TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Jevgeni Vereštšagin 031687

PÄÄSTEVAHENDITE JA TULEKUSTUTUSVARUSTUSE SERTIFITSEERIMISDOKUMENTIDE LOOMISE TARKVARA ANALÜÜS JA PROTOTÜÜP

Bakalaureusetöö

Juhendaja: Inna Švartsman
Lektor

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: Jevgeni Vereštšagin

09.05.2018

Annotatsioon

Antud projekti raames oli tehtud MS Access tarkvara eeliste ja puudujääkide analüüs eesmärgiga üle kanda vana MS Accessi põhise paastevahendite ja tulekustutusvarustuse sertifikaatide loomise ja hoidmise süsteem uute tehnoloogiate peale ning vältida MS Access-i tarkvara peamisi puudujääke.

Oli läbi viidud populaarsemate programmeerimiskeelte ja nendes kirjutatud raamistike analüüs, eesmärgiga leida vahendid, mis võimaldavad kiiresti arendada erinevatel platvormidel töötavat tarkvara. Tuginedes erinevatele teadusartiklidele, populaarsuse reitingutele, GitHubis olevale statistikale, jõudlustestidele ja kliendi poolt esitatud nõuetele, olid välja valitud optimaalsed arendusvahendid.

Lisaks oli detailselt lahti kirjutatud veebirakenduse arenduskäik koos koodinäidetega ja kasutajaliidesest piltidega ning esitatud plaanitavate täienduste nimekiri.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 31 leheküljel, 5 peatükki, 8 joonist, 2 tabelit.

Abstract

Firefighting and lifesaving equipment certificates creation software analysis and prototype

In this thesis, the author has analyzed the advantages and disadvantages of MS Access software in order to move old firefighting and lifesaving equipment certificates creation system to new technologies and avoid the main disadvantages of MS Access.

Most popular programming languages and frameworks written in them have been analyzed in order to find the most suitable tools for quick developing of cross-platform software. The choice of tools relies on scientific papers, popularity charts, GitHub statistics, performance tests and client requirements.

Software development workflow is described in detail, providing code samples and user interface screenshots. Non-implemented functionalities are listed in the list of future work.

The thesis is in Estonian and contains 31 pages of text, 5 chapters, 8 figures, 2 tables.

Lühendite ja mõistete sõnastik

API	*Application programming interface*, rakendusliides ehk programmiliides ehk rakendustarkvara liides
SQL	*Structured Query Language*, struktuurpäringukeel
JSON	*JavaScript Object Notation*, lihtsustatud andmevahetusvorming, mis põhineb JavaScripti programmeerimiskeele alamhulgal
XSS	*Cross-site Scripting*, turvalisuse haavatavuse tüüp, mis võib veebirakendustes leiduda. XSS rünnak toimub kõige sagedamini siis, kui kasutaja sisend ei olnud kontrollitud ega töödeldud ning sisendis olnud skript käivitub kasutaja brauseris.
REST	*Representational State Transfer*, tarkvaraarhitektuuri stiil, mida kasutatakse hüpermeedia hajussüsteemide (hajusarvutuse) valdkonnas
ORM	*Object-relational mapping*, objektimudeli kujutamine relatsiooniliseks andmemudeliks
MIT License	on vaba tarkvara (*free software*) litsens, mis on loodud Massachusettsi Tehnoloogia Instituudis (MIT)
XML	*Extensible Markup Language* – laiendatav märgistuskeel on W3C välja töötatud ja soovitatud standardne üldotstarbeline märgistuskeel.

Sisukord

Jooniste loetelu

Tabelite loetelu

1 Sissejuhatus

Käesoleva töö raames analüüsitakse päästevahendite ja tulekustutusvarustuse sertifitseerimisdokumentide loomise tarkvara arendamisel kasutatud vahendit ja valitakse välja uued kaasaegsemad lahendused, mida hakatakse kasutama uue süsteemi arendamiseks.

Antud töö eesmärkideks on:

- Kaasaegsete tehnoloogiate analüüs tuginedes arendatava süsteemi spetsiifikale ja parima lahenduse valik.
- Kirjeldada süsteemi arendamiskäiku.
- Luua uue süsteemi prototüüp.

Eespool nimetatud esmärkide saavutamiseks vajalikud ülesanded:

- Selgitada välja vana süsteemi loomisel kasutatud tarkvara eelised ja puudujäägid.
- Analüüsida populaarsemad *back-end* loomisel kasutatavad programmeerimiskeeled ja valida välja sobivaim.
- Analüüsida kaasaegsed *back-end* raamistikud ja valida välja sobivaim.
- Analüüsida kaasaegsed *front-end* raamistikud ja valida välja sobivaim.
- Luua uue süsteemi prototüüp kasutades valitud tehnoloogiad.
- Tuua välja plaanitavad täiendused (kirjeldada lõputöö raames mitterealiseeritud funktsionaalsusi).

Lõputöö teises peatükis tuuakse välja MS Access tarkvara positiivsed ja negatiivsed küljed.

Kolmandas peatükis analüüsitakse populaarsemaid programmeerimiskeeli, *back-end* ja *front-end* veebiraamistikke ning seejärel valitakse välja sobivamad.

Seejärel neljandas peatükis kirjeldatakse veebirakenduse arenduskäiku ja esitatakse arendatava rakenduse prototüüp.

Viiendas peatükis raägitakse plaanitavatest täiendustest, mida on vaja enne kliendile üle andmist veebirakenduses implementeerida.

1.1 Infosüsteemi kirjeldus

Arendatav infosüsteem on tellitud kliendi poolt. Infosüsteemi andmebaasis hoitakse sertifitseerimisdokumente ja andmeid laevapäästevahendite ja tulekustutusvarustuse kohta (näiteks päästevestide, tulekustutite seisukord ja arv, päästeparved, tulekustutussüsteemid, hüdrotermoülikonnad, muud märkused). Alguses süsteemi kasutab väike arv (< 10) võrdsete õigustega inimesi. Sertifikaate saab luua ning hiljem andmebaasist otsida ja nende andmeid vaadata. Andmebaasis olevatest sertifikaatidest saab genereerida pdf failid. Süsteemis võib hoida ka alltöövõtjate sertifikaate. Sertifikaatide kehtivuse lõppemisel kasutajat teavitatakse. Lisaks on süsteemil administraator, kes lisab kasutajad ja ühikuid andmebaasi. Seega on süsteemis kaks kasutajaprofiili: administraator ja kasutaja. Kasutaja saab vaadata ja luua kõikide sertifikaatide pdf-id. Kasutaja saab redigeerida ja kustutada sertifikaate.

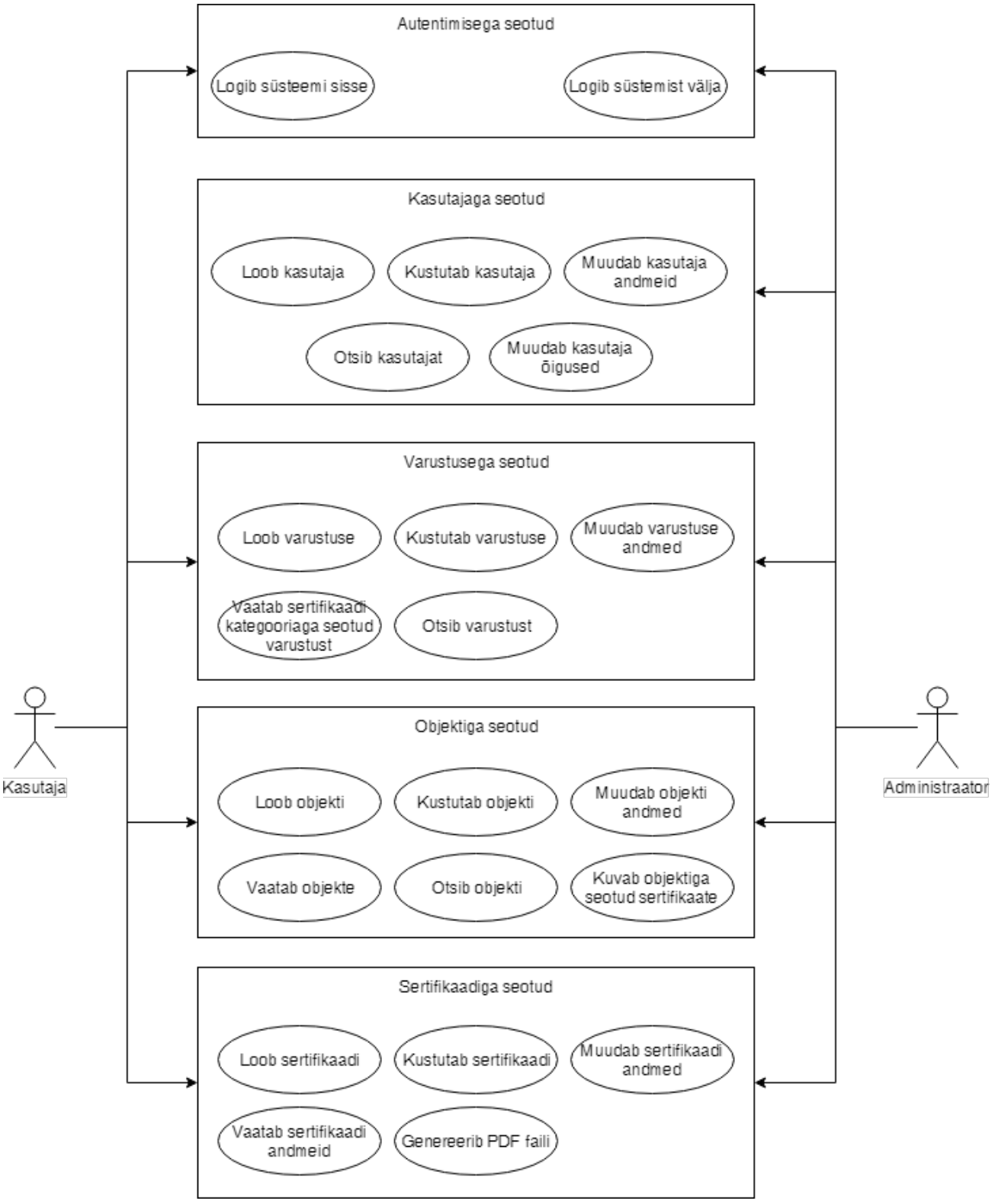
Andmebaasis hoitakse kõik loodud sertifikaadid. Iga sertifikaat on seotud ühe sertifitseemisobjektiga. Sertifikaadi loomisel kasutaja valib kategooria, kuhu kontrollitav varustus kuulub. Sõltuvalt kategooriast kuvatakse sertifikaadi vormid ning kategooriaga seotud staatilised andmed. Sertifikaadi vorm on kinnitatud rahvusvaheliste registrite poolt ja omab versiooni sõltuvalt sisemisest kvaliteedi süsteemist, mis võib aja jooksul koos vormiga rahvusvaheliste nõuete pärast muutuda. Sertifikaate saab väljastada ettevõte, kes on saanud tegevusloa ühest või mitmest regionaalsest registrist. Sertifikaadid on rahvusvahelised ja kehtivad kõikide riikide territooriumil. Sertifikaadid luuakse check-list-ide alusel (tulevikus sisestatakse ka neid andmebaasi ja need on sertifikaadi laienduseks). *Check-list* on andmed, mida täidab kontrolliv isik otseselt laeval või kohal kus toimub varustuse kontroll.

Nendes hoitakse kõik nimetused, mõõtmised ja tööd, mis olid kontrollitavate ühikutega läbi viidud, lisaks ka kogu infot, mida sertifikaadil ei näidata. Iga kontrollitav varustuse ühik peab vastama rahvusvaheliselt kehtestatud standardile. Juhul kui varustus ei läbinud kontrolli, tehakse remonti ja kalibreeriimist, parandatakse, taastatakse järgides rahvusvahelisi norme või läheb mahakandmisele.

Sertifikaati on võimalik kustutada ja redigeerida juhul kui andmete sisestamisel oli tehtud viga. Pärast sertifikaadi aktsepteerimist ja andmebaasi salvestamist on võimalik genereerida sellest PDF fail, mida saab alla kirjutada digitaalselt ja e-posti teel saata või väljastada paberkandjal koos allkirja ja pitsatiga.

Sertifikaatide väljastamise ja otsingu süsteemile saavad ligi vaid sertifitseerimisettevõtte töötajad ja see on loodud kõigepealt nende mugavuseks ja töö lihtsustamiseks. Lisaks võimaldab süsteem jälgida lõppeva kehtivusega sertifikaate, mis aitab õigeaegselt teavitada klienti uuest kontrollist. Juhul kui klient kaotab väljastatud sertifikaadi, on võimalik pöörduda sertifitseerimisettevõtte poole ja tellida vana sertifikaadi uue versiooni.

Kõik süsteemi põhilised kasutusjuhud näidetud allpool oleval joonisel (Joonis 1).



Joonis 1 *Use Case* diagramm.

Tarkvara funktsionaalsus:

- Autentifikatsioon
- Andmebaasi muutmine läbi administraatori paneeli
- Objekti otsing
- Objekti lisamine
- Objekti redigeerimine
- Objekti kustutamine
- Objektiga seotud sertifikaatide kuvamine
- Lõppeva kehtivusajaga sertifikaatide kuvamine esilehel
- Sertifikaadi loomine
- Sertifikaadi redigeerimine
- Sertifikaadi kustutamine
- PDF faili genereerimine sertifikaadist
- Varustuse otsing
- Varustuse lisamine

2 Olemasoleva süsteemi loomisel kasutatud vahendi MS Access tarkvara analüüs

Hetkel kasutusel oleva süsteemi arendamiseks kasutati tarkvara MS Access. Allpool tuuakse selle tarkvara positiivsed ja negatiivsed aspektid. Selle analüüsi tulemusi arvestatakse uue süsteemi loomisel.

MS Access eelised

- Ei ole vajadust osta lisatarkvara juhul kui on soetatud Microsoft Office Pro või kõrgem.
- MS Access on üks populaarsemaid andmebaasisüsteeme. [1]
- MS Access on üks populaarsemaid desktop andmebaasisüsteeme. [2]
- MS Access on laiemalt toetatud teiste tarkvara tootjate poolt.

MS Access puudujäägid [3]

- Töötab ainult Ms Windowsi all.
- Raske kasutada samaaegselt mitme kasutaja poolt.
- Piiratud ühilduvus operatsioonisüsteemi uuemate versioonidega.
- Kulukas lahendus juhul kui tarkvara plaanitakse kasutada interneti teel mitme kasutaja poolt.
- Kasutajaliidese loomise piiratud võimalused.
- Puudub lihtne lahendus kuidas kasutada tarkvara mobiilsetel seadmetel.
- Lisakulud üleminekul uue süsteemi peale.
- Tarkvara kasutamine on piiratud litsentsi tingimustega.

Vaadates ülalpool mainitud MS Accessi puudujääke võib järeldada, et selle tarkvara suurimaks probleemiks on võimatus seda kasutada erinevatel platvormidel mitme kasutaja poolt korraga. Samuti on keeruline seda toetada üleminekul uuemale operatsioonisüsteemile. Nende puudujääkide vältimiseks oli otsustatud üle minna veebipõhisele infosüsteemile. Selline lahendus võimaldab vältida litsentsidega seotud kulusid ja laiendada kasutajaliidese arendamise võimalusi.

3 Uue süsteemi analüüs ja vahendite valik

Veebipõhiste süsteemide eeliseks on võimalus kasutada neid erinevate operatsioonisüsteemide (Windows, GNU/Linux, MacOS, iOS, Android jne) alt. Samuti on väga lihtne kohandada veebirakendusi nii, et nende kasutamine oleks mugav erineva suurusega ekraanide peal. Kuna on olemas väga palju vahendeid, mille abil saab luua veebirakendusi, tuli kõigepealt läbi viia vahendite analüüs, et teha selgeks, mis vahendite kasutamine oleks antud projekti puhul optimaalne. Selleks, et tarkvara oleks lihtsam arendada, oli otsustatud arendada *back-end* ja *front-end* osasid iseseisvalt. Tänu sellele lähenemisele võib tulevikus teha *front-end* muudatusi serveripoolset osa puutumata. Suhtlemine serveripoolse osaga toimub sel juhul REST protokolli kaudu. [4]

Tänapäeval on olemas kaks kõige populaarsemat formaati andmete esitamiseks läbi REST arhitektuuri. Need on XML ja JSON. Mõlemal formaadil on hulk eeliseid ja puudujääke, aga kuna meie süsteemis kasutatakse JavaScript *front-end*-i ja meile on tähtis andmete edastamise kiirus läbi REST arhitektuuri, minu arvates on mõttekam kasutada JSON formaati. [5]

3.1 *Back-end*

Enne *back-end* raamistikku valimist olid läbi vaadatud mõned kõige populaarsemad programmeerimiskeeled: Java, Python, PHP and Ruby. [6]

3.1.1 Programmeerimiskeele valik

Programmeerimiskeel Java on ülalnimetatud keeltest üks keerulisematest. Tavaliselt kirjutatakse Javas mahukaid projekte suurte ettevõttete jaoks. [7]

Java keeles kirjutatud raamistikud nagu Spring ja Vaadin on mahukad ja nad on suhteliselt raskelt õpitavad. See võib olla takistuseks juhul kui on vaja arendada mingit projekti väga piiratud ajaga. [8]

Programmeerimiskeel PHP on tänapäeval ikka veel väga populaarne veebirakenduste arendamise keel. Selles on kirjutatud suur hulk suuri projekte nagu näiteks Facebook, WordPress, Drupal jne. Vaatamata sellele on PHP ka olulisi puudujääke: nõrk tüpiseerimine, mis võib viia ootamatute vigadeni, mida on raske üles leida. Samuti ei ole

selles programmeerimiskeeles head vigade raporteerimist: tihti ei ole võimalik vea kirjeldusest ja selle tekkimiskohast aru saada.

Suureks PHP puudujäägiks, mis võib algaja programmeerija töö efektiivsust vähendada on ka selle programmeerimiskeele ebaloogilisus ja vasturääkivused. Programmeerija peab meelde jätma palju erandeid, puudub ühine funktsioonide nimetamise standard (näiteks “str_ireplace”, “str_pad”, “str_repeat” ja samal ajal “strcasecmp”, “strchr”, “strcmp” jne). Mõnede PHP funktsioonide nimed ei ole kooskõlas implementatsiooniga. [9]

Programmeerimiskeel Ruby on kergesti õpitav, sobib hästi algajatele programmeerijatele ja on väga modulaarne. Tavaliselt seostatakse Ruby-t kõige populaarsema veebiraamistikuga Ruby on Rails. Juhul kui selle raamistiku funktsionaalsus ei lase implementeerida projekti täielikult, tuleb lisada lisamooduleid, mis nõuavad keele põhjalikku õppimist, mis võib projekti kirjutamisele aega oluliselt suurendada. Suhteliselt raske on leida põhjalikku ja lihtsalt loetavat dokumentatsiooni. Kõikidest ülaltoodud keeltest Ruby peetakse kõige aeglasemaks koodi käivitamise ja töötlemise osas. [10]

Programeerimiskeel Python on lihtne ja seda on kerge õppida ka algajatel. Pythonis kirjutatud kood on väga lihtsalt loetav. Tänu suurele hulgale baasteeke see keel omab suurt funktsionaalsust ilma lisamoodulite paigaldamiseta. Pythonis on kirjutatud palju raamistikke, nende hulgast saab valida sobiva erineva suuruse projektide loomiseks. Suuremas osas Pythonis kirjutatud raamistikest on sisseehitatud testimise süsteem, mis võimaldab kiiret vigade kõrvaldamist. Sellised suured ettevõtted nagu Yahoo, Google, IBM, NASA, Nokia and Disney eelistavad ja soovitavad programmeerimiskeelt Python.

Pythoni puudujäägiks võib nimetada täitmise kiirust. Kuna Python on interpreteeritud keel, selles kirjutatud koodi täitmine on aeglasem kui keeltes kus toimub eelkompileerimine. Vaatamata sellele Pythonit uuendatakse pidevalt ja selle kiirus paraneb. [11] [12]

Projekti arendamiseks hakatakse kasutama programmeerimiskeelt Python. See keel sobib kiireks arendamiseks ning on kergesti õpitav. Python on väga populaarne programmeerimiskeel ning selles on kirjutatud hulk raamistikke, mis võimaldavad kasutada implementeeritud mooduleid ning vältida oma implementatsioonide kirjutamist tüüpiliste ülesannete lahendamisel. Pythonis kirjutatud raamistikud on hästi

dokumenteeritud ning abi saab ka paljude kanalite kaudu, näiteks Gitter, IRC jne. Pythonis kirjutatud veebirakenduste majutus on lihtne ja suhteliselt odav.

3.1.2 *Back-end* raamistike analüüs

Analüüsitavad *back-end* raamistikud olid valitud vastavalt veebilehel https://hotframeworks.com avaldatud skoorile. [13] Raamistike populaarsuse üldskoor on moodustatud GitHub ja Stack Overflow skooride põhjal. Edaspidi analüüsitakse kolm kõige populaarsemat Python veebiraamistikku: Django, Flask ja Tornado (Tabel 1).

Parameeter	Django	Flask	Tornado
Forks on GItHub (2018-04-14)	14064	10616	4528
Contributors on GitHub (2018-04-14)	1566	458	280
Built-in ORM	+	-	-
Vormide valideerimine	+	+	+
Built-in Admin panel	+	-	-
Encode a object to JSON and return as response in ms (lower is better) [14]	103.36	65.68	136.87
Load a response from remote server and return as response in ms (lower is better) [14]	6918.64	6824.88	1928.31
Load data from database with ORM and render to template in ms (lower is better) [14]	2632.36	1465.25	910.06
Testid	+	+	+

Litsents	3-Clause BSD	3-Clause BSD	Apache-2.0

Tabel 1. Python *back-end* raamistike võrdlus

Võrdlustabelis toodud parameetritest lähtudes võib järeldada, et

- Django on nende kolme raamistiku seast kõige populaarsem ja seda on kõige lihtsam seadistada, sest see sisaldab kohe kogu meile vajalikku funktsionaalsust. Puudujäägiks on selle raamistiku suurus. Jõudluse testid näitavad keskmiste väärtuste puhul halvimat tulemust.

- Flask on keskmise populaarsusega raamistik. Baasfunktsionaalsus on piiratud ja selle laiendamiseks on vaja installeerida lisamooduleid. Jõudluse poolest on Flask kolmest analüüsitavast raamistikust keskmine.

- Tornado on vähem populaarne raamistik ja nagu Flaski puhul peab lisamooduleid eraldi installeerima ja seadistama. Üldjõudluse poolest on aga Tornado parim.

Projekti arendamiseks kõige loogilisemaks valikuks tundus Django veebiraamistik. Django võimaldab administraatori paneeli kasutamist ilma et peaks lisamooduleid installeerima. Administraatori paneeli olemasolu on oluline selle pärast, et tänu sellele klient saab ise kirjeid andmebaasi lisada ja neid muuta ning selleks ei pea luua eraldi kasutajaliidest, mis võimaldaks kõikide vajalike tabelite poole pöörduda. Suureks plussiks on sisseehitatud ORM, mis võimaldab teha päringuid andmebaasi poole otse Python-ist vajaduseta SQL lauseid kirjutada. Django lai sisseehitatud fuktsionaalsus aitab lihtsustada projekti installeerimist ja seadistamist ning seega hoiab aega kokku.

Lisaks Django on populaarseim veebiraamistik GitHub-i keskkonnas, seepärast on vajalikku infot võrreldes vähem populaarsete raamistikega lihtsam leida ning suuremast kogukonnast tõenäosem kiiresti abi saada juhul kui tekkivad küsimused või probleemid. Jõudluse poolest ei ole Django analüüsitavatest veebiraamistikest parim, kui meie projekti puhul ei ole see aspekt kõige olulisem.

"*Encode an object to JSON and return as response*", mida hakkame meie projektis kasutama, näitab keskmist tulemust.

3.2 *Front-end* raamistike analüüs

Front-end raamistiku valimisel oli põhirõhk järgmistele aspektidele: jõudlus, kogukonna suurus, uuendused. [16] [17] Need parameetrid olid valitud selle pärast, et need on lihtsalt hinnatavad. Lähtudes github.com veebilehel olevast statistikast [18] võib järeldada, et suurim kogukond on järgmistel raamistikel: Angular, React, Vue. Edaspidi analüüsitakse neid kolm raamistikku (Tabel 2).

Parameeter	Angular	React	Vue
Forks on GItHub	8,546+	17000+	13000+
Contributors on GitHub	615 (2018-04-11)	1178 (2018-04-11)	188(2018-04-11)
Performance tests for Keyed implementations *Duration / Fastest* [15]	1,15	1,09	1,15
Performance tests for non Keyed implementations *Duration / Fastest* [15]	1,12	1,10	1,15
Uuendused (*Last release*)	2018-04-07	2018-04-03	2018-03-24
Litsents	MIT	MIT	MIT
Allalaadimiste arv kuus (Nov. 2016 - Okt. 2017)	616,135 -> 1,081,796 (+75.6%)	2,564,601 -> 7,040,410 (+174.5%)	6231 -> 874,424 (13933.4%)

Tabel 2. JavaScript *front-end* raamistike võrdlus

Front-end raamistike võrdlustabelist on näha, et nii React kui ka Angular ja Vue on aktiivselt arendatavad, kuid React raamistik on hetkel kõige populaarsem ja selle populaarsus kasvab hoogsalt. Populaarse ja suurema kogukonna raamistiku puhul on lihtsam leida vastuseid erinevatele küsimustele, mis võivad arendamise käigus tekkida.

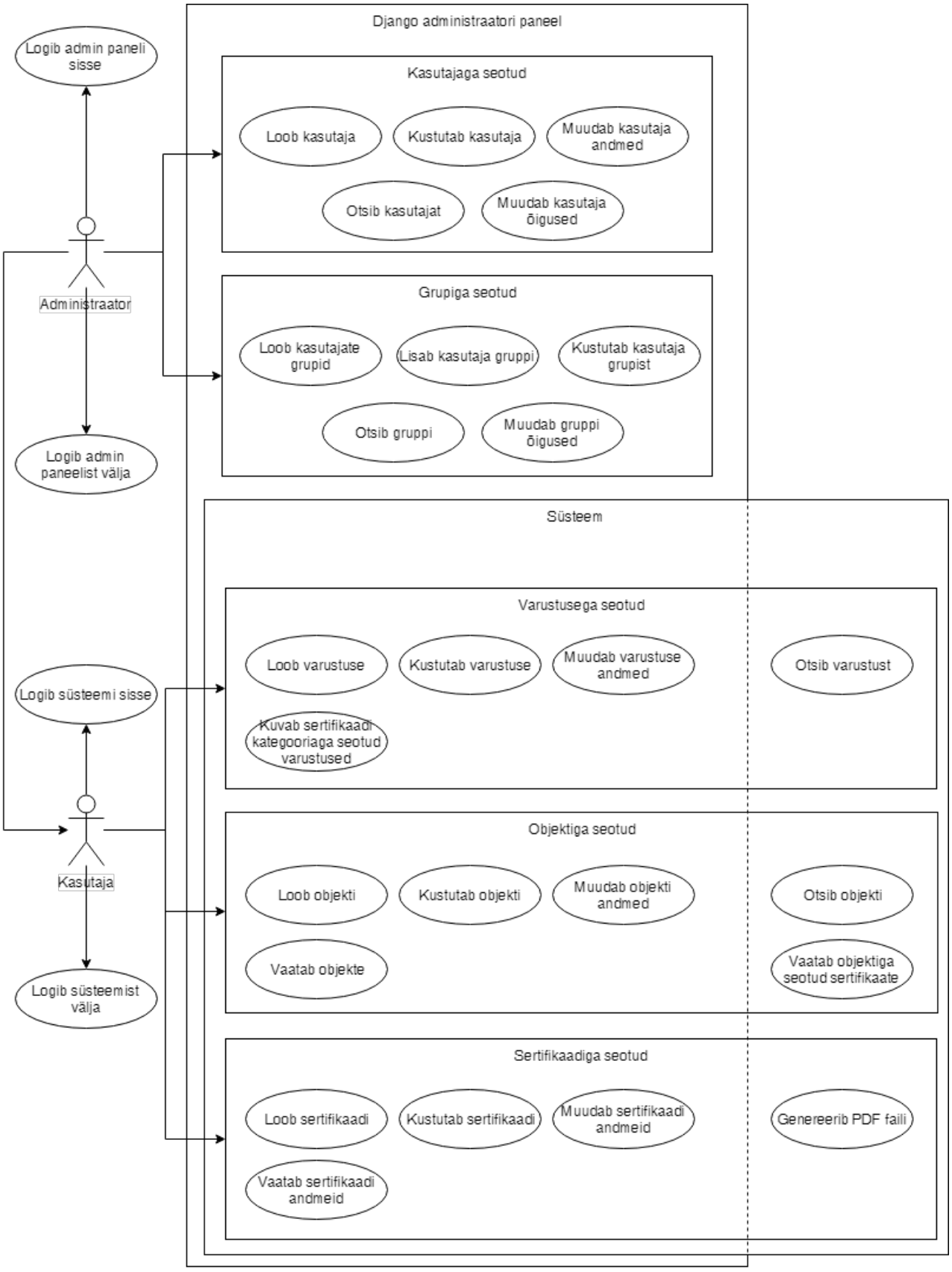
Kõik analüüsitavad raamistikud on saadaval MIT litsentsi all, mis tähendab, et neid saab kasutada ka kommertstoodete arendamisel ja vajadusel võib koodi modifitseerida.

Jõudluse testid olid tehtud arvestades kirjete loomisele, muutmisele, uuendamisele kulutud aega nii keyed kui ka non-keyed implementatsioonidel. Tabelis on antud kõikide positsioonide keskmised väärtused (kestvus / parim tulemus). Lähtudes nendest koeffitsientidest võib järeldada, et React raamistik on hetkel nendest kolmest jõudluse poolest parim. Tuginedes kõikidele ülalpool olevas tabelis mainitud andmetele teeme järelduse, et arendatava süsteemi front-end raamistikuks sobib React kõige paremini.

4 Süsteemi arenduskäik ja prototüüp

Selleks, et arendamine toimuks võimalikult kiiresti, oli otsustatud kasutada Django + React põhja. Selleks oli alla laaditud valmis põhi GitHub repositoorimist cookiecutter-django-reactjs. [19] Tänu sellele sain vajalikud vahendid paigaldada ja projekti seadistada väga kiiresti.

Arendatava veebirakenduse kasutusjuhud on näidatud allpool oleval joonisel (Joonis 2). Administraatoriga seotud kasutusjuhud implementeeritud Django administraatori paneeli abil ning kasutaja seotud kasutusjuhud on implementeeritud eraldiseisva süsteemina. Enamust kasutajaga seotud kasutusjuhte võib teha ka läbi administraatori paneeli.



Joonis 2. Arendatava süsteemi *use-case* diagramm.

4.1 *Back-end* ja andmebaas

Valitud koodipõhi eeldab PostgreSQL andmebaasisiüsteemi kasutamist. Kuna PostgreSQL on populaarseim vabavaraline SQL andmebaasisüsteem ning arendatav projekt ei eelda SQL lausete kirjutamist tänu Django ORM-ile, see andmebaasisüsteemi valik tundus loogiline ja suuremat analüüsi sellel teemal ei tehtud.

Projekti arendamise esimeseks sammuks oli luua andmebaas. See oli tehtud läbi Pythoni koodi. Selleks, et luua tabel PostgreSQL andmebaasi, tuleb luua Django mudel. Selleks käivitatakse projecti *root* kataloogis järgmise käsu:

```
python3 manage.py startapp appname
```

Seejärel lisatakse loodud mudel installitud rakenduste nimekirja:

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    ‘appname’,
]
```

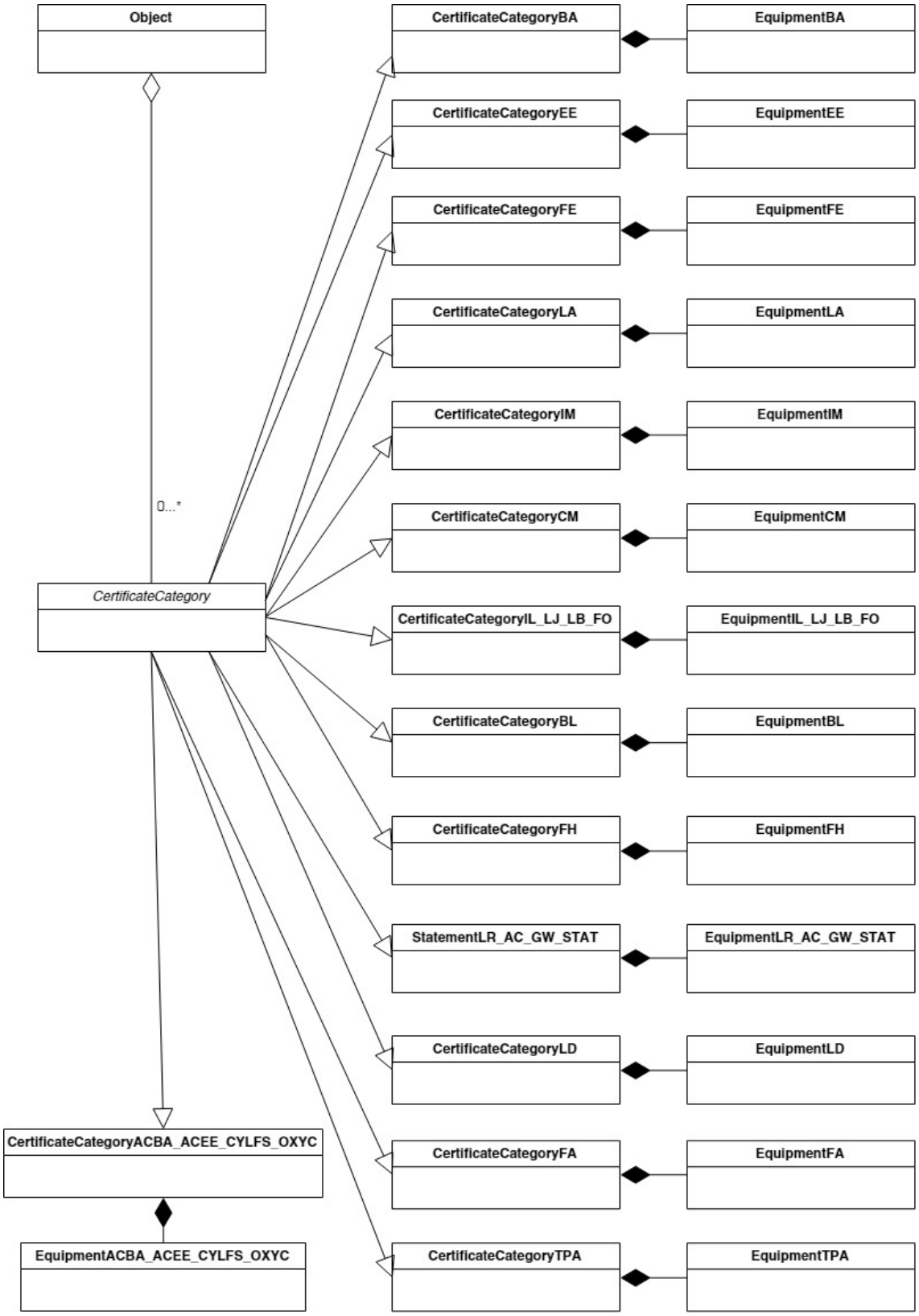
Failis *appname/models.py* kirjeldadatakse mudelit, mis vastab SQL tabelile.

```
from django.db import models
from django.utils import timezone
class ModelExample(models.Model):
    field = models.CharField(max_length=200)
    def __str__(self):
        return self.title
```

Kui mudeli väljade tüübid on määratud, on vaja luua mudelile vastava tabeli PostgreSQL andmebaasis. Selleks käivitame järjest järgmised käsud:

```
python3 manage.py makemigrations appname
python3 manage.py migrate appname
```

Projekti andmebaasi struktuur on näidatud järgmisel joonisel (Joonis 3):



Joonis 3. Süsteemi andmebaasi struktuur

Antud projekti kekskseteks mudeliteks on sertifikaadi mudel (implementeeritud abstraktse mudelina) ja selle alamklassid. Abstraktses klassis on kõikide sertifikaatide ühised väljad ning alamklassides on iga sertifikaaditüübi spetsiifilised väljad. Selline objekt-orienteeritud lähenemine aitab vältida liigse koodi kirjutamist. Klassi abstraktsus määratakse Django raamistikus klassi meta andmetes. Lisaks sertifikaatide mudelitele on mudel Object, mida kasutatakse laeva/tellija andmete manipuleerimiseks. Sertifikaat on seotud objektiga mitu-ühele sidemega. Iga sertifikaaditüübiga (*CertificateCategory*) on seotud varustuse tabel (näiteks *EquipmentFE*). Kuigi siin võiks samamoodi kasutada varustuse abstraktset klassi *Equipment*, meie süsteemi puhul ei ole varustuse tabelites ühtegi ühist välja. Kui ühised väljad siiski tekkivad, võib mudelite sekka lisada abstrakse klassi *Equipment*, ning laiendada olemasolevad varustuse klassid (näiteks *class EquipmentFE(Equipment)*). Allpool olev kood näitab klasside (mudelite) omavahelist seost.

```
class Object(models.Model):
    {Class fields}


class CertificateCategory(models.Model):
    object = models.ForeignKey(Object, on_delete=models.CASCADE)
    {Class fields}


    class Meta:
        abstract = True


class CertificateCategoryFE(CertificateCategory):
    {Class fields}


class EquipmentFE(models.Model):
    certificateCategoryFE =
models.ForeignKey(CertificateCategoryFE,
on_delete=models.CASCADE)
    {Class fields}
```

Selleks et muuta, kustutada ja vaadata mudeliga seotud tabeli kirjeid, võib kasutada Django sisseehitatud administraatori paneeli. Selleks et mudeliga seotud andmed oleksid administraatori paneelis nähtavad, tuleb lisada faili *admin.py* järgmised koodiread:

```
# appname/admin.py
from django.contrib import admin
from .models import ModelExample
admin.site.register(ModelExample)
```

Administraatori paneeli kasutamiseks tuleb luua *superuser*:

```
python3 manage.py createsuperuser
```

Seejarel pannakse server käima kasuga

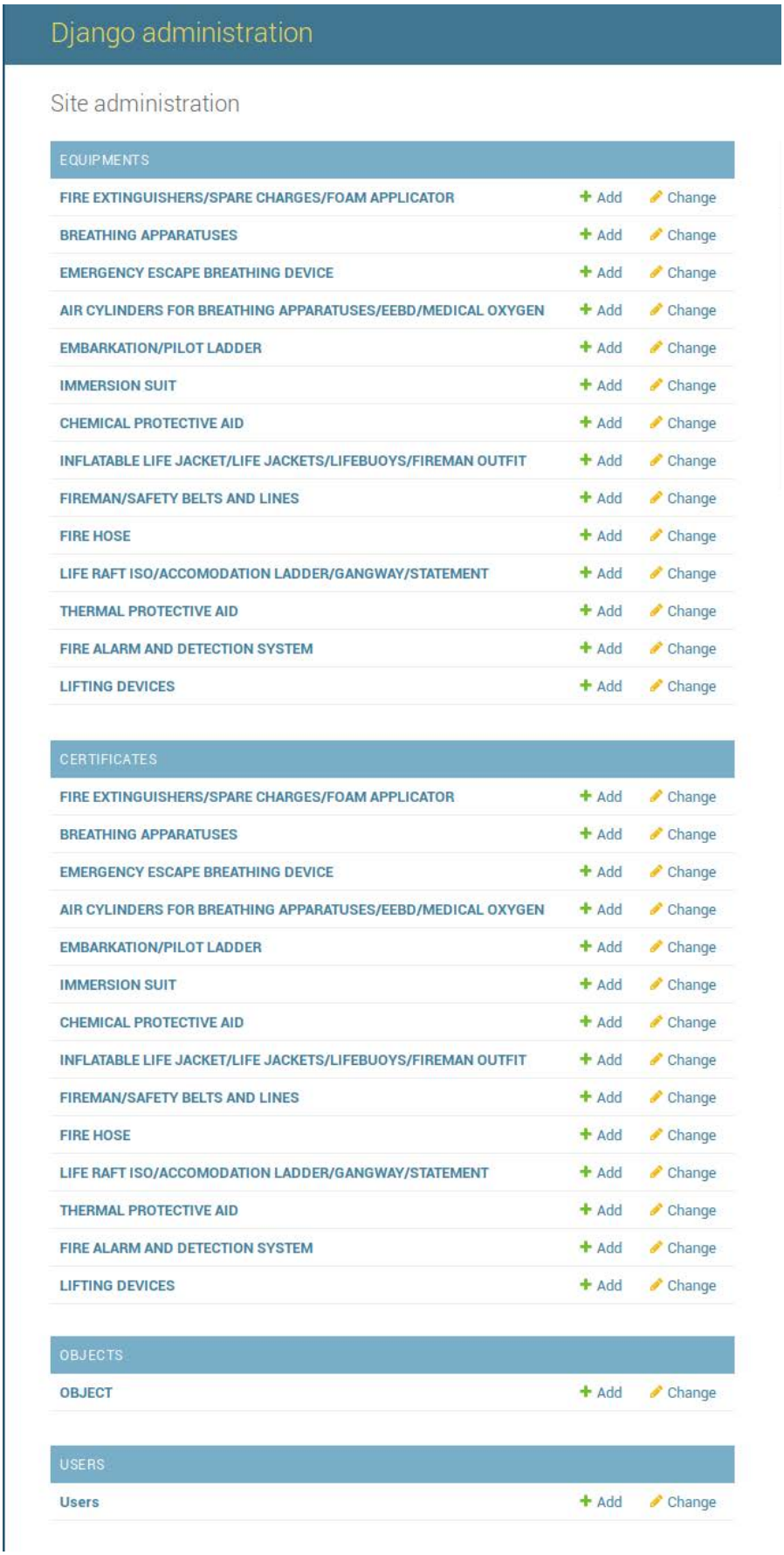
```
python manage.py runserver
```

Administraatori paneel on nüüd saadaval lingil *http://127.0.0.1:8000/admin/* (Joonis 4)

Peale *models.py* failis loodud mudelite saab administraatori paneelis näha ka Django poolt loodud tabeleid nagu näiteks kasujate (*Users*) või gruppide (*Groups*) tabel.

Vaikimisi on tabelite nimed administraatori paneelis automatselt genereeritavad ja inimeste poolt on raskelt loetavad. Selle parandamiseks oli klasside metaandmetesse lisatud nii nimetatud *verbose_name* ehk nimetus mida hakatakse administraatori paneelis kuvama.

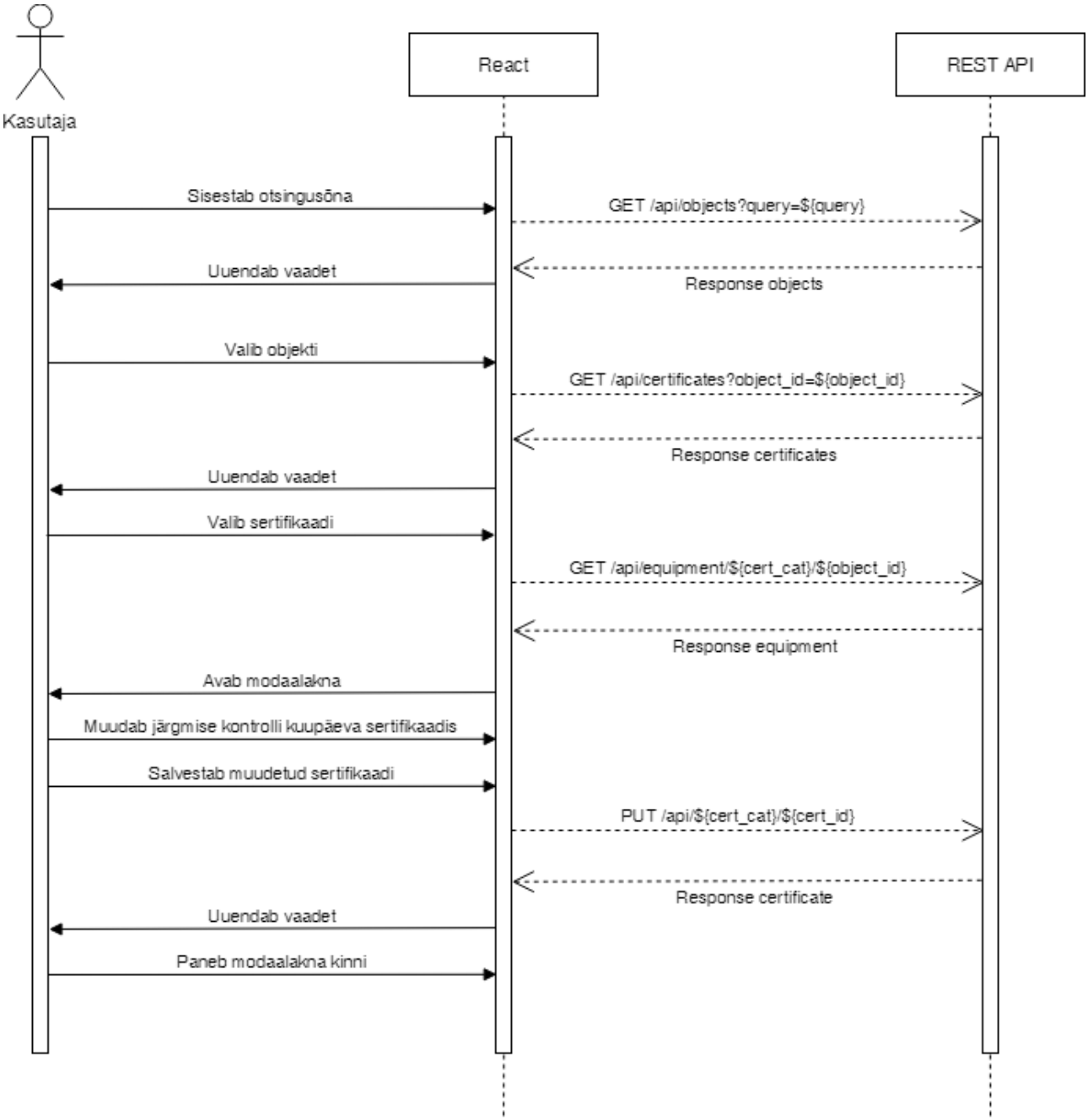
```
class CertificateCategoryFE(CertificateCategory):

    {Class fields}


    class Meta:

        verbose_name = 'FIRE EXTINGUISHERS/SPARE CHARGES/FOAM APPLICATOR'

        verbose_name_plural = verbose_name
```



Joonis 4. Arendatava veebirakenduse administraatori paneel.

4.2 *Front-end ja suhtlus API-ga*

Projekti struktuur on järgmine: *front-end* ja *back-end* on eraldiseisvad ning suhtlevad API kaudu (Joonis 5). Selleks kasutatakse raamistiku Django REST. Selline lahendus oli valitud eesmärgiga eraldada kliendi- ja selveripoolset osa. Tänu sellele võib teha mitu klienti, mis võivad teha päringuid serveri poole API kaudu. [4] Andmed liiguvad kliendi poolt serverile ja tagasi JSON formaadis.



Joonis 5. Sertifikaadi välja uuendamise jadadiagramm.

Kliendi poolt saadetakse päringuid Django REST API poole. Näiteks uue objekti loomiseks saadab kliendipoolne osa järgmiseid andmeid JSON formaadis:

```
 {
    "vessel_name": "Vessel Name",
    "customer_name": "Customer Name",
    "imo": "Imo",
    "flag": "Flag",
    "register": "Register"
  }
```

Andmete edastamine serverile toimub järgmiselt:

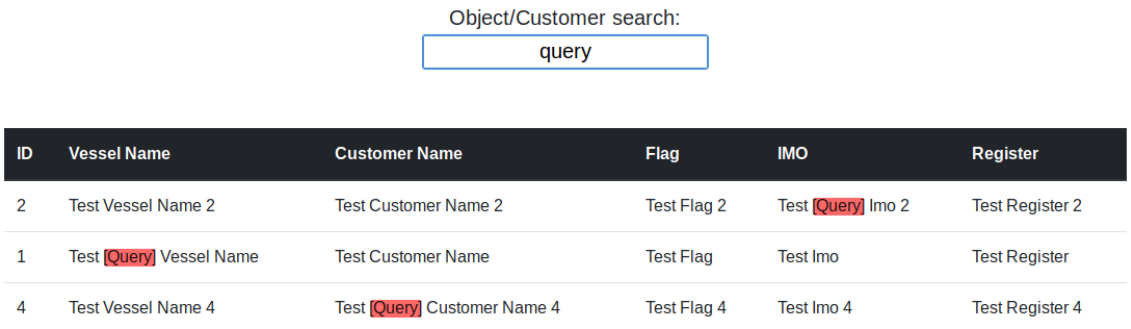
```
function createObject(objectData) {
     const url = `${BASE_URL}/api/objects`;
     return axios.post(url, objectData).then(response =>
response.data);
}
```

Seejärel tagastatakse serverilt vastuseks loodud objekti andmeid:

```
 {
    "id": "42",
    "vessel_name": "Vessel Name",
    "customer_name": "Customer Name",
    "imo": "Imo",
    "flag": "Flag",
    "register": "Register"
  }
```

Päringute saatmiseks kasutatakse axios [20]. Andmete pärimiseks kasutatakse GET meetodit, uuendamiseks PUT meetodit ja uute andmete lisamiseks POST meetodit.

Veebirakenduse esilehel on otsing, mille abil kasutaja saab leida vajaliku *Object*-i. Kuna *Object* mudelis on mitu välja (*vessel_name*, *customer_name*, *imo*, *flag*, *register*), siis oli otsustatud implementeerida otsingut *full-text* otsinguna (Joonis 6), mis tähendab, et päringusõna otsitakse kõikides valitud väljades. See teeb otsingu lihtsamaks selles mõttes, et kasutaja võib sisestada sellist informatsiooni *Object*-i kohta, mis tal meeles on ja ei pea sobivat välja valima. *Full-text* otsing on Django raamistikku PostgreSQL andmebaasisüsteemiga töötav sisseehitatud funktsioon.



Joonis 6. *Full-text* otsingu näide. Punasega on esile tõstetud otsingusõna.

Full-text otsingu jaoks on vaja teha järgmist importi:

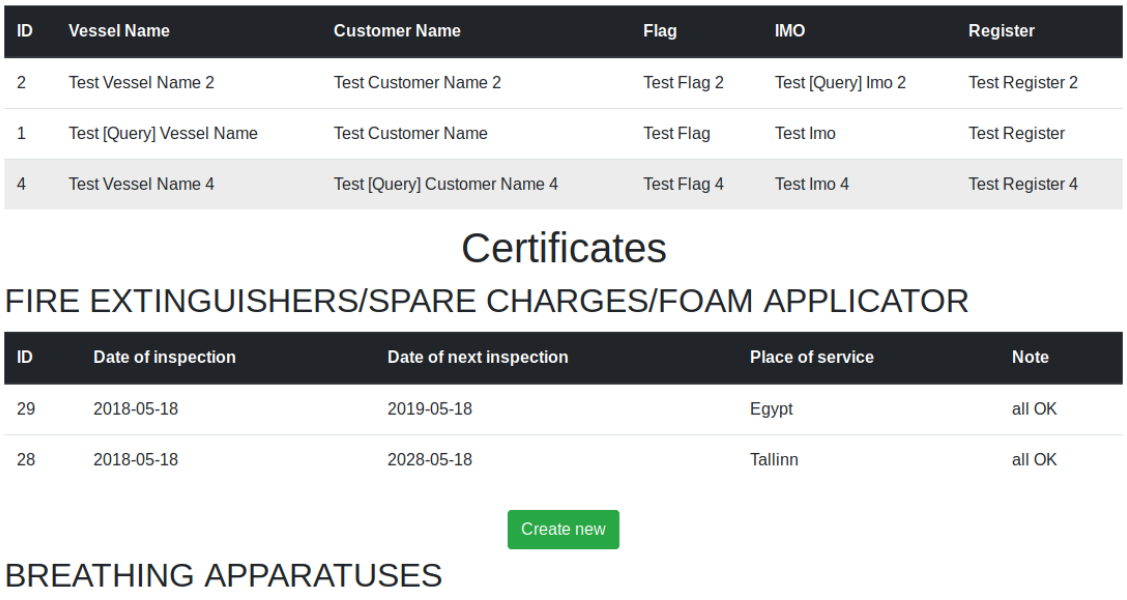
```
from django.contrib.postgres.search import SearchVector
```

Päring näeb välja niimoodi:

```
queryset =
self.get_queryset().annotate(search=SearchVector('vessel_name')
+ SearchVector('customer_name') + SearchVector('flag') +
SearchVector('imo') +
SearchVector('register'),).filter(search=user_query)
```

SearchVector on kõik väljad, kus hakatakse andmebaasisüsteemi tabelis otsima ning *user_query* on kasutaja päringusõna.

Tulemused kuvatakse tabelis ja sobivale reale klikkides ilmub allapoole veel üks React-i komponent *Certificates*, kus kuvatakse objektiga seotud sertifikaatide nimekirja (Joonis 7).



ID	Vessel Name	Customer Name	Flag	IMO	Register
2	Test Vessel Name 2	Test Customer Name 2	Test Flag 2	Test [Query] Imo 2	Test Register 2
1	Test [Query] Vessel Name	Test Customer Name	Test Flag	Test Imo	Test Register
4	Test Vessel Name 4	Test [Query] Customer Name 4	Test Flag 4	Test Imo 4	Test Register 4

Certificates

FIRE EXTINGUISHERS/SPARE CHARGES/FOAM APPLICATOR

ID	Date of inspection	Date of next inspection	Place of service	Note
29	2018-05-18	2019-05-18	Egypt	all OK
28	2018-05-18	2028-05-18	Tallinn	all OK

Create new

BREATHING APPARATUSES

Joonis 7. Valitud objektiga seotud sertifikaatide nimekiri.

Selleks, et pärida kõik objektiga seotud sertifikaadid, tuleb teha päringud kõikidesse sertifikaatide tabelitesse. Selle saavutamiseks kasutati Django REST raamistiku filtri *ObjectMultipleModelAPIView*. Kõigepealt tuli selle importida:

```
from drf_multiple_model.views import ObjectMultipleModelAPIView
```

Seejärel lisasime kõikide päringute vastused *querylist*-i ning tagasime selle:

```
class ListObjectCertificates(ObjectMultipleModelAPIView):

    def get_querylist(self):

        object_id = self.request.query_params.get('object_id')

        querylist = [

            {'queryset':
models.CertificateCategoryFE.objects.filter(object_id=object_id)
, 'serializer_class':
serializers.CertificateCategoryFESerializer},

            {'queryset':
models.CertificateCategoryBA.objects.filter(object_id=object_id)
, 'serializer_class':
serializers.CertificateCategoryBASerializer},

    ]

    return querylist
```

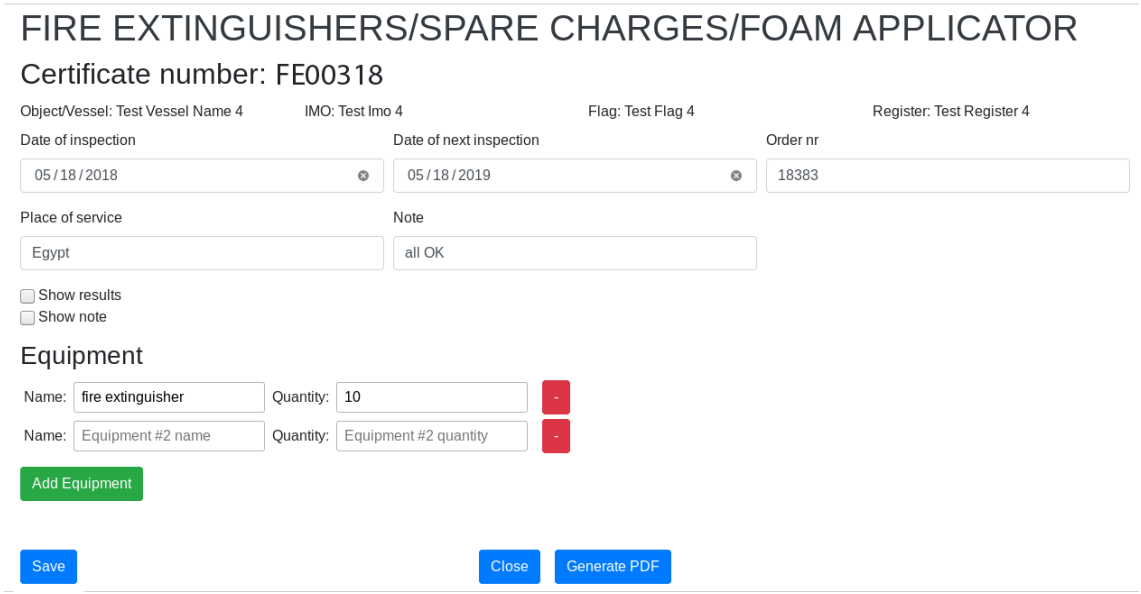
React raamistik põhineb komponentide süsteemil. Vajadusel komponente näidatakse/peidetakse. Meie süsteemi puhul oli loodud kolm komponenti: *Objects*, *Certificates* ja *CertificateForm*. Komponendi andmeid hoitakse selle muutujas *state*:

```
class Objects extends Component {

constructor(props) {

    super(props);

    this.state = { objects: [], certificates: {}, formValues:
{"query":""}, chosenObject: null, certificatesShown: false };

    }

}
```

Komponent *Objects* hoiab objektide andmeid ning vajadusel edastab objekti andmeid komponendile *Certificates* ja kuvab seda komponenti.

```
render() {

     …

     return (

          …

               {this.state.certificatesShown &&
this.state.chosenObject ? <Certificates
certificates={this.state.certificates}
chosenObject={this.state.chosenObject} /> : null}

     );

}
```

Komponent *Certificates* hoiab valitud objektiga (*Object*) seotud andmeid ning vajadusel annab neid andmeid edasi komponendile *CertificateForm*, mis vastutab sertifikaadi detailse info kuvamise ja info muutmise eest (Joonis 8). Juhul kui komponendile *CertificateForm* ei edastata sertifikaadi andmeid, vormi salvestamisel luuakse PUT-meetodiga uus sertifikaat.



Joonis 8. Sertifikaadi kuvamine ja muutmine.

5 Järgmised sammud

Mõned vajalikud funktsionaalsused ei olnud projekti raames implementeeritud, kuid need tuleb kindlasti lisada enne kliendile üleandmist:

- PDF genereerimine

  Salvestatud sertifikaatidest peab olema võimalik genereerida PDF faili, mida saadetakse tellijale paberkandljal posti teel või elektrooniliselt e-posti aadressile. Selle ülesande lahendamiseks hakkan kasutama ReportLab [21] või selle taolist vahendit.

- Kasutaja teavitamine sertifikaatide kehtivuse lõppemisel

  Lisada esilehele komponent, kus kuvatakse kõikide objektide varsti lõppevate sertifikaatide nimekirja.

- Turvalisus

  Veebirakenduse puhul on väga oluline tagada selle turvalisus. Üheks oluliseks aspektiks on vormide valideerimine. Enne salvestamist andmebaasi tuleb valideerida kõik kasutaja poolt sisestavad andmed. Nad peavad vastama nii tüübi kui ka pikkuse poolest. Lisaks on vaja töödelda sümboleid, mille kasutamine võib põhjustada XSS rünnakut.

- Autentifikatsioon

  Arendatav veebirakendus on mõeldud kasutamiseks kindlale sihtrühmale, kes omavad ligipääsu õigusi. Autentifikatsiooniks hakatakse kasutama populaaset auth0.com teenust.

- Testid

  Selleks, et uute funktsionaalsuste lisamine ei tooks vigu sisse, tuleb kirjutada regressioonitestid. Samuti plaanitakse kirjutada kasutajaliidese testid, et testida tüüpilised stsenaariumid (objektide lisamine, sertifikaatide loomine jne).

- Kasutajaliidese täiendamine

Kuna arendatavat veebirakendust hakatakse kasutama erinevatel seadmetel, on vaja kohandada seda erineva suurusega ekraanidega. Lisaks peab läbi viima testimine kasutajate poolt eesmärgiga selgitada välja ning lahendada kasutajamugavuse probleeme.

- Vigade töötlemine

  Vigade puhul peab kasutaja saama adekvaatse tagaside tekkinud vea kohta.

- Veebimajutuse valik ja projekti paigaldus serverile.

Kokkuvõte

Käesoleva lõputöö raames olid analüüsitud kaasaegsed tarkvaraarenduse vahendid, mis oleksid tellitud süsteemi arendamiseks sobivad. Oli tehtud populaarsemate programmeerimiskeelte, *front-end* ja *back-end* raamistike detailne võrdlus nii tabelite kui ka detailse analüüsi kujul.

Tulemusena said valitud programmeerimiskeelt Python, *back-end* raamistik Django ning *front-end* raamistik React. Nende vahendite abil sai loodud veebirakenduse baasprototüüp, mis laseb hallata objekte ning nendega seotud sertifikaate. Lisaks sellele on detailselt kirjeldatud veebirakenduse arendusprotsess ning on välja toodud täiendused, mida peab tulevikus implementeerima selleks, et rakenduse saaks kasutusele võtta.

Kasutatud kirjandus

[1] „DB-Engines Ranking,“ Mai 2018. [Võrgumaterjal]. Available: https://db-engines.com/en/ranking. [Kasutatud 11 Mai 2018].

[2] „Best Desktop Database Software,“ [Võrgumaterjal]. Available: https://www.g2crowd.com/categories/desktop-database. [Kasutatud 11 Mai 2018].

[3] F. Djikic, K. Brown ja K. Yerino, „What are the pros and cons of using Microsoft Access?,“ 12 Märts 2017. [Võrgumaterjal]. Available: https://www.quora.com/What-are-the-pros-and-cons-of-using-Microsoft-Access. [Kasutatud 20 Mai 2018].

[4] W. Zhou, L. Li, M. Luo ja W. Chou, „REST API Design Patterns for SDN Northbound API,“ 2014. [Võrgumaterjal]. Available: https://sci-hub.tw/http://ieeexplore.ieee.org/abstract/document/6844664/. [Kasutatud 13 Mai 2018].

[5] Z. U. Haq, G. F. Khan ja T. Hussain, „A Comprehensive analysis of XML and JSON web,“ [Võrgumaterjal]. Available: http://www.inase.org/library/2015/vienna/bypaper/CSSCC/CSSCC-14.pdf. [Kasutatud 13 Mai 2018].

[6] T. software, „TIOBE Index for May 2018,“ Mai 2018. [Võrgumaterjal]. Available: https://www.tiobe.com/tiobe-index/. [Kasutatud 11 Mai 2018].

[7] R. Kenneth, „Which is best for web development, Java, Python, Ruby, or PHP?,“ 13 Juuli 2016. [Võrgumaterjal]. Available: https://www.quora.com/Which-is-best-for-web-development-Java-Python-Ruby-or-PHP. [Kasutatud 11 Mai 2018].

[8] N. Bubniuk, „The 7 Best Java Frameworks for 2016,“ 22 Oktoober 2016. [Võrgumaterjal]. Available: https://dzone.com/articles/7-best-java-frameworks-for-2016. [Kasutatud 11 Mai 2018].

[9] „PHP Sadness,“ 05 Jaanuar 2017. [Võrgumaterjal]. Available: http://phpsadness.com/. [Kasutatud 11 Mai 2018].

[10] Y. J. Shah, „Advantages and Disadvantages of Ruby on Rails,“ 5 August 2016. [Võrgumaterjal]. Available: https://www.businesslabs.org/learning-rooms/advantages-disadvantages-ruby-on-rails/. [Kasutatud 11 Mai 2018].

[11] P. Hegde, „What are advantages and disadvantages of Python?,“ 16 Märts 2018. [Võrgumaterjal]. Available: https://www.quora.com/What-are-advantages-and-disadvantages-of-Python. [Kasutatud 11 Mai 2018].

[12] P. Krill, „A developer's guide to the pros and cons of Python,“ 24 Veebruar 2015. [Võrgumaterjal]. Available: https://www.infoworld.com/article/2887974/application-development/a-developer-s-guide-to-the-pro-s-and-con-s-of-python.html. [Kasutatud 12 Mai 2018].

[13] „Python,“ 04 Mai 2018. [Võrgumaterjal]. Available: https://hotframeworks.com/languages/python. [Kasutatud 11 Mai 2018].

[14] „Измерение производительности популярных python web-фреймворков,“ 15 Juuni 2015. [Võrgumaterjal]. Available: http://klen.github.io/python-web-benchmarks.html. [Kasutatud 12 Mai 2018].

[15] S. Krause, „Results for js web frameworks benchmark – round 7,“ 20 November 2017. [Võrgumaterjal]. Available: http://www.stefankrause.net/js-frameworks-benchmark7/table.html. [Kasutatud 14 Mai 2018].

[16] A. Pano, D. Graziotin ja P. Abrahamsson, „Factors and actors leading to the adoption of a JavaScript,“ 9 Märts 2018. [Võrgumaterjal]. Available: https://arxiv.org/pdf/1605.04303.pdf. [Kasutatud 12 Mai 2018].

[17] J. Schae, „A Real-World Comparison of Front-End Frameworks with Benchmarks,“ 31 Märts 2018. [Võrgumaterjal]. Available: https://medium.freecodecamp.org/a-real-world-comparison-of-front-end-frameworks-with-benchmarks-2018-update-e5760fb4a962. [Kasutatud 12 Mai 2018].

[18] „Front-end JavaScript frameworks,“ [Võrgumaterjal]. Available: https://github.com/collections/front-end-javascript-frameworks. [Kasutatud 13 Mai 2018].

[19] „Cookiecutter Django is a framework for jumpstarting production-ready Django and ReactJS projects quickly.,“ [Võrgumaterjal]. Available: https://github.com/chopdgd/cookiecutter-django-reactjs. [Kasutatud 14 Mai 2018].

[20] „Promise based HTTP client for the browser and node.js,“ [Võrgumaterjal]. Available: https://github.com/axios/axios. [Kasutatud 20 05 2018].

[21] „Outputting PDFs with Django,“ [Võrgumaterjal]. Available: https://docs.djangoproject.com/en/2.0/howto/outputting-pdf/. [Kasutatud 20 Mai 2018].