TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Priit Tohver 134655IAPB

# FÜSIOTERAAPIA ETTEVÕTTE INFOSÜSTEEMI PATSIENTIDE JA VISIITIDE HALDAMISE ALLSÜSTEEM VEEBIRAKENDUSE KUJUL

Bakalaureusetöö

Juhendaja: Ago Luberg
MSc

Tallinn 2018

# 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: Priit Tohver

10.01.2018

# Annotatsioon

Lõputöö eesmärgiks on arendada kliendi poolt tellitud infosüsteemi allsüsteemi realiseeriv veebirakendus. Realiseeritav allsüsteem tegeleb patsientide ja visiitide haldamisega. Süsteemi lahendust sooviti veebirakenduse kujul, et seda oleks võimalik kasutada iga uuema veebilehitsejaga sõltumata platvormist. Veebirakenduse tagakomponendi arendamiseks kasutati Spring Boot raamistikku ning esikomponendi arendamiseks AngularJS raamistikku. Andmebaas realiseeriti PostgreSQL andmebaasisüsteemis. Veebirakendus võimaldab patsientidel endale valitud arsti juurde visiit registreerida ning arstidel hallata süsteemis olevaid patsiente ja visiite. Lisaks on arstidel lubatud kalendrisse märkida aegu, kuhu ei saa visiite registreerida ning võimalus eksportida andmebaasist infot kõikide temaga seotud visiitide kohta Microsoft Exceliga avatava failina.

Töö annab ülevaate infosüsteemi analüüsist ning allsüsteemi realisatsioonis kasutatud tehnoloogiatest. Lisaks võrreldakse kasutatud tehnoloogiaid teiste hetkel kasutusel olevatega ning loodavat rakendust teiste olemasolevate lahendustega.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 30 leheküljel, 3 peatükki, 20 joonist, 1 tabelit.

# Abstract

THE DEVELOPMENT OF THE PATIENT AND VISIT MANAGEMENT SUBSYSTEM AS A WEB APPLICATION FOR A PHYSIOTHERAPY COMPANY'S INFORMATION SYSTEM

The purpose of this thesis is the development of an information system's subsystem for managing patients and visits as a web application, requested by a client that owns a physiotherapy company. The client requested a web application that would be useable with every newer web browser regardless of platform. The Spring Boot framework was used for the development of the back-end of the application and the AngularJS framework was used for the front-end of the application. The database was created using the PostgreSQL database system. The web application allows patients to register visits with a specific doctor and doctors to manage the patients and the visits that exist in the system. Additionaly, doctors are allowed to choose dates and times where patients cannot register for visits and have the ability to export data from the database regarding visits, related to the doctor, as an Microsoft Excel file.

The thesis gives an overview of the analasys of the system and the technologies used for the implementation of the web application. The author will also compare the used technologies with other currently relevant technologies and the web application to other currently available solutions.

The thesis is in Estonian and contains 30 pages of text, 3 chapters, 20 figures, 1 table.

# Lühendite ja mõistete sõnastik

| | |
|---|---|
| AngularJS | Dünaamiliste veebirakenduste loomiseks mõeldud esikomponendi raamistik |
| API | *Application Programming Interface*, määratud reeglistik, mille vahendusel erinevad tarkvarakomponendid omavahel suhtlevad |
| BCrypt | Paroolide räsimise algoritm |
| CSS | *Cascading Style Sheets*, veebidokumentidele stiili lisamise keel |
| Docker | Rakenduste ehitamise ja käivitamise platvorm |
| Esikomponent | *Front-end*, kasutaja arvutis töötav veebirakenduse osa |
| Git | Hajutatud versioonihaldustarkvara |
| GitLab | Giti repositooriumite loomise ja haldamise rakendus |
| HTML | *HyperText Markup Language*, enim levinud keel veebidokumentide loomiseks |
| Java | Platvormist sõltumatu objektorienteeritud programmeerimiskeel |
| Jenkins | Pidevintegratsiooni konveierite loomise vahend |
| JSON | *JavaScript Object Notation*, andmevahetusformaat, alternatiiv XML-le |
| jUnit | Java koodi ühiktestimiseks kasutatav teek |
| JWT | *JSON Web Token*, kasutaja tuvastamiseks kasutatav pääsuluba |
| Kanban | Agiilne tarkvara arendamise metoodika |
| Oracle | Relatsiooniline andmebaasisüsteem |
| PostgreSQL | Relatsiooniline andmebaasisüsteem |
| REST | *Representational State Transfer*, hajussüsteemides (nt veebis) kasutatav tarkvaraarhitektuuri stiil |
| SASS | *Syntactically Awesome Style Sheets*, CSS-i laiendus, mis lisab teistele programmeerimiskeeltele omaseid võimalusi (nt muutujad) |
| Scrum | Agiilne tarkvara arendamise metoodika |
| Selenium | Veebilehitsejas tehtavate tegevuste automatiseerimise vahend |
| Spring Boot | Java põhjal ehitatud võrgurakenduste loomise raamistik |
| SQL | *Structured Query Language*, struktuurpäringukeel, mida kasutatakse relatsioonilise andmebaasiga suhtlemiseks |

| | |
|---|---|
| SSH | *Secure Shell*, krüptograafiline võrguprotokoll võrguteenuste turvaliseks kasutamiseks üle turvamata võrgu |
| Tagakomponent | *Back-end*, serveris töötav veebirakenduse osa |
| Veebilehitseja | *Web browser*, veebidokumentide vaatamiseks mõeldud programm |
| XML | *Extensible Markup Language*, andmevahetusformaat, mille eesmärgiks on andmete jagamine erinevate infosüsteemide vahel |

# Sisukord

# Jooniste loetelu

# Tabelite loetelu

# 1 Sissejuhatus

Käesoleva bakalaureusetöö eesmärk on arendada kliendi poolt tellitud infosüsteemi allsüsteem. Allsüsteem tegeleb füsioteraapia ettevõtte patsientide ja nendega seotud visiitide haldamisega. Seni on kliendil kasutusel olnud Exceliga loodud tabelid, kuid patsientide hulga suurenedes on nende täitmine ja haldamine tülikaks muutunud. Allsüsteemi realisatsioonina loodava veebirakenduse peamiseks eesmärgiks on lihtsustada ja seeläbi kiirendada kliendi igapäevaseid toiminguid patsientide ja visiitide andmetega.

## 1.1 Soovitav infosüsteem

Klient soovis uut infosüsteemi, kuna hetkel kasutusel olev tabelite süsteem ei ole enam piisavalt efektiivne. Soovitud infosüsteem peaks lihtsustama patisentide ja nendega seotud visiitide lisamist ning haldamist ja võimaldama patsientidel soovi korral ise ennast visiidile registreerida. Samas peaks säilima MS Excelis juba olemasolev vajalike andmete otsimise funktsionaalsus. Infosüsteemi implementatsiooni soovis klient veebirakenduse kujul, aga lisas, et rakendus peaks piiratud funktsionaalsusega olema kasutatav ka ilma võrguühenduseta.

## 1.2 Ülesandepüstitus

Antud bakalaureusetöö raames keskendutakse ainult patsientide ja visiitide haldamise allsüsteemi arendusele. Arenduse tulemusena valmiv veebirakendus peab lubama patsientidel ennast visiitidele registreerida ning arstidel patsiente ja nendega seotud visiite hallata. Arendust teostatakse töö käigus loodud infosüsteemi analüüsist lähtudes.

## 1.3 Ülevaade tööst

Lõputöö on jaotatud kolmeks peatükiks. Esimeses peatükis antakse ülevaade kliendiga toimunud suhtlusest ja kirjeldatakse realiseeritava allsüsteemi ning osaliselt ka kogu infosüsteemi analüüsi. Teises peatükis põhjendatakse rakenduse loomisel kasutatud

tehnoloogiate valikut ning võrreldakse neid teiste laialt kasutusel olevate tehnoloogiatega. Lisaks selgitatakse, miks ei võetud kasutusele mõnda juba olemasolevat sarnase ülesande jaoks mõeldud infosüsteemi ning seejärel kirjutatakse lähemalt rakenduse loomisest. Kolmandas peatükis tuuakse välja tähtsamad punktid, millele rakenduse edasises arenduses rõhku pannakse.

# 2 Infosüsteemi analüüs

## 2.1 Kohtumised kliendiga

Kuna rakendus on kliendi poolt tellitud, lähtutakse analüüsil ja arendamisel võimalikult palju kliendilt saadud tagasisidest kasutajaliidese ja funktsionaalsuse osas. Kliendiga kohtumised toimuvad keskmiselt korra kuus ning kohtumistel demonstreeritakse kliendile juba olemasolevat funktsionaalsust ning arutatakse uute nõuete lisandumist ja vanade nõuete muutmist. Kohtumiste tulemusena täiendatakse pidevalt tarkvara edukaks arendamiseks vajalikku funktsionaalsete ja mittefunktsionaalsete nõuete analüüsi, mille põhjal toimub kogu edasine arendus.

## 2.2 Nõuded

Käesolevas alapeatükis tuuakse välja planeeritava infosüsteemi mittefunktsionaalsed nõuded ning kasutusjuhud. Lõputöö raames osaliselt või täielikult mitte realiseeritavad nõuded on tähistatud tärniga (*) ning lühidalt selgitatud.

### 2.2.1 Mittefunktsionaalsed nõuded

Tabel 1. Infosüsteemi mittefunktsionaalsed nõuded.

| **Tüüp** | **Nõude kirjeldus** |
|---|---|
| Keel | Valminud rakenduse kasutajaliidest peab olema võimalik kasutada eesti, vene ja inglise keeles. (* – esialgu soovis klient rakendust eesti keeles ning alles valminud rakendusele teiste keelte lisamist) |
| Töökiirus | Päringute tegemisel ei tohi vastuste kuvamine ega andmemuudatuste salvestamine võtta kauem kui 5 sekundit. |
| Töökindlus | Valminud rakenduse tõrgeteta töö on vajalik ettevõtte tõrgeteta töötamiseks. Kuna rakenduse kaudu toimub ettevõtte põhiandmete haldamine, siis põhjustaks süsteemi tõrge ettevõtte töö seisaku.<br>Juhul kui tekkinud tõrke tagajärjel andmebaas või rakendus kahjustub, tuleb need taastada viimase varukoopia põhjal ühe tunni jooksul peale tõrke põhjuse kõrvaldamist. Maksimaalselt võivad kaduma minna viimase 24 tunni andmed. (* – hetkel andmebaasist varukoopiaid ei tehta kuna andmebaas sisaldab vaid testandmeid) |

| Tüüp | Nõude kirjeldus |
|---|---|
| | Kuna tegu on ettevõtte korrektseks tööks vajaliku süsteemiga, peab rakendus olema piiratud funktsionaalsusega kasutatav ka internetiühenduse puudumisel. (* – klient ei ole nõutavat funktsionaalsust veel täpsustanud) |
| Kasutajaliides | Rakendus on mõeldud kasutamiseks arvutis, seega ei pea suurt rõhku panema mobiilis/tahvelarvutis kasutatavale kasutajaliidesele.<br>Nõuded kasutajaliidese ülesehitusele:<br>▪ Rakenduse ülemises servas peab igal lehel olema navigeerimise riba<br>▪ Kuupäevad tuleb esitada formaadis DD/MM/YYYY<br>▪ Kellaajad tuleb esitada formaadis HH:MM (24h)<br>▪ Vormi kinnitades tuleb kasutajale näidata, millised vormiväljad on ebakorrektselt täidetud ning kuvada abistavaid sõnumeid vigade kohta (* – osaliselt realiseeritud, aga vajab kliendi tagasisidet)<br>▪ Kaalu kuvatakse kilogrammides (kg) ja pikkust sentimeetrites (cm)<br>▪ Enne igat kustutamisoperatsiooni tuleb kasutajalt küsida kinnitust |
| Turvalisus | Patsientide andmetele ligi pääsemiseks peab kasutaja süsteemi sisse logima. Uusi kasutajaid tohib luua ainult olemasolev, administraatori õigustega kasutaja. (* – kasutajate haldamine ei ole osa realiseeritavast allsüsteemist)<br>Andmebaasis hoitav parool ei tohi olla avatekst, vaid peab olema olema räsiväärtus, mis on kasutaja paroolist genereeritud.<br>Sisselogimiseks kasutatavad e-maili aadress ning parool peavad olema vastavalt tõstutundetu ning tõstutundlik. Sisselogimiseks peab olema võimalik kasutada ka ID-kaarti või Mobiil-ID-d.<br>(* – allsüsteemi raames on realiseeritud ainult e-maili aadressi ning parooliga sisselogimine, et testida kasutaja autentimist vajavaid süsteemi osi)<br>Rakenduse esi- ja tagakomponendi vaheliseks suhtluseks tuleb kasutada turvalist HTTPS (*HyperText Transport Protocol Secure*) protokolli. (* – turvalisusega tegeletakse ülejäänud funktsionaalsuse valmimise järel) |
| Testitavus | Rakendus peab olema loodud selliselt, et selle põhjal oleks võimalikult kerge kirjutada automaatteste. Testidega peavad olema kaetud tagakomponendi osad, mis tegelevad patsientide |

| Tüüp | Nõude kirjeldus |
|---|---|
| | ning visiitide info käsitlemisega. (* – aja kokkuhoiu huvides jääb testide kirjutamine suures osas lõputööst välja) |
| Skaleeritavus | Rakendus peab olema loodud selliselt, et soovi korral oleks võimalikult lihtne rakendusele uut funktsionaalsust lisada. Rakenduse loomisel tuleb lähtuda kõrge sidususe ja madala sidestuse (*high cohesion, low coupling*) põhimõtetest. |

### 2.2.2 Kasutusjuhud

**Kasutusjuht:** Visiidile registreerimine

**Tegutsejad:** Patsient

**Tüüpiline sündmuste järjestus:**

1. Patsient soovib uuele visiidile registreerida.
2. Süsteem avab vormi, kus patsient saab sisestada oma nime ning kontaktandmed ja valida arsti, kelle juurde visiit registreerida. Lisaks on patsiendil võimalik lisada märkmeid selle kohta, miks ta visiidile registreerida soovib.
3. Süsteem kontrollib, et sisestatud andmetega patsient eksisteeriks andmebaasis.
4. Kui vorm on korrektselt täidetud, avab süsteem valitud arsti vabasid vastuvõtuaegu sisaldava kalendri.
5. Sobivale kuupäevale vajutades avaneb aken, kus patsient saab valida antud kuupäeva vabade vastuvõtuaegade seast endale sobiva.
6. Peale patsiendi kinnitust salvestab süsteem visiidi andmebaasi.
7. Kui patsiendi e-maili aadress on süsteemile teada, saadetakse sellele kinnitus visiidi registreerimisest (* – e-mailide saatmise funktsionaalsus tuleb kliendiga üle täpsustada).

**Laiendus** (Kui patsient soovib esimest korda visiidile registreerida)**:**

2. Süsteem avab vormi, kus lisaks kontaktandmetele küsitakse patsiendilt ka vajalikke lisaandmeid (pikkus, kaal, amet, töö iseloom, spordi harrastamist) ning eelnevate diagnooside/vigastuste ajalugu, et arstil oleks parem ülevaade uuest patsiendist (lisaandmete täitmine ei ole kohustuslik).
3. Süsteem genereerib patsiendile unikaalse koodi.
6. Süsteem salvestab visiidi ning uue patsiendi andmebaasi.

**Kasutusjuht:** Arsti autentimine

**Tegutsejad:** Arst

**Tüüpiline sündmuste järjestus:**

1. Arst soovib ennast autentida selleks, et süsteemi kasutada.
2. Süsteem avab vormi, kuhu saab sisestada e-maili aadressi ning parooli.
3. Kui sisestatud andmed klapivad andmebaasis olevatega, saab arst õiguse süsteemi kasutada.

**Laiendus** (Arst soovib süsteemi siseneda Mobiil-ID või ID-kaardiga) **:**

2. Vormi ülemises servas on võimalus Mobiil-ID või ID-kaardiga autentimiseks.
3. Autentimine toimub läbi Sertifikaadikeskuse.
   (* – arstide autentimine ei ole realiseeritava allsüsteemi osa ning realiseeritakse vaid osaliselt)

**Kasutusjuht:** Uue patsiendi lisamine

**Tegutsejad:** Arst

**Tüüpiline sündmuste järjestus:**

1. Arst soovib süsteemiväliselt visiidile registreerinud patsiendi andmed andmebaasi lisada.
2. Süsteem avab vormi, kuhu saab sisestada patsiendi isikuandmed ning lisada täiendavaid märkmeid patsiendi kohta.
3. Kui vorm on korrektselt täidetud, salvestatakse uus patsient andmebaasi.

**Kasutusjuht:** Patsiendi andmete vaatamine

**Tegutsejad:** Arst

**Tüüpiline sündmuste järjestus:**

1. Arst soovib vaadata kindla patsiendi andmeid.
2. Süsteem avab otsingu, kus arst saab patsienti otsida nime, koodi või telefoninumbri järgi.
3. Süsteem kuvab leitud patsientide nimekirja, kus on näha patsiendi nimi, kood, sugu, vanus ning telefoninumber (kui see on patsiendile lisatud).
4. Arst valib otsingutulemuste seast sobiva patsiendi.
5. Süsteem kuvab valitud patsiendi isikuandmed ning temaga seotud märkmed, failid ja visiidid.

**Laiendus** (Kui otsitavat patsienti ei leita)**:**

3. Arstile kuvatakse kiri „Vastet ei leitud“.

**Kasutusjuht:** Patsiendi andmete muutmine

**Tegutsejad:** Arst

**Tüüpiline sündmuste järjestus:**

1. Arst soovib patsiendi andmeid muuta.
2. Arst navigeerib patsiendi andmete lehele (vt Kasutusjuht: Patsiendi andmete vaatamine, lk 16) ning avab muutmise lehe.
3. Süsteem kuvab patsiendi olemasolevate andmetega täidetud vormi, kus on võimalik muuta patsiendi nime, sünniaega, sugu, e-posti aadressi, telefoninumbrit, pikkust, kaalu ning patsiendi kohta kirjutatud lisamärkmeid.
4. Kui vorm on korrektselt täidetud, salvestatakse muudetud info andmebaasi.

**Kasutusjuht:** Patsiendile faili lisamine

**Tegutsejad:** Arst

**Tüüpiline sündmuste järjestus:**

1. Arst soovib patsiendiga seotud faili lisada.
2. Arst navigeerib patsiendi info lehele (vt Kasutusjuht: Patsiendi andmete vaatamine, lk 16) ning avab faili lisamise akna.
3. Arst valib lisatava faili ning lohistab selle faili lisamise aknasse. Korraga on võimalik lisada ainult üks fail. Kõik failitüübid on lubatud, aga süsteem ei luba sisestada faile mis on suuremad kui 10MB.
4. Lisatud faili sisu salvestatakse andmebaasi baitide massiivina.

**Kasutusjuht:** Patsiendi faili kustutamine

**Tegutsejad:** Arst

**Tüüpiline sündmuste järjestus:**

1. Arst soovib patsiendiga seotud faili kustutada.
2. Arst navigeerib patsiendi info lehele (vt Kasutusjuht: Patsiendi andmete vaatamine, lk 16).
3. Arst vajutab soovitava faili juures kustutamise nuppu.
4. Süsteem küsib kinnitust kustutamiseks.
5. Fail kustutatakse andmebaasist.

**Kasutusjuht:** Patsiendile visiidi lisamine

**Tegutsejad:** Arst

**Tüüpiline sündmuste järjestus:**

1. Arst soovib patsiendile uut visiiti lisada.
2. Arst navigeerib patsiendi info lehele (vt Kasutusjuht: Patsiendi andmete vaatamine, lk 16) ning avab visiidi lisamise akna.
3. Arst valib kalendrist sobiva kuupäeva ning seejärel on võimalik valida antud kuupäeva vabade aegade seast sobiv aeg. Lisaks on arstil võimalus visiidi kohta lisamärkmeid kirjutada.
4. Süsteem salvestab uue visiidi andmebaasi.

**Kasutusjuht:** Patsiendi visiidi andmete muutmine

**Tegutsejad:** Arst

**Tüüpiline sündmuste järjestus:**

1. Arst soovib olemasoleva visiidi andmeid muuta.
2. Arst navigeerib patsiendi info lehele (vt Kasutusjuht: Patsiendi andmete vaatamine, lk 16) ning avab soovitud visiidi
3. Avanenud aknast valib arst visiidi muutmise.
4. Arstile kuvatakse olemasolevate märkmetega täidetud tekstikast, kus saab teksti muuta ning dünaamiline inimkeha joonis, mille peal on võimalik märkida patsiendi probleemsed piirkonnad (visiidi kuupäeva ja kellaaega muuta ei saa).
   (* – probleemsete piirkondade märkimine tuleb enne realiseerimist kliendiga põhjalikumalt läbi arutada)
5. Süsteem salvestab uuendatud andmed andmebaasi.

**Kasutusjuht:** Kindla kuupäeva ülevaate saamine

**Tegutsejad:** Arst

**Tüüpiline sündmuste järjestus:**

1. Arst soovib saada ülevaadet ühest kindlast kuupäevast.
2. Süsteem kuvab kalendri.
3. Arst valib sobiva kuupäeva ning avab sellele vajutades selle kuupäeva visiite ning vabade aegade seast eemaldatud ajaperioode sisaldava akna.

**Kasutusjuht:** Kindlal kuupäeval vabade aegade eemaldamine

**Tegutsejad:** Arst

**Tüüpiline sündmuste järjestus:**

1. Arst soovib kindlal kuupäeval vabasid aegu eemaldada, et nendele visiite ei saaks registreerida.
2. Arst avab soovitud kuupäeva ülevaate (vt Kasutusjuht: Kindla kuupäeva ülevaate saamine, lk 18) ning avab vabade aegade eemaldamise akna.
3. Arst sisestab avanenud aknas ajaperioodi alguse ja lõpu. Kui arst sisestab ainult alguse aja, määratakse lõpu ajaks päeva lõpp. Kui arst sisestab ainult lõpu aja, määratakse alguse ajaks päeva algus.
4. Süsteem ei lase valitud ajavahemikus visiite registreerida.

**Laiendus** (Arst soovib terve päeva eemaldada):

2. Vabade aegade eemaldamise akna avamise asemel vajutab Arst terve päeva eemaldamise nupule.

**Kasutusjuht:** Registreeritud visiidi või eemaldatud aja kustutamine

**Tegutsejad:** Arst

**Tüüpiline sündmuste järjestus:**

1. Arst soovib süsteemis olevat visiiti või eemaldatud ajaperioodi kustutada.
2. Arst avab soovitud kuupäeva ülevaate (vt Kasutusjuht: Kindla kuupäeva ülevaate saamine, lk 18).
3. Arst vajutab sobiva visiidi või kinnise aja juures kustutamise nuppu.
4. Süsteem küsib kinnitust kustutamiseks.
5. Valitud element kustutatakse andmebaasist.

**Kasutusjuht:** Andmebaasist visiitide info Excel kujul eksportimine

**Tegutsejad:** Arst

**Tüüpiline sündmuste järjestus:**

1. Arst soovib kõikidest enda juurde registreeritud visiitidest MS Exceli tabelit.
2. Süsteem genereerib andmebaasis olevate visiitide põhjal *.xlsx* laiendiga faili ning avab selle allalaadimise akna

**Kasutusjuht:** Arsti andmete muutmine

**Tegutsejad:** Arst

**Tüüpiline sündmuste järjestus:**

1. Arst soovib enda vastuvõtuaegu, telefoninumbrit, nime või parooli muuta.
2. Süsteem avab arsti andmete muutmise vormi.
3. Arst sisestab uuenenud andmed.
4. Süsteem salvestab uuenenud andmed andmebaasi.

   (* – arstide haldamine ei ole osa realiseeritavast allsüsteemist)

**Kasutusjuht:** Uue arsti registreerimine

**Tegutsejad:** Administraator

**Tüüpiline sündmuste järjestus:**

1. Süsteemi soovib kasutada uus arst, kellele tuleb kasutaja luua.
2. Süsteem avab uue arsti registreerimise vormi.
3. Administraator sisestab avanenud vormi vajalikud arsti andmed.
4. Süsteem genereerib uuele arstile parooli ning salvestatab arsti info andmebaasi.
5. Arsti e-maili aadressile saadetakse teade kasutaja loomisest.

   (* – arstide haldamine ei ole osa realiseeritavast allsüsteemist)

# 3 Veebirakenduse realisatsioon

Selles peatükis kirjeldatakse erinevaid veebirakenduste loomise tehnoloogiaid, võrreldakse lühidalt neist populaarsemaid ning põhjendatakse valikuid rakenduse loomisel kasutatavate tehnoloogiate osas. Seejärel selgitatakse lähemalt rakenduse loomist ning valitud tehnoloogiate kasutamist.

## 3.1 Tehnoloogiate valik ja võrdlus

### 3.1.1 Esikomponendi tehnoloogiad

Kõikide veebirakenduste esikomponentide aluseks on HTML (*hyper text markup language*), CSS (*cascading style sheets*) ja JavaScript. Kuigi on võimalik ehitada terveid kasutajaliideseid ilma mingeid raamistikke kasutamata, on targem neid kasutada. Raamistikud teevad arendamise mugavamaks, kiiremaks ja lihtsustavad arenduse käigus tekkivate keerukate probleemide lahendamist.

Esikomponendi raamistikud võib nende eesmärgi järgi jagada kaheks: stiiliraamistikud ja funktsionaalsed raamistikud. Stiiliraamistike eesmärk on lihtsustada dünaamiliste kasutajaliideste tegemist. Need võimaldavad kasutada erinevaid sisseehitatud CSS-i lehti vaadete kujundamiseks, kuid ei ole mõeldud lihtsustama serveriga suhtlust või vaadetes sisestatud andmete haldamist. Teisalt ka enamus funktsionaalseid raamistikke võimaldavad luua dünaamilisi kasutajaliideseid, kuid põhirõhk on pandud just veebirakenduste loomise funktsionaalsusele. Sellised raamistikud lihtsustavad näiteks HTTP (*hyper text transfer protocol*) sõnumite saatmist.

Vaieldamatult on tänapäeval kõige populaarsemaks stiiliraamistikuks Bootstrap[1] [1]. See ei ole ilmtingimata teistest parem, aga populaarsuse tõttu on sellel teistest rohkem erinevaid juhendeid ja lisasid, mis lihtsustavad selle õppimist. Bootstrapile pakub kõige

[1] https://getbootstrap.com/

rohkem konkurentsi Foundation[1] raamistik, mida kasutavad mitmed suured ettevõtted nagu Amazon, Mozilla, Ebay jpt [2]. Neile kes soovivad midagi vähem mahukat võib soovitada Pure[2] raamistikku. Pure on väiksem ja seetõttu ka vähem võimas kui eelnevalt mainitud raamistikud, kuid võimaldab siiski teha lihtsaid ja mugavaid kasutajaliideseid.

Erinevalt stiiliraamistikest, mis põhinevad suures osas CSS-l, on funktsionaalsed raamistikud ehitatud JavaScripti baasil. Kõige levinumateks näideteks on AngularJS[3] ja React[4] [3]. Erinevalt AngularJS-st on React mõeldud kasutamiseks erinevate teekide ja moodulitega [4]. See tähendab, et React iseseisvalt ei ole sama võimas kui AngularJS, aga Reacti on lihtsam ise kohandada vastavalt projekti vajadustele. Teine suur erinevus seisneb selles, et AngularJS kasutab vaadetega suhtlemiseks kahesuunalist andmete sidumist [5], React aga ühesuunalist [6]. Seega kui AngularJS-ga loodud rakenduse vormi väljas olevat teksti muuta, muutub ka sellega seotud muutuja väärtus ning vastupidi. Reacti puhul muutub vormi välja väärtus kui muutub sellega seotud muutuja väärtus, aga vastupidine ei ole tõsi. See võimaldab AngularJS-ga teha kiiremini reageerivaid ja lihtsamini kontrollitavaid vorme, aga suurendab ka erinevate potentsiaalsete kõrvalnähtude teket (nt SQL süstimine).

Rakenduse loomisel lähtuti nõudest, et rakendus peab piiratud funktsionaalsusega töötama ka internetiühenduseta. Sellest tulenevalt planeeriti esialgu vältida suuri raamistikke ning kasutada esikomponendi loomisel tavalist JavaScripti. Seda sellepärast, et erinevatel veebilehitsejatel on erineval hulgal mälu internetiühenduseta töötamiseks vajaliku rakenduse lähtekoodi salvestamiseks. Terve funktsionaalse raamistiku asemel võeti esialgu kasutusele vaid AngularJSi lehtede vahelise navigeerimise teek[5], aga rakenduse kasvades otsustati koodi loetavuse, kasutajaliidese dünaamilisuse ning rakenduse kiiruse huvides kasutusele võtta terve AngularJS raamistik. Stiiliraamistikke rakenduse loomisel ei kasutatud.

---

[1] https://foundation.zurb.com/

[2] https://purecss.io/

[3] https://angularjs.org/

[4] https://reactjs.org/

[5] https://docs.angularjs.org/api/ngRoute/service/$route

### 3.1.2 Tagakomponendi tehnoloogiad

Erinevalt esikomponendi raamistikest, millel on kõigil samad aluskomponendid, on tagakomponendi raamistikke võimalik ehitada praktiliselt iga olemasoleva programmeerimiskeele baasil. Levinumad neist on Python[1] ja Java[2] [7]. Lisaks erinevatele programmeerimiskeeltele on olemas ka PHP[3] (*hypertext preprocessor*) ja Node.js[4]. PHP on veebirakenduste loomiseks disainitud serveripoolne skriptimiskeel, mille jaoks on loodud palju erinevaid raamistikke [8], kuid mida on võimalik mugavalt kasutada ka ilma nendeta [9]. Node.js on JavaScripti põhjal loodud ja veebirakenduste serveri poole jaoks mõeldud käitusmootor.

Java põhjal loodud veebirakendused on tuntud oma kiiruse ja stabiilsuse poolest, kuid Java rakenduste arendamine on võrreldes teistega aeglasem, sest samade probleemide lahendamiseks on vaja rohkem ridu koodi [10]. Lisaks vajavad Java rakendused normaalseks tööks rohkem ressursse kui näiteks PHP-d kasutavad rakendused [11]. Pythoni tugevaks küljeks on kergesti arusaadav ja kirjutatav kood. Lisaks sellele on Pythoni rakendusi võrreldes Javaga palju lihtsam serverisse tööle panna ja neid seal hallata. Pythoni nõrk külg on selle kiirus. Kuna tegu on interpreteeritava mitte kompileeritava keelega, on see aeglasem kui Java rakendused [12]. Sama kehtib ka PHP kohta [11]. PHP on kõige levinum veebilehtede serveripoolne keel [13]. PHP eeliseks on selle lihtsus ja paindlikkus, aga samad asjad on ka PHP nõrkuseks. Kuna PHP on nii paindlik, on liiga kerge kirjutada halba koodi, mis teeb rakenduse skaleerimise keeruliseks ning seetõttu on PHP enamasti kasutusel ainult väiksemate rakenduste puhul. Sarnaselt PHP-le on Node.js mõeldud spetsiaalselt veebirakenduste loomiseks. Node.js-i kõige suuremaks eeliseks teiste ees on JavaScripti kasutamise võimalus. Sellest tulenevalt on funktsionaalsuse liigutamine esikomponendist tagakomponenti või vastupidi kiire ja mugav. Node.js-i nõrkuseks on see, et terve rakendus töötab alati maksimaalselt läbi ühe lõime [14], mille pärast võib terve rakenduse serveri pool hanguda, kui üks kasutajatest teeb mõne suure päringu.

---

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

[2] https://www.java.com/en/

[3] http://www.php.net/

[4] https://nodejs.org/en/

Veebirakenduse loomisel kasutati Javat, sest sellega oldi antud hetkel kõige rohkem tuttav. Uue keele õppimine oleks olnud liiga ajakulukas ning kuna Java sobis kõikide projekti nõuetega, ei nähtud mingit põhjust otsustada mõne teise keele kasuks. Raamistikuna võeti kasutusele Spring Boot, sest see on vaieldamatult kõige levinum Java veebirakenduste arendamise raamistik [15].

### 3.1.3 Andmebaasi tehnoloogiad

Andmebaasisüsteemid võib nende tööpõhimõtete järgi jagada kaheks: relatsioonilised ja mitte-relatsioonilised. Relatsioonilistes ehk SQL (*structured query language*) andmebaasisüsteemides loodud andmebaasid jaotavad andmed kindlaks määratud väljadega tabelitesse. Iga tabeli välja kohta peab täpsustama, mis tüüpi andmeid selles hoitakse ning igale väljale saab lisada eraldi kitsendusi. Tabelitesse ei ole võimalik lisada andmeid, mille jaoks väli puudub ja andmebaasiga saab suhelda kasutades SQL keelt. Mitte-relatsioonilistes ehk noSQL andmebaasisüsteemides loodud andmebaasid hoiavad andmeid dokumentides, võti-väärtus paarides või muudes andmebaasisüsteemile omastes kogumites. Erinevalt relatsiooniliste andmebaaside tabelitest ei ole noSQL kogumid kindla struktuuriga, seega nendesse saab lisada selliseid andmeid nagu parasjagu vaja. NoSql andmebaasiga suhtlemiseks on igal andmebaasisüsteemil oma käsud [16].

See kumba andmebaasisüsteemi kasutada tuleks, sõltub projekti vajadustest ning öelda, et üks neist on parem kui teine, ei oleks õige. Mõlemad täidavad sama ülesannet, andmete haldamine, aga lähenevad sellele erinevalt. Kui on täpselt teada, milliseid andmeid andmebaasis hoidma hakatakse, on oluline andmete turvalisus ja valideerimine ning erinevad andmebaasiobjektid peavad olema omavahel seotud, oleks mõistlikum luua relatsiooniline andmebaas. Kui aga ei ole kindlalt teada, milliseid andmeid andmebaasis tuleb hoida või samade andmebaasiobjektide erinevatel eksemplaridel on erinevad väljad (nt erinevate tööstusmasinate puhul tuleb ühel mõõta pöördeid minutis, teisel aga temperatuuri), oleks loogilisem valik mitte-relatsiooniline andmebaas. NoSQL andmebaasid on üldiselt kiiremad kui sarnaselt disainitud SQL andmebaasid, sest nendes ei ole vaja keerulisi päringuid ja tabelite ühendamisi, et andmebaasist andmeid pärida [17]. See kiirus tuleb andmebaasi suuruse kasvamise arvelt. Kuna mitte-relatsioonilised andmebaasid ei ole mõeldud erinevate andmete vahel ühenduste loomiseks, tuleb paljusid andmeid dubleerida [17]. Teisalt, on isegi halvasti disainitud noSQL andmebaase lihtsam skaleerida kui SQL andmebaase, sest ei tule muretseda erinevate kitsenduste ja

välisvõtmeveergude pärast (nt relatsioonilise andmebaasi tabelisse ei saa lisada uut kohustuslikku välja kui tabelis on juba andmeid). Samas, SQL andmebaasides on lihtsam tagada andmeterviklus, sest andmebaasi tabelitele on võimalik määrata kohustuslikke väljasid ning kontrollida väljadesse sisestatavaid andmeid.

Rakenduse loomisel võeti kasutusele relatsiooniline andmebaasisüsteem, kuna oli täpselt teada, milliseid andmeid andmebaasis hoidma hakatakse ning relatsioonilised andmebaasisüsteemid võimaldavad mugavamalt luua seoseid erinevate andmebaasiobjektide vahel. Tabeli väljadele kitsendusi lisades on ka lihtsam kontrollida sisestavate andmete korrektsust, mis oli antud rakenduses oluline. Kuna Spring Boot raamistik lubab andmebaasiga suhelda kasutades Java koodi, ei olnud väga oluline, millist andmebaasisüsteemi rakendus kasutab. Ainukeseks nõudeks oli, et valitav lahendus peab olema tasuta. Algselt plaaniti kasutada Oracle[1] andmebaasisüsteemi, kuid lähemalt tutvudes selgus, et tasuta versioon ei lubanud andmebaasi piiramatut kasutust [18]. Lisaks kaaluti ka MySQL-i[2] kasutamist, aga kuna sellega oldi juba varasemalt kokku puutunud ning selle omapärasest käitumisest mõningate päringutega teadlik [19], otsustati lõpuks andmebaas realiseerida kasutades PostgreSQL[3] andmebaasisüsteemi.

## 3.2 Olemasolevad infosüsteemid

Patsientide ja visiitide haldamine ei ole uuema aja probleem ning sellest tulenevalt on antud teemal tehtud juba lugematu arv erinevaid rakendusi ja infosüsteeme nii eesti kui ka muudes keeltes. Eesti e-tervise sihtasutuse kodulehelt võib leida nimekirja kõikidest eestikeelsetest meditsiiniasutustele mõeldud tarkvaralahendustest [20], aga need kõik jagavad ühist probleemi. Sealsed rakendused on mõeldud üldiseks kasutamiseks eriarstide poolt, sõltumata valdkonnast. Kui tuua näide eKliiniku[4] rakenduse kohta, mis tundus antud lehel olevatest kõige lubavam, oli küll hästi lahendatud patsientide, visiitide ning isegi arstide haldamine, kuid kasutajaliides oli kliendi jaoks segaselt disainitud ning puudulik oli soovitud erialane lähenemine. Lisaks vajavad enamus lehel leiduvatest

[1] https://www.oracle.com/database/index.html

[2] https://www.mysql.com/

[3] https://www.postgresql.org/

[4] http://connected.ee/ekliinik-site/

rakendustest internetiühendust, kuid veebirakendusega oli tegu vaid ühe puhul. Eelpool mainitut arvestades, otsustas klient, et ei soovi maksta igakuist kasutustasu rakenduse eest, millega ta rahul ei ole ning soovib ikkagi füsioteraapia vaatepunktist loodud rakendust.

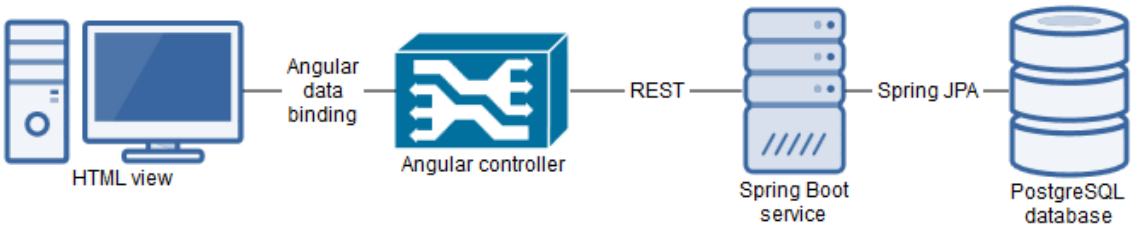
## 3.3 Rakenduse loomine

### 3.3.1 Arendusmetoodika

Rakenduse arendusel kasutati suures osas nn kauboi (*cowboy coding*) stiili [21]. Kauboi stiilil puudub igasugune reeglistik ning arendaja võib arendada omale sobival ajal ja nii palju kui parasjagu soovib. Kuna rakendust arendati üksinda, mõeldi, et valitud stiil sobib paremini kui suurte meeskondade jaoks loodud arendusmetoodikad nagu scrum[1] või kanban[2].

Rakenduse versioonihalduseks kasutati Giti ning repositoorium loodi GitLabi[3] serverisse (Lisa 1). GitLab on tasuta ja avatud lähtekoodiga Giti repositooriumite haldamise tarkvara. Arendatava rakenduse väiksemateks osadeks jagamiseks ja nende haldamiseks kasutati GitLabis olevat probleemide (*issue*) haldamise võimalust.

### 3.3.2 Rakenduse arhitektuur




Joonis 1. Rakenduse loogiline arhitektuur

Rakenduse aluseks valitud loogiline arhitektuur (Joonis 1) võimaldab esi- ja tagakomponendi iseseisvat arendust. Tänu sellele on uute tehnoloogiate ilmudes lihtsustatud rakenduse erinevate komponentide uuendamine või vahetamine teistest komponentidest sõltumatult. Vältimaks otseühendust esikomponendi ja andmebaasi

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

[2] https://www.atlassian.com/agile/kanban

[3] https://about.gitlab.com/

vahel toimub andmete töötlemine ja haldamine tagakomponendis Java ning Spring Boot raamistiku abil realiseeritud REST (*Representational State Transfer*) [22] teenustes.

Esikomponendist päringute tegemiseks on kasutusel AngularJS-i *$http* teenus, millega on võimalik mõne reaga asünkroonseid REST päringuid koostada (Joonis 2). Päringutele vastamiseks on võimalik Spring Booti poolt kasutatava Jacksoni[1] abil teisendada Java objekte JSON (*JavaScript Object Notation*) kujule, et neid oleks lihtsam JavaScriptiga hallata.

```
$http({
  method: 'GET',
  url: 'http://85.222.235.135:8080/api/allDoctors/'
  }).then(function successCallback(response) {
    $scope.doctorList = response.data
  }, function errorCallback(response) {
    console.log(response);
});
```

Joonis 2. Näide AngularJS *$http* teenusega koostatud päringust.

Relatsiooniliste andmete Java objektideks kaardistamiseks on tagakomponendis kasutusel Spring JPA[2] (*Java Persistance API*). Spring JPA võimaldab relatsioonilise andmebaasi tabeleid defineerida Java objektidena, lihtsustades sellega andmebaasi päringute loomist.

### 3.3.3 Projekti põhja genereerimine

Kuna Spring Boot raamistikuga veebirakendusi luues tuleb kirjutada mitmeid konfiguratsioonifaile ning lisada suurel hulgal sõltuvusi, kasutati projekti põhja genereerimiseks JHipsterit[3]. JHipster on käsureal töötav, avatud lähtekoodiga, Javat (Spring Booti) ja AngularJS-i kasutavate veebirakenduste generaator.

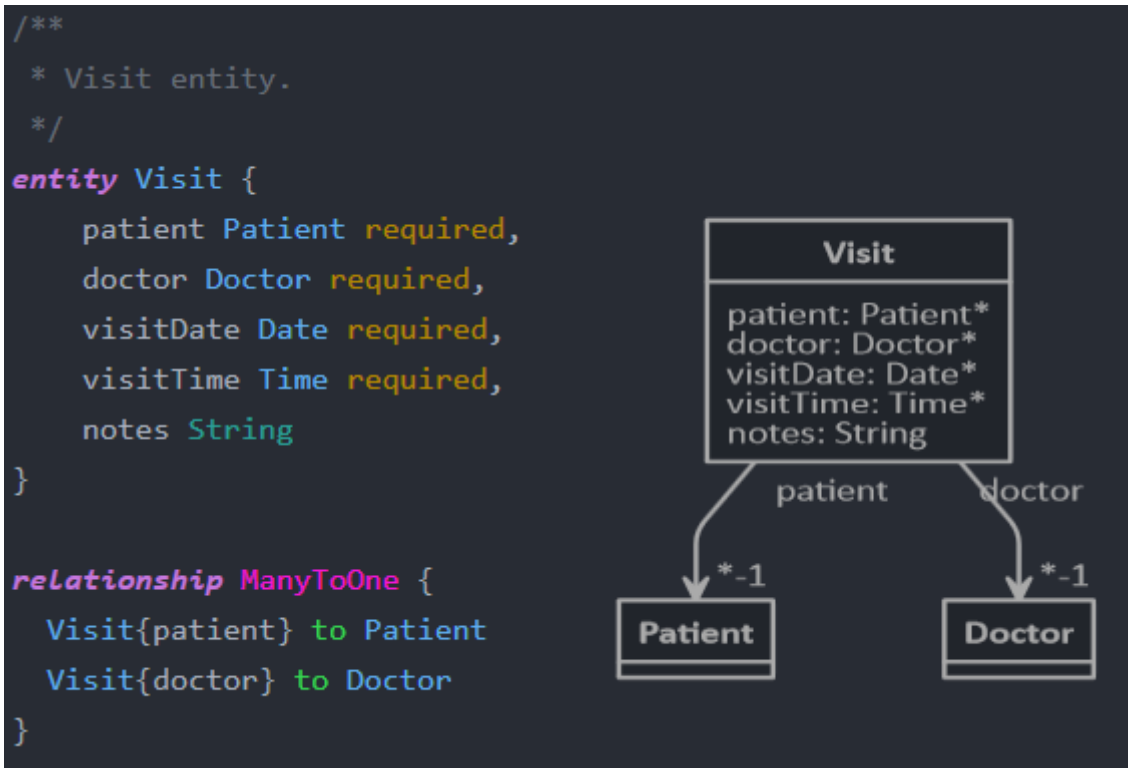
JHipsteriga rakendust genereerides on võimalus konfigureerida muude sätete hulgas näiteks andmebaasiühendus, kasutajate autentimine ning testimiseks kasutatavad raamistikud [23]. Lisaks on rakenduses kasutatavate Java objektide genereerimiseks

[1] https://github.com/FasterXML/jackson

[2] https://docs.spring.io/spring-data/jpa/docs/current/reference/html/

[3] http://www.jhipster.tech/

loodud JDL-Studio[1] (*JHipster Domain Language Studio*), mille abil on võimalik määrata objektide parameetrid koos kitsendustega ning objektide omavahelised seosed (Joonis 3). Genereeritud mudelite põhjal saab luua Java objektid, nendega seotud teenused ja kontrollerid ning JPA repositooriumid.




Joonis 3. JDL-Studios loodud visiidi olem ning selle põhjal automaatselt genereeritud mudel.

### 3.3.4 Andmebaasi tabelite loomine ja testandmed

Andmebaasi tabelite loomise lihtsustamiseks on JHipsteri poolt genereeritud rakendustel alati juures Liquibase'i sõltuvus. Liquibase[2] on avatud lähtekoodiga, andmebaasisüsteemist sõltumatu teek, mis võimaldab hallata ja luua andmebaasi skeeme. Kõik skeemi muudatused salvestatakse skeemifailidesse ning on identifitseeritavad muudatuse id ja autori järgi. Toetatud skeemifaili formaadid on XML (JHipster kasutab seda), YAML, JSON ja SQL, aga soovi korral on võimalik kasutada laiendusi, mis toetavad ka teisi formaate [24].

[1] https://start.jhipster.tech/jdl-studio/

[2] http://www.liquibase.org/

Kuna Liquibase on andmebaasist sõltumatu ja kõik andmebaasi skeemi muudatused on Liquibase'i skeemifailides, on rakenduse poolt kasutatava andmebaasi migreerimine tehtud triviaalseks. Liquibase'i skeemifailid avatakse iga kord kui rakendus tööle panna ning kui nendes esineb muudatusi, mida andmebaasis ei kajastata, uuendatakse andmebaasi. Seega, kui rakendus ühendada uue andmebaasiga, luuakse sinna rakenduse esmasel käivitamisel kõikides skeemifailides olevad andmebaasiobjektid (nt tabelid, kitsendused, indeksid).

Liquibase'i skeemifailides olevad muudatused on antud rakenduses objektorienteeritud programmeerimise põhimõtetest lähtuvalt tehtud võimalikult väiksed. Iga tabeli, indeksi ja suurema kitsenduse jaoks on loodud eraldi muudatus (Joonis 4) ning kõik ühe tabeliga seonduv on koondatud ühte skeemifaili. Tänu sellele on muudatused lihtsamini hallatavad ning vigaste muudatuste puhul vead kergemini leitavad.

```
<changeSet id="create_table_doctor" author="tohver">
    <createTable tableName="doctor">
        <column name="id" type="bigint" autoIncrement="true">
            <constraints primaryKey="true" nullable="false"/>
        </column>
        <column name="name" type="varchar(255)">
            <constraints nullable="false" />
        </column>
        <column name="phone_number" type="varchar(255)">
            <constraints nullable="false" unique="true"/>
        </column>
        <column name="email" type="varchar(255)">
            <constraints nullable="false" />
        </column>
    </createTable>
</changeSet>
```

Joonis 4. Liquibase'i muudatus arsti tabeli loomiseks.

Lisaks tabelitele, kitsendustele ja indeksitele on võimalik Liquibase'i abil andmebaasi ka andmeid lisada. Andmeid sisaldava skeemifaili võib kirjutada ise (testandmed) või genereerida andmebaasis olevatest andmetest (andmebaasi migreerimine) [25]. Antud rakenduses on Liquibase'i kasutatud testandmete lisamiseks. Kõikide testandmete muudatused on koondatud ühte faili, kuid iga tabeli andmed on loodud eraldi muudatusena (Joonis 5).

```
<changeSet id="data_for_doctor" author="Tohver">
    <insert tableName="doctor">
        <column name="name" value="Jooksev Meeter"/>
        <column name="email" value="jooksevmeeter54@hotmail.com"/>
        <column name="phone_number" value="56321456"/>
    </insert>
    <insert tableName="doctor">
        <column name="name" value="Maizi Pulgad"/>
        <column name="email" value="maiz26@gmail.com"/>
        <column name="phone_number" value="58103247"/>
    </insert>
</changeSet>
```

Joonis 5. Liquibase'i muudatus arsti tabelisse andmete lisamiseks.

### 3.3.5 Patsiendi visiidile registreerimine

Patsiendipoolne visiidile registreerimine koosneb kahest osast: patsiendi tuvastamine (või esmakordse patsiendi puhul uue patsiendi loomine) ja sobiva aja valimine.

Patsiendi tuvastamiseks on vaja patsiendi ees- ja perekonnanime ning e-maili aadressi või telefoninumbrit. Nende põhjal saadetakse REST päring tagakomponendis olevasse kontrollerisse, mis vaatab, kas andmebaasis esineb selliste andmetega patsient. Kui andmete sisestamisel on tehtud viga või sellist patsienti süsteemis ei eksisteeri, ei lubata visiiti registreerida.

Kui patsient soovib ennast esmakordselt visiidile registreerida, genereeritakse talle tagakomponendis unikaalne kood (nt 171104-001). Kood koosneb patsiendi registreerimise kuupäevast (kujul YYMMDD) ning järjekorranumbrist, mis iga päev uuesti nullist alustab (Joonis 6).

```
if (Integer.valueOf(dateFormat.format(date)) > currentDate) {
       patientCounter = 0;
       currentDate = Integer.valueOf(dateFormat.format(date));
}
```

Joonis 6. Järjekorranumbri nullimine.

Visiidi aja valimiseks kuvatakse patsiendi poolt valitud arsti vabade vastuvõtuaegade põhjal genereeritud kalender (Lisa 2). Kalendri genereerimist teostav AngularJS-i teenus pärib tagakomponendilt valitud kuu kõiki vabasid vastuvõtuaegu. Päringu vastus koosneb võti-väärtus paaride loendist, kus iga elemendi võtmeks on päeva number ning väärtuseks massiiv vabade aegadega (Joonis 7). Saadud vastuse põhjal valitakse iga kalendripäeva kuvamiseks sobiv värv: valge, kui vabasid aegu on piisavalt, kollane, kui antud päeval on

vähem kui pooled ajad vabad, punane kui mitte ükski aeg ei ole vaba. Hetkel on iga arsti vastuvõtuajad vahemikus 8:00-18:00.

```
{
  "d1" : [ "08:00", "08:30", "09:30", "15:00", "15:30" ],
  "d4" : [ "11:00", "11:30", "12:00", "12:30", "13:00", "13:30" ]
}
```

Joonis 7. Lihtsustatud näide ühe kuu vabade aegade päringu vastusest.

### 3.3.6 Arsti autentimine

Lõputöö raames realiseeritud kasutaja autentimise funktsionaalsus toimub e-maili aadressi ja parooli baasil. Paroolide krüpteerimiseks kasutab tagakomponent BCrypt algoritmi [26]. Kuna lõppserveris on võimalik kasutatavaid ressursse juurde osta, sobib valitud algoritm antud rakendusele, sest sõltuvalt serveri võimsusest on võimalik paroolide räsimise keerukust ja ressursi kulu tõsta või alandada. Andmebaasis hoitakse paroole krüpteeritud kujul.

Tagakomponendis genereeritakse korrektse autentimise tulemusena kasutajale JWT[1] (*JSON Web Token*) pääsuluba. Kõik tagakomponendi teenused, mida ei tohi olla võimalik kasutada ilma sisse logimata, nõuavad, et esikomponendilt saadetud päringu päis sisaldaks kehtivat pääsuluba (Joonis 8).

```
$http({
      method: 'GET',
      url: 'http://85.222.235.135:8080/api/loggedin/patient/' +
$scope.patient.code + '/visits',
      headers: {
        'Authorization': 'Bearer ' + sessionStorage.token
      }
    }).then(function successCallback(response) {
        $scope.visits = response.data
      }, function errorCallback(response) {
        console.log(response);
      });
```

Joonis 8. JWT pääsuluba kasutav päring.

Esikomponendis autentimise tulemusena vastuseks saadud JWT pääsuluba hoitakse veebilehitseja sessioonimälus (*session storage*). Erinevalt küpsistest (*cookie*) ja

[1] https://jwt.io/

lokaalmälust (*local storage*) kustutatakse sessioonimälu koos avatud akna sulgemisega [27]. Tänu sellele, kui sulgub rakendust kasutav aken, logitakse kasutaja automaatselt välja. Samas erinevalt muutujasse salvestamisest, säilib sessioonimälus olev ka lehte värskendades (*refresh*). Kui arst on töö lõpetanud ja otsustab ise välja logida, kustutatakse sessioonimälus olev JWT pääsuluba manuaalselt (Joonis 9).

```
function logOut() {
  sessionStorage.removeItem('token')
  location.href = '#/'
}
```

Joonis 9. JWT pääsuloa kustutamine sessioonimälust.

### 3.3.7 Patsientide andmete vaatamine ja muutmine

Kindla patsiendi andmete nägemiseks on rakenduses implementeeritud patsientide otsimise funktsionaalsus. Patsiente saab otsida nende nime, koodi või telefoninumbri järgi ning otsingu põhjal saadetakse tagakomponendile päring automaatselt 0.5 sekundit pärast viimast otsinguvälja muutust. Otsingu tulemustest genereeritakse HTML elemendid, millele vajutades suunatakse arst edasi patsiendi vaatesse (Lisa 3).

Kuna rakenduses navigeerimiseks kasutatav AngularJS-i teek *$route* ei võimalda ühelt vaatelt teisele liikudes andmeid edastada (antud juhul oli vaja patsiendi koodi), on arsti patsiendi vaatesse ümbersuunamine lahendatud kasutades CSS-i atribuuti *display* (Joonis 10). Vaatele, mida kasutaja hetkel näeb, on määratud atribuudi *display* väärtus *block* ning vaadetele, mida kasutaja hetkel ei näe, on määratud atribuudi *display* väärtuseks *none*. Sellest tulenevalt, erinevalt ülejäänud rakendusest, kui patsiendi vaates vajutada veebilehitseja tagasi nuppu ei suunata arst mitte patsiendi otsingu lehele vaid lehele, kus ta oli enne otsingu kasutamist.

```
$scope.editPatient = function() {
  $('.patientViewPanel').css("display", "none")
  $('.editPatientForm').css("display", "block")
}
```

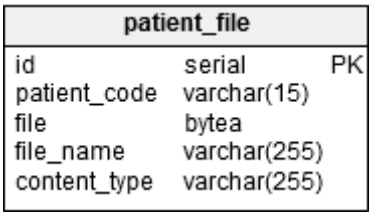
Joonis 10. Ühelt vaatelt teisele liikumine kasutades CSS-i.

Kui patsiendi vaatest liikuda edasi patsiendi andmete muutmise vaatesse, luuakse AngularJS-i kontrolleris muutmiseks mõeldud koopia muudetavast patsiendi objektist. Muutmise lõppedes salvestatakse andmebaasi esialgse objekti muudetud koopia ning vaates olevad andmed asendatakse uute, muudetud andmetega.

### 3.3.8 Patsientide failide süsteem

PostgreSQL andmebaasisüsteem võimaldab faile hoida kahel kujul: BLOB (*binary large object*) tüübina või bytea ehk pika tekstina [28]. Antud rakenduses on kasutusel bytea tüüp, sest seda on lihtsam Spring JPA-d kasutava rakendusega liidestada ning patsientidele lisatavad failid ei ole piisavalt suured, et oleks vaja kasutada BLOB tüüpi.

Faili üleslaadimiseks on patsiendi vaates võimalik avada failide lisamise aken. Avanenud aknasse on võimalik lohistada soovitud fail, mis seejärel automaatselt tagakomponendile saadetakse. Peale faili saatmist uuendatakse patsiendi vaates nähtavat failide nimekirja. Serveris olevas teenuses teisendatakse fail ümber baitide massiiviks ning salvestatakse andmebaasi koos faili nime ja tüübiga (Joonis 11).




Joonis 11. Patsientide faile sisaldava tabeli mudel.

Faili allalaadimiseks tuleb veebilehitseja aken ümber suunata faili tagastava Spring Booti kontrolleri aadressile. Kuna veebilehitseja suunamisel uuele lehele ei ole võimalik kaasa anda päringu päist nii nagu seda on REST päringute puhul [29], ei ole võimalik kontrollida, kas kasutaja on sisse logitud. Seetõttu on vajalik failidele turvalise ligipääsu tagamiseks luua abistav failikoodide süsteem. Selle süsteemi eesmärk on veebilehitseja ümbersuunamisel kontrollida JWT pääsuloa olemasolu.

Koodide süsteem genereerib ühekordselt kasutatava faili koodi, mille abil on võimalik fail kasutaja arvutisse salvestada. Kui esikomponendilt tuleb faili allalaadimise päring, mille päis sisaldab kehtivat JWT pääsuluba, genereeritakse serveris soovitavale failile unikaalne kood (Joonis 13). Genereeritud kood lisatakse serveris hoitavasse failikoodide nimekirja ning saadetakse vastuseks esikomponendilt tulnud päringule. Kui esikomponent on soovitud koodi kätte saanud, suunatakse veebilehitseja ümber saadud koodi kasutavale aadressile, mille peale algab faili allalaadimine. Kohe peale kasutamist kustutatakse serveris olevast failikoodide nimekirjast kasutatud kood (Joonis 13Joonis 13).

```
public static String generateFileHash(Long fileID) {
	String hash = UUID.randomUUID().toString();
	fileHashes.put(hash, fileID);
	return hash;
}
```

Joonis 12. Faili koodi genereerimine.

```
public static Long useFileHash(String hash) {
	return fileHashes.remove(hash);
}
```

Joonis 13. Faili koodi kustutamine.

Loodud süsteem tagab, et faile ei saaks alla laadida ilma faili allalaadimiseks vajaliku koodita ning failile ei saa koodi genereerida, kui kasutaja ei ole süsteemi poolt autentitud.

### 3.3.9 Kalendri eksport

Java abil Microsoft Office-i dokumentide loomiseks ja muutmiseks on võimalik kasutada Apache POI[1] teeki. See sisaldab meetodeid Exceli faili loomiseks, sinna ridade ja lahtrite lisamiseks ning nende kujundamiseks.

Kui arst soovib kõiki endaga seotud visiite andmebaasist Exceli kujul eksportida, käivitatakse andmebaasipäring, mis tagastab kõigi arstiga seotud visiitide nimekirja. Selle nimekirja põhjal genereeritakse, Apache POI teeki kasutades, Exceli fail (Joonis 14) ning saadetakse see esikomponendile. Esikomponendis suunatakse arst edasi faili allalaadimise lehele (samamoodi nagu patsiendi failide allalaadimise puhul).

```
for (Visit visit : visits) {
	row = sheet.createRow(rowNum++);
	row.createCell(0).setCellValue(visit.getPatient().getCode());
	row.createCell(1).setCellValue(visit.getVisitDate().toString());
	row.createCell(2).setCellValue(visit.getVisitTime().toString());
	row.createCell(3).setCellValue(visit.getNotes());
}
```

Joonis 14. Exceli faili ridade genereerimine ning täitmine.

### 3.3.10 Rakenduse käivitamine serveris

Töö käigus selgus, et kliendi kasutuses oleval virtuaalserveril puudub juurkasutajale ligipääs ning sellest tulenevalt oli võimalik serverit konfigureerida ainult teenusepakkuja

[1] https://poi.apache.org/index.html

poolt loodud piiratud funktsionaalsusega veebipõhises kasutajaliideses. Kuna olemasolev virtuaalserver ei võimaldanud käitada Java rakendusi ning andmebaasisüsteemina kasutusele võtta PostgreSQL-i, ei olnud võimalik selles tööle seada arendatavat rakendust. Lahendusena oli klient nõus tellima uue, rohkemate võimalustega, virtuaalserveri. Uuel serveril oli SSH (*Secure shell*) vahendusel olemas ligipääs juurkasutajale.

Esialgselt oli plaanis rakenduse pidevintegratsiooni (*continuous integration*) jaoks kasutusele võtta Jenkinsi[1] konveier (*pipeline*) ning rakenduse ehitamiseks ja käivitamiseks kasutada Dockeri[2] kujutist (*image*). Kuna virtuaalserveril puudus Dockeri tööks vajalik ligipääs füüsilise serveri operatsioonisüsteemi kernelile, ei olnud võimalik Dockerit kliendi serveris tööle seada. Sellest tulenevalt jäi esialgu ka Jenkinsi kasutuselevõtt tahaplaanile.

Serveris töötava rakenduse uuendamiseks loodi käivitatav kestaskripti (*shell script*) fail (Joonis 15). Käivitatav skript lõpetab alustuseks ära eelnevalt tööle pandud versiooni rakendusest ning seejärel laetakse alla kõige uuem versioon. Uue versiooni allalaadimisel parooli ei küsita, kuna serverisse konfigureeriti SSH võti Giti serveri ühenduseks. Allalaetud kaustast võetakse esikomponendi kõik elemendid ning kopeeritakse virtuaalserverisse loodud esikomponendi lähtekoodi jaoks mõeldud kausta. Tagakomponent ehitatakse Maveni käsuga ning pannakse tööle Java käsuga selliselt, et avatud terminaliakna sulgemisel, rakendus jätkaks tööd.

```
#!/bin/bash
pkill java
cd home/fysioweb
git pull
rsync -r FysioWeb/src/main/webapp/ /var/www/html
mvn -f FysioWeb/ package
setsid java -jar FysioWeb/target/fysio-web-0.0.1-SNAPSHOT.war
```

Joonis 15. Rakenduse uuendamise skript.

[1] https://jenkins-ci.org/

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

# 4 Edasine töö rakendusega

Lõputöö raames sai infosüsteem vaid osaliselt realiseeritud ning seetõttu jätkub töö rakendusega ka peale lõputöö esitamist. Selles peatükis on kirjeldatud edasist tööd projektiga.

## 4.1 Turvalisus

Kuna rakendust kasutatakse privaatsete isikuandmete haldamiseks, on oluline, et rakenduse turvalisus oleks piisav, et tagada isikuandmete kaitse seaduse[1] korrapärane täitmine. Selle tagamiseks on plaanis rakendusele lisada võimalus arstil ennast ID-kaardi või mobiil-ID vahendusel autentida ning rakenduses toimuvates päringutes HTTP protokolli asemel kasutusele võtta HTTPS protokoll.

ID-kaardiga kasutaja tuvastamise realiseerimiseks rakenduses tuleb kõigepealt ette valmistada vajalikud sertifikaadifailid. Kui rakendus reaalsesse kasutusse panna, tuleb Sertifitseerimiskeskuselt[2] tellida kinnitus oma sertifikaadifailile (esialgseks testimiseks võib selle sammu vahele jätta). Sertifitseerimiskeskuselt kinnituse tellimine on tasuline ning kinnituse kasutamiseks on vaja tasuda igakuist makset. Seejärel tuleb veebiserverit seadistada selliselt, et see kasutaks üle võrgu suhtlemiseks tavalisest HTTP protokollist turvalisemat HTTPS protokolli (seda saab teha muutes Spring Booti konfiguratsiooni). Ning lõpuks, tuleb ID-kaardi kasutamise funktsionaalsus lisada veebirakendusele [30].

Mobiil-ID vahendusel kasutaja tuvastamiseks tuleb, sarnaselt ID-kaardiga kasutaja tuvastamisele, kasutada Sertifitseerimiskeskust. Erinevalt ID-kaardiga autentimise süsteemist toimub mobiil-ID kasutamine läbi sertifitseerimiskeskuse veebiteenuse DigiDocService [31]. Ligipääs veebiteenuse APIle toimub veebiserveri IP-aadressi

[1] https://www.riigiteataja.ee/akt/748829

[2] https://www.sk.ee/

põhiselt. Sarnaselt ID-kaardi jaoks vajalikule sertifikaadifaili kinnitusele, on DigiDocService veebiteenuse API kasutamiseks vaja tasulist lepingut.

## 4.2 Pidevintegratsioon

Kuigi kliendi serveris ei ole võimalik kasutada Dockerit rakenduse ehitamiseks, on siiski mõttekas rakenduse uute versioonide automaatseks integreerimiseks lisada Jenkinsi (või mõne muu pidevintegratsiooni pakkuva teenuse) konveier. Konveieri abil on võimalik automatiseerida rakenduse ehitamist, testimist ning käivitamist (Joonis 16).

```
node {
    stage('Build') {
        echo 'Building....'
    }
    stage('Test') {
        echo 'Building....'
    }
    stage('Deploy') {
        echo 'Deploying....'
    }
}
```

Joonis 16. Näidis Jenkinsi konveieri konfiguratsioonist [32].

Rakendust arendades ei pruugi alati olla vajalik iga uuenenud faili pärast konveierit käivitada. Selle lahendamiseks on võimalik luua uus Giti haru ning konveieri konfiguratsioonis ära täpsustada, et konveier tuleb käivitada ainult uues harus faile muutes. See välistab võimaluse, et iga kord kui arendaja uue koodi Giti üles laeb, lõpetatakse serveris rakenduse töö ning pannakse uus, antud hetkel poolik, versioon serveris käima.

Lisaks Giti harudele on Spring Bootiga võimalik erinevate profiilide loomine [33]. See võimaldab serveri jaoks mõeldud profiilil kasutada erinevat konfiguratsiooni kui arenduskeskonnas kasutataval profiilil (näiteks arenduskeskonnas kasutatakse erinevat andmebaasi kui serveris käivas rakenduses (Joonis 17)). Kasutatavat Springi Booti profiili saab määrata rakenduse ehitamisel ja seega on seda võimalik täpsustada konveieri konfiguratsioonis.

```
spring:
    profiles:
        active: dev
    datasource:
        type: com.zaxxer.hikari.HikariDataSource
        url: jdbc:postgresql://localhost:5432/fysioWeb
        username: FysioWeb
        password: FysioWeb
```

Joonis 17. Dev profiilis kasutatav andmebaasiühendus.

## 4.3 Esikomponendi optimeerimine

Rakenduse esikomponendi arendust alustades kasutati AngularJS-i vaid vaadete vaheliseks navigatsiooniks. Kuna rakenduse kasvades võeti AngularJS laiemalt kasutusele, esineb esikomponendi koodis palju ebakorrapära. Osades kohtades on kasutajaliidese dünaamilisus saavutatud elementide stiili muutes (vt Joonis 10 lk 32) ning teistes kohtades AngularJS-i abil (Joonis 18).

```
<tr class="overview" ng-show="dayState == 'overview'">
    ...
</tr>


$scope.goToOverview = function() {
    $scope.dayState = 'overview'
}
```

Joonis 18. Elemendi nähtavaks muutmine AngularJS-ga.

Selline ebakõla teeb koodi raskemini hallatavaks ning rakenduse skaleerimise tülikamaks. Mõistlik oleks suur osa rakenduse esikomponendist ümber kirjutada selliselt, et funktsionaalsus säiliks, kuid sarnase ülesandega koodijupid oleksid sarnase ülesehitusega. Kuna see nõuaks rohkem aega kui lõputöö valmimiseks on, jääb esikomponendi optimeerimine lõputöö skoobist välja.

## 4.4 Testimine

Testjuhitud arendus (*test driven development*) on efektiivne moodus vigade koheseks leidmiseks ja koodi korrasolu kontrolliks [34]. Kuna käesolev rakendus on ainult ühest arendajast koosneva meeskonna jaoks küllaltki suur projekt ning ajaressurss on piiratud, siis lõputöö raames testide kirjutamisele suur rõhku ei pandud.

Rakenduse testimine on siiski vajalik ning teste täiendatakse ja lisatakse kindlasti enne rakenduse lõpliku versiooni väljalaskmist juurde. Rakenduse esi- ja tagakomponenti testitakse eraldi. Väiksemaid tagakomponendi utiliite, mis toetavad suuremaid rakenduses olevaid teenuseid, testitakse standardsete jUnit[1] ühiktestidega. Suuremaid tagakomponendi teenuseid testitakse Spring Boot raamistikus sisalduvate testimiseks mõeldud meetoditega, mille abil on võimalik näiteks teenuse poolt tagastatava HTTP sõnumi erinevate osade kontrollimine. Tänu sellele on võimalik rakendust integratsioonitestidega testida seda tööle panemata (Joonis 19) [35].

```
@Test
public void testDoctorListLength() {
        MvcResult result = mockMvc.perform(get("/api/allDoctors"))
             .andExpect(status().isOk())
             .andReturn();
        List<Doctor> doctorList = new ObjectMapper()
             .readValue(result.getResponse().getContentAsString()
             , List.class);
        assertEquals(doctorList.size(), 2);
}
```

Joonis 19. Kõikide arstide nimekirja saamise päringu põhjal kirjutatud test.

Esikomponendi vaadete testimiseks kasutatakse Seleniumit[2]. Seleniumi abiga on võimalik koostada teste, mis testivad vaadete kiirust ning funktsionaalsust. AngularJS-i testimiseks on võimalik kasutada AngularJS-i enda poolt pakutavat ühiktestimise võimalust (Joonis 20) või erinevaid AngularJS-i testimiseks mõeldud raamistikke [36].

```
it('Convert number of month (0-11) to string', function() {
  var filter = $filter('monthFilter');
  expect(filter(0)).toEqual('JAANUAR');
});
```

Joonis 20. AngularJS-i põhjal kirjutatud ühiktest.

[1] http://junit.org/junit5/

[2] http://www.seleniumhq.org/

# 5 Kokkuvõte

Bakalaureuse töö eesmärgiks oli luua kliendi poolt tellitud infosüsteemi patsientide ja visiitide haldamise allsüsteemi realiseeriv veebirakendus. Lõputöös anti ülevaade koostööst kliendiga ning nõuete analüüsist, võrreldi erinevaid populaarsemaid veebirakenduste loomise tehnoloogiaid ja kirjeldati rakenduse põhifunktsionaalsuse loomist. Lisaks kaeti lühidalt rakendusega seotud arenduse jätkumine tulevikus.

Kuigi suur osa soovitud funktsionaalsusest sai realiseeritud, selgus arenduse käigus, et realiseeritav allsüsteem on mahukam kui esialgse analüüsi käigus arvati. Sellest tulenevalt ei ole rakendus jõudnud staadiumisse, kus klient oleks valmis seda kasutusele võtma. Realiseerimata jäi privaatsete isikuandmete haldamiseks nõutav turvalisus ning arenduse soodustamiseks kasulik pidevintegratsioon. Lisaks ei ole rakenduse tähtsamad komponendid täielikult testidega kaetud. Kuna arenduse jooksul võeti kasutusele tehnoloogiad, mida esialgu kasutada ei planeeritud, on ka valminud esikomponendi lähtekoodi kvaliteet soovitust kehvem.

Arendamise efektiivsusele oleks tingimata kaasa aidanud mõne levinud arendusmetoodika kasutuselevõtt. See oleks taganud stabiilse arendamise tempo, mis oleks välistanud tihti esinenud päevadepikkuseid lünkasid rakenduse loomisel.

Kasutatud tehnoloogiate valikuga võib rahule jääda. Kuigi kliendi esialgset serverit arvestades oleks olnud mõistlikum kasutada tagakomponendi loomiseks PHP-d ja MySQL-i, oleks uue keele õppimiseks kuluv aeg rakenduse valmimist veel edasi lükanud. Sama võib öelda ka esikomponendi loomisel kasutatud AngularJS-i kohta.

Kokkuvõttes võib projekti, vaatamata rakenduse poolikusele, siiski lugeda õnnestunuks, kuna klient on tehtud tööga rahul. Lisaks sai autor eeskätt kogeda seda, kui ajakulukas on üksinda suure veebirakenduse loomine ning kui tähtis on sobiva arendamismetoodika valik.

# Kasutatud kirjandus


[1] I. Gerchev, “The 5 Most Popular Frontend Frameworks of 2017 Compared,” 17.05.2017 [Võrgumaterjal]. Saadaval: https://www.sitepoint.com/5-most-popular-frontend-frameworks-compared/. Kasutatud 03.01.2018

[2] “Brands using Foundation,” Foundationi tutvustuses. [Võrgumaterjal]. Saadaval: https://foundation.zurb.com/showcase/about.html. Kasutatud 03.01.2018

[3] S. Sinitsky, “Top 5 JavaScript Frameworks For Your Projects,” 06.10.2017 [Võrgumaterjal]. Saadaval: https://anadea.info/blog/top-javascript-frameworks. Kasutatud 03.01.2018

[4] Dominik T, “Angular vs React—the DEAL BREAKER,” 17.01.2017 [Võrgumaterjal]. Saadaval: https://hackernoon.com/angular-vs-react-the-deal-breaker-7d76c04496bc. Kasutatud 03.01.2018

[5] “Data Binding in AngularJS Templates,” AngularJS-i dokumentatsioonis. [Võrgumaterjal]. Saadaval: https://docs.angularjs.org/guide/databinding. Kasutatud 03.01.2018

[6] “Step 2: Build A Static Version in React,” Reacti dokumentatsioonis. [Võrgumaterjal]. Saadaval: https://reactjs.org/docs/thinking-in-react.html. Kasutatud 03.01.2018

[7] M. Weinberger, “The 15 most popular programming languages, according to the 'Facebook for programmers',” 11.10.2017 [Võrgumaterjal]. Saadaval: http://www.businessinsider.com/the-9-most-popular-programming-languages-according-to-the-facebook-for-programmers-2017-10. Kasutatud 03.01.2018

[8] Jamsheer K, “42 PHP Frameworks to Watch Out in 2017,” 13.01.2017 [Võrgumaterjal]. Saadaval: http://acodez.in/best-php-frameworks/. Kasutatud 03.01.2018

[9] B. Savage, “You don’t need a framework,” 08.01.2014 [Võrgumaterjal]. Saadaval: https://www.brandonsavage.net/you-dont-need-a-framework/. Kasutatud 03.01.2018

[10] “JAVA,” “PHP,” “PYTHON,” “NODE.JS,” Erinevate programmeerimiskeelte koodi võrdluses. [Võrgumaterjal]. Saadaval: https://excelwithbusiness.com/blog/say-hello-world-in-28-different-programming-languages/. Kasutatud 03.01.2018

[11] PHP programs versus Java. - [Võrgumaterjal]. Saadaval: http://benchmarksgame.alioth.debian.org/u32/compare.php?lang=php&lang2=java. Kasutatud 03.01.2018

[12] Python 3 programs versus Java. - [Võrgumaterjal]. Saadaval: https://benchmarksgame.alioth.debian.org/u64q/python.html. Kasutatud 30.12.2017

[13] Usage Statistics for Programming Language technologies. [Võrgumaterjal]. Saadaval: https://trends.builtwith.com/framework/programming-language. Kasutatud 30.12.2017

[14] R. Posa, “Node JS Architecture – Single Threaded Event Loop,” 21.06.2016 [Võrgumaterjal]. Saadaval: https://www.journaldev.com/7462/node-js-architecture-single-threaded-event-loop. Kasutatud 03.01.2018

[15] S. Maple, “Java Tools and Technologies Landscape Report 2016,” 14.06.2016 [Võrgumaterjal]. Saadaval: https://zeroturnaround.com/rebellabs/top-4-java-web-

frameworks-revealed-real-life-usage-data-of-spring-mvc-vaadin-gwt-and-jsf/. Kasutatud 17.12.2017

[16] C. Buckler, “SQL vs NoSQL: The Differences,” 18.09.2015 [Võrgumaterjal]. Saadaval: https://www.sitepoint.com/sql-vs-nosql-differences/. Kasutatud 03.01.2018

[17] C. Hadjigeorgiou, “RDBMS vs NoSQL: Performance and Scaling Comparison,” Magistritöö, Edinburghi Ülikool, Edinburgh, Šotimaa, 2013. [Võrgumaterjal]. Saadaval: https://static.epcc.ed.ac.uk/dissertations/hpc-msc/2012-2013/RDBMS%20vs%20NoSQL%20-%20Performance%20and%20Scaling%20Comparison.pdf. Kasutatud 17.12.2017

[18] “License Rights,” Oracle andmebaasisüsteemi kasutustingimustes. [Võrgumaterjal]. Saadaval: http://www.oracle.com/technetwork/licenses/database-11g-express-license-459621.html. Kasutatud 17.12.2017

[19] Solomonoff's Secret, “Is MySQL really this bad,” 18.11.2007 [Foorumipostitus]. Saadaval: https://arstechnica.com/civis/viewtopic.php?f=20&t=92525. Kasutatud 17.12.2017

[20] “Tarkvara,” Eesti e-tervise sihtasutuse kodulehel. [Võrgumaterjal]. Saadaval: http://www.e-tervis.ee/index.php/et/2012-07-22-08-57-49/liidestumine-tervise-infosysteemiga/tarkvara. Kasutatud 18.12.2017

[21] M. Rouse, “cowboy coding,” 12.2014 [Võrgumaterjal]. Saadaval: http://searchsoftwarequality.techtarget.com/definition/cowboy-coding. Kasutatud 01.01.2018

[22] R. T. Fielding, ”Architectural Styles and the Design of Network-based Software Architectures,” Doktoritöö, California Ülikool, Berkley, California, USA, 2000. [Võrgumaterjal]. Saadaval: https://www.ics.uci.edu/%7Efielding/pubs/dissertation/top.htm. Kasutatud 03.01.2018

[23] ”Creating an application,” jhipsteri dokumentatsioonis. [Võrgumaterjal]. Saadaval: http://www.jhipster.tech/creating-an-app/. Kasutatud 03.01.2018

[24] ”Other Changelog formats,” liquibase-i dokumentatsioonis. [Võrgumaterjal]. Saadaval: http://www.liquibase.org/documentation/other_formats.html. Kasutatud 03.01.2018

[25] ”Generating Change Logs,” liquibase-i dokumentatsioonis. [Võrgumaterjal]. Saadaval: http://www.liquibase.org/documentation/generating_changelogs.html. Kasutatud 03.01.2018

[26] S. K. Bansal, ”Securing Passwords with Bcrypt Hashing Function,” 10.04.2014 [Võrgumaterjal]. Saadaval: https://thehackernews.com/2014/04/securing-passwords-with-bcrypt-hashing.html. Kasutatud 29.12.2017

[27] N. C. Zakas, “Introduction to sessionStorage,” 21.07.2009 [Võrgumaterjal]. Saadaval: https://www.nczonline.net/blog/2009/07/21/introduction-to-sessionstorage/. Kasutatud 29.12.2017

[28] “BinaryFilesInDB,” PostgreSQL-i wikis. [Võrgumaterjal]. Saadaval: https://wiki.postgresql.org/wiki/BinaryFilesInDB. Kasutatud 29.12.2017

[29] Quentin, “window.open with headers,” 01.12.2010 [Foorumipostitus]. Saadaval: https://stackoverflow.com/questions/4325968/window-open-with-headers. Kasutatud 03.01.2018

[30] “Kasutaja tuvastamine veebis,” 20.04.2014 [Võrgumaterjal]. Saadaval: https://eid.eesti.ee/index.php/Kasutaja_tuvastamine_veebis. Kasutatud 29.12.2017

[31] “Kuidas Mobiil-ID oma infosüsteemiga siduda,” 25.02.2012 [Võrgumaterjal]. Saadaval: https://www.id.ee/index.php?id=30388. Kasutatud 30.12.2017

[32] “Using a Jenkinsfile,” Jenkinsi dokumentatsioonis. [Võrgumaterjal]. Saadaval: https://jenkins.io/doc/book/pipeline/jenkinsfile/. Kasutatud 30.12.2017

[33] “Profiles,” Spring Booti dokumentatsioonis. [Võrgumaterjal]. Saadaval: https://docs.spring.io/spring-boot/docs/current/reference/html/boot-features-profiles.html. Kasutatud 31.12.2017

[34] K. Beck, *Test-Driven Development By Example*, 2000. [Võrgumaterjal]. Saadaval: https://www.eecs.yorku.ca/course_archive/2003-04/W/3311/sectionM/case_studies/money/KentBeck_TDD_byexample.pdf. Kasutatud 03.01.2018

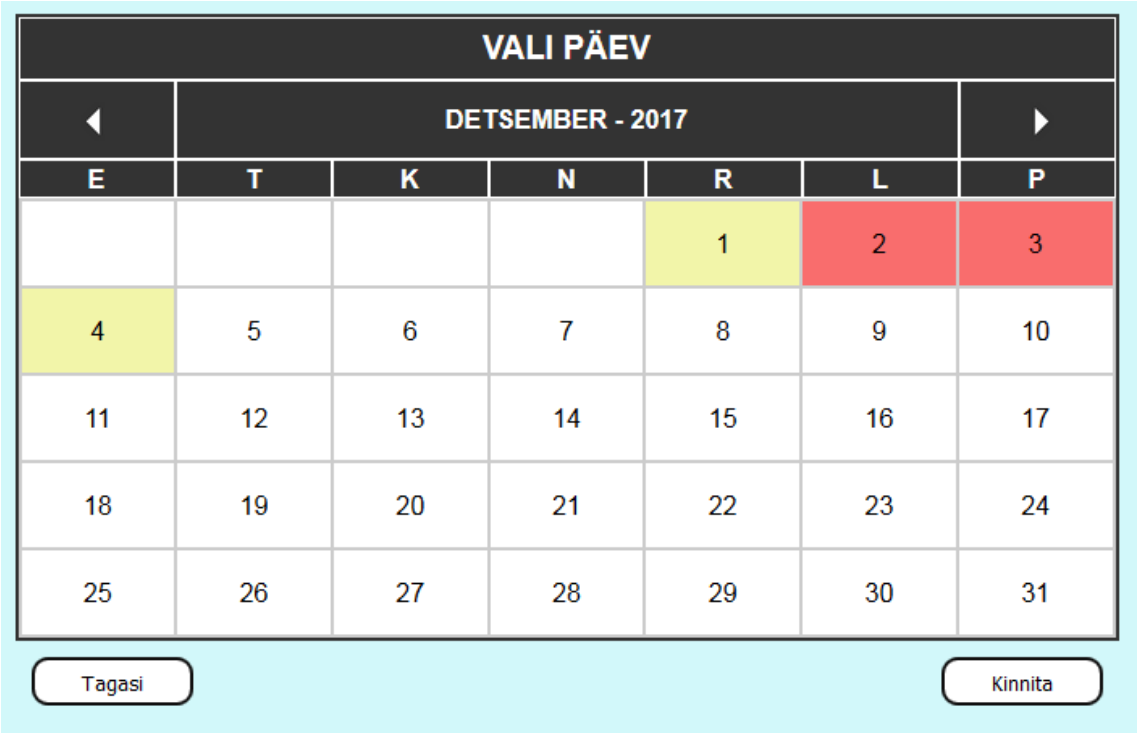
[35] “Testing,” Spring Booti dokumentatsioonis. [Võrgumaterjal]. Saadaval: https://docs.spring.io/spring-boot/docs/current/reference/html/boot-features-testing.html. Kasutatud 03.01.2018

[36] “Unit Testing,” AngularJS-i dokumentatsioonis. [Võrgumaterjal]. Saadaval: https://docs.angularjs.org/guide/unit-testing. Kasutatud 30.12.2017

# Lisa 1 – Loodava rakenduse repositooriumi aadress

https://gitlab.cs.ttu.ee/Priit.Tohver1/baka

# Lisa 2 – Patsiendile kuvatav kalender



## Lisa 3 – Patsiendi vaade