TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Gen Metsaveer 155019IADB

Juurutusjuhiste modelleerimis- ja publitseerimisvahend Eesti tervishoiu teenustele

Bakalaureusetöö

Juhendaja: Igor Bossenko
Magistrikraad

Tallinn 2022

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: Gen Metsaveer

16.05.2022

Annotatsioon

Käesoleva bakalaureusetöö eesmärk on luua veebirakenduse näol minimaalne töötav toode Health Level 7 poolt loodud Fast Healthcare Interoperability Resources standardi juurutusjuhiste loomiseks. Loodav lahendus peaks lahendama peamised probleemid antud standardi juurutusjuhiste loomisel, mis aitab kasutajatel kiiremini ja mugavamini neid luua.

Töös käsitletavaks probleemiks on lihtsate ja tasuta kättesaadavate HL7 FHIR juurutusjuhiste loomise rakenduste olemasolu.

Antud töö analüüsib olemasolevaid lahendusi nii veebirakenduste kui ka käsurea programmide näol. Nende alusel leitakse lahenduste eelised ja puudujäägid võrreldes neid omavahel. Seejärel seatakse kriteeriumid loodavale lahendusele ning valitakse tehnoloogiad selle elluviimiseks.

Viimases peatükis kirjeldatakse loodava lahendust ning töö käigus tehtud valikuid ja otsuseid.

Bakalaureusetöö raames realiseeriti minimaalne töötav toode, et valmistada HL7 FHIR juurutusjuhiseid. See seab alused, mille põhjal on võimalik luua turul konkureerivate toodetega samaväärne toode.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 22 leheküljel, 7 peatükki, 9 joonist, 0 tabelit.

Abstract

Implementation Guide Building and Publishing Tool for Estonian Healthcare Services

The aim of this bachelro's thesis ist o create a minimum vibale product in the form of a web application to create implementation guides using Health Level 7 Fast Healthcare Interoperability Resources standard. The solution should address key issues in creating standard deployment instructions that help users create them faster and more conveniently.

The problems addressed in this work are the availability of simple and free applications for creating Health Level 7 Fast Healthcare Interoperability Resources implementation guides.

This work analyzes the existing solutions in the form of both web applications and command line programs. Based on them, the advantages and disadvantages of the solutions are found in comparison with each ohter. Then the criteria are set for the solution to be created and the technologies for its implementation are selected.

The last chapter describes the solution to be created and the choices and decisions made during the work.

As part of the bachelor's thesis, the minimum viable product was realized for creating Health Level 7 Fast Healthcare Interoperability Resources implementation guides. This lays the Foundation for a product that is equivalent to competing product.

The thesis is in Estonian and contains 22 pages of text, 7 chapters, 9 figures, 0 tables.

Lühendite ja mõistete sõnastik

API	*Application Programming Interface* ehk rakendusliides
FHIR	*Fast Healthcare Interoperability Resources*
Github	Veebimajutusteenus tarkvaraarendusele
HL7	*Health Level 7* organisatsioon
HTTP	*HyperText Transfer Protocl,* ehk hüperteksti edastuse protokoll
ImplementationGuide	HL7 FHIR juurutusjuhise ressurss
JSON	*JavaScript object notation,* ehk JavaScripti objektide notatsioon
REST	*Representational state transfer*
SOAP	*Simple Object Access Protocol,* ehk lihtne objektipöördusprotokoll

Sisukord

Jooniste loetelu

1 Sissejuhatus

Eesti meditsiiniliste andmete talletamine on liikumas uue standardi peale - Health Level 7 Fast Healthcare Interoperability Resources ehk HL7 FHIR. See on oma loomult piisavalt erinev, et varasemad standardi kasutamise dokumenteerimise viisid pole enam sobilikud ja vajavad väljavahetamist. See toob kaasa palju dokumenteerimise eeskirjade standardiseerimist, mis omakorda peab olema kõrge kvaliteediga, et andmeedastuse standardi vahetus liiguks sujuvalt ja uue kasutuselevõtt nii riiklikul tasandil kui ka erasektoris oleks kõigile arusaadavas vormis.

HL7 FHIR standard näeb ette andmeedastust sündmuse põhiselt ning kasutab REST arhitektuuri disaini mustreid. See on standard, mis kirjeldab andmevorminguid ja elemente ning rakenduste programmeerimisliidest elektrooniliste meditsiiniandmete vahetamiseks. Üheks standardi ressursiks on *ImplementationGuide* ehk juurutusjuhis, mille kaudu on võimalik kirjeldada standardi kasutamise reegleid ja kuidas mingi konkreetne probleem lahendatakse.

Kuigi antud standardi arhitektid on mõelnud ka dokumenteerimise standardiseerimise peale, siis selle kasutamise viisid võivad kasutajatele olla keerulised ja kohmakad. Seega on turul paar lahendust, mis seda aitavad lihtsustada, kuid neil kõigil on omad probleemid.

Käesolevas lõputöös analüüsitakse probleemi ja võrreldakse olemasolevaid HL7 FHIR juurutusjuhiste valmistamiste tööriistu ja terviklahendusi. Antud probleemi lahendamiseks töötab autor välja minimaalse töötava HL7 FHIR 4 juurutusjuhiste valmistamise veebirakenduse, mis seab alused turul samalaadsete toodetega konkureerivale lahendusele. Antud töö kirjutamisel kasutatakse töö kirjutamise hetkel hilisemat ametlikku versiooni R4.

Veebirakenduse eesmärk on luua kasutajatele keskkond, mille abil on minimaalse väljaõppega võimalik luua HL7 FHIR juurutusjuhiseid. Selle abil kaob ära rakenduste kohalikud paigaldused, ebaühtivad sõltuvused ja vajadus keerulisematele arvuti kasutamise oskustele.

Vastavalt eesmärkidele on autor seadnud endale järgnevad ülesanded:

- Disainida lahenduse arhitektuur
- Luua tagarakendus vastava arhitektuuri põhjal
- Luua eesrakendus suhtlemaks tagarakendusega

2 Probleemi püstitus ja eesmärk

Käesolevas peatükis kirjeldatakse teema aktuaalsust, olemasolevat lahendust ja tausta. Tuuakse välja lahendatav probleem ning kirjeldatakse probleemi uurimiseks ja lahendamiseks valitud metoodikat. Viimaks tuuakse välja töö eesmärgid.

2.1 Aktuaalsus

Eesti tervise infosüsteemide andmevahetusstandard on liikumas HL7 FHIR standardile [1]. See toob kaasa suuri muudatusi andmete edastamises ja talletamises. Uue standardi juurutamine vajab kvaliteetset dokumentatsiooni ning HL7 FHIR loojad on selle jaoks kirjutanud mõningad soovitused. Selle alusel on loodud mõningaid tööriistu ja veebirakendusi, mis aitavad HL7 FHIR juurutusjuhiseid valmistada.

2.2 Taust

Eesti on kasutanud meditsiiniliste andmete edastamiseks Health Level Seven organisatsiooni poolt loodud XML-l põhinevat dokumentide andmevahetusstandardit V3 juba üle kümne aasta. Vahepeal on sama organisatsiooni poolt loodud uus ja kaasaegsem standard FHIR, mis võimaldab SOAP asemel kasutada REST arhitektuuri ning muudab dokumendipõhise andmeedastuse sündmusepõhiseks.

2.3 Probleemipüstitus

Peamised HL7 FHIR juurutusjuhiste valmistamise tööriistad nõuavad vähemalt üldisi teadmisi tarkvara arendusest, kuigi peamised juurutusjuhiste loojad ei pruugi olla tarkvara arendajad. Turul on saadaval mõningaid veebirakendusi, kuid need on kas aegunud või nõuavad tasulist litsentsi enamus funktsioonide jaoks.

Ajakohased veebirakendused pole avatud lähtekoodiga, mis võib tähendada uute funktsioonide lisamise või paranduste viivitusega. Avatud lähtekoodiga lahendustel on võimalik kõikidel sisse viia nii parandusi kui ka edendusi.

Kõik eelnev on tekitanud olukorra, kus juurutusjuhiste loojad peavad, kas maksma suuri litsentsisummasid või õppima uusi tehnoloogiaid. Need on kõik lisakulud, mida saaks vältida tasuta veebirakenduse olemasolul, kus pole vajalik omada erilist kogemust infotehnoloogiast.

Tarkvara välearenduse meetodeid arvestades, on lihtsasti kättesaadav ja selge dokumentatsioon vajalik nii arendajatele kui ka analüütikutele. See on üks eeldusi, et loodav lahendus oleks kõrge kvaliteediga ja jälgiks võimalikult täpselt ettekirjutatud reegleid ja soovitusi.

2.4 Eesmärk

Käesoleva töö eesmärk on analüüsida FHIR põhiseid juurutusjuhiste loomise ja publitseerimise tööriistu ning leida nende puudujäägid Eesti tervishoiu tarkvaraliste lahenduste kontekstis. Luua tehnilise lahenduse prototüüp, mis seab alused, et luua lõplik lahendus, mis eelnevad puudujäägid kõrvaldab ja kujundada veebirakendusest minimaalne elujõuline toode, mille abil on võimalik koostada FHIR põhiseid juurutusjuhiseid.

2.5 Metoodika

Käesolev bakalaureusetöö algab probleemi püstitusega, kirjeldatakse selle tausta ja seatakse eesmärgid loodavale lahendusele.

Analüüsi käigus tuuakse välja olemasolevate lahenduste puudujäägid ning selgitatakse välja kriteeriumid, mille alusel on võimalik luua uus lahendus. Analüüsitulemuste põhjal leitakse sobivad tehnoloogiad lahenduse elluviimiseks.

Rakendust luuakse väikeste iteratsioonide kaupa. Lahenduse loomisel teostatakse paralleelselt kasutajakogemuse uuringut, mis aitab lõpplahendusel valmida võimalikult kasutajasõbralikult.

Lahenduse valmimisel sõnastatakse tehtud töö tulemused ning tuuakse välja edasiarenduse võimalused.

3 Analüüs

Käesoleva peatüki eesmärk on välja selgitada sarnaste lahenduste võrdlemise tulemusena võimaliku lahenduse nõuded. Esmalt tuuakse välja sarnased lahendused. Seejärel kirjeldatakse, milliseid nõudeid peab lõpplahendus täitma.

3.1 Olemasolevad lahendused

Olemasolevaid rakendusi ja valdkonnapõhiseid keeli on varasemalt loodud üksikuid. Neid kõiki ühendab ühine eesmärk - lihtsustada FHIR põhiste juurutusjuhiste kirjutamist ja nende publitseerimist [2].

Sarnaste lahenduste otsimiseks kasutati Google otsingumootorit ja HL7 Github hoidlat. Analüüsimiseks valiti need lahendused, mis autori hinnangul on ajakohased ja populaarsed. Ajakohasuse ja populaarsuse hinnangus lähtuti keskkondade kasutatavuse uurimisel: veebirakenduste puhul nende sisemiste juurutusjuhiste nimekirjade vaatlemisel; tööriistade puhul Github hoidla aktiivsuse uurimisel.

Analüüsi käigus uuriti rakendusi ja keeli, mis olid tasuta kättesaadavad, kuna puudus ligipääs tasulistele lahendustele.

3.1.1 Olemasolevate lahenduste analüüs

Olemasolevatest lahendustest analüüsiti nelja rakendust ja ühte valdkonnapõhist keelt (Lisa 2). Need lahendused võib jaotada kahte gruppi, kus ühed on veebirakendused ja teised tööriistad, mis vajavad kohalikku arvutisse paigaldust. Analüüsi käigus uuriti lahendusi järgmiste küsimuste põhjal olulisuse järjekorras:

1. Kas tarkvara nõuab kasutaja arvutisse kohalikku arvutisse paigaldust või on kättesaadav pilveteenusena?
2. Kas lahendus rahuldab kõiki juurutusjuhise valmistamise vajadusi?
3. Kas juurutusjuhist on võimalik importida ja eksportida taasesitavas vormis?
4. Kas lahendus pakub võimalust ehitada lõpptulemus mitmesse keelde?

Analüüsi käigus jagas töö autor leitud olemasolevad lahendused kahte gruppi: tööriistad ja veebirakendused. Tööriistade alla liigitatud lahenduste hulka kuuluvad HL7 FHIR IG Publisher, FHIR Shorthand ja SUSHI Unshortens ShortHand Inputs. Veebirakenduste hulka kuuluvad Simplifier ja Trifolia-on-FHIR.

Tööriistade puhul võib võimekamaks lugeda HL7 FHIR IG Publisheri. See on loodud samade autorite poolt, kes on loonud HL7 FHIR standardi ja on selle aktiivsed edasiarendajad. See võimaldab luua juurutusjuhiseid nelja sammuna [3]:

1. Tööriista kohalik paigaldus
2. Ressursside kirjeldamine
3. Struktuuri ja narratiivi kirjeldamine
4. Tööriista kasutamine eelnevalt loodud failidega

Antud tööriistaga on võimalik juurutusjuhiseid valmistada kasutades kindlat kaustade ja failide struktuuri, mida on võimalik kohati muuta. Kuna töö käigus kasutatakse ainult loodud faile, siis on võimalik seda kasutada koostöö jaoks mõeldud versioonihaldustarkvaradega. Peamiseks puudujäägiks antud tööriista puhul võib tuua mitmekeelsuse puudumise, kuid see on plaanitud lisada hilisemates väljalasetes.

FHIR Shorthand on domeenipõhine keel, mis võimaldab lihtsamini kirjeldada HL7 FHIR profiile. Selle eesmärk on võimaldada juurutusjuhendite loojatel oma kavatsusi otsesemalt väljendada, tunda vähem muret FHIR-i aluseks oleva mehaanikate pärast, ja tõhusalt toota kvaliteetseid FHIR-i juurutusjuhiseid. See on mõeldud kasutamiseks koos HL7 FHIR IG Publisheri tööriistaga ning ainuke erinevus on ressursside kirjeldamiseks kasutatav keel. Selleks, et FHIR Shorthandi sisend teha HL7 FHIR IG Publisheri jaoks kasutatavaks on vaja kasutada SUSHI Unshortens ShortHand Inputs kompilaatorit. Selle kompilaatori töö on luua FHIR Shorthandi sisend loetavaks HL7 FHIR IG Publisheri jaoks, ehk muudab etteantud keele süntaksiga failid JSON ja XML kujule.

Veebirakenduste puhul võimekamaks võib lugeda Simplifier keskkonda, mida kasutavad maailmas enamus suuremad riigid ja nende riiklikud tervishoiuasutused. Simplifier on FHIR-i spetsifikatsioonide arendamise, koostöö ja avaldamise platvorm, kus lisaks juurutusjuhistele on võimalik hallata ka kõiki selle juurde käivaid muid ressursse. Selle suurimaks eelduseks on kohene lehtede esitamine. Kui teiste lahendustega peab kõigepealt leheküljed valmis kirjutama ja käivitama kompilaatorid, siis Simplifieri puhul

toimub see käigupealt ja tulemust on võimalik koheselt näha. See annab kiire ülevaate tehtavast ning kiirendab arendusprotsessi. Simplifieri suurimaks miinuseks võib lugeda selle tasuta litsentsi piirangud, ehk enamus häid funktsioone on võimalik kasutada ainult tasuliste litsentsidega.

Viimane uuritav veebirakendus on Trifolia-on-FHIR, mis oma olemuselt on vormipõhise kasutajaliidesega rakendus, kus on võimalik luua ja muuta FHIR ressursse ja juurutusjuhiseid. Sellega on lihtsasti võimalik muuta kõiki juurutusjuhise välju, lisada uusi lehekülgi ja neid muuta. Lisaks sellele on kohene sisu valideerimine, kuid töö lõpptulemuseks on HL7 FHIR IG Publisheri kasutamine Trifolia-on-FHIR tagarakenduses ning tihtipeale eesrakenduse validaator pole piisav. Trifolia-on-FHIR suurimaks eeliseks on ressursside loomine otse veebirakenduses. Suurimaks puudujäägiks võib lugeda ajakohasus, ehk antu veebirakendust ei uuendata tihti, kuigi FHIR standard, eriti kõik seonduv juurutusjuhistega on pidevas muutumises.

Analüüsist järeldab töö autor, et ükski lahendus ei ole ideaalne ning eelnevalt tõstatatud küsimustele vastates on tööriistade alla jaotatud lahendused küll head vahendid, kuid tavakasutajatele võivad need osutuda keeruliseks ning veebirakenduse näol lahendus on lihtsamini kasutatav. Nii Simplifieril kui ka Trifolia-on-FHIRil on häid kui ka halbu külgi, näiteks esimeses pole võimalik otse veebirakenduses luua ressursse, kuid viimases on. Simplifieris on aga võimalus koheselt näha töö tulemusi ning Trifolia-on-FHIRis peab selleks käivitama pika kompileerimise protsessi.

3.2 Mittefunktsionaalsed nõuded

Rakenduse kasutatavuse, laiendatavuse ja muude mittefunktsionaalsete standardite põhjal on autor välja toonud järgmised kvaliteediatribuudid.

- Rakenduse äriloogika peab olema andmebaasist eraldi sõltumatus rakenduskihis.
- Rakendust peab olema võimalik kasutada igas seadmes kasutades veebibrauserit.
- Rakendus peab olema skaleeritav.
- Rakenduse arhitektuur peab olema jaotatud erinevate loogiliste kihtide vahel.
- Rakendus peab olema kõikidele kasutajatele tasuta kättesaadav.
- Rakenduse kasutajaliidest peab olema võimalik kasutada eesti keeles.

3.3 Funktsionaalsed nõuded

Rakenduse kirjeldamiseks, selle süsteemi ja komponentide käitumise nõuetele on autor seadnud järgmised nõuded.

- Rakenduses peab olema võimalus näha kõiki süsteemis olevaid juurutusjuhiseid.
- Rakenduses peab olema võimalus luua juurutusjuhiseid ning täita kõiki selle välju, mis on kirjeldatud HL7 FHIR versioon 4 ImplementationGuide ressursis.
- Rakenduses peab olema võimalus luua ja muuta juurutusjuhisega seotud lehti.
- Rakenduses peab olema võimalus importida HL7 FHIR profiile ja neid kasutada lehtede valmistamisel.
- Rakenduses peab olema võimalus eksportida juurutusjuhiseid.

4 Tehnoloogiate valik

Antud peatükis analüüsitakse erinevate tehnoloogiate valikut lõpplahenduse loomisel. Tehnoloogiate valik on üks olulisemaid osasid antud töö juures, kuna õigete valikute tegemine kiirendab töö valmimist ning edendab edasist arendust. Tehnoloogiate valikul lähtutakse peamiselt autori pädevuses valikus olevate tehnoloogiate osas ja nende populaarsuses tervishoiu infotehnoloogia valdkonnas. Alljärgnevalt on välja toodud iga lõpplahenduse osa programmeerimiskeelte, raamistike, suurtemate teekide ja muude tehnoloogiate valikud.

4.1 Eesrakendus

Eesrakenduse valmistamise raamistike ja teeke on olemas palju ning enamused on sobilikud antud töö kirjutamiseks. Neist ühed populaarsemad on Angular, React.js ja Vue.js [4]. Raamistiku valimisel üheks oluliseks kriteeriumiks autori arvates on TypeScript tugi, sest see aitab oluliselt parandada lähtekoodi loetavust, leida kiiremini vigu ning aitab kaasa koodi staatilisel analüüsimisel [5]. Teiseks oluliseks kriteeriumiks võib tuua raamistiku jõudluse. Tuginedes GitHub koodihoidlas olevale js-framework-benchmarkile võib öelda, et eelpool nimetatud raamistikest on parim Vue.js, kuid teised kaks palju maha ei jää. Enamus teostatud testides on kõik raamistikud üsna tasavägised [6].

Tuginedes autori varasemale kogemusele eesrakenduste arendusel, on autor otsustanud eesrakenduse raamistikuks valida Angulari. Selle valiku tegemisel väldib autor uue raamistiku õppimist ning seega kiirendab arendusprotsessi.

Eesrakenduse komponentide teegid pakuvad võimalust standardiseerida arendust, vähendada koodi kordumist, parandada meeskondade vahelist koostööd ja suurendada skaleeritavust. Seega on oluline valida oma vajadustele vastav õige lahendus [7]. Lisaks on hea kasutajaliides (UI) ja kasutajakogemus (UX) on rakenduse üheks olulisemaks teguriks. Põhjus on selles, et kasutajaliides ja kasutajakogemus on lüli kasutaja ja süsteemi vahel. Hea kasutajaliides ja kasutajakogemus aitavad kasutajatel saada teavet

vastavalt sellele, mida kasutajad vajavad [8]. Populaarseimad kasutajaliidese komponentide teegid Angular raamistikule GitHub koodihoidla tähtede järgi on NG-ZORRO, PrimeNG ja Angular Material.

Töö autor proovis teha lihtsamaid kasutajaliideseid kõikide eelpool mainitud komponentide teekidega ning on siinkohal otsustanud PrimeNG kasuks, kuna see on subjektiivsel hinnangul kõige paremini kasutatav ja parimate võimalustega. Lisaks on sellel palju erinevaid sisseehitatud stiiliteemasid ning lihtsasti muudetav stiil.

Komponentide stiliseerimiseks on autor valikusse võtnud 2 populaarsemat stiili raamistikku: Bootstrap ja Tailwind CSS. Autor on otsustanud Tailwind CSS kasuks peamiselt varasemast kogemusest lähtudes ja rohkemate CSS tarbeklasside olemasolu põhjal.

4.2 Tagarakendus

Tagarakenduse programmeerimiskeeli on palju, kuid keeli, millel on juba valmis HL7 FHIR API on väheseid. Github koodihoidlast leidis autor kolm programmeerimiskeelt, mille jaoks on varasemalt kirjutatud HL7 FHIR rakendusliideseid.

Peamised kasutatavad programmeerimiskeeled on Java, C# ja JavaScript. Keele valikul on oluline arvestada FHIR teekide olemasoluga, mis lihtsustab kordades arendusprotsessi ja hõlbustab nendega ümber käimist. Suurimad Java FHIR teekide kogumid on HAPI FHIR ja IBM FHIR. C# keele jaoks on olemas Firely .NET SDK ja Microsofti FHIR Server. Kõik on oma loomult sarnased ainult mõningate erisustega ning kuna autor on varasemalt kasutanud HAPI FHIR teeke, siis on valik kallutatud just Java poole. Seega, autori hinnangul on parim keel antud probleemi lahendamiseks Java. Valiku põhjenduseks on eelnev kogemus programmeerimiskeelega ja selle jaoks kirjutatud FHIR teekidega, selle eelistus Eesti tervishoiu rakendustes [9] ning võimalik lihtne ühilduvus HL7 FHIR standardi jaoks ehitatud teekidega [10].

Tagarakenduse raamistikele ainukesed nõuded on, et kasutatakse REST arhitektuuri ja oleks piisavalt sisseehitatud võimalusi ja teeke erinevate probleemide lahendamiseks. See tagab olukorra, kus peab vähem kirjutama tüüpsisu ning saab keskenduda äriloogilistel probleemidel. Autor valis välja antud kriteeriumidele vastavaid kaks raamistikku: Spring Boot ja Micronaut. Mõlemad on oma arhitektuuril väga sarnased ja on piisavalt

funktsioonide ja teekidega täiendatud. Kui võrrelda mõlema raamistiku võimalusi, siis Spring Bootil on neid kindlasti rohkem. Lisaks on Spring Bootil suurem kasutajaskond ja seega ka arendusprotsessis välja tulnud probleemidele on suurem tõenäosus leida lahendusi veebist. Siiski, võttes arvesse autori varasemat kogemust Micronaut raamistikuga on autor otsustanud viimase kasuks. See valik kiirendab oluliselt arendusprotsessi, ehk saab vältida uue raamistiku õppimisega seotud protsessidega.

4.3 Andmebaasihaldussüsteem

Andmebaasihaldussüsteemid võib oma mudeli järgi jaotada kahte gruppi: relatsioonilised ja mitte relatsioonilised.

Relatsiooniline andmebaas on andmeüksuste kogum, mille vahel on eelnevalt määratletud seosed. Need üksused on jaotatud tabelitesse veergude ja ridadega. Tabeleid kasutatakse teabe hoidmiseks andmebaasis esitatavate objektide kohta [11], [12].

Mitte relatsioonilised andmebaasid on loodud konkreetsete andmemudelite jaoks ja neil on paindlikud skeemid kaasaegsete rakenduste loomiseks. Sellised andmebaasid kasutavad andmetele juurdepääsuks ja nende haldamiseks mitmesuguseid andmemudeleid. Seda tüüpi andmebaasid on optimeeritud spetsiaalselt rakenduste jaoks, mis nõuavad suurt andmemahtu, madalat latentsust ja paindlikke andmemudeleid, mis saavutatakse mõne teiste andmebaaside andmete järjepidevuse piirangute leevendamisega [13].

Eelnevat arvesse võttes on kindel otsus kasutada relatsioonilisi andmebaase vigade vältimise ja kindla struktuuri hoidmiseks. Populaarseimad autori poolt välja valitud relatsioonilised andmebaasihaldussüsteemid on PostgreSQL ja MySQL. Mõlemad haldussüsteemid on võrdväärsed ning valik lähtub täielikult autori isiklikust eelistusest. Lisaks, lähtudes autori varasemast kogemusest on autor otsustanud lõpplahenduse jaoks kasutada PostgreSQLi.

5 Lahendus

Antud peatükk annab ülevaate tehnilise lahenduse valmimisest vastavalt analüüsi käigus kirjeldatud nõuetele kasutades eelnevalt välja valitud tehnoloogiaid. Peatükk on jaotatud kahte ossa, kus esimene kirjeldab tagarakenduse arendust ning teine kirjeldab eesrakenduse arendust. Kuigi rakenduse osad on jaotatud erinevatesse peatükkidesse, siis arendati neid samaaegselt väikeste osade kaupa, et vähendada kommunikatsioonihäirete tõenäosust ja hõlbustada lahenduse sujuvat arendamist.

5.1 Tagarakenduse arendus

Tagarakenduse arendusel on arvestatud, et kliendipoolsed vaated genereeritakse eraldi eesrakendusega veebilehitsejas, ehk kasutatakse kliendipoolset renderdamist. Tagarakenduse ülesanne on vastutada HTTP päringute töötlemise eest ning saata vastused JSON kujul eesrakendusele arvesse võttes REST arhitektuuri stiili. Taga- ja eesrakenduse eraldamine annab võimaluse luua paremini eraldatavad rakenduse osad ning võimaluse kasutada tagarakenduse API-t ka teiste rakenduste jaoks.

5.1.1 Projekti põhi

Tagarakenduse projektipõhi loodi kasutades Micronaut Launch keskkonda, mis võimaldab kiiresti ja mugavalt seadistada rakenduse peamised atribuudid. Rakenduse loomiseks valiti Micronaut-i versioon 3.4.0 kasutades Java versiooni 17. Rakenduse ehitustööriistaks valiti Gradle ning testimise raamistikuks Spock. Antud keskkonnas lisati arenduse jaoks kõik vajalikud teegid, mis sõltuvad Micronaut raamistikust.

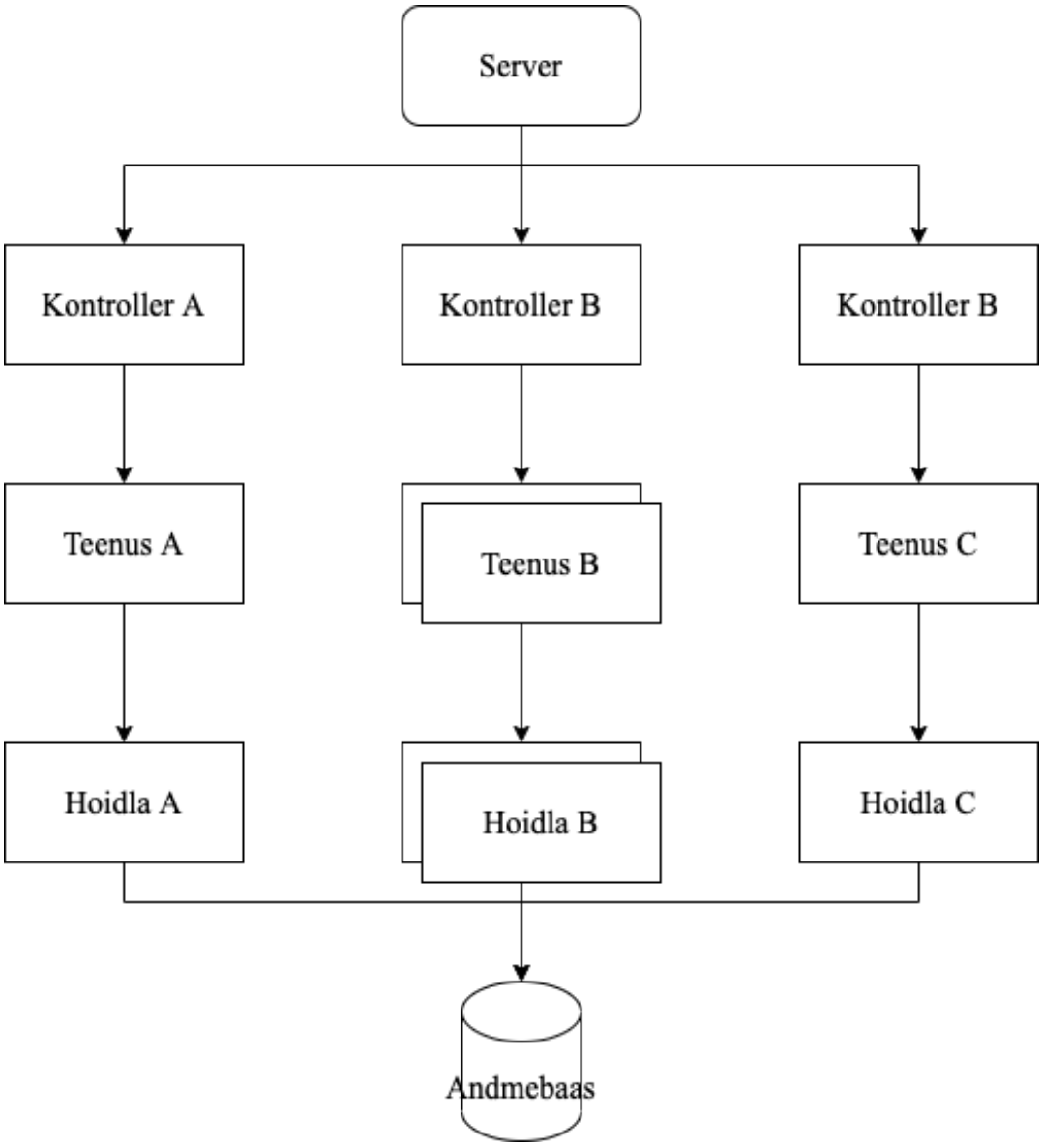
Lisaks Micronaut teekidele võeti kasutusele HAPI FHIR domeenimudelid ja süntaksianalüsaatorid lihtsustamaks ümber käimist FHIR resurssidega.

5.1.2 Struktuur

Tagarakenduse arenduses on arvesse võetud REST API disainimustrite parimaid tavasid [14]. Selline lähenemine aitab kaasa rakenduse tulevikukindluse ning andmekvaliteedi osas. Tagarakenduse koodibaasi loogika on jaotatud erinevate kihtide vahel, millel igaühel on eraldiseisev osa ning on üksteisest sõltumatud. See aitab kiiremat ja lihtsamatt

integratsiooni teiste rakendustega, vajadusel kihtide loogika muutmist ning muudab koodibaasi hallatavamaks [15]. Seega on koodibaas jaotatud kolme järgneva kihi vahel (Joonis 1):

- Kontrollerite kiht - vastutab ainult töö delegeerimise eest ning ei sisalda äriloogikat ega manipuleeri andmeid. Peamine eesmärk on HTTP päringute kätte saamine, nende sisu edasi saatmine järgmisele kihile, ehk teenustele ja sealt saadud vastuste tagastamine.
- Teenusekiht - vastutab äriloogiliste ülesannete eest, ehk manipuleerib andmetega ning lisatoimingute teostamine. Selles kihis on palju erinevaid teenuseid, millel igaühel on kindel ja spetsiifiline ülesanne.
- Püsitalletuse kiht - vastutab andmete talletamise eest andmebaasis, ehk suhtleb andmebaasihaldussüsteemiga. See toimib vahendajana rakenduse ärifunktsioonide ja andmebaasis talletavate andmete vahel.



Joonis 1. . Tagarakenduse üldine arhitektuur.

5.1.3 Kirjeldus

Rakenduse arenduses on olulist rõhku pandud FHIR RESTful rakendusliidese reeglitele ja soovitustele [21], et tulevikus FHIR serveritega ühildumist lihtsustada. ka eeskuju võetud IBM FHIR serverist [19], HAPI FHIR [20] teekide kogumist ja KeFHIR [22] serveri arhitektuurist ja disainimustritest.

Ressursid, mida kasutatakse rakenduse käigus on kõik üsnagi sarnased ja on laiendatud baas ressurssidest. Seega on autor kirjutanud ressursside töötlemise jaoks baas teenuse ja hoidla Java abstraktsed klassid, mida kõik teised ressurssidega töötavad klassid kasutavad ja laiendavad. Selline muster aitab ühtlustada ressurssidega ümber käimist ning võimalike lisavõimaluste juurde lisamist lihtsustada. Lisaks väldib programmikoodi mitmekordset

kirjutamist. Kui on vaja mingi ressursi töötlemiseks teisi operatsioone, mida teiste ressurssidega ei tehta, siis on võimalik neid juurde kirjutada vastava ressursiga seotud klassi juures. Näide abstraktsest alushoidla klassi deklareerimisest ning kahe meetodi sisu, mis võimaldab CRUD operatsioone ressurssidega on välja toodud joonisel 2.

```
public abstract class ResourceRepository<T extends Resource> extends
BaseRepository<T> implements IResourceRepository<T> {
  private final JdbcTemplate jdbcTemplate;
  private final FhirNamedClass<T> fhirNamedClass;

  public ResourceRepository(FhirContext fhirContext, JdbcTemplate
jdbcTemplate, FhirNamedClass<T> fhirNamedClass) {
    super(fhirContext);
    this.jdbcTemplate = jdbcTemplate;
    this.fhirNamedClass = fhirNamedClass;
  }

  @Override
  public List<T> findAll() {
    String sql = """
            select * from resource where resource ->> 'resourceType' = ?;
            """;

    return jdbcTemplate.query(sql, this::rowMapper,
fhirNamedClass.resourceType().name());
  }

  @Override
  public boolean update(String id, T resource) {
    PGobject jsonObject = toJsonb(resource);
    String sql = """
            update resource set resource_id = ?, resource = ?::jsonb where
resource_id = ? and resource ->> 'resourceType' = ?;
            """;

    return jdbcTemplate.update(sql, id, jsonObject,
resource.getIdElement().getIdPart(), fhirNamedClass.resourceType().name()) ==
1;
  }
}
```

Joonis 2. Abstraktse alushoidla klassi deklaratsioon.

Selline lahendus aitab lihtsalt uute hoidla klasside kirjutamist ja nende muutmist. Kõik seda klassi laiendavad klassid saavad kasutada selle meetodeid ja *ResourceRepository*-sse uute meetodite lisamisel samuti. Näide antud abstraktse klassi kasutamisest *ImplementationGuide* ressursi hoidla näitel on joonisel 3.

```
public class ImplementationGuideRepository
    extends ResourceRepository<ImplementationGuide> {

  public ImplementationGuideRepository(FhirContext fhirContext, JdbcTemplate
jdbcTemplate) {
    super(fhirContext, jdbcTemplate,
        new FhirNamedClass<>(ImplementationGuide.class,
            ResourceType.ImplementationGuide));
  }
}
```

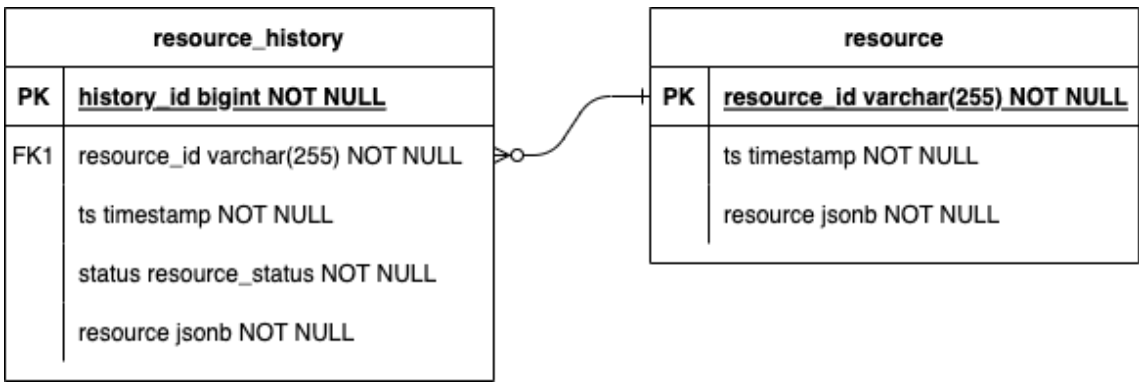
Joonis 3. ImplementationGuide ressursi hoidla.

5.1.4 Olem-seos-mudel

Tagarakendus suhtleb andmebaasihaldussüsteemiga, milleks on PostgreSQL. See on relatsioonilise andmebaasi haldussüsteem, kus olemid on omavahelistes seostes [16].

HL7 FHIR andmemudelid on keerulised ja struktuuriga, mille sisu võib olla etteaimatav. Igal ressursi elemendil võivad olla laienduse alamelemendid, mis omavad lisateavet, mis ei kuulu ressursi põhidefinitsiooni hulka [17]. Selliselt on tavapärase relatsioonilise olem-seos-mudeli loomine keerukas ja veaohtlik. Selleks on töö autor otsustanud kasutada PostgreSQL andmetüüpi jsonb, mis hõlbustab ressursside hoidmist andmebaasis. Ressursside vahelised suhted hoitakse neis endis ning olemite vahelised suhted hoitakse olemites võõrvõtmetena.

Andmebaasimudeli loomisel lähtuti suuresti KeFHIR serveri [22], IBM FHIR serveri [19] ja HAPI FHIR andmebaasimudelitest [20], mille alusel töö autor otsustas minimaalse lahenduse ja rakenduse lihtsuse huvides kasutada lihtsustatud kujul sarnaseid mudeleid. Ressursside ajalugu ja muudatused hoitakse eraldi tabelis, millel on seos põhitabeliga. Olem-seos-mudeli diagramm on kirjeldatud joonisel 4 ja on loodud draw.io keskkonnas.



Joonis 4. Olem-seos-mudeli diagramm.

Ressursside sisestamisel andmebaasi muudetakse need kõigepealt PostgreSQL jaoks sobilikku JSON formaati. Peamisesse tabelisse, ehk resource tabelisse lisatakse ka viimase muudatuse ajatempel. Iga operatsioon mingi kirje kohta tekitab sellele eelneva seisu kopeerimine resource_history tabelisse ning seejärel muudetakse resource tabelis antud kirjet. Selline lahendus on saavutatud PostgreSQL funktsioonide ja päästikutega, kus joonisel 5 on toodud näide kirje lisamise funktsioonis ja päästikust.

```
create or replace function resource_insert() returns trigger as
$$
begin
    insert into resource_history
        (resource_id, status, resource)
    values (new.resource_id, 'active', new.resource);
    return new;
end;
$$
    language plpgsql;

create trigger resource_insert_trigger
    after insert
    on resource
    for each row
execute procedure resource_insert();
```

Joonis 5. Kirje tekitamise funktsioon ja päästik.

Analoogselt on tehtud ka uuenduse funktsioon ja päästik ning ainuke erinevus kirje kustutamise funktsioonis on, et kirjet reaalselt tabelist ei kustutata (Joonis 6). Andmete võimaliku taasesitamise ja ajaloo säilitamise mõttes on kõikidel kirjetel kas *active* või *inactive* staatus, ehk ainult viimasel versioonil kirjest on olemas active staatus. Kui kirjet tahetakse kustutada, siis muudetakse see inactive staatusesse.

```
create or replace function resource_delete() returns trigger as
$$
begin
    update resource_history
    set status = 'inactive'
    where status = 'active' and resource_id = old.resource_id;
    update resource
    set status = 'inactive'
    where status = 'active' and resource_id = old.resource_id;
    return null;
end;
$$
    language plpgsql;

create trigger resource_delete_trigger
    before delete
    on resource
    for each row
execute procedure resource_delete();
```

Joonis 6. Kirje kustutamise funktsioon ja päästik.

5.2 Eesrakenduse arendus

Eesrakenduse ülesanne on tagarakendusega suhtlemine ja andmete kuvamine loetaval kujul. Lisaks on vajalik andmete esitamine tagarakendusele, et neid talletada andmebaasis.

5.2.1 Projekti põhi

Eesrakenduse projektipõhi on loodud kasutades Angulari käsurea liidest, mis võimaldab kiirelt püsti seada alusprojekti ning teha esimesed otsused koodibaasi stiili osas. Projekti alustamise hetkel kasutati Angulari versiooni 13.2.6 ning lisati vajalikud angulari poolt loodud teegid. Lisaks sellele lisati PrimeNG komponentide teek ning Tailwind CSS stiili raamistik. HL7 FHIR domeenimudelitega ümber käimiseks kasutatakse DefinitelyTyped poolt loodud TypeScript tüüpe.

5.2.2 Struktuur

Angulari rakenduste arhitektuur sõltub kindlastest fundamentaalsetest kontseptsioonidest. Põhilised Angulari ehitusplokid on Angulari komponendid, mis on jaotatud Angulari mooduliteks (NgModule). Angulari rakendustel on alati vähemalt

juurmoodul, mis võimaldab alglaadimist, ning tüüpiliselt mitmeid muid erimooduleid [18].

Eesrakendus on jaotatud põhiliselt kahe kihi vahel. Esimene kiht on vaate kiht, mille ülesanne on kuvada kliendile andmed ilusal kujul ja võimaldada kasutajaliidese kaudu liikumist. Teine kiht on teenuse kiht, mille ülesandeks on tagarakendusega ja vaate kihiga andmete vahetamine.

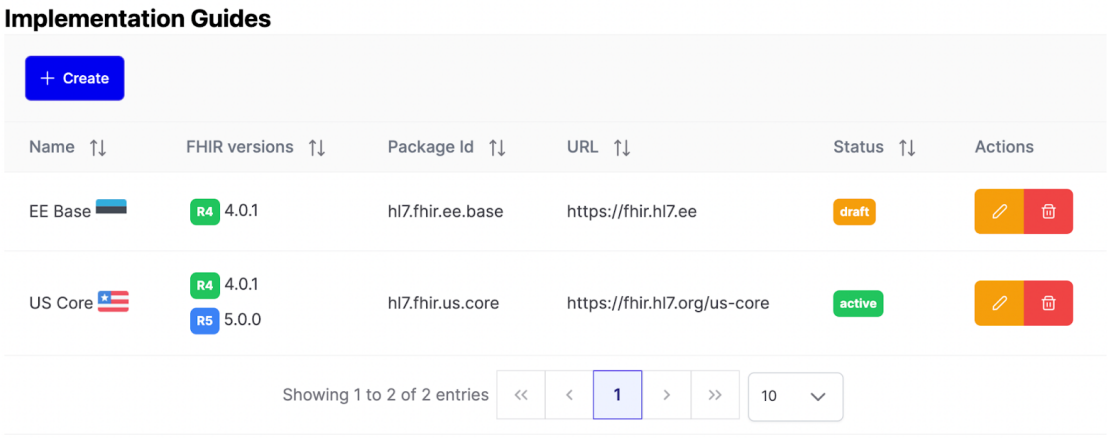
Rakenduse lähtekoodi kirjutamisel on lähtutud sarnaselt tagarakendusega, kus erinevate FHIR ressursside jaoks on loodud alusklassid teenusekihis, mida iga ressursi jaoks on võimalik laiendada.

5.2.3 Vaated

Rakenduses on kasutusel erinevad vaated, millel igalühel on oma ülesanne. Minimaalse elujõulise toote jaoks realiseeriti rakenduse põhifunktsionaalsust omavad vaated, kus on võimalik juurutusjuhiseid näha, muuta ja uut valmistada. Kõiki vaateid on võimalik näha mitmes keeles ning antud töö raames realiseeriti need nii inglise kui ka eesti keeles.

Vaadete stiliseerimisel on kasutatud PrimeNG valmiskomponente ja stiliseeritud TailwindCSS stiiliraamistiku abil.

Kasutajal on võimalik näha kõiki antud süsteemis olevaid juurutusjuhiseid ja nende põhiandmeid nimekirja vaates (Joonis 7). Samast vaatest on võimalik juurutusjuhiseid kustutada ning liikuda järgmistesse vaadetesse, kas uue juurutusjuhise loomiseks või olemasoleva muutmiseks.

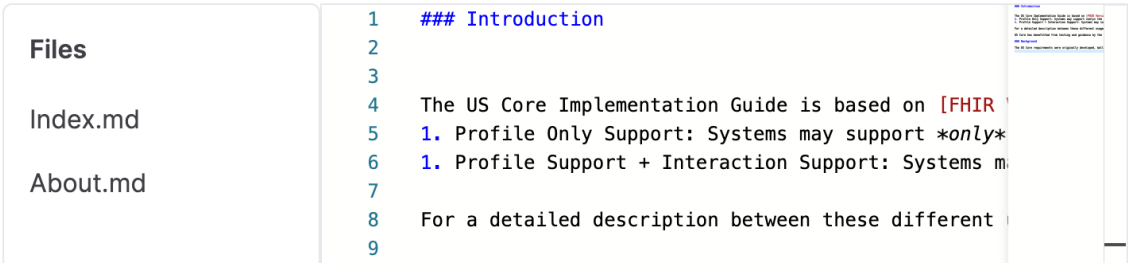


Joonis 7. Juurutusjuhiste nimekirja vaade.

Uue juurutusjuhise loomise vaates (Joonis 8) on võimalik vormi näol täita kohustuslikud ImplementationGuide ressursi väljad ning vormi edukal esitamisel liigutakse edasi juurutusjuhise muutmise vaatesse. Muutmise vaates on analoogsed vormielemendid, mille abil on võimalik kõiki ImplementationGuide ressursi välju täita ning uusi elemente vajadusel lisada.

Joonis 8. Juurutusjuhiste loomise vaade.

Juurutusjuhiste juurde käivaid lehekülgi on võimalik luua ja muuta lehekülgede vaates antud juurutusjuhise juures. Selles vaates on nimekirjana konkreetse juurutusjuhise leheküljed ning Microsofti poolt loodud esirakendustele mõeldud tekstiredaktor *Monaco Editor* nende muutmiseks (Joonis 9).



Joonis 9. Juurutusjuhiste lehekülgede vaade.

6 Tulemus

Käesolevas peatükis annab töö autor ülevaate loodud lahendusest, saavutatud tulemustest ning kirjeldab edasiarengute võimalusi.

Bakalaureusetöö raames valmis minimaalne elujõuline veebirakendus, kus on võimalik luua juurutusjuhiseid. Töö autor realiseeris eelnevalt kirja pandud nõuete kohaselt nii ees- kui ka tagarakenduse. Peamine rõhk töö käigus oli luua alused täieliku toote valmistamiseks ning kirjutada lähtekood selliselt, et oleks võimalik lihtsalt luua uusi funktsioone ja täiendusi.

Antud tööd on võimalik edasi arendada igast küljest ning rakenduse kõik osad on selliselt kirjutatud, et lisaarenduste tegemine oleks lihtne. Töö käigus jõudis autor arusaamale, et tavalise tagarakenduse olemasolu ei ole nii tähtis ning pigem oleks vaja täieulatuslikku FHIR standardi spetsiifilist serverit. Samas lisaks see omajagu keerukust, kuid annab võimaluse FHIR ressurssidega ümerkäimisel maksimaalset paindlikkust, nende valideerimist ja palju muud. Kindlasti on täieliku toote jaoks vaja luua kasutajate süsteem, et võõrad kasutajad ei saaks muute teiste loodud juurutusjuhiseid.

Eesrakendust on võimalik edasi arendada lisades lehtede loomise juurde mallide võimalus, et tavalise Markdown keele asemel kasutada kohandatavaid HTML komponente. Üheks suurimaks edasiarenduseks võib lugeda ressursside loomist otse veebirakenduses, milleks üks lihtsaim võimalus on tekstiredaktoriga JSON, XML või mõnes muus formaadis failide muutmine.

7 Kokkuvõte

Käesoleva bakalaureusetöö eesmärk oli analüüsida Health Level 7 Fast Healthcare Interoperability Resources standardi juurutusjuhiste loomise võimalusi pakkuvaid lahendusi ja programme ning luua selle alusel minimaalne töötav toode, mis loob eeldused täieliku lahenduse valmistamiseks.

Töö jooksul analüüsiti erinevaid turul olevaid lahendusi ning leiti nende head ja halvad küljed, mille põhjal töö autor koostas kriteeriumid lõpplahendusele. Seejärel tegi töö autor lõpplahenduse valmistamise tehnoloogilised valikud koos põhjendustega ning leidis suuremad teekide kogumid, mida realiseerimisel kasutada. Viimaks kirjeldati lõpplahenduse valmimist, milleks oli veebirakendus nii esi- kui ka tagarakenduse näol. Lahenduse kirjeldamisel kirjeldati projekti kõiki osi, ehk alustades projekti loomisest kuni selle lõpptulemusteni. Lisaks kirjeldati ka andmebaasimudelit ning selle kasutamist tagarakenduse kaudu.

Töö tulemuseks on veebirakendus, mis valmis minimaalse elujõulise tootena ning seab eeldused täieliku konkurentsivõimelise veebirakenduse valmimiseks.

Püstitatud eesmärgid said täidetud ning autor täiendas teadmisi antud teemal piisavalt, et valmistada konkureeriv toode. Rakendust arendatakse edasi ning saab kasutust Eesti tervishoius uue standardile ülemineku protsessides.

Kasutatud kirjandus

[1] Tervise ja Heaolu Infosüsteemide Keskus, *Teabekeskus*, [Online]. Loetud aadressil: https://www.tehik.ee/teabekeskus. Kasutatud 30.11.2021.

[2] Health Level Seven, *HL7.FHIR.UV.SHORTHAND\FHIR Shorthand - FHIR v4.0.1*, [Online]. Loetud aadressil: http://hl7.org/fhir/uv/shorthand/2020May/index.html. Kasutatud 2.12.2021.

[3] Confluence Pages of Health Level 7 (HL7) International, *IG Publisher Documentation*, 2022 [Online]. Loetud aadressil: https://confluence.hl7.org/display/FHIR/IG+Publisher+Documentation Kasutatud 24.04.2022.

[4] Stack Overflow, *Stack Overflow Developer Survey 2021*, [Online]. Loetud aadressil: https://insights.stackoverflow.com/survey/2021#section-most-popular-technologies-web-frameworks Kasutatud 24.04.2022.

[5] TypeScript, *The TypeScript Handbook*, [Online]. Loetud aadressil: https://www.typescriptlang.org/docs/handbook/intro.html Kasutatud 24.04.2022.

[6] Github, *js-framework-benchmark*, [Online]. Loetud aadressil: https://github.com/krausest/js-framework-benchmark Kasutatud 24.04.2022.

[7] Studio By UXPin, *What Is a Component Library and Why Should You Use One for UI Development?*, [Online]. Loetud aadressil: https://www.uxpin.com/studio/blog/ui-component-library Kasutatud 24.04.2022.

[8] T. Pratama και A. Cahyadi, "Effect of User Interface and User Experience on Application Sales," IOP Conference Series: Materials Science and Engineering, τ. 879, σ. 012133, 08 2020.

[9] Tervise ja Heaolu Infosüsteemide Keskus, *Sotsiaalministeeriumi haldusala riist- ja tarkvara ning e-teenuste üldine haldamise ja arendamise infotehnoloogiline standard*, 2021 [Online]. Loetud aadressil: https://www.tehik.ee/sites/default/files/2021-04/AV-IT-Profiil-280920-1316-38.pdf Kasutatud 24.04.2022

[10] HAPI FHIR, *Working With Resources*, [Online]. Loetud aadressil: https://hapifhir.io/hapi-fhir/docs/model/working_with_resources.html Kasutatud 24.04.2022

[11] IBM, Relational Databases, [Online]. Loetud aadressil: https://www.ibm.com/cloud/learn/relational-databases Kasutatud 24.04.2022

[12] Amazon, *What is a Relational Database?*, [Online]. Loetud aadressil: https://aws.amazon.com/relational-database/ Kasutatud 24.04.2022

[13] Amazon, *What is NoSQL?*, [Online]. Loetud aadressil: https://aws.amazon.com/nosql/ Kasutatud 24.04.2022

[14] Stack Overflow Blog, *Best practices for REST API design*, 2020 [Online]. Loetud aadressil: https://stackoverflow.blog/2020/03/02/best-practices-for-rest-api-design/ Kasutatud: 25.04.2022

[15] Oracle, *Service Layer Architecture*, [Online]. Loetud aadressil: https://docs.oracle.com/cd/E93130_01/service_layer/service%20layer%20API/Content/Serv

ice%20Layer%20Architecture/Service%20Layer%20Architecture.htm Kasutatud: 25.04.2022

[16] PostgreSQL, *About*, [Online]. Loetud aadressil: https://www.postgresql.org/about/ Kasutatud: 25.04.2022

[17] HL7 FHIR, *Extensibility*, [Online]. Loetud aadressil: https://www.hl7.org/fhir/extensibility.html Kasutatud: 25.04.2022

[18] Angular, *Introduction to Angular concepts*, [Online]. Loetud aadressil: https://angular.io/guide/architecture Kasutatud: 25.04.2022

[19] HAPI FHIR, *Database Schema*, [Online]. Loetud aadressil: https://hapifhir.io/hapi-fhir/docs/server_jpa/schema.html Kasutatud: 15.05.2022

[20] Github, *The IBM FHIR Server - Schema Design and Management*, 2022, [Online]. Loetud aadressil: https://github.com/IBM/FHIR/blob/main/fhir-persistence-schema/docs/SchemaDesign.md Kasutatud 15.05.2022

[21] HL7 FHIR, *Http*, 2019 [Online]. Loetud aadressil: https://www.hl7.org/fhir/http.html Kasutatud 15.05.2022

[22] Gitlab, *KeFhir*, 2022 [Online]. Loetud aadressil: https://gitlab.com/kodality-public/kefhir Kasutatud 16.05.2022

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

Mina, Gen Metsaveer

1. Annan Tallinna Tehnikaülikoolile tasuta loa (lihtlitsentsi) enda loodud teose „Juurutusjuhiste modelleerimis- ja publitseerimisvahend Eesti tervishoiu teenustele“, mille juhendaja on Igor Bossenko
   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.

16.06.2022

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

Lisa 2 - Olemasolevate lahenduste analüüsis kasutatud lahendused

Rakendused

- HL7 FHIR IG Publisher - https://github.com/HL7/fhir-ig-publisher
- FHIR Shorthand - https://build.fhir.org/ig/HL7/fhir-shorthand/
- Trifolia-on-FHIR - https://trifolia-fhir.lantanagroup.com/lantana_hapi_r4/home
- Simplifier - https://simplifier.net/

Valdkonnapõhised keeled

- SUSHI Unshortens ShortHand Inputs - https://fshschool.org/docs/sushi/