsust võrreldes olemasolevate veebiraamistikega.

Magistritöö tulemusena valmib automaattestitud serveri- ja kliendipoolne veebiraamistik koos integreeritud mallimootoriga ning avatud lähtekoodiga tuumkood, mida on võimalik kasutada taoliste veebilehtede arendamiseks.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 39 leheküljel, 7 peatükki, 25 joonist ja 12 tabelit.

Abstract

C++-based Full-Stack Web Framework with Integrated Template Engine

The aim of this thesis is to create a server- and client-side web framework with an integrated template engine that would allow web developers to create high-quality websites as quickly as possible, as the framework provides a specific structure and interface components.

There are many different web frameworks to choose from for developing websites, but when creating a new website, the frameworks need to be matched and time spent adding standard functionality. This work offers a solution that allows you to create new web pages more conveniently and with less workload.

The first half of this paper will focus on exploring existing web frameworks and setting the requirements for developing the framework. The second part of the paper describes and uses the created framework for migrating an existing customer's website to the new system and examines whether the created framework provides sufficient functionality by comparing the framework with existing web frameworks.

The master's thesis results in an automated tested server- and client-side web framework with an integrated template engine and open-source core that can be used to develop such websites.

The thesis is in Estonian and contains 39 pages of text, 7 chapters, 25 figures, 12 tables.

Lühendite ja mõistete sõnastik

API	*Application Programming Interface*, rakendusliides ehk programmiliides ehk rakendustarkvara liides
C++	üldotstarbeline staatiliste andmetüüpidega multifunktsionaalne programmeerimiskeel
CMS	*Content management system*, sisuhaldustarkvara
DBMS	*Database management system*, andmebaasihaldussüsteem
GET	HTTP meetod, mis küsib serverist andmeid
HTML	*HyperText Markup Language*, hüperteksti märgistuskeel
HTTP	*Hypertext Transfer Protocol*, hüperteksti edastusprotokoll
ID	*Identifier*, unikaalne kood
Java	objektorienteeritud programmeerimiskeel
JavaScript	objektorienteeritud programmeerimiskeel, mida kasutatakse peamiselt veebilehtede skriptimiseks
JSON	*JavaScript Object Notation*, lihtsustatud andmevahetusvorming
kompileerima	ühes arvutikeeles kirjutatud lähtekoodi teise arvutikeelde tõlkima
küpsis	*Cookie*, tekstikujuline andmeplokk kliendi veebibrauseris
lokaat	*Locale*, selgesõnaline emakeelse keskkonna mudel ja määratlus
MVC	*Model–view–controller*, tarkvara arhitektuurimuster, mis jagab tarkvararakenduse kolmeks omavahel seotud osaks: mudel, vaade ja kontroller
npm	*Node Package Manager*, JavaScripti programmeerimiskeele jaoks loodud paketihaldur
PHP	*Hypertext Preprocessor*, skriptimiskeel, mida kasutatakse peamiselt serveripoolsetes lahendustes dünaamiliste veebilehtede loomisel
POST	HTTP meetod, mis saadab andmeid kliendilt serverile
päring	*Query*, Andmebaasikeele lause andmebaasist andmete otsimiseks
Python	üldotstarbeline interpreteeritav programmeerimiskeel, mida algselt arendati skriptimiskeeleks
REST	*Representational State Transfer*, Tarkvaraarhitektuuri laad, mis seab veebirakenduse loomisele kindlad piirid
SQL	*Structured Query Language*, struktuurpäringukeel
TypeScript	vabavaraline programmeerimiskeel, mis võimaldab lisada staatilist tüübikirjeldust ja kompileerib Javascriptiks

URI	*Uniform Resource Identifier*, ühtne ressursiidentifikaator
URL	*Uniform Resource Locator*, internetiaadress ehk üldine infoallika asukohamääraja
XML	*Extensible Markup Language*, laiendatav märgistuskeel

Sisukord

Jooniste loetelu

Tabelite loetelu

1 Sissejuhatus

Antud magistritöö eesmärgiks on luua serveri- ja kliendipoolne veebiraamistik koos integreeritud mallimootoriga, mis paneb rõhku kiirusele ja funktsionaalsusele ning võimaldab arendajal alustada uue veebilehe loomist võimalikult efektiivselt, sest raamistik paneb paika kindla struktuuri, mille abil saab raamistikku hõlpsasti ja mitmekülgselt kasutada erinevate projektide raames.

1.1 Taust ja probleem

Tänapäeval arendatavad veebilehed ja -rakendused peaksid olema kõrge kvaliteediga, sest need on ligipääsetavad kõikvõimalikest erinevatest seadmetest üle kogu maailma. Jaanuar 2021 seisuga oli kogu maailmas 4,66 miljardit aktiivset interneti kasutajat - 59,5% maailma elanikkonnast. Sellest kogusest 92,6 protsenti (4,32 miljardit) kasutas internetti mobiilseadmete kaudu. [1]

Igal modernsel veebilehel on konkreetne stiil vastavalt veebilehe omaniku brändile ning disain on loodud nii, et annaks sõnumi kasutajale edasi võimalikult isikupäraselt. Kuigi veebilehed võivad välimuse poolest erineda kohati väga drastiliselt, siis tegelikult on veebilehtedel väga palju sarnasusi, millest kasutaja ei pruugi teadlik olla. Iga veebilehe toimimiseks on vajalik server, mis kommunikeerib brauseriga, andes edasi informatsiooni, millist sisu milliselt lingilt on võimalik vaadata. Veebilehe dünaamilisuse tagavad andmebaasis olevad andmed. Veebilehtede sisud koosnevad paljudel juhtudel sarnastest elementidest ning funktsionaalsusest, nagu näiteks vormid, nupud, rippmenüüd, navigatsiooniriba, otsing, sisselogimine, jmt. Selliste veebilehtede loomiseks, mis sisaldavad endas taolisi elemente ning funktsionaalsust, kasutatakse tavaliselt raamistikke.

Veebi arendamine võib olla suur väljakutse isegi kogenud arendajatele, kuna see on tavaliselt seotud keeruliste raamistike, teekide ja paljude konfiguratsioonifailidega. Lisaks põhineb enamik veebiraamistikke tõlgendatud keeltel ja keerukatel komponentide koostoimetel, mis võivad jõudlust negatiivselt mõjutada. [2]

Sõna "raamistik" on laialdaselt levinud veebiarendamise valdkonnas, mis lahti kirjutatuna võiks tähendada järgnevat: “toetav struktuur, mille abil saab midagi ehitada.” Raamistikud jaotuvad laialdaselt kaheks: serveri- ja kliendipoolseteks. [3] Serveripoolseteks raamistikeks on raamistikud, mida kasutatakse suhtluse toimimiseks serveri ja kliendi vahel ning kliendipoolseteks raamistikeks on raamistikud, mida kasutatakse lõppkasutajale sisu kujundamiseks.

Veebilehtede loomisel on võimalik valida väga paljude erinevate raamistike vahel, mis on iseenesest hea, kuid uue veebilehe loomisel on iga kord vaja uuesti seadistada kasutatavad raamistikud ning kulutada aega baasi loomisele. Mitmed erinevad raamistikud kasutavad antud ülesande lahendamiseks erinevat lähenemist. Kui konkreetse ülesande jaoks valitakse vale raamistik, võib see veebirakendust negatiivselt mõjutada, mõjutades omakorda selle jõudlust ja mastaapsust. [4]

1.2 Ülesande püstitus

Töö eesmärgiks on luua raamistik, mis pakuks veebilehtede loomiseks lahendust, millega oleks võimalik uusi veebilehti luua võimalikult kiiresti. Arendatav raamistik võimaldab alustada uute veebilehtede arendamist olemasolevatest raamistikest palju efektiivsemalt, kuna loodav raamistik pakub modernsete veebilehtede loomiseks järgnevaid vajalikke komponente:

- server, mis sisaldab marsruutide süsteemi, liidestust andmebaasiga;
- mallimootor, mis genereerib valiidse HTML [5] koodi;
- baasliideskomponendid, mida läheb tavaliselt iga veebilehele tegemisel vaja;
- päringute tegemine lõimitud loogikaga;
- keeletugi tõlkefailidena.

1.3 Uurimisküsimused

U1. Kuidas erinevad päringute tegemiste ajad olemasolevatel C++ raamistikel?

U2. Kuidas erinevad olemasolevad mallimootorid jõudluse poolest ja kas nende funktsionaalsus on piisav dünaamiliste veebilehtede arendamiseks?

U3. Kas olemasolevate raamistikega on optimaalne taasluua olemasolev kliendi veebileht?

1.4 Metoodika

Raamistiku arendamisel uuritakse erinevaid olemasolevaid raamistikke, tehakse kindlaks loodava süsteemi nõuded ning pannakse paika kindel süsteemi struktuur, et kood oleks skaleeritav ning loodav kood jaotatakse vastavalt loogilisteks repositooriumiteks. Raamistik kirjutatakse C++ [6] keeles, kuna vajalik on koodi kompileerimine, mille läbi on võimalik tuvastada ilmnenud koodivigu juba enne lõpliku koodi käivitamist. Koodi kirjutamiseks kasutatakse põhiliselt CLion [7] rakendust, kuna CLion pakub soovitusi C++ koodi optimeerimiseks.

Loodud raamistiku testimiseks luuakse automaattestid, mis aitavad vähendada arenduse käigus muudatuste tegemisel tekkivaid võimalikke probleeme ning et süsteemi edasiarendamisel oleks võimalik testida, kas kõik eelnev funktsionaalsus toimib vastavalt nõuetele. Automaattestid kirjutatakse põhiliselt mallisüsteemile, kuna mallisüsteem peab olema võimeline tõlgendama arendaja poolt kirjutatud dünaamilist HTML koodi ning olema töökindel, et lõppkasutajale nähtaval veebilehel ei oleks tõrkeid.

1.5 Ülevaade tööst

Lõputöö on jagatud neljaks osaks.

Lõputöö esimeses osas keskendutakse olemasolevate raamistike uurimisele ja pannakse paika raamistiku funktsionaalsed ja mittefunktsionaalsed nõuded.

Teise osa eesmärk on luua raamistik, mis vastaks eelmises peatükis püstitatud nõuetele ning kirjeldada loodud raamistiku tehnilist osa.

Kolmandas osas analüüsitakse ja testitakse valminud rakendust ning võrreldakse seda teiste olemasolevate raamistikega. Lisaks kirjeldatakse ka ühe veebilehe taasloomist uut raamistikku kasutades.

Lõputöö neljandas osas tuuakse välja raamistiku võimalikud edasiarendused.

2 Tausta analüüs

Raamistiku loomiseks on vajalik tutvuda olemasolevate tehnoloogiate ning raamistikega, et loodav süsteem oleks võrreldes leitud raamistikega funktsionaalselt sama võimekas, kuid samas pakuks lisaväärtusi, et loodav raamistik oleks paremaks alternatiiviks.

2.1 Olemasolevad raamistikud

Antud peatükis võrreldakse olemasolevaid raamistikke ning tuuakse välja nende tugevad ja nõrgad küljed.

Tänapäeval kõige laialdasemalt kasutatavad raamistikud põhinevad JavaScriptil ning nendeks on React.js [8], jQuery [9], Express [10], Angular [11] ning teistel keeltel põhinevad populaarseimad raamistikud on ASP.NET Core [12], Flask [13], Spring [14]. [15]

Raamistike võrdlemisel võetakse arvesse järgnevat funktsionaalsust: andmebaas, mallimootor, tõlked, marsruutimine, küpsised.

Andmebaas on organiseeritud struktureeritud teabe või andmete kogum, mis on tavaliselt salvestatud elektrooniliselt arvutisüsteemi. Andmebaasi juhib tavaliselt andmebaasihaldussüsteem (DBMS). Andmeid ja DBMS-i koos nendega seotud rakendustega nimetatakse andmebaasisüsteemiks, mida sageli lühendatakse lihtsalt andmebaasiks. [16]

Mallimootor võimaldab kasutada staatilisi mallifaile. Käitusajal asendab mallimootor mallifailis olevad muutujad tegelike väärtustega ja teisendab malli kliendile saadetavaks HTML-failiks. Seeläbi on võimalik muuta lehe genereerimine dünaamiliseks. [17]

Mitmekeelne veebileht aitab jõuda suurema vaatajaskonnani. See loob kuvandi rahvusvahelisest publikust ja seeläbi suurendab veebiliiklust ja loob suhteid uute vaatajaskondadega. Samuti aitab teksti lugemine oma emakeeles inimestel paremini veebilehe sisu mõista.

Marsruutimine määrab kindlaks, kuidas rakendus vastab kliendi päringu konkreetsele lõpp-punktile, milleks on URI (või *path*) ja konkreetne HTTP päringu meetod (GET,

POST ja nii edasi).

Igal marsruudil võib olla üks või mitu käitleja funktsiooni, mis käivitatakse, kui sobiv marsruut on leitud. [18]

Dünaamiline URL on veebilehe aadress, mille sisu sõltub muutuvatest parameetritest, mis serverile edastatakse. Parameetrid võivad URL-is endas juba olemas olla või olla kasutaja sisestatud. Dünaamiline URL sisaldab endas sageli teatud märke, mis on nähtavad brauseri aadressiribal. Nendeks märkideks võivad olla näiteks: `& $ + = ? %`. [19]

Veebiküpsis on väike andmeosa, mille server saadab kasutaja veebibrauserile. Brauser võib küpsise salvestada ja saata selle hilisemate päringutega samale serverile tagasi. Tavaliselt kasutatakse HTTP-küpsist selleks, et teha kindlaks, kas kaks päringut pärinevad samast brauserist – näiteks kasutajat hoitakse sisselogituna. See jätab meelde olekuta HTTP-protokolli olekuteabe.

Küpsiseid kasutatakse peamiselt kolmel eesmärgil:

- seansi info – sisselogimised, ostukärud, mängude tulemused või kõik muu, mida server peaks meeles pidama;
- isikupärastamine – kasutaja eelistused, teemad ja muud seaded;
- jälgimine – kasutaja käitumise salvestamine ja analüüsimine. [20]

2.1.1 Express

Express on minimaalne ja paindlik Node.js veebirakenduste raamistik, mis pakub veebi- ja mobiilirakenduste arendamiseks robustseid võimalusi. HTTP utildiidid ning vahevara funktsionaalsus võimaldavad kiirelt ja lihtsalt luua robustseid API-sid. Express pakub lisaks Node.js-ile fundamentaalseid meetodeid veebirakenduste loomiseks. Mitmed populaarsed Node.js raamistikud on ehitatud kasutades Expressi: [10] [21]

- **Feathers** on kergekaaluline veebiraamistik reaalajarakenduste ja REST API-de loomiseks TypeScript või JavaScript keeles; [22]
- **Kraken** on turvaline ja skaleeritav kiht Expressile, mis pakub ühtsemat struktuuri projekti loomisel; [23]

- **Sails** on serveripoolne MVC veebiraamistik praktiliste Node rakenduste loomiseks; [24]
- **NestJS** on samuti serveripoolne veebiraamistik, millega on võimalik luua tõhusaid ja skaleeritavaid Node rakendusi. NestJS pakub konkreetset struktuuri sarnaselt Krakeni raamistikule, läbi mille on võimalik luua paremini testitavaid ja hallatavaid rakendusi. [25]

Expressile pole **andmebaasi** tuge otseselt sisseehitatud ning toe lisamiseks tuleb käsitsi paigaldada sobilik *„npm pakett“*. Populaarseimad toetatud andmebaasid on järgmised: [26]

- **MySQL** on populaarseim avatud lähtekoodiga relatsiooniline andmebaasisüsteem, mis kasutab standardiseeritud programmeerimiskeelt SQL; [27]
- **PostgreSQL** on avatud lähtekoodiga võimas objekt-relatsiooniline andmebaasisüsteem, mis kasutab ning laiendab SQL programmeerimiskeelt. PostgreSQL on 30 arendusaastaga teeninud tugeva reputatsiooni oma arhitektuuri ja usaldusväärsusega; [28]
- **Cassandra** on avatud lähtekoodiga NoSQL hajus andmebaas, mis on mõeldud massiivsete andmemahtudega töötamiseks, pakkudes kõrget käideldavust ning hajutatust; [29]
- **MongoDB** on saadaoleva lähtekoodiga dokument-orieteeritud andmebaas, mis ei kasuta SQL programmeerimiskeelt. MongoDB kasutab andmete hoidmiseks JSON-laadseid dokumente valikuliste skeemidega; [30]
- **Redis** on avatud lähtekoodiga mälusisene andmestruktuuride salvestusruum, mida kasutatakse andmebaasi, vahemälu ja sõnumite vahendajana; [31]
- **MariaDB** on üks populaarsemaid avatud lähtekoodiga relatsioonilisi andmebaase. See on loodud MySQL algsete arendajate poolt ning selle kood on garanteeritud jääma avatuks. MariaDB on saadaval enamikes andmebaasimajutust pakkuvates teenustes ning enamike Linuxi distributsioonide vaikeseade. [32]

Mõned populaarsed **mallimootorid**, mis Expressiga töötavad, on Pug [33], Mustache [34] ja EJS [35]. Expressi rakenduste generaator kasutab vaikimisi Jade'i, kuid see toetab ka mitmeid teisi. [17]

- **Pug** on Node'i ja brauseri mallimootor, mis kompileeritakse HTML-i ja sellel on lihtsustatud süntaks. Pug teeb lihtsaks nii korduvkasutatava HTML-i kirjutamise kui ka andmebaasist või API-st võetud andmete renderdamise. [36]
- **Mustache** on loogikavaba JavaScripti malli süntaks, mida saab kasutada HTML-i, konfiguratsioonifailide, lähtekoodi – kõige jaoks. See toimib, laiendades malli silte, kasutades räsi või objekti väärtusi. See on "loogikavaba", kuna puuduvad tingimuslaused ja silmused, mille asemel on ainult sildid. Mõned sildid asendatakse väärtusega, mõned mitte millegagi ja teised väärtuste jadadega. [34]
- **EJS** on lihtne mallikeel, mis võimaldab luua HTML-märgistust tavalise JavaScriptiga [35].

Expressi kõige enim levinud **tõlgete toe** "*npm pakett*" on "*express-translate*", mille nädalane allalaadimiste arv on ~500 [37], mis on võrreldes populaarsemate npm pakettidega väga vähe. Seega saab järeldada, et see ei ole Expressi arendajate seas väga levinud.

Express-translate tõlgib võtmed, mis on objektis määratud võtme => tõlkestringi vastenduses valitud keelte jaoks. Toetatud on stringide interpoleerimine ja kogu pahatahtlik sisu on vaikimisi HTML-ist välja jäetud.

Marsruudi parameetrid on fikseeritud osad URL-ist, mida kasutatakse URL-is nende osade väärtustamiseks. Jäädvustatud väärtused sisestatakse objekti „*req.params*", kusjuures marsruudi parameetri nimi on määratud teele vastavate võtmetena. Express toetab dünaamilisi marsruute läbi regulaaravaldiste ning parameetrite nimede. [18]

Küpsiste kasutamiseks veebis on olemas npm pakett nimega "*cookie-parser*", mille nädalane allalaadimiste arv on ~2.3 mln [38]. Nagu näha, siis võrreldes eelpool mainitud express-translate'ga on see arv oluliselt suurem.

2.1.2 Flask

Flask on tasuta avatud lähtekoodiga kiire ja paindlik Pythoni mikro-veebiraamistik. Flask on algselt välja töötatud Pinaxi arendajate poolt ja on välja antud avatud lähtekoodiga platvormina 2011. aasta juulis.

Mikroraamistikus olev "mikro" tähendab, et Flaski eesmärk on hoida tuum lihtsana, kuid laiendatavana. Flask ei tee arendaja eest palju otsuseid ära ja kõik otsused on lihtsasti muudetavad. Arendaja seadistada on kõik muu, ilma millegi üleliigseta.

Vaikimisi ei sisalda Flask andmebaasi abstraktsioonikihti, vormi valideerimist ega midagi muud, kus on juba olemas erinevad teegid, mis sellega hakkama saavad. Selle asemel toetab Flask laiendusi, et lisada rakendusele selline funktsionaalsus, nagu oleks see Flaskis endas rakendatud. Arvukad laiendused pakuvad andmebaasi integreerimist, vormide valideerimist, üleslaadimise käsitlemist, erinevaid avatud autentimistehnoloogiaid ja palju muud. [39]

Flask'i on sisseehitatud SQLite **andmebaas**, mis ei nõua eraldi andmebaasiserveri seadistamist. Kui aga samaaegsed päringud üritavad samal ajal andmebaasi kirjutada, aeglustuvad need, kuna iga kirjutamine toimub järjestikku. Väikese rakenduse puhul ei ole see tähtis, kuid suuremate rakenduste puhul tuleb kasutada mõnda teist andmebaasi, vastasel juhul on süsteem väga aeglane. [40]

Flask konfigureerib Jinja2 [41] **mallimootori** automaatselt. Malli renderdamiseks saab kasutada meetodit *render_template()*. Peab vaid sisestama malli nime ja muutujad, mida mallimootorile märksõnaargumentidena edastada.

Mallides on juurdepääs konfiguratsiooni, päringu, seansi ja objektidele ning funktsioonidele kasutades *url_for()* ja *get_flashed_messages()*.

Kui nimi sisaldab HTML-i, eemaldatakse see automaatselt. Kui tegemist on turvalise HTML-iga, on võimalik see turvaliseks märkida, kasutades klassi Markup või kasutades mallis |*safe* filtrit. [42]

Flask-Babel on Flaski laiendus, mis lisab igale Flaski rakendusele rahvusvahelistumise ja lokaliseerimise toe. Phrase on **tõlkehaldustööriist**, millel on võimas kontekstisisene redaktor, mis muudab tõlkimise protsessi mugavamaks. [43]

Kaasaegsed veebirakendused kasutavad kasutajate abistamiseks sisukaid URL-e. Kasutajad tulevad tõenäolisemalt tagasi, kui leht kasutab tähenduslikku URL-i, mida nad mäletavad ja mida lehe külastamiseks otse kasutada.

Võimalik on muuta **URL**-i osad **dünaamiliseks** ja lisada funktsioonile mitu reeglit. [42]

Küpsistele juurdepääsuks on võimalik kasutada küpsiste atribuuti. Küpsiste seadmiseks on võimalik kasutada vastuseobjektide meetodit „*set_cookie*“. Päringuobjektide küpsiste atribuut on sõnastik kõigi kliendi edastatavate küpsistega. Kui on soov kasutada seansse, siis on vajalik kasutada Flaski seansse, mis lisavad küpsistele vajaliku turvalisuse. [42]

2.1.3 Spring

Spring on avatud lähtekoodiga Java veebiraamistik, mis muudab programmeerimise kiiremaks, lihtsamaks ja turvalisemaks. Spring'i keskendumine kiirusele, lihtsusele ja tootlikkusele on teinud sellest maailma populaarseima Java raamistiku.

Spring'i paindlikke teeke usaldavad arendajad üle kogu maailma ning see pakub iga päev miljonitele lõppkasutajatele meeldivaid kogemusi – olgu selleks siis teleri voogesitus, veebiostlemine või lugematu hulk muid uuenduslikke lahendusi. Spring'i on panuse andnud ka kõik tehnikavaldkonna suured nimed, sealhulgas Alibaba, Amazon, Google, Microsoft ja teised.

Spring'i paindlik ja kõikehõlmav laienduste komplekt ja kolmandate osapoolte teegid võimaldavad arendajatel luua peaaegu kõiki võimalikke rakendusi. Põhimõtteliselt loovad Spring Frameworki juhtimise inversioon (IoC) ja sõltuvuse süstimise (DI) funktsioonid aluse laiaulatuslikule funktsioonide ja funktsioonide komplektile. Olenemata sellest, kas luua veebi jaoks turvalisi, reaktiivseid pilvepõhiseid mikroteenuseid või ettevõtte jaoks keerulisi andmevoogusid, on Springil abiks tööriistad. [14]

Integreeritud **andmebaas** on oma kerge olemuse tõttu kasulik projekti arendusfaasis - lihtne konfigureerida, kiire käivitusaeg, testitavus on vaid mõned integreeritud andmebaasi kasutamise eelised.

Pakett „*org.springframework.jdbc.datasource.embedded“* pakub tuge manustatud Java andmebaasimootoritele. HSQL-i [44], H2 [45] ja Derby [46] tugi on saadaval platvormipõhiselt. Uute manustatud andmebaasitüüpide ja andmeallika rakenduste ühendamiseks on ka laiendatav API. [47]

- **HSQLDB (HyperSQL DataBase)** on Java keeles kirjutatud SQL relatsiooniandmebaasisüsteem. [44]

- **H2** on Java SQL andmebaas, mille peamised omadused on väga kiire, avatud lähtekoodiga JDBC API. [45]
- **Apache Derby** on Apache DB alamprojekt, mis on avatud lähtekoodiga relatsiooniandmebaas ning on täielikult arendatud Javas. [46]

Üks valdkondi, milles Spring paistab silma, on vaatetehnoloogiate eraldamine ülejäänud MVC raamistikust. **Renderdamisüsteemina** saab kasutada näiteks: [48]

- **JavaServer Pages** võimaldab dünaamilist sisu sisestada staatilisesse sisusse, kasutades Javat ja Java Servlette. [49]
- **Groovy Markup Templates** on mallimootor, mille peamine eesmärk on genereerida XML-i sarnaseid märgistusi. [50]
- **Thymeleaf** on kaasaegne serveripoolne Java mallimootor nii veebi- kui ka eraldiseisvate keskkondade jaoks. [51]

Springi rakenduse **lokaliseerimiseks** on vajalik kogu staatiline tekst ekstraktida mallidest lokaliseerimisfaili. Hiljem kasutatakse seda vajalike tekstibittide otsimiseks vastavalt valitud keelele. Vaikefailivorminguks on Spring *MessageSource*'i ressursipaketid, mida kasutatakse enamiku Java rakenduste lokaliseerimiseks. Springil on ka suurepärane tugi nende failide tõlgete talletamiseks ja pärimiseks.

Et rakendus teaks, millist keelt praegu kasutada, tuleb konfigureerida *LocaleResolver*. *SessionLocaleResolver* salvestab valitud lokaadi (= keele) kasutaja seanssi.

Lokaati saab muuta URL-is päringuparameetri kaudu valitavaks lisades *LocaleChangeInterceptor'*i. See pealtkuulaja võimaldab praeguse lokaadi üle kirjutada, pakkudes *lang* parameetri kaudu lokaadi nime.

Oma projektile teise keele lisamiseks tuleb luua koopia esialgsetest „*messages.properties*“-st. Järgmisena lisada lokaadi nimi (st „*messages_de.properties“*) ja tõlkida faili sisu. [52]

Spring'is saab kasutada **dünaamilisi URL-e** ning on võimalik parameetritele määrata ka vastavad muutuja tüübid. [53]

Küpsiste kasutamiseks tuleb kõigepealt seadistada *Spring Session*. Pärast seadistamist on võimalik seansiküpsist konfigureerida läbi *CookieSerializer* klassi. [54]

2.2 Olemasolevad mallimootorid

Antud peatükis võrreldakse olemasolevaid mallimootoreid C++ keeles ning tuuakse välja nende tugevad ja nõrgad küljed.

2.2.1 Jinja2C++

Jinja2C++ on avatud lähtekoodiga mallimootor, mis on kirjutatud C++ keeles ning on loodud eesmärgiga implementeerida täpselt sama funktsionaalsust nagu Jinja2 mallimootoril. [55]

Jinja2C++ kasutatavuse analüüsimiseks klooniti alla kõige uuem stabiilne versioon 1.2.1, mille kompileerimisel esinesid vead, kuna CMake [56] kompileerimisskript oli vigane ning vajas manuaalselt süntaksi parandamist. Pärast süntaksi vigade eemaldamist kompileerus mallimootor ning oli kasutatav, kuid esmamulje võib mõjutada mallimootori kasutamist uute projektiga, kui selle kasutamiseks on vajalik kompileerida palju suuremahulisi teeke ning parandada kompileerimisskripte.

Jinja2C++ arendamiseks on kasutatud mitmeid suuremahulisi väliseid teeke nagu näiteks Boost ning C++ standardfunktsionaalsuse asemel on kasutatud *fmtlib/{fmt}* ning *std* võimalusi taasloovaid teeke. JSON toe lisamiseks on kasutatud *„nlohmann/json“* [57] teeki, mis on populaarseim JSON-i teek C++ keeles kasutamiseks.

Samuti peab ära märkima, et ametlik Jinja2C++ veebileht [58] pole hetkeseisuga kättesaadav, kuigi seda on mainitud arendusrepositooriumi lehel ning seda veebilehte on ka otse viidatud paigaldamise juhiste, täpsemate võimaluste ning dokumentatsiooni vaatamiseks.

Vastates uurimisküsimusele U3, ei ole see mallimootor eelnevalt mainitud puuduste tõttu optimaalne olemasoleva kliendi veebilehe taasloomiseks.

2.2.2 Inja

Inja on samuti Jinja mallimootorist inspireeritud C++ keeles kirjutatud mallimootor, kuid selle eesmärgiks ei ole tagada täpselt sama funktsionaalsust nagu Inja mallimootoril.

Mallimootoril pole suuri väliseid teeke nagu Jinja2C++ mallimootoril, mille tõttu on Inja mallimootorit kasutada C++ koodipäist kaasates, läbi mille on Inja kasutamise alustamine tunduvalt lihtsam.

Sellel on olemas piisavalt funktsionaalsust nagu näiteks muutujad, silmused ning tingimuslaused, mis tagavad selle sobivuse kasutamaks väikesemamahuliste projektide arendamiseks.

Inja kasutab malli andmete sisestamiseks samuti „*nlohmann/json*“ teeki, kuid sellel puudub tugi dünaamilise JSON andmete edastamiseks otse mallimootorist ning selle funktsionaalsuse arendamine ei ole ka plaanis [59], mille tõttu ei ole Inja sobilik dünaamiliste JSON tuge vajavate veebilehtede loomiseks.

3 Nõuete analüüs

3.1 Funktsionaalsed nõuded

Antud peatükis kirjeldatakse süsteemile püstitatud funktsionaalseid nõudeid.

3.1.1 Serveri pordi muutmine

Tabel 1. FR1 nõue: Serveri pordi muutmine

Nõude ID	**FR1**
Nõue	Arendajal peab olema võimalik serveri porti muuta.
Tegutseja	Arendaja
Eeltingimus	Arendajal on ligipääs serveri failidele
Järeltingimus	Serveri port on muudetud
Põhistsenaarium	Arendaja avab serveri konfiguratsiooni faili ning muudab pordi numbri ära.

3.1.2 Tingimuslaused

Tabel 2. FR2 nõue: Tingimuslaused

Nõude ID	**FR2**
Nõue	Arendajal peab olema võimalik defineerida erinevaid tingimuslauseid.
Tegutseja	Arendaja
Eeltingimus	Mallimootor toetab tingimuslauseid.
Järeltingimus	Uus tingimuslause on loodud.
Põhistsenaarium	Arendaja avab faili ning lisab sinna mõne tingimuslause.

3.1.3 Iteratsioonid

Tabel 3. FR3 nõue: Iteratsioonid

Nõude ID	**FR3**
Nõue	Arendajal peab olema võimalik defineerida erinevaid iteratsioone.
Tegutseja	Arendaja

Eeltingimus	Mallimootor toetab iteratsioone.
Järeltingimus	Uus iteratsioon on loodud.
Põhistsenaarium	Arendaja avab faili ning lisab sinna mõne iteratsiooni.

3.1.4 Funktsioonid

Tabel 4. FR4 nõue: Funktsioonid

Nõude ID	**FR4**
Nõue	Arendajal peab olema võimalik defineerida erinevaid funktsioone.
Tegutseja	Arendaja
Eeltingimus	Mallimootor toetab funktsioone.
Järeltingimus	Uus funktsioon on loodud.
Põhistsenaarium	Arendaja avab faili ning lisab sinna mõne funktsiooni.

3.1.5 Marsruudid

Tabel 5. FR5 nõue: Marsruudid

Nõude ID	**FR5**
Nõue	Arendaja peab saama defineerida erinevaid marsruute.
Tegutseja	Arendaja
Eeltingimus	Mallimootor toetab marsruute.
Järeltingimus	Uus marsruut on süsteemi lisatud.
Põhistsenaarium	Arendaja avab serveri faili ning lisab sinna marsruudi.

3.1.6 Andmebaasi laiendamine

Tabel 6. FR6 nõue: Andmebaasi laiendamine

Nõude ID	**FR6**
Nõue	Eeldefineeritud andmebaaside klassi peab olema võimalik laiendada.
Tegutseja	Arendaja
Eeltingimus	Eeldefineeritud andmebaasi klass on olemas, mida soovitakse laiendada.

Järeltingimus	Uus klass, mis laiendab eeldefineeritud andmebaasi klassi on loodud.
Põhistsenaarium	Arendaja loob uue faili, kuhu lisab uue klassi, mis on laienduseks juba olemasolevale andmebaasi klassile.

3.2 Mittefunktsionaalsed nõuded

Antud peatükis kirjeldatakse süsteemile püstitatud mittefunktsionaalseid nõudeid.

3.2.1 Turvalisus

Vaikimisi pole serveris ükski fail lubatud, et ei pääseks failidele ligi kui selleks pole eraldi luba antud.

Tabel 7. NFR1 nõue: Turvalisus

Nõude ID	**NFR1**
Nõue	Kasutaja pääseb ainult nendele failidele ligi, milleks on luba antud.
Tegutseja	Lõppkasutaja
Eeltingimus	Süsteemi kasutaja olemasolu
Järeltingimus	Lubatud failide sisu on nähtav ning mittelubatud failide sisu ei ole kättesaadav.
Põhistsenaarium	Kasutaja siseneb süsteemi ja tahab saada infot. Kui tal on ligipääs sellele infole olemas, siis see kuvatakse talle ekraanile. Kui tal ei ole ligipääsu, siis ta ei näe, et selline asi üldse eksisteeriks.

3.2.2 Paralleelühendused

Server toetab paralleelsete ühenduste saamist ja saatmist. Vastasel juhul peab iga järgneva ühenduse jaoks eelnev ühendus olema lõpule viidud.

Tabel 8. NFR2 nõue: Paralleelühendused

Nõude ID	**NFR2**
Nõue	Server toetab paralleelsete ühenduste saamist ja saatmist.
Tegutseja	Mitu lõppkasutajat samaaegselt

Eeltingimus	Iga serverisse sissetuleva ühenduse jaoks luuakse uus lõim.
Järeltingimus	Ühe kliendi aeglane internet ei mõjuta teise kliendi päringu tegemise kiirust.
Põhistsenaarium	Mitu lõppkasutajat kasutavad samaaegselt süsteemi. Ühel kasutajal läheb päringu tegemisega väga kaua aega, kuid teisi kasutajaid see ei mõjuta.

3.2.3 Päringu kiirus

Kohalikus masinas serverilt vastuse saamine peab võtma aega vähem kui 100 ms.

Tabel 9. NFR3 nõue: Päringu kiirus

Nõude ID	**NFR3**
Nõue	Kohalikus masinas serverilt vastuse saamine peab võtma aega vähem kui 100 ms.
Tegutseja	Lõppkasutaja
Eeltingimus	Serverile on tehtud päring.
Järeltingimus	Serverilt on saadud vastus kiiremini kui 100ms.
Põhistsenaarium	Lõppkasutaja saadab serverile päringu. Server tagastab vastuse 100ms jooksul.

3.2.4 Skaleeritavus

Süsteem peab olema üksteisest sõltumatumaks arendamiseks jaotatud kolmeks osaks: tuum, raam ja klient

Tabel 10. NFR4 nõue: Skaleeritavus

Nõude ID	**NFR4**
Nõue	Süsteem peab olema üksteisest sõltumatumaks arendamiseks jaotatud kolmeks osaks: tuum, raam ja klient.
Tegutseja	Arendaja
Eeltingimus	Süsteemi on võimalik jaotada erinevatesse osadesse.
Järeltingimus	Süsteem on jaotatud kolmeks osaks.

Põhistsenaarium	Süsteem on jaotatud kolmeks osaks ning eriklientide jaoks on võimalik kasutada sama tuum- ja raamkoodi.

3.2.5 Mitmekeelsustugi

Süsteemis on sisseehitatud tugi keelte ja tõlkefailide kasutamiseks.

Tabel 11. NFR5 nõue: Mitmekeelsustugi

Nõude ID	**NFR5**
Nõue	Saab kasutada erinevaid keeli ja tõlkefaile.
Tegutseja	Arendaja, lõppkasutaja
Eeltingimus	Arendaja on süsteemi sisestanud erinevad keeled ja tõlkefailid.
Järeltingimus	Süsteemis on sisseehitatud tugi keelte ja tõlkefailide kasutamiseks ning keeled ja tõlkefailid on sisestatud.
Põhistsenaarium	Lõppkasutajal on võimalik navigeerida lehel erinevate keelte vahel.

3.2.6 Mallimootori automaattestid

Süsteem peab olema kaetud automaattestidega ning koodi kaetus peab olema 100%. See vähendab võimalust, et süsteemi uuendamisel ja muudatuste tegemisel midagi lakkaks toimimast.

Tabel 12. NFR6 nõue: Mallimootori automaattestid

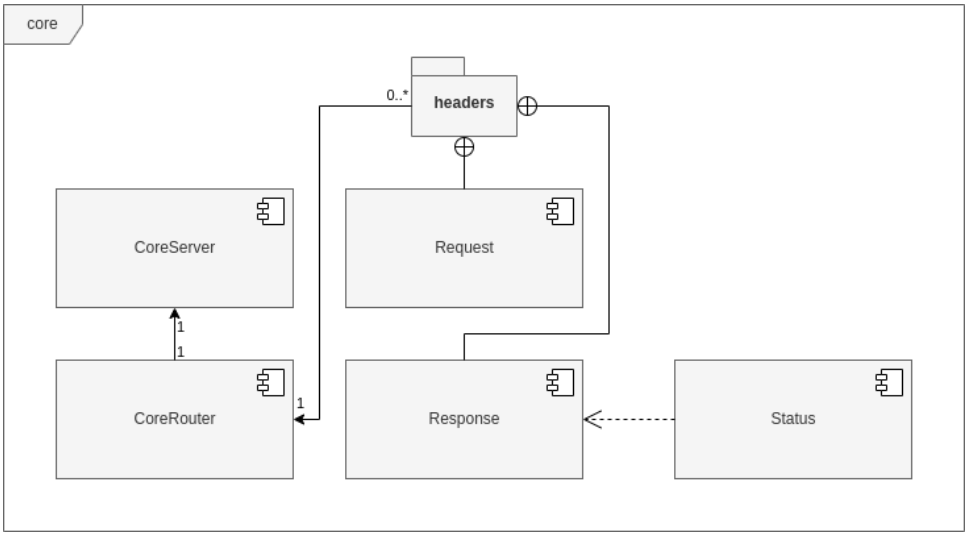
Nõude ID	**NFR6**
Nõue	Süsteem peab olema kaetud automaattestidega ning mallimootori koodikaetus peab olema vähemalt 90%.
Tegutseja	Arendaja
Eeltingimus	Mallimootorit on võimalik testida.
Järeltingimus	Mallimootorile on kirjutatud automaattestid, mille koodikaetus on vähemalt 90%.
Põhistsenaarium	Arendaja loob automaattestid mallimootorile ning loodud testide koodikaetus on vähemalt 90%.

4 Raamistiku teostus

Süsteemi teostuse saab jaotada konkreetseteks etappideks, et arendatav süsteem vastaks püstitatud nõuetele ning oleks võimalikult skaleeritav ning kohandatav vastavalt arendaja äranägemisele.

4.1 Tuuma ülesehitus

Tuumkood on võimalikult minimalistliku serveri käivitamiseks ning tööshoidmiseks, mis pakub võimalust marsruutimiseks ning päringutele vastamiseks. Tuumkood koosneb viiest moodulist (Joonis 1), mis panevad aluse loodavale raamistikule. Tuumkood on kättesaadav avalikult GitHubi repositooriumi lehelt[1].



Joonis 1. Tuuma ülesehitus

4.1.1 Server

Kõige algsem ning tähtsam klass loodava süsteemi jaoks on põhituuma server, mille eesmärgiks on serveri käivitamine, defineerides operatsioonsüsteemi jaoks kasutatavad soklid, läbi mille on võimalik ühendada ning teha päringuid serverisse. Serveri klass hoolitseb selle eest, et sokli defineerimine, loomine ning seadistamine õnnestuksid ning

[1] https://github.com/kristjan-vool/core

arendaja poolt valitud porti oleks võimalik pealt kuulata, et päringute tegemine serverile oleks võimalik. Teine tähtis osa serverist on funktsionaalsus serverit jäädavalt töös hoida, kuni server pannakse kinni arendaja poolt või ilmunud vea tõttu. Vaikimisi luuakse iga ühenduse jaoks uus lõim, et serverisse tulevad ühendused oleksid üksteisest sõltumatud ning ei mõjutaks üksteise jõudlust.

4.1.2 Marsruutija

Ruuteri klassi eesmärgiks on vastata sissetulevatele päringutele. Ruuteri klass hoolitseb ka selle eest, et marsruutimine toimiks ehk kuidas kindlale aadressile vastatakse. Vaikimisi vastatakse päringutele veakoodiga *404*, kui vastavat marsruuti pole eelnevalt defineeritud.

4.1.3 Päised

Iga sissetuleva ühenduse jaoks luuakse *Request* ja *Response* objektid, läbi mille on võimalik lugeda teavet ühenduse kohta ning saata vastus loodud ühendusele.

Saabuva ühenduse jaoks on *Request* klass, mis hoiab endas teavet päringu kohta. Kõige tähtsamaks päringu informatsiooniks loetakse HTTP meetodit, URL-i, HTTP versiooni, päringu sisu, olemasolevad brauseri küpsised.

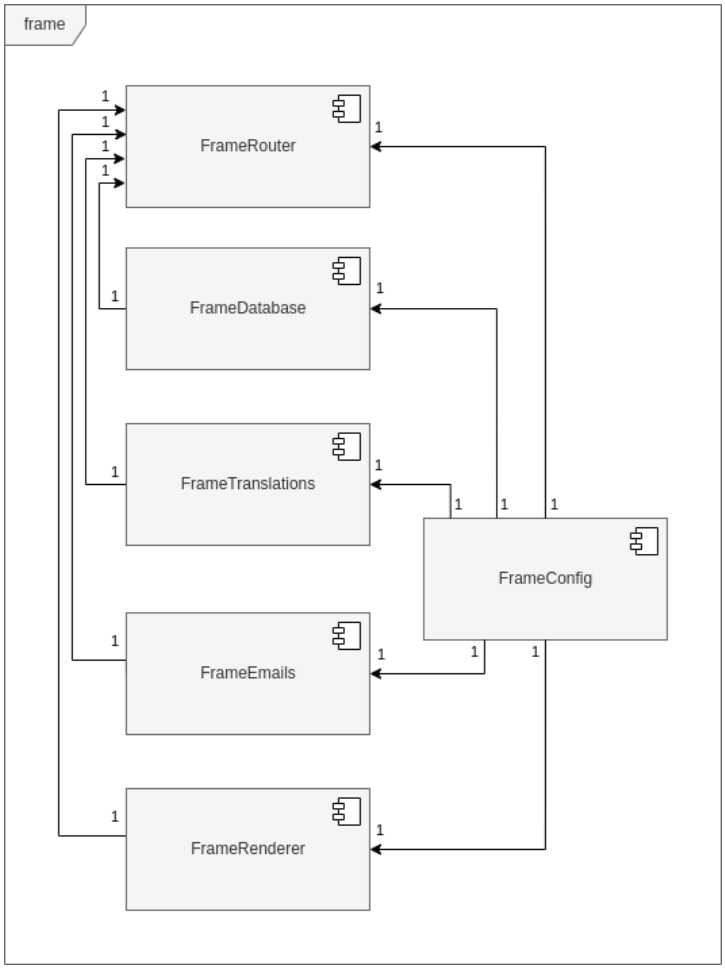
Ühendusele vastamiseks on *Response* klass, mis sisaldab endas vajaminevat funktsionaalsust, et ühendusele vastata vastavalt HTTP standarditele. Vastuse saatmisel on võimalik defineerida saadetava sisu tüüpi (vaikimisi *text/html*), staatuskood (vaikimisi *200 – OK*), sisu ennast ning brauseriküpsiseid.

4.1.4 HTTP staatus

Abistav klass, kuhu on defineeritud HTTP staatuskoodide kirjeldused vastavalt HTML standarditele, mis on vajalikud HTTP päringutele vastamiseks. Vastuse saatmisel on HTTP staatuskoodi määramine kohutuslik, kuna see on üks osa paika pandud HTML suhtlusprotokollist.

4.2 Baasi ülesehitus

Põhikood, mis täiendab tuumkoodi, lisades juurde moodulid toetamaks suhtlust andmebaasiga, mitmekeelsustuge, tõlkefailide kasutatavust, mallimootorit ja liidesekomponente (Joonis 2).



Joonis 2. Baasi ülesehitus

4.2.1 Seadistusklass

Seadistusklass, mis defineerib väljad, läbi mille on võimalik seadistada serveri toimimist. Saab määrata, millised kaustad ja failitüübid on lubatud ning seadistada meiliaadressi hosti, kasutajanime, parooli ja porti. Vastavalt sellele, millist andmebaasi kasutada, on eraldi defineeritud seadistusobjektid erinevate andmebaaside jaoks.

4.2.2 Andmebaas

Virtuaalklass, kus on defineeritud virtuaalsed meetodid, mida on vaja implementeerida. Kaasas on automaatne tugi MongoDB ja MariaDB jaoks. Automaatse andmebaasi toeks on vajalik kasutada defineeritud struktuuri.

MariaDB

Lehtede tabeli struktuur (Joonis 3), mis on vajalik päringute tegemisel lehe kuvamiseks. Näidistabelis on *"_est"* ja *"_eng"* veerud vajalikud mitmekeelsustoe jaoks. *"inline"* veerg on otse sisestatud teksti kuvamiseks lehel, *"redirect"* on ümbersuunamiseks teise URL peale, *"mask"* on olemasoleva URLi maskeerimiseks sisestatud URLi peale, mis kuvab teise lehe sisu praegusel lehel, *"content"* (Joonis 4) on määramaks, kuidas sisu kuvatakse *("inline", "redirect", "mask").*

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra
1	**id**	int(11)			No	*None*		AUTO_INCREMENT
2	**title_est**	text	utf8_general_ci		No	*None*		
3	**url_est**	text	utf8_general_ci		No	*None*		
4	**title_eng**	text	utf8_general_ci		No	*None*		
5	**url_eng**	text	utf8_general_ci		No	*None*		
6	**inline**	text	utf8_general_ci		Yes	*NULL*		
7	**redirect**	text	utf8_general_ci		Yes	*NULL*		
8	**mask**	text	utf8_general_ci		Yes	*NULL*		
9	**content**	int(11)			No	1		
10	**position**	int(11)			No	0		

Joonis 3. Andmebaasi struktuur: pages

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra
1	id	int(11)			No	*None*		AUTO_INCREMENT
2	type	text	utf8_general_ci		No	*None*		

Keyname	Type	Unique	Packed	Column	Cardinality	Collation	Null	Comment
PRIMARY	BTREE	Yes	No	id	6	A	No	
pages_content	BTREE	No	No	content	6	A	No	

Joonis 4. Andmebaasi struktuur: pages_content

Vaated (Joonis 5) defineeritakse, et ühte vaadet oleks võimalik kasutada mitme lehe peal, et poleks vajalik koodi dubleerida (Joonis 6). Vaated seotakse relatsiooniliselt lehega (Joonis 7), et kuvada lehe sisu vaadetena õiges järjekorras.

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra
1	id	int(11)			No	*None*		AUTO_INCREMENT
2	view	varchar(100)	utf8_general_ci		No	*None*		
3	title_est	text	utf8_general_ci		No	*None*		
4	description_est	text	utf8_general_ci		No	*None*		
5	title_eng	text	utf8_general_ci		No	*None*		
6	description_eng	text	utf8_general_ci		No	*None*		

Joonis 5. Andmebaasi struktuur: views

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra
1	id	int(11)			No	*None*		AUTO_INCREMENT
2	page_id	int(11)			No	*None*		
3	view_id	int(11)			No	*None*		
4	position	int(11)			No	*None*		

Joonis 6. Andmebaasi struktuur: pages_views

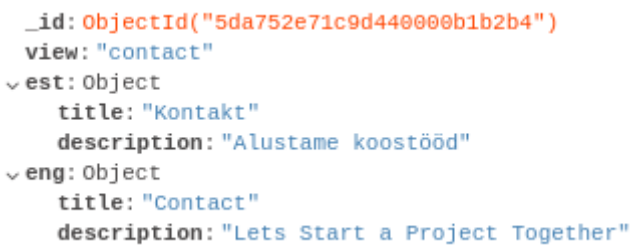
Keyname	Type	Unique	Packed	Column	Cardinality	Collation	Null	Comment
PRIMARY	BTREE	Yes	No	id	22	A	No	
page_id	BTREE	No	No	page_id	22	A	No	
view_id	BTREE	No	No	view_id	22	A	No	

Joonis 7. Andmebaasi struktuur: pages_views relations

MongoDB

Lehtede kollektsioon (Joonis 8 ja Joonis 9), mis on vajalik, et kasutada raamistikku lisatud MongoDB andmebaasi klassi automaatselt. Sarnaselt MongoDB struktuurile on vajalik defineerida vajalikud väljad mitmekeelsustoe jaoks.

Joonis 8. MongoDB andmebaasi lehtede struktuur: lehe näidisdokument



Joonis 9. MongoDB andmebaasi vaadete struktuur: vaate näidisdokument

4.2.3 E-kirjad

Klass, mis seadistab vajaminevad andmed e-kirjade saatmiseks läbi seadistusklassi ning pakub lihsustatud meetodeid meilide saatmiseks, põhinedes avatud lähtekoodiga *"karastojko/mailio"* [60] teegile.

4.2.4 Tõlketugi

Genereerib automaatselt tõlked mallimootorile kasutamiseks. Tõlked imporditakse sisse serveri käivitamisel *.po* laiendiga tõlkefailidest.

4.2.5 Mallimootor

Mallimootor, mis sisaldab endas erinevat funktsionaalsust dünaamiliste lehtede loomiseks, kasutab andmete vahendamiseks *"nlohmann/json"* JSON teeki ning sisaldab endas tavapärast funktsionaalsust nagu muutujaid, pesastatud objekte andmete saatmisel, massiive, tingimuslauseid, silmuseid ja meetodeid. Kaasatulevad meetodid sisaldavad endas vajaliku tuge tõlgete ja liidesekomponentide jaoks.

Loodud mallimootor erineb olemasolevatest mallimootoritest lisaks funktsionaalsusele ka süntaksi poolest. Olemasolevad mallimootorid kasutavad koodiplokkide, nagu näiteks silmuste ja tingimuslausete defineerimiseks vastavalt *"{% for ... %}{% endfor %}"* ja *"{% if ... %}{% endif %}"*, kuid loodud mallimootoril on lihtsustatud süntaks (Joonis 10) ning koodiplokkide lõpetamiseks eraldi eritähendusega koodisilte kirjutama ei pea.

```
{{ for example of examples:
  {{ example }}
}}
```

```
{{ if variable == “variable”:
  variable equals
}}
```

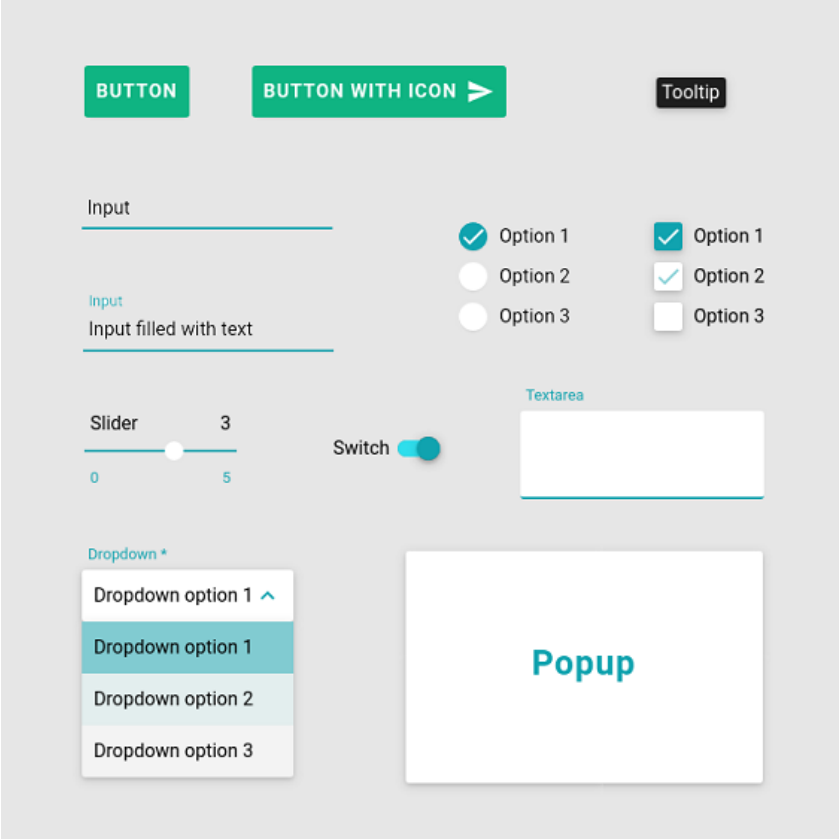
Joonis 10. Loodud mallimootori näidiskood silmuste ja tingimuslausete kohta

4.2.6 Marsruutija

Tuumkoodi marsruutijat täiustav klass, mis pakub vajaliku lisafunktsionaalsust veebilehtede saatmiseks. Päringud on jaotatud kaheks suuremaks osaks: lehed ja failid ning päringutele vastatakse vastavalt eelnevalt seadistatud seadistusklassile ning andmebaasi lisatud lehtedele.

4.2.7 Liidesekomponendid

Liidesekomponendid (Joonis 11) on loodud võttes arvesse veebilehtedel enimkasutatavaid komponente. Loodud liidesekomponentcide hulka kuuluvad näiteks nupud, rippmenüüd, ühe- ja mitmevalikuga kastid, liugur, lülitid, vormilahtrid, hüpikaknad, hüpikvihjed.



Joonis 11. Kõige tavapärasemad liidesekomponendid

Liidesekomponendid genereeritakse dünaamiliselt mallimootori poolt, andes genereeritavale komponendile kaasa andmed JSON kujul dünaamiliselt, mis poleks võimalik kasutades Inja mallimootorit. (Joonis 12 ja Joonis 13)

```
<textarea-container field="{{ id }}"
  {{ if required: required="" }}
>
  <textarea type="text" name="{{ id }}" border=""></textarea>
  <label>{{ label }}{{ if required:  *}}</label>
</textarea-container>
```

Joonis 12. Liideskomponendi tekstiala mallikood

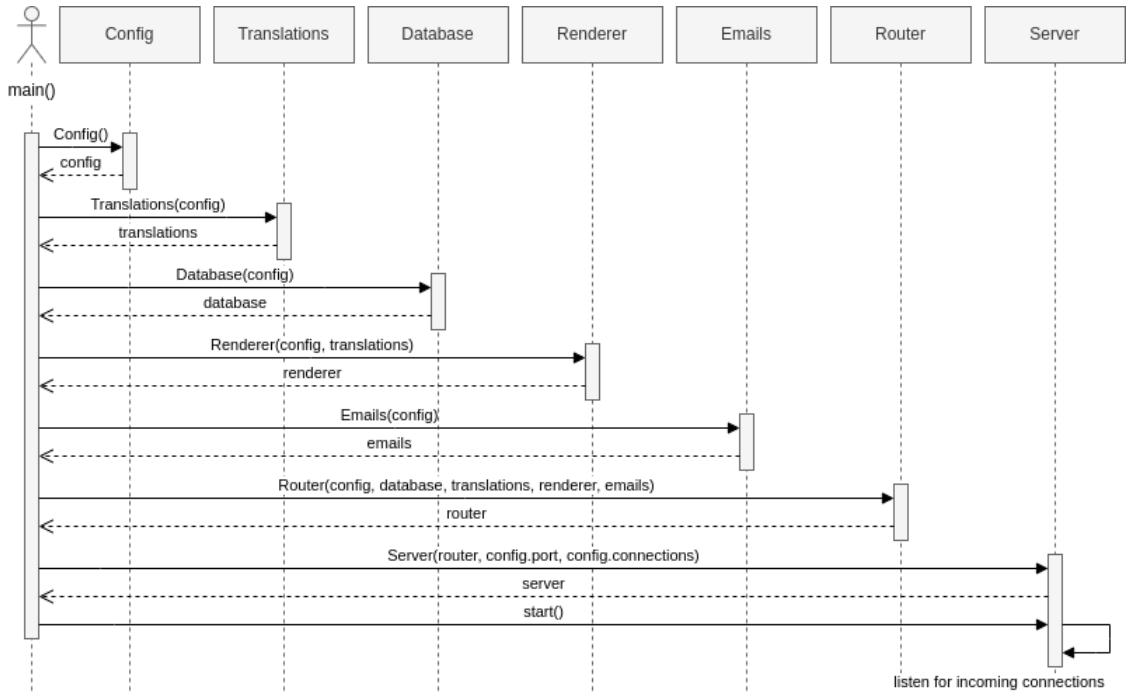
```
{{ textarea({
 "id": "description",
 "label": "{{ translate("contact_details") }}",
 "required": true
}) }}
```

Joonis 13. Liideskomponendi tekstiala kasutamise näidiskood

4.3 Põhilised kasutusmallid

4.3.1 Serveri käivitamine

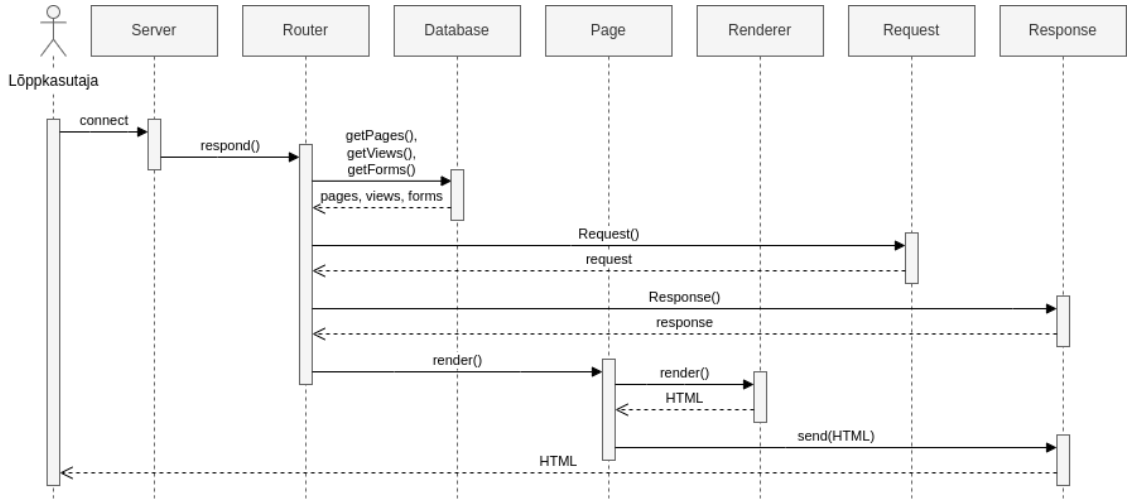
Serveri käivitamiseks (Joonis 14) raamistikku kasutades peab looma objektid, mis on vajalikud päringutele vastamiseks. Objektid tuleb luua konkreetses järjekorras, kuna objektid sõltuvad üksteistest. Pärast objektide loomist on vajalik kutsuda *Server* klassi objektil välja *start()* meetod, mis paneb serveri käima ning alustab sissetulevate ühenduste jälgimist seadistatud pordi peal.



Joonis 14. Serveri käivitamine

4.3.2 Päringule vastamine (Joonis 15)

Lõppkasutaja teeb päringu serverile. *Server* saadab saabunud ühenduse edasi *Routerile*, mis kontrollib, et päring oleks valiidne ning päringu URL vastaks defineeritud marsruutidele. Loodud ühenduse kohta luuakse vastavalt *Request* ja *Response* objektid, mis saadetakse edasi *Page* objektile. *Page* objekt kutsub välja vajaminevad renderdamise meetodid mallimootorist, mis pannakse kokku HTML tekstiks. Lõplik HTML saadetakse edasi *Response* objekti, mille läbi vastatakse tehtud päringule.



Joonis 15. Päringutele vastamine

5 Raamistiku valideerimine

Antud peatükis testitakse loodud raamistikku, võrreldakse olemasolevate raamistikega ja maalimootoritega ning taasluuakse olemasolev kliendi veebileht kasutades uut raamistikku.

5.1 Automaattestid

Automaattestid luuakse CLion rakenduses kasutades GoogleTest raamistikku, sest see on automaatselt integreeritud CLion rakendusega. GoogleTest raamistik võimaldab testida loodud klasse, luues nendest objektid. Tehtud testide kohta koostatakse detailne raport, milles kuvatakse teavet lausekate, harukate, testide õnnestumise ning kestuse kohta.

Kokku loodi mallimootorile 72 testi: 4 üldist, 10 muutujatele, 10 pesastatud objektidele, 9 massiividele, 31 tingimuslausetele, 4 silmustele, 4 meetoditele, millest raamistiku arendamise lõpuks kõik testid edukalt läbisid ning katsid 94% koodikaetusega (Joonis 16).

method.cpp	100% lines covered	56% branches covered
method.hpp		
renderable.cpp	100% lines covered	65% branches covered
renderable.hpp		
renderer.cpp	69% lines covered	40% branches covered
renderer.hpp		
statement_conditional.cpp	100% lines covered	53% branches covered
statement_conditional.hpp		
statement_if.cpp	100% lines covered	57% branches covered
statement_if.hpp		
statement_loop.cpp	100% lines covered	51% branches covered
statement_loop.hpp		

Joonis 16. Automaattestide koodikaetus CLion rakendusest teste käivitades

5.2 Serveri jõudluse võrdlus olemasolevate raamistikega

Raamistike serveri jõudluse testimiseks viidi läbi testid kasutades *apache bench'i* tööriista, mis teeb n korda päringuid määratud URL-ile. Testid käivitati käsurealt käsuga *"ab –n 1000 http://localhost:8000/"*, kus URL määrab päritava serveri. Pärast konkreetse

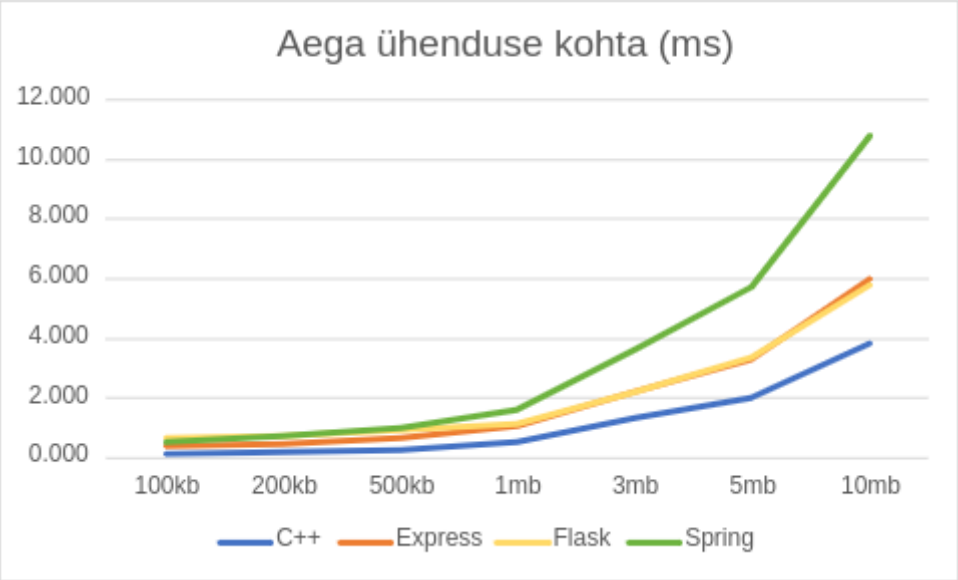
failisuurusega testi muudeti ära saadetava fail ning taaskäivitati serverid. Iga testitud faili suurusega käivitati test 1000 päringuga, et arvutada keskmine tulemus 1000 päringu pealt ning seeläbi oleks testid usaldusväärsed. (Joonis 17, Joonis 18 ja Joonis 19)

Jõudlustestid viidi läbi järgnevat süsteemi kasutades:

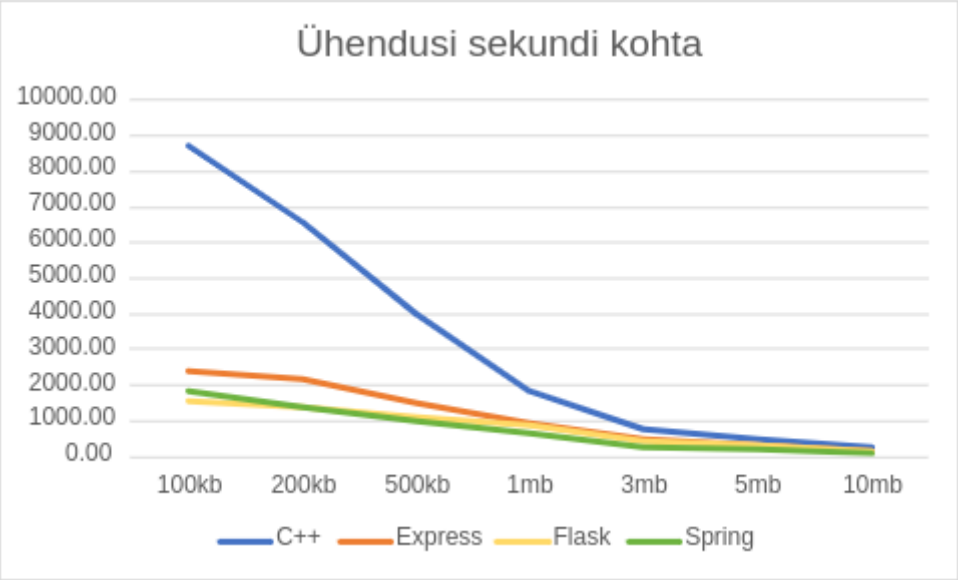
- **CPU:** Intel Core i7-10750H @ 2.60GHz
- **GPU:** NVIDIA GeForce RTX 3070 Laptop GPU
- **RAM:** 16 GB
- **SSD:** Toshiba XG6 KXG60ZNV512G
- **OS:** Arch Linux 64-bit
- **KERNEL:** 5.15.8-arch1-1

Serveri jõudlustestid viidi läbi järgnevate versioonidega raamistikel:

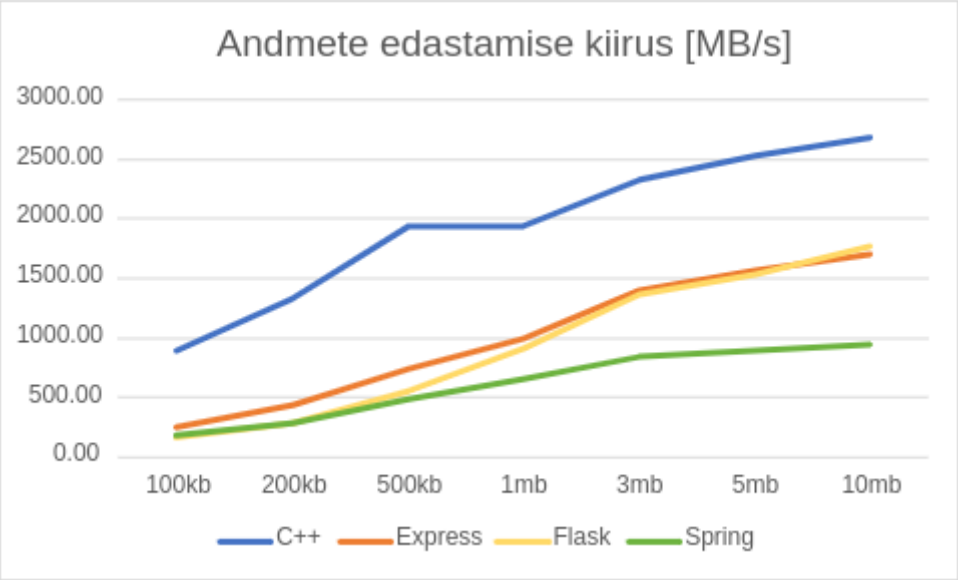
- **Express:** - v4.17.1
- **Flask:** - v2.0.2
- **Spring:** - v5.3.14



Joonis 17. Jõudlustestid. Aega ühenduse kohta.



Joonis 18. Jõudlustestid. Ühendusi sekundi kohta.



Joonis 19. Jõudlustestid. Andmete edastamise kiirus.

Testide tulemustest võib järeldada, et loodud raamistik on kordades kiirem, kui olemasolevad raamistikud, kuna veebilehtedel olevad failid on üldjuhtudel võimalikult väikese mahuga, et veebilehed oleksid võimalikult kiiresti nähtavad lõppkasutajale. Loodud raamistik on tasemel olemasolevate raamistikega ning see on ideaalselt sobilik kasutamaks veebilehtede arendusteks. See analüüs ühtlasi vastab ka uurimisküsimusele U1.

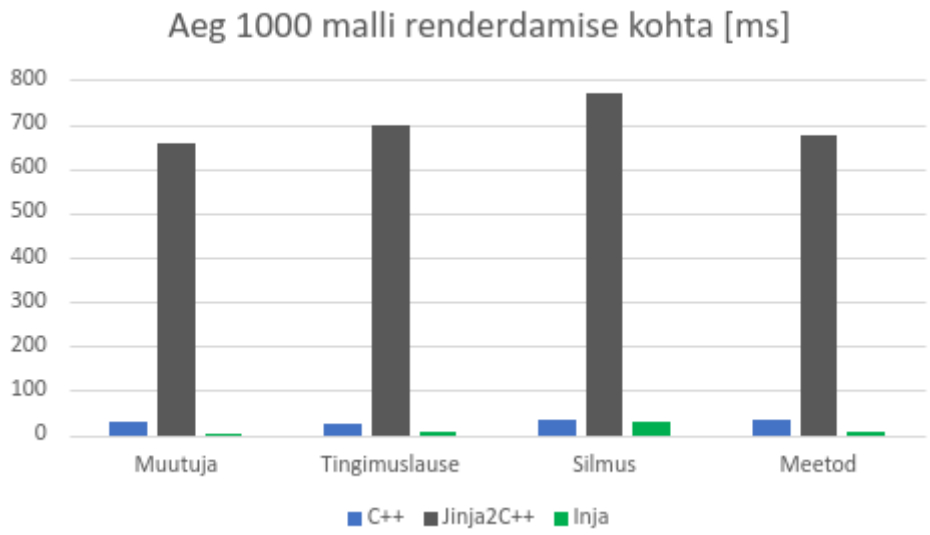
5.3 Mallimootorite jõudluse võrdlemine

Et vastata uurimisküsimusele U2, sai teostatud eksperiment mallimootorite jõudluste testimiseks võrreldes loodud C++ mallimootoriga, loodi mallid, mille genereeritud tekstid olid samad ning igat malli renderdati 1000 korda, et arvutada välja malli renderdamisele kuluv aeg (Joonis 20). Aega mõõdeti koodisiseselt *chrono* [61] teeki kasutades.

Testid sisaldasid endas mallimootorite funktsionaalsuse testimist, nagu näiteks muutujate kuvamine, tingimuslaused, silmused ja meetodid.

Mallimootorite jõudlustestid viidi läbi järgnevate versioonidega raamistikel:

- **Jinja2C++** - v1.2.1
- **Inja** - v3.3



Joonis 20. Aeg 1000 malli renderdamise kohta

Läbiviidud testidest on näha, et loodud mallimootor pole kõige kiirem võrreldes olemasolevate mallimootoritega, kuid samas on sarnase kiirusega nagu Inja, mis on tunduvalt kiirem kui Jinja2C++. Kuna loodud mallimootor pakub lisafunktsionaalsust JSON toe kasutamiseks, mille läbi on arendajal võimalik mallifaile lihtsamini ning kiiremini arendada.

5.4 Kliendi veebilehe taasloomine raamistikku kasutades

Algselt oli kliendi veebileht kirjutatud kasutades PHP-d, marsruutimiseks kasutati Apache *.htaccess* faili ning andmebaasiks oli MariaDB. Veebilehe põhifunktsionaalsuse alla kuulusid portfoolio, meeskonna liikmed, mitmekeelsustugi, e-kirjade saatmine läbi kontakti vormi.

Kõik veebilehe funktsionaalsused konverteeriti edukalt ümber kasutades loodud raamistikku ning uusi mallifaile kasutades oli kood tunduvalt lühem kui algne PHP lähtekood. Järgnevalt on välja toodud näited raamistiku kasutamisest.

5.4.1 Portfoolio

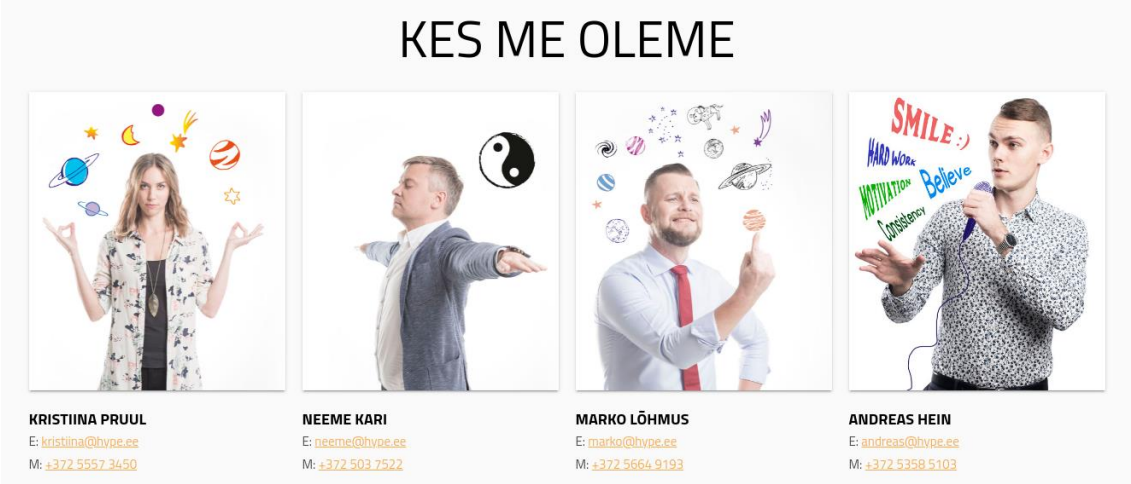
Portfoolio (Joonis 21) andmed on sisestatud andmebaasi ning loodud sai kliendi repositoorumisse andmebaasi klass mis laiendab raami andmebaasi klassi. Andmebaasi klassi sai juurde lisatud üldine portfoolio kategooriate andmete pärimise funktsioon ning ühe konkreetse portfoolio kategooria informatsiooni pärimise funktsioon. Portfoolio andmete hoidmiseks sai loodud andmeklassid, mis tehti ümber JSON kujule integreeritud mallimootoris kasutamiseks. Portfoolio kuvamiseks kasutati tõlketuge ning mallimootori silmust portfoolio kohandatud meetodi väljakutsmiseks andes kaasa ühe portfoolio andmed JSON kujul.

Joonis 21. Portfoolio vaade esilehel

5.4.2 Meeskond

Sarnaselt portfooliole loodi vastavad andmebaasi meetodid ja andmete klassid ka meeskonna (Joonis 22) jaoks, mida kasutati meeskonna andmete pärimiseks ning JSON kujul kasutamiseks meeskonna mallifailides. (Joonis 23 ja Joonis 24)

Vana PHP objekt, mida kasutati meeskonna liikmete HTML koodi genereerimiseks, oli 159 koodirida pikk ning mallimootorit kasutades on meeskonna liikme mallifail koosneb vaid 18 koodireast (Joonis 25).



Joonis 22. Meeskonna vaade esilehel

```
<client-view
  type="secondary" view="team"
  url="{{ pages["6"].url }}"
>
  <view-title h1>{{ translate("team") }}</view-title>
  <view-inner>
    {{ for team_member in team:
      {{ team_small({{ team_member }}) }}
    }}
  </view-inner>
</client-view>
```

Joonis 23. Meeskonna vaade meeskonna liikmete genereerimiseks

```
<client-team size="small">
  <team-pictures card>
    {{ for team_picture in pictures:
      <team-picture
        hidden="{{ team_picture != 1 ? "true" : "false" }}
        style="background-image: url(
          /public/img/team/{{ id }}_{{ team_picture }}.jpg
        );”>
      </team-picture>
    }}
    <a href="{{ pages["6"].url }}/{{ url }}"></a>
  </team-pictures>

  <team-name h5>{{ name }}</team-name>
  <team-email>
    <a href="mailto:{{ email }}">{{ email }}</a>
  </team-email>
  <team-mobile>
    <a href="tel:{{ mobile }}">{{ mobile }}</a>
  </team-mobile>
</client-team>
```

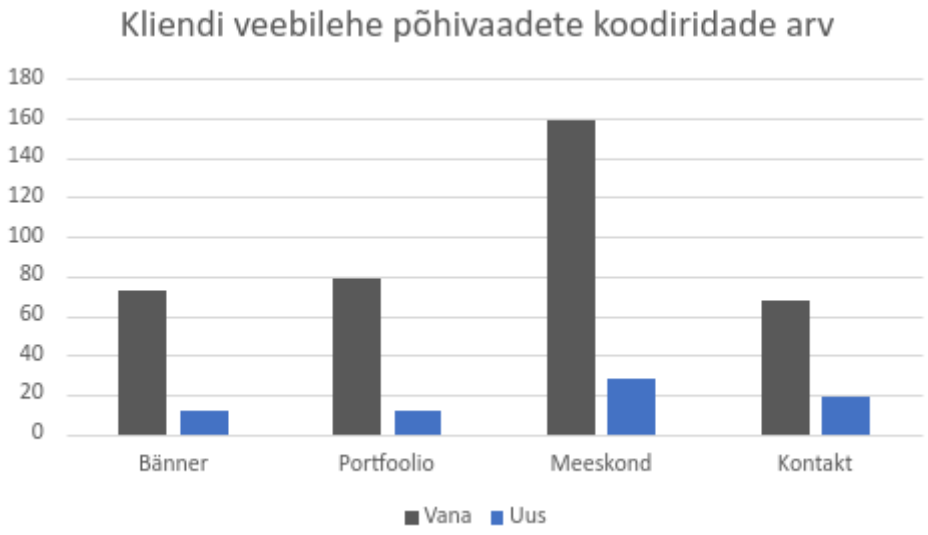
Joonis 24. Meeskonna objekt meeskonna vaatesse lisamiseks

5.4.3 Mitmekeelsustugi

Veebileht pidi toetama nii eesti- ja inglise keelt ning selle lisamiseks kasutati raami jaoks arendatud keeletuge. Vajalik oli konfiguratsiooni lisada kaks keeleobjekti ning andmebaasi defineerida vastavalt „*est*“ ja „*eng*“ veerud ning vastavad tõlkefailid „*translations*“ kausta lisada, peale mida oli mitmekeelsustugi lisatud ning automaatselt toimis ka kahe keele vahel vahetamine veebilehel navigeerides.

5.4.4 Koodiridade võrdlus

Vanal veebilehel oli vaadete genereerimiseks kasutatud PHP objekte, kuhu olid defineeritud meetodid andmete pärimiseks ning kasutamiseks HTML koodi genereerimiseks. Kliendi veebilehe põhilisteks vaadeteks on bänner, portfoolio, meeskond, kontakt.



Joonis 25. Vana ja uue veebilehe põhivaadete koodiridade arvu võrdlus

Jooniselt 24 on näha, et mallimootorit kasutades on koodiridasid kordades vähem, kuna pole vajalik defineerida klasse ega meetodeid, läbi mille on kood loetavam, kuna mallifailides on selgelt näha genereeritav HTML struktuur, kus on vajalikud lüngad täidetud andmetega.

6 Võimalikud edasiarendused

Antud peatükis kirjeldatakse võimalike edasiarendusi, mille läbi oleks loodud raamistikku võimalik edukamalt kasutada loodavate projektide raames.

Raamistiku lihtsamaks kasutamiseks oleks võimalik luua automatiseeritud paigaldusskript, mida käivitades küsitaks arendajalt hulk suunatuid küsimusi, mis funktsionaalsust ning kuidas seda plaanitakse kasutada. Näiteks oleks võimalik arendajalt küsida valitavat serveri porti, andmebaasihaldussüsteemi ning kas andmebaasi struktuur soovitakse automaatselt genereerida raamistiku spetsifikatsiooni järgi. Sarnast esmase skripti kasutamise loogikat kasutab näiteks Node rakenduste loomiseks käsuga *npm init*, läbi mille küsitakse arendajalt küsimusi projekti kohta ning luuakse vastav paketi konfiguratsioonifail. [62]

Valminud raamistiku mallimootoris C++ koodi kasutamiseks on arendajal vajalik defineerida kohandatud meetodid, läbi mille on võimalik kirjutatud koodi käivitada. Sellele lisaks võiks olla võimalus käivitada C++ koodi otse mallimootorist sarnaselt EJS mallimootorile, mis võimaldab mallifailides otse JavaScripti koodi käivitada.

Projektidele, mis kasutaksid raamistikku ning millel oleks andmebaasi struktuur vastavalt raamistiku spetsifikatsioonile, võiks olla võimalik juurde integreerida sisuhaldussüsteem, läbi mille oleks võimalik hallata veebilehel olevaid lehti, andmeid ja vaateid.

7 Kokkuvõte

Antud töö eesmärgiks oli luua serveri- ja kliendipoolne veebiraamistik C++ keeles koos integreeritud mallimootoriga, mis oleks alternatiiviks olemasolevatele veebiraamistikele, kuid samas oleks jõudluse poolest kiirem ning pakuks rohkem funktsionaalsust võrreldes teiste raamistikega.

Loodud veebiraamistiku ja mallimootori arendamiseks analüüsiti ning võrreldi olemasolevaid raamistikke ja mallimootoreid, et loodud raamistik oleks sama võimekas kui analüüsitud raamistikud ja mallimootorid.

Raamistiku serveri jõudluse testimiseks tehti jõudlustestid võrreldes Express, Flask ja Spring raamistikega. Jõudlustestidest tuli välja märkimisväärne erinevus testitud raamistike vahel, kuna loodud raamistik oli kohati kordades kiirem, kui teised raamistikud.

Mallimootorile loodi automaattestid ning jõudluse testimiseks võrreldi loodud raamistikku teiste C++ mallimootoritega. Loodud mallimootor ei olnud testides kõige kiirem, kuid pakkus selle eest lisafunktsionaalsust, mis teistel mallimootoritel puudus.

Raamistiku valideerimiseks taasloodi olemasolev kliendi veebileht kasutades arendatud raamistikku ning mallimootorit, et testida nende kasutatavust ning funktsionaalsust, mis osutusid edukaks, sest kliendi veebilehte oli võimalik taasluua täpselt sama funktsionaalsusega, kuid veebilehe lähtekoodi maht vähenes tänu mallimootorile drastiliselt. Lõputöö tegemise käigus leiti vastused igale püstitatud uurimisküsimustele.

Raamistiku ning mallimootori täiustamiseks pakuti välja ka võimalikud edasiarendused tulevikuks, läbi mille oleks võimalik muuta kogu raamistik ning mallimootor klientide ning arendajate jaoks veel atraktiivsemaks.

Kasutatud kirjandus

[1] „Internet users in the world 2021,“ [Võrgumaterjal]. Available: https://www.statista.com/statistics/617136/digital-population-worldwide/. [Kasutatud 11 oktoober 2021].

[2] H. Lima ja M. M. Eler, „C++ Web Framework: A Web Framework for Web Development using,“ *Proceedings of the 23rd International Conference on Enterprise Information Systems,* kd. 2, nr https://www.scitepress.org/Papers/2021/104577/104577.pdf, pp. 76-87, 2021.

[3] „Cambridge Dictionary,“ [Võrgumaterjal]. Available: https://dictionary.cambridge.org/dictionary/english/framework. [Kasutatud 12 oktoober 2021].

[4] S. P. Mishra ja S. K. Srivastava, „Web development frameworks and its performance analysis - a review,“ %1 *Smart Computing*, Pauri, Garhwal, Uttarakhand, India, CRC Press, 2020, pp. 337-343.

[5] „Introduction to HTML,“ [Võrgumaterjal]. Available: https://www.w3schools.com/html/html_intro.asp. [Kasutatud 23 detsember 2021].

[6] „C++20 - cppreference.com,“ [Võrgumaterjal]. Available: https://en.cppreference.com/w/cpp/20. [Kasutatud 23 detsember 20212].

[7] „CLion koduleht,“ [Võrgumaterjal]. Available: https://www.jetbrains.com/clion/. [Kasutatud 23 detsember 2021].

[8] „ReactJS koduleht,“ [Võrgumaterjal]. Available: https://reactjs.org/. [Kasutatud 23 detsember 2021].

[9] „jQuery koduleht,“ [Võrgumaterjal]. Available: https://jquery.com/. [Kasutatud 23 detsember 2021].

[10] „Express JS koduleht,“ [Võrgumaterjal]. Available: https://expressjs.com/. [Kasutatud 12 detsember 2021].

[11] „Angular koduleht,“ [Võrgumaterjal]. Available: https://angular.io/. [Kasutatud 23 detsember 2021].

[12] „ASP.NET dokumentatsioon,“ [Võrgumaterjal]. Available: https://docs.microsoft.com/en-us/aspnet/core/?view=aspnetcore-6.0. [Kasutatud 23 detsember 2021].

[13] „Flask koduleht,“ [Võrgumaterjal]. Available: https://flask.palletsprojects.com/en/2.0.x/. [Kasutatud 12 detsember 2021].

[14] „Spring koduleht,“ [Võrgumaterjal]. Available: https://spring.io/. [Kasutatud 12 detsember 2021].

[15] S. Liu, „statista,“ 19 August 2021. [Võrgumaterjal]. Available: https://www.statista.com/statistics/1124699/worldwide-developer-survey-most-used-frameworks-web. [Kasutatud 13 oktoober 2021].

[16] „What Is a Database | Oracle,“ [Võrgumaterjal]. Available: https://www.oracle.com/database/what-is-database/. [Kasutatud 19 detsember 2021].

[17] „Express - Using template engines with Express,“ [Võrgumaterjal]. Available: https://expressjs.com/en/guide/using-template-engines.html. [Kasutatud 23 detsember 2021].

[18] „Express - Express routing,“ [Võrgumaterjal]. Available: https://expressjs.com/en/guide/routing.html. [Kasutatud 23 detsember 2021].

[19] „What is dynamic URL? Definition from WhatIs.com,“ [Võrgumaterjal]. Available: https://whatis.techtarget.com/definition/dynamic-URL. [Kasutatud 23 detsember 2021].

[20] „Using HTTP cookies - HTTP | MDN,“ [Võrgumaterjal]. Available: https://developer.mozilla.org/en-US/docs/Web/HTTP/Cookies. [Kasutatud 23 detsember 2021].

[21] „Express - Frameworks built on Express,“ [Võrgumaterjal]. Available: https://expressjs.com/en/resources/frameworks.html. [Kasutatud 23 detsember 2021].

[22] „Feathers koduleht,“ [Võrgumaterjal]. Available: http://feathersjs.com/. [Kasutatud 23 detsember 2021].

[23] „Kraken koduleht,“ [Võrgumaterjal]. Available: http://krakenjs.com/. [Kasutatud 23 detsember 2021].

[24] „Sails koduleht,“ [Võrgumaterjal]. Available: http://sailsjs.org/. [Kasutatud 23 detsember 2021].

[25] „NestJS koduleht,“ [Võrgumaterjal]. Available: https://github.com/nestjs/nest. [Kasutatud 23 detsember 2021].

[26] „Express - Database integration,“ [Võrgumaterjal]. Available: https://expressjs.com/en/guide/database-integration.html. [Kasutatud 15 detsember 2021].

[27] „MySQL koduleht,“ [Võrgumaterjal]. Available: https://www.mysql.com/. [Kasutatud 23 detsember 2021].

[28] „PostgreSQL: About,“ [Võrgumaterjal]. Available: https://www.postgresql.org/about/. [Kasutatud 23 detsember 2021].

[29] „Apache Cassandra koduleht,“ [Võrgumaterjal]. Available: https://cassandra.apache.org/_/index.html. [Kasutatud 23 detsember 2021].

[30] „What is MongoDB? -- MongoDB Manual,“ [Võrgumaterjal]. Available: https://docs.mongodb.com/manual/. [Kasutatud 23 detsember 2021].

[31] „Redis koduleht,“ [Võrgumaterjal]. Available: https://redis.io/. [Kasutatud 23 detsember 2021].

[32] „MariaDB koduleht,“ [Võrgumaterjal]. Available: https://mariadb.org/. [Kasutatud 22 detsember 2021].

[33] „Pug GitHub leht,“ [Võrgumaterjal]. Available: https://github.com/pugjs/pug. [Kasutatud 23 detsember 2021].

[34] „Mustache - npm,“ [Võrgumaterjal]. Available: https://www.npmjs.com/package/mustache. [Kasutatud 14 detsember 2021].

[35] „EJS koduleht,“ [Võrgumaterjal]. Available: https://ejs.co/. [Kasutatud 14 detsember 2021].

[36] „Pug HTML Template Engine: A Beginner's Guide - SitePoint,“ [Võrgumaterjal]. Available: https://www.sitepoint.com/a-beginners-guide-to-pug/. [Kasutatud 15 detsember 2021].

[37] „express-translate - npm,“ [Võrgumaterjal]. Available: https:/www.npmjs.com/package/express-translate. [Kasutatud 23 detsember 2021].

[38] „NPM - Cookie Parser,“ [Võrgumaterjal]. Available: https://www.npmjs.com/package/cookie-parser. [Kasutatud 16 detsember 2021].

[39] „Flask - Foreword,“ [Võrgumaterjal]. Available: https://flask.palletsprojects.com/en/2.0.x/foreword/. [Kasutatud 14 detsember 2021].

[40] „Flask - Define and Access the Database,“ [Võrgumaterjal]. Available: https://flask.palletsprojects.com/en/2.0.x/tutorial/database/. [Kasutatud 14 detsember 2021].

[41] „Jinja2 koduleht,“ [Võrgumaterjal]. Available: https://palletsprojects.com/p/jinja/. [Kasutatud 20 detsember 2021].

[42] „Flask - Quickstart,“ [Võrgumaterjal]. Available: https://flask.palletsprojects.com/en/2.0.x/quickstart/. [Kasutatud 14 detsember 2021].

[43] „Flask App Tutorial on Localization,“ [Võrgumaterjal]. Available: https://phrase.com/blog/posts/python-localization-flask-applications/. [Kasutatud 14 detsember 2021].

[44] „HSQLDB koduleht,“ [Võrgumaterjal]. Available: http://www.hsqldb.org/. [Kasutatud 15 detsember 2021].

[45] „H2 koduleht,“ [Võrgumaterjal]. Available: http://www.h2database.com/html/main.html. [Kasutatud 15 detsember 2021].

[46] „Apache Derby koduleht,“ [Võrgumaterjal]. Available: https://db.apache.org/derby/. [Kasutatud 15 detsember 2021].

[47] „Spring docs - Embedded database support,“ [Võrgumaterjal]. Available: https://docs.spring.io/spring-framework/docs/3.0.0.M4/reference/html/ch12s08.html. [Kasutatud 15 detsember 2021].

[48] „Spring docs - View technologies,“ [Võrgumaterjal]. Available: https://docs.spring.io/spring-framework/docs/4.3.0.RC2/spring-framework-reference/html/view.html. [Kasutatud 14 detsember 2021].

[49] „JSP - Guide to JavaServer Pages,“ [Võrgumaterjal]. Available: https://www.baeldung.com/jsp. [Kasutatud 15 detsember 2021].

[50] „Groovy Docs - The MarkupTemplateEngine,“ [Võrgumaterjal]. Available: https://docs.groovy-lang.org/latest/html/documentation/markup-template-engine.html. [Kasutatud 15 detsember 2021].

[51] „Thymeleaf koduleht,“ [Võrgumaterjal]. Available: https://www.thymeleaf.org/. [Kasutatud 15 detsember 2021].

[52] „How to localize Spring applications like a Pro,“ [Võrgumaterjal]. Available: https://phrase.com/blog/posts/how-to-localize-spring-applications-like-a-pro/. [Kasutatud 15 detsember 2021].

[53] „Spring Docs - Creating RESTful services,“ [Võrgumaterjal]. Available: https://docs.spring.io/spring-framework/docs/3.0.0.M3/reference/html/ch18s02.html. [Kasutatud 15 detsember 2021].

[54] „Spring Session - Custom Cookie,“ [Võrgumaterjal]. Available: https://docs.spring.io/spring-session/docs/2.4.2/reference/html5/guides/java-custom-cookie.html. [Kasutatud 15 detsember 2021].

[55] „Jinja2C++ repositoorium,“ [Võrgumaterjal]. Available: https://github.com/jinja2cpp/Jinja2Cpp. [Kasutatud 30 detsember 2021].

[56] „CMake koduleht,“ [Võrgumaterjal]. Available: https://cmake.org/. [Kasutatud 30 detsember 2021].

[57] „Nlohmann JSON Github,“ [Võrgumaterjal]. Available: https://github.com/nlohmann/json. [Kasutatud 16 detsember 2021].

[58] „Jinja2C++ koduleht,“ [Võrgumaterjal]. Available: https://jinja2cpp.dev . [Kasutatud 30 detsember 2021].

[59] „Inja GitHubi probleem JSON toe lisamiseks,“ [Võrgumaterjal]. Available: https://github.com/pantor/inja/issues/212. [Kasutatud 30 detsember 2021].

[60] „Mailio Github,“ [Võrgumaterjal]. Available: https://github.com/karastojko/mailio. [Kasutatud 16 detsember 2021].

[61] „Cppreference - Date and time utilities,“ [Võrgumaterjal]. Available: https://en.cppreference.com/w/cpp/chrono. [Kasutatud 18 detsember 2021].

[62] „NPM - init,“ [Võrgumaterjal]. Available: https://docs.npmjs.com/cli/v8/commands/npm-init. [Kasutatud 15 detsember 2021].

[63] intelegain, „intelegain,“ 6 August 2019. [Võrgumaterjal]. Available: https://intelegain-technologies.medium.com/what-are-web-frameworks-and-why-you-need-them-c4e8806bd0fb. [Kasutatud 12 Oktoober 2021].

[64] D. Goyala, P. Jain ja B. Bhushan, „Enhancement of Security using Various Web Development Frameworks,“ %1 *Proceedings of the International Conference on Innovative Computing & Communications (ICICC) 2020*, https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3600740, 2020.

[65] D. Dinh ja Z. Wang, „Modern front-end web development – How libraries and frameworks transform everything,“ TURKU UNIVERSITY OF APPLIED SCIENCES, Turku, 2020.

[66] „MongoDB koduleht,“ [Võrgumaterjal]. Available: https://www.mongodb.com/. [Kasutatud 23 detsember 2021].

[67] „PostgreSQL koduleht,“ [Võrgumaterjal]. Available: https://www.postgresql.org/. [Kasutatud 23 detsember 2021].

Lisa 1 – Lihtlitsents lõputöö reprodutseerimiseks ja lõputöö üldsusele kättesaadavaks tegemiseks[2]

Mina, Kristjan Vool

1. Annan Tallinna Tehnikaülikoolile tasuta loa (lihtlitsentsi) enda loodud teose „Serveri- ja kliendipoolne veebiraamistik C++ keeles koos integreeritud mallimootoriga“, mille juhendaja on Gert Kanter
   1.1. reprodutseerimiseks lõputöö säilitamise ja elektroonse avaldamise eesmärgil, sh Tallinna Tehnikaülikooli raamatukogu digikogusse lisamise eesmärgil kuni autoriõiguse kehtivuse tähtaja lõppemiseni;
   1.2. üldsusele kättesaadavaks tegemiseks Tallinna Tehnikaülikooli veebikeskkonna kaudu, sealhulgas Tallinna Tehnikaülikooli raamatukogu digikogu kaudu kuni autoriõiguse kehtivuse tähtaja lõppemiseni.
2. Olen teadlik, et käesoleva lihtlitsentsi punktis 1 nimetatud õigused jäävad alles ka autorile.
3. Kinnitan, et lihtlitsentsi andmisega ei rikuta teiste isikute intellektuaalomandi ega isikuandmete kaitse seadusest ning muudest õigusaktidest tulenevaid õigusi.

04.01.2022

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