Tartu Ülikool

Loodus- ja täppisteaduste valdkond

Tehnoloogiainstituut

Kristjan Laugesaar

ELEKTROONIKAINSENEERIA ETTEVÕTTE TARBEKS TOOTE ELUTSÜKLI HALDUSSÜSTEEMI LOOMINE

Bakalaureusetöö (12 EAP)

Arvutitehnika eriala

Juhendajad:

Renno Raudmäe

Mihkel Heidelberg

Tartu 2023

Resümee/Abstract

ELEKTROONIKAINSENEERIA ETTEVÕTTE TARBEKS TOOTE ELUTSÜKLI HALDUSSÜSTEEMI LOOMINE

Hetkel antud elektroonikainseneeria ettevõttel puudub lahendus ülevaate saamiseks valmis ja arenduskäigus olevatest toodetest. Toodetest hea ülevaate puudumisel aeglustub tootearendus ning tellijatele toe pakkumine on raskendatud. Bakalaureusetöö eesmärk on luua lahendus ettevõttele, et hõlbustada manussüsteemidele toe pakkumist pärast tootmist ning võimaldada seadmetest parema ülevaate saamine. Töö raames seati ülesse PostgresSQL andmebaas koos REST API liidesega, veebirakendus JavaScriptis ja Pythoni moodul seadmete tootmises registreerimiseks. Andmebaas ning veebirakendus seati ülesse DigitalOceani Droplet virtuaalmasinas.

CERCS: T120 Süsteemitehnoloogia, arvutitehnoloogia; T130 Tootmistehnoloogia.

Märksõnad: manussüsteemid, tootmine, andmebaasid, veebirakendus.

THE DEVELOPMENT OF A PRODUCT LIFECYCLE MANAGEMENT SYSTEM FOR A ELECTRONICS ENGINEERING COMPANY

Currently an electronics engineering company is missing a solution to keep track of devices produced and devices in the prototyping stage. Having no overview of a product slows down the development process and makes it harder to provide support for the client. The aim of this bachelor’s thesis is to create a support platform for embedded systems after manufacturing and to help give a better overview of the products. The result consisted of a PostgreSQL database interfaced with a REST API written in Python using the FastAPI library, a webpage written in JavaScript using the SolidJS framework and a Python library for registering devices in production. The database and webpage were set up in a DigitalOcean Droplet virtual machine.

CERCS: T120 Systems engineering, Computer technology; T130 Production technology.

Keywords: embedded systems, manufacturing, databases, web application.

Sisukord

Jooniste ja tabelite loetelu

Lühendid, konstandid, mõisted

ACID – *Atomicity; consistency; isolation; durability*, atomaarsus; järjepidevus; isoleeritus; vastupidavus. Omadused andmebaasil.

API – *application programming interface*, rakendusliides.

CSS – *Cascading style sheets*, veebilehe kujundamise keel.

DNS – *Domain name system*, domeeni nime süsteem.

HTML – *HyperText markup language* ,hüperteksti märgistuskeel.

HTTPS – *HyperText transfer protocol secure*, turvaline hüperteksti edastusprotokoll.

IMEI – *International mobile equipment identity*, rahvusvaheline mobiilseadme identiteet.

IoT – *Internet of things*, asjade internet, nutistu.

JSM – *Jira service management*, teenuse haldamise tarkvara.

JWT – *JSON web token*, jsoni formaadis veebi tooken.

MAC – *Media access control*, meediumipöörduse juhtimise aadress.

OOP – *Object oriented programming*, objekt orienteeritud programmeerimine.

PLM – *Product lifecycle management*, toote elutsükli haldamine.

QR – *Quick read*, ruutkood.

REST – *Representational state transfer*, tarkvara arhitektuuriline stiil.

SQL – *Structured query language*, struktueeritud päringute keel.

TCP – *Transport control protocol*, edastusohje protokoll.

TLS – *Transport layer security*, transpordikihi turbeprotokoll.

TOML – *Toms obvious minimal language*, konfiguratsioonifailide failivorming.

URL – *Uniform resource locator*, üldine ressursi asukohamääraja.

WSGI – *Web server gateway interface*, veebi serveri värava liidese spetsifikatsioon Pythoni programmeerimiskeele jaoks.

1 Sissejuhatus

Kiire IoT areng on revolutsioneerinud tehnoloogiaga kokkupuutumist igapäeva elus, võimaldades kergesti jälgida ja kontrollida süsteeme kauguselt. Nutistu on kasutust leidnud erinevates sektorites nagu meditsiin, autotööstus, targad majad ja ühistransport ning Cisco interneti raporti kohaselt on pooled võrguseadmetest maailmas just IoT seadmed. Asjade interneti areng ei näita märke peatumisest ning mida rohkem IoT seadmeid toodetakse seda olulisem on pidada arvet nende üle. [1]

Toote elutsükli haldamise ehk PLM-i (ing.k. *Product Lifecycle Management)* protsess hõlmab tervet toote elutsüklit: disaini, tootmist, kasutajatuge ning ka toote amortiseerumist. PLM-i põhimõtted on eksisteerinud juba kaheksakümnendatest aga alles hiljutised arengud tarkvara tööriistaahelates on võimaldanud seda tõeliselt efektiivselt implementeerida. [2]

1.1 Probleemi tutvustus

Antud ettevõttes genereeritakse prototüüpimise protsessi käigus tihti mitu riistvara iteratsiooni, millest igaüks viib mitme prototüübi loomiseni. Selleks, et leida nende prototüüpide seast mõni seade, mida saab kasutada testimiseks või doonoriks teise seadme jaoks on ajamahukas. Tootes prototüüpe mitme kliendi jaoks, kaob suure kvanditeedi tõttu tootest ülevaade.

Agiilse prototüüpimise üks oluline osa on suhtlus tellijaga. Kui tellija tahab prototüübi arendamise käigus täiendada nõudeid, siis peab tal olema võimalus anda neid arendajale läbipaistvalt ja selgelt. Kui see protsess toimub mitme osapoolega meilivestluse teel siis tekib tulemusena infomüra, kust on tähtsat infot keeruline leida. Selle tagajärjel aeglustub tellija jaoks toote arendamine.

1.2 Töö eesmärk ja ülevaade

Bakalaureuse töö eesmärk on luua ning võtta kasutusele lahendus, mis võimaldab seadmete ajalugu jälgida prototüüpimise, tootmise ning kasutuses olemise jooksul. Ajaloo jälgimine peaks hoidma kasutaja interaktsioonid võimalikult minimaalsed ning eemaldama arenduse käigus tehtavaid operatsioone. Niimoodi vähendatakse korduvaid tegevusi ja hoitakse kokku ajalisi kulusid toodete arendamisel. Lisaks on eesmärkideks kvaliteedikontrolli hõlbustamine ning toote seisukorrast ülevaate andmise võimaldamine.

Töö teoreetilises osas antakse ülevaade probleemist ning analüüsitakse turul olevaid lahendusi. Arutatakse, kas on tarvilik välja töötada uus lahendus või on võimalik integreerida olemasolevaid platvorme. Samuti seatakse nõuded, mida lahendus peab täitma. Töö praktilises osas valmistatakse veebirakendus, andmebaas ja rakise moodul, ning selgitatakse tööriistade valikut.

1.3 Nõuded

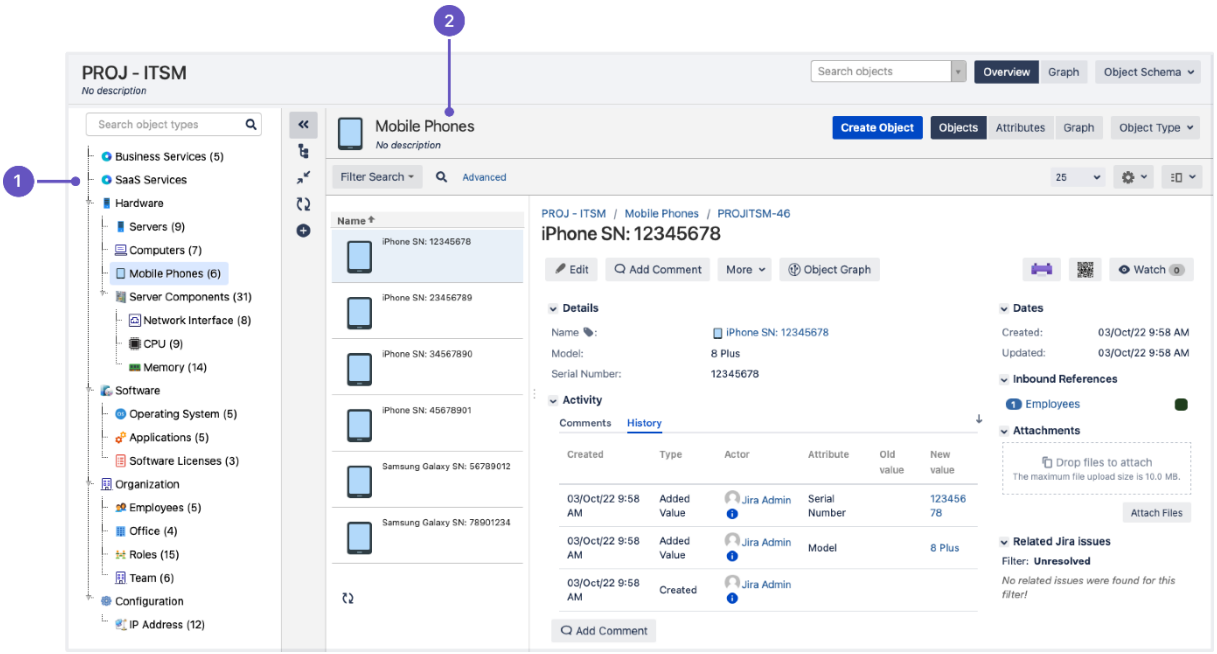
Lahendus peab võimaldama:

1. Seadmete jälgimist;
2. Seadme kohta tootmisandmete hoiustamist;
3. Seadmega logide sidumist;
4. Seadmega piltide sidumist;
5. Seadme logide ja piltide pärimist;
6. Minimaalse vaevaga kasutust;
7. Kasutust telefonist ja arvutist;
8. Ligipääsu autentimist;
9. Andmesuhtluse krüpteerimist;
10. Lihtsat tootmisprotsessidega integreerimist;
11. Vähemalt 100 000 seadme hoiustamist.

2 Ülevaade probleemist

2.1 Olemas olevad lahendused

Atlassian on firma, mis pakub klientidele produktiivsus tarkvara erivorme [3]. Üheks selliseks on Jira Service Management (edaspidi JSM), mis pakub erinevaid malle teenuste ülesse seadmiseks ja haldamiseks.[4] JSM-ile on võimalik lisada võimekust nimega Assets, mis võimaldab pidada arvet firma varade kohta.



Joonis 1. Jira Service Management Assets rakendus

Nagu kõikide Atlassiani tarkvaradega on ka Assets väga funktsionaalse kasutajaliidesega. Jooniselt 1. on võimalik näha kasutajaliidese võimekusi, milline näeb välja kirje andmebaasis ja milline on andmestruktuur. Igat toodetud IoT seadet on võimalik hoida ühe *asset´*ina ning selle info välju on kerge muuta ja vaadata. Lisaks on olemas ka REST API, et teha päringuid seadmete lisamiseks ja nende kohta andmete pärimiseks. See võimaldab seadmete lisamist testrakisest ja ka Jira poolt genereeritud QR koodide saamist. Seadmetega saab ka siduda Jira pileteid, mida saavad arendajad kasutada tekkinud probleemide parandamiseks.

Assets pole mõeldud suuremahulise tootmisinfo hoidmiseks vaid just firma varade ülevaateks nagu näiteks arvutid, telefonid jm. firma varad. Hetkel on varade piiranguks 10000, mis on piiravaks suurema andmemahu hoiustamisel[5]. Kasutajaliides on küll paljude funktsionaalsustega, kuid samas on ka liiga võimekas. See muudab suuremate andmehulkade

juures kasutuse kohmakaks. Sellest järeldub, et süsteem on mitteskaleeruv suurtemate andmehulkade juures.

Asset Panda on sarnast teenust pakkuv platvorm, mis on suunatud firmadele, et jälgida varasid veebikeskkonnas. See teenus pakub mobiilirakendust, vöötkoodi lugemist, platvormi ümberkujundamist vastavalt vajadustele jm võimekusi. Puuduvad andmed , et antud tarkvara oleks kasutatav tootmisandmete hoiustamise jaoks. Kuna hind kasvab vastavalt lisatud seadmete arvule, siis ei ole see jätkusuutlik valik [6].

2.2 Olemasolevate lahenduste puudujäägid

Välja toodud lahenduste põhiprobleemiks on mitteskaleeruvus. Jira Service Management Assets ja Asset Panda on disainitud väiksema mahuliste kasutusjuhtude jaoks. Ulatuslikuma tootmise ja prototüüpimise korral jõuame kiiresti andmemahtudeni, mis ületavad mainitud süsteemide võimekusi. Lisaks on väljatoodud süsteemid liiga võimekad, sest nad on disainitud laialdasele kasutajaskonnale ja need lisavõimekused teevad kasutuse kohmakaks. Otsitav lahendus peaks olema vastupidine ning soodustama minimaalsete sammudega kasutamist.

Valmis oleva lahenduse kasutusele võtmisel võib selle ajaline maht kujuneda sama suureks, kui enda süsteemi tegemisel, aga tulemusena jäädakse seotuks valitud keskkonnaga. Samuti ei ole soovitud alternatiiv mõne partner tehase poolt pakutavav keskkond kuna see ei pruugi pakkuda muid võimekusi peale tootmisinfo hoidmise ning lisaks seob süsteemi kolmanda osapoolega. Sobiva lahenduse mitte leidmise tõttu loodi oma süsteem.

3 Metoodika

Töö praktiline osa koosneb süsteemi arendamisest tarkvaras ning serveri ülesseadmisest. Süsteemi diagramm on kujutatud joonisel 2. Süsteemil on kolm osa:

- Andmebaas ja selle liides;
- Pythoni moodul;
- Veebirakendus.

3.1 Funktsionaalsed nõuded süsteemi osistele.

Andmebaas peab olema:

- Skaleeruv;
- ACID nõuetele vastav;
- Avatud lähtekoodiga;
- Relatsiooniline.

Andmebaasi liides peab võimaldama:

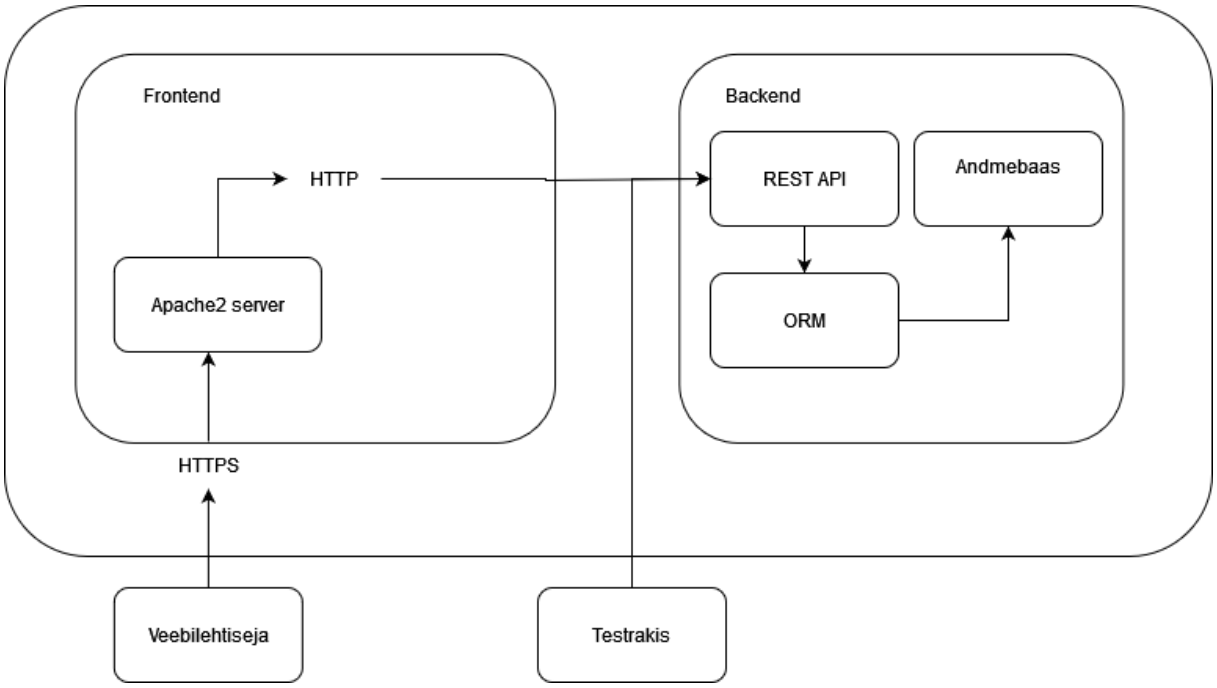
- Kirjete lisamist, lugemist ja kustutamist;
- Operatsioonde tegemist, nii kasutajaliidesest kui ka rakise moodulist;
- Teistest osistest kerget lahtisidumist;
- Uute funktsionaalsuste kerget lisamist;
- Kasutaja autentimist.

Testrakise moodul peab võimaldama järgnevaid funktsionaalsusi:

- Testprogrammist seadmete lisamist andmebaasi;
- Andmebaasist seadme ID pärimist;
- Lihtsat integreerimist testkoodi;
- Moodulina paigaldamist.

Kasutajaliides peab võimaldama:

- Seadme andmete kiiret pärimist QR koodi skanneriga ja otsingulahtriga;
- Seadmetele logide lisamist;
- Piltide lisamist;
- Seadme logisõnumite vaatamist;
- Seadme piltide vaatamist;
- Ligipääsu läbi veebilehitseja;
- Minimaalsete toimingutega kasutust.



Joonis 2. Süsteemi diagramm

3.2 Andmebaas

Relatsioonilises andmebaasis on andmed organiseeritud tabelitesse ning relatsioonid ehk seosed võimaldavad tabeleid omavahel siduda. Tabeli ridadeks on kirjed ning kirjete eristamiseks ja seoste loomiseks on kirjetel unikaalsed ID-d. Kirjetel on omadused ja need on organiseeritud veergudesse. Relatsioonid võimaldavad luua suuri andmebaase, mis on hallatavad ja tõhusad. Lisaks vähendavad relatsioonid andmete kordumist andmebaasis, säästes ruumi. [7]

PostgreSQL on üks kõige populaarsematest avatud lähtekoodiga relatsioonilistest andmebaasidest, mida on aktiivselt arendatud 35 aastat. Töös kasutati 2023 aasta veebruaris välja lastud PostgreSQL versiooni 15.2. [8] [9]

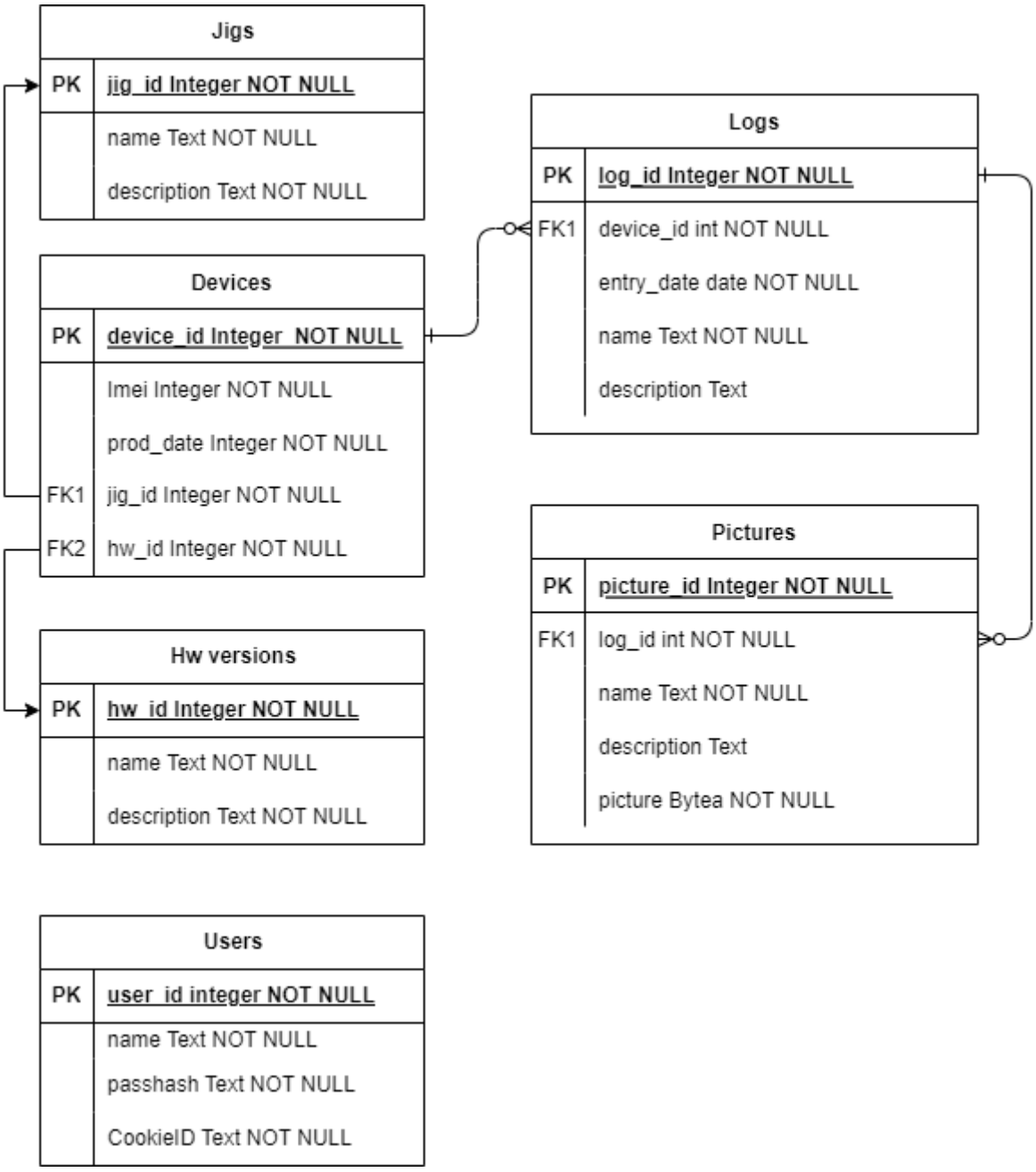
3.2.1 Andmebaasi skeem

Andmebaasi skeemi tuumaks on seadmete tabel. Seadme tabelis on järgnevad väljad: seadme kood, riistvara tuvastuskood, tootmiskuupäev, riistvara versiooni ID ja testrakise ID. Seadme ID on andmebaasi poolt genereeritud võti, millega saab seadet andmebaasist leida.

Riistvara versioonide tabelis on kirjas toodetud ja tootmises olevad riistvara versioonid. Iga seade on seoses ühe riistvara kirjega. Riistvara tabelis on kaks veergu: riistvara nimi ja kirjeldus. Testrakiste tabelis on kirjas kõik andmebaasi külge ühendunud rakised. Igal rakisel on nimi ja kirjeldus. Rakised ja riistvarad on eraldi tabelites, sest need väärtused hakkavad seadme tabelis korduma. Lisaks on riistvara ja rakised seadmest eraldi vaadeldavad objektid.

Samuti peab olema võimalik ühe seadme kohta lisada mitu logisõnumit. Logisõnumite tabelis on järgnevad veerud: logi nimi, kirjeldus, kuupäev, seadme id ning logi võti, millest viimast kasutatakse logisõnumitega piltide sidumise jaoks. Piltide tabelis on veerud pildi nime ja kirjelduse jaoks ning viide logisõnumi kirjele. Pildi fail andmebaasis on *Bytea* tüübina.

Viimaseks on kasutajate tabel, kus on kirjas kasutaja nimed ja nende paroolide räsid. Kasutajate paroolid on räsitud kasutades PostgreSQL laiendust pgcrypto. Kasutajate tabelisse kasutaja lisamisel räsib pgcrypto parooli crypt funktsiooniga ja hoiustab seda tabelis [10]. Loodud skeemi on näha joonisel 3.



Joonis 3. Relatsiooni skeem

3.3 Taustaprogramm

3.3.1 REST API

Ehitatav süsteem nõuab, et andmebaas oleks kättesaadav nii kasutajaliidesele kui ka testrakistele. Kasutajaliides peab võimaldama andmebaasi logide ja piltide lisamist ning testrakisele teenust pakkuv moodul peab võimaldama seadmete lisamist andmebaasi. Üks lahendus on luua kahe otspunkti vahel veebisokkel. Veebisokkel võimaldab luua kiire dupleks ühenduse, kuid andmebaasiga suhtlemise jaoks ei ole ilmtingimata vaja sellist kiiret ühendust ning nendega tulevad kaasa ka olekud serveris. [11]

REST API (ingl. k. *Representful state transfer application programming interface*) on liides, mis lubab ligipääsu ressurssidele kasutades hüperteksti edastusprotokolli ehk HTTP (ingl. k. *HyperText pransfer protocol*) päringuid. REST API eripärasus tuleneb sellest, et päringu tegemiseks pole vaja eelnevat olekut ja sissetulev päring teenindatakse ära ainult seal sisalduva informatsiooniga. Lisaks on REST API lahendus horisontaalselt skaleeruv, ehk uusi otspunkte on lihtne lisada. [12]

Kuna töö autoril on kõige rohkem kogemust just Pythoni programmeerimiskeelega ja Python on suure toetajaskonnaga, paljude teekidega ning võimekas keel, siis osutus praktilises osas kasutatavaks keeleks just Python. Pythoni teek Django on populaarne raamistik veebirakenduse ülesse seadmiseks ning võimaldab ka REST API seadistamist. Tuntud Django kasutajad on näiteks Instagram ja Pinterest.[13] FastAPI on Pythoni teek, mis võimaldab üles seada REST API-t. FastAPI on veidi uuem kui Django ning kasutab hiljutisemaid Pythoni arenguid. FastAPI pole nii võimekas kui Django, näiteks ei ole FastAPI teegil sisse ehitatud ORM funktsionaalsust. FastAPI on siiski eesmärgi täitmiseks sama võimekas ja väiksemahuline valik ning seetõttu valiti REST API seadistamiseks FastAPI teek versioon 0.95.0 .[14]

3.3.2 Andmebaasi liides

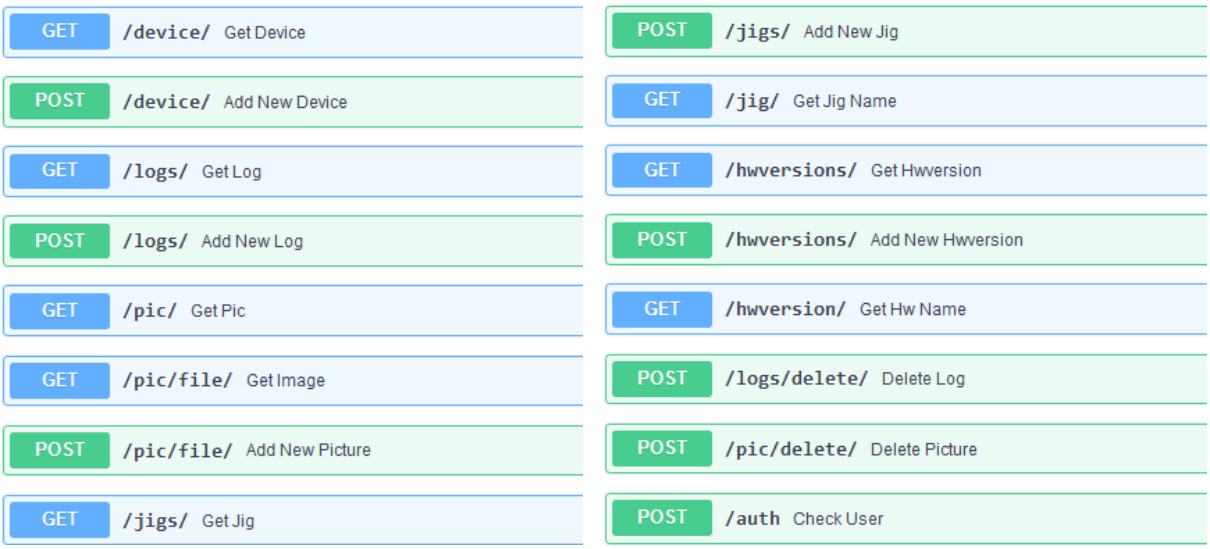
PostgreSQL andmebaasi liidestamiseks Pythoniga saab kasutada andmebaasi adapterit, millega saab teha struktueeritud päringu keele ehk SQL (ingl. k. *Structured Query Language*) päringuid otse Pythoni koodis. Üks adapter, mida PostgreSQL pakub on libpq. See on C programmeerimiskeeles implementeeritud adapter, mis lubab teha päringuid otse koodist[15]. Kuna töös kasutatav REST API raamistik kasutab Pythoni programmeerimiskeelt, siis on mõistlik valida ka andmebaasi liides, mis on samas keeles implementeeritud. Üks selline adapter on psycopg2. Selle Pythoni teegiga on võimalik luua funktsioonid iga nõutud

operatsiooni jaoks, kuid enda SQL päringute koostamine ja nendega liidese loomine on mahukas protsess. Mõistlikum ja ajasäästlikum on kasutada objekti relatsioonide kaardistamise ehk ORM (ingl. k. *Object Relational Mapping*) teeki nagu SQLAlchemy, mis madalamal tasemel kasutab juba psycopg2 adapterit [16]. Töös kasutati SQLAlchemy versiooni 2.0.12.

ORM lahendus võimaldab kasutada andmebaasi andmeid kui objekte ning samuti koodi puhtamana hoida. Selle võimaldamiseks peab defineerima objektid ja objekti väljad, mis vastavad andmebaasis olevatele tabelitele ning nende veergudele. Tehes päringuid tagastab ORM andmetega täidetud valitud objekti või objetkid vastavalt päringu väärtusele. Selline abstraktsioonikiht teeb andmebaasist päringute lisamise kiiremaks ning lühendab just süsteemiga seotud koodi hulka.

3.3.3 Lõpp-punktid

Lõpp-punktid on asukohad, kus asuvad REST API poolt pakutavad ressursid. Pythoni teek FastApi võimaldab kergelt defineerida lõpp-punkte ning siduda nendega funktsioone. REST API lõpp-punktid, mis süsteemis loodi on välja toodud joonisel 4.



Joonis 4. Nimekiri REST API lõpp-punktidest

Iga lõpp-punktil on defineeritud aadress, ehk URL (ingl. k. *Uniform Resource Locator*). Kui klient tahab ligipääsu mingile ressursile siis peab klient tegema HTTP päringu vastavale aadressile. Näiteks tehes HTTP *get* päringu aadressile *http://{serveri_ip}:{RESTapi}/device/* käivitub funktsioon *get_device()*.

```
 96 @app.get("/device/")
 97 def get_device(
 98     skip: int = 0,
 99     limit: int = 10,
100     imei: int | None = None,
101     deviceid: int | None = None,
102     db: Session = Depends(get_db),
103     hmac: int = Depends(check_hmac),
104 ):
105     if deviceid:
106         device = crud.get_device(db, deviceid)
107     elif imei:
108         device = crud.get_device_by_imei(db, imei)
109     else:
110         device = crud.get_devices(db, skip, limit)
111     if device is None:
112         raise HTTPException(status_code=404)
113     return device
114
```

Joonis 5. REST API lõpp-punkti näide

Aadressile on võimalik lisada päringu parameetreid lisades kujul *...device/?deviceid=1&skip=0*. Päringu parameetrid võimaldavad kaasa anda lisa argumente, millega kujundatakse päring andmebaasist. Joonisel 5 olevas funktsioonis, parameetrite defineerimata jätmisel, tagastab päring nimekirja seadmetest. See nimekiri on vaikimisi piiratud esimese kümne kirjega. Vastavalt seatud parameetritele kutsutakse välja erinev andmebaasi päring.

FastAPI võimaldab seada sõltuvusi lõpp-punktidele, mis käivitatakse päringu tegemisel enne kui hakatakse lõpp-punktile vastavat funktsiooni täitma. Näiteks sõltuvus *get_db* teeb ühenduse andmebaasiga ning annab selle sessiooni edasi muutujas *db*. Sessiooni kasutatakse andmebaasist päringute tegemiseks ning lõpuks, kui päring on tehtud, pannakse sessioon kinni.

Kuna REST API on avalik, siis on vajalik päringute autentimine. Üks võimalus, et autentida päringute usaldusväärsust on räsitud sõnumi autentimise koodiga ehk HMAC (ingl. k. *Hashed message authentication code*) allkirjastamine. HMAC põhineb sümmeetrilisel krüptograafial. Sümmeetriline krüptograafia on andmete loetamatuks muutmine jagatud saladuse abil. Kuna

sümmeetrilise krüptograafia puhul hoitakse krüpteerimise võtit ka kliendi poolel, on selle saladusena hoidmine keeruline. [17]

HMAC-i asemel kasutab REST API teenus JSON veebi tokenit, ehk JWT (ingl. k. *JSON web token*), mis allkirjastatakse serveri poolt ning antakse kasutajale, siis kui on sisestatud korrektsed sisselogimise andmed. JWT on idee poolest sarnane HMAC allkirjastamise meetodile. Allkiri koostatatkse JWT sisust ja krüpteeritakse võtmega, kuid JWT tokenit genereerib ja kontrollib ainult server. Pahatahtlikul osapoolel pole võimalik genereerida JWT tokeneid ise ning ei ole võimalik ka kasutada vanu tookeneid, sest nendel on sätitud kasutusaeg 20 minutit loomisest. Sissetulevat tookenit ei aksepteerita, kui see on vanem kui 20 minutit. Järelikult on JWT tokeniga ligipääsu andmine hea lahendus REST API-le, kuna JWT kohta ei hoita serveris informatsiooni, mis on REST stiilile kohane. JWT tookeni lisamiseks REST API koodi kasutati pyjwt teegi versiooni 2.6.0. [18], [19]

Päringutega kaasnevate andmete korrektsuse tuvastamiseks on võetud kasutusele andmeklassid, mis ebakorrektse sisendi puhul annavad vea enne andmebaasi sisestamist. Andmeklassid implementeeriti Pythoni Pydantic teeki kasutades. Andmeklasside defineerimine aitab luua ühtset arusaama, mis kujul peaks lõpp-punkti andmeid saatma.[20]

REST API jooksutati kasutades Gunicorni versiooni 20.1.0, mis on liides veebiserveri ja Pythoni programmi vahel. Gunicorn edastab veebiserverisse sissetulevad päringud edasi Pythoni koodile, mis võimaldab FastAPI lõpp-punkte käivitada. [21] Valminud API lähtekood on üleval Gitlab koodrepositooriumis. [22]

3.4 Pythoni moodul

3.4.1 Mooduli disain

Testrakis on tööriist, mis võimaldab testida, kas seade on töökorras ja teha viimased sammud tootmises. Mooduli eesmärk on integreerida testrakisesse andmebaasiga ühenduse loomine ning sinna seadmekirjete lisamine. Testrakise jaoks peab mooduli kasutus olema võimalikult lihtne. Moodul esineb testrakise koodis kahe objektina. Nendeks on andmebaasi ühenduse objekt ning lisatava seadme objekt.

3.4.2 Implementatsioon

Mooduli kirjutamisel lähtuti puhtakoodi kirjutamise suunistest. Kood kirjutati võimalikult ennast kirjeldavalt ja funktsioonid hoiti mõistlikult lühikesed. Moodulis loodi kaks klassi, mida rakis saab kasutama hakata.

Seadme klass on lihtne andmeklass, mille loomisel võetakse sisenditeks loetud seadme riistvaraline tuvastuskood ja riistvara versiooni ID.

Andmebaasi klass teeb kahte asja: initsialiseerib ühenduse andmebaasiga, et saada JWT tooken ning lisab seadme objekt andmebaasi. Initsialiseerimisel loeb klass sätted argumendina kaasa antud *toml* failist, kus on kirjas serveri aadress, autentimise väärtused, rakise info ja riistvara versiooni info [23]. Kuna JWT tooken aegub 20 minuti jooksul pärast selle andmist siis kontrollib rakis tookeni aegumist igakord, kui saadab uue päringu. Kui tooken on aegunud, küsib moodul uue.

```
1  [server]
2  url = "https://siilipesa.ee/api"
3  name = "Testing"
4  passw = "Testing"
5
6  [jig]
7  name = "Manual"
8  description = "This device has been added manually"
9
10 [device]
11 hw_ver = "rev1.0"
12
13
```

Joonis 6. Näide rakise mooduli sätete failist

3.4.3 Kasutus

Loodud koodist tehti moodul, mida on võimalik installeerida Pythoni paketi paigaldamise tööriistaga Pip. Paketi integreerimise automatiseerimine hoiab rakiste koodihoidlat puhtamana.[24]

Joonisel 4 on toodud näide, kuidas loodud moodulit kasutada. On näha, et testrakise töö jaoks ebaolulised mooduli funktsionaalsused on ära abstraheeritud ning ei ole esil kõrgema taseme koodis. Loodud mooduli koodi hoitakse Gitlabi koodirepositooriumis. [25]

```
18 database_conn = main.Database(logger)
19 database_conn.read_config_file("example.toml")
20 database_conn.connect()
21
22 while(True):
23     logger.info("Enter imei")
24     imei = input()
25
26     new_dev = main.Device(imei, 3)
27     n = database_conn.register_device(new_dev)
28
29     logger.info(f"got {n}")
30     f = open("label.zpl", "w")
31     f.write(f"^XA^FO125,30^BQN,2,8^FDMM,A{n}^FS^XZ")
32     f.close()
33     os.system("lp -d zd421 -o raw label.zpl")
```

Joonis 7. Näide rakise mooduli kasutusest

3.5 Veebirakendus

Kasutajaliidese koodi kirjutamise jaoks valiti JavaScript, sest enamjaolt igal kasutusel oleval veebilehitsejal on tugi JavaScripti koodi jooksutamiseks. Samuti on JavaScript väga laialdaselt kasutuses ning tugi keele jaoks jätkub veel aastateks. Lisaks on JavaScriptile tehtud suurel hulkal raamistike. Need annavad JavaScriptile lisa võimekusi nagu reaktiivsus. Reaktiivsus veebilehes on võimekus uuendada veebilehe osi ilma lehte värskendamata. [26]

Töö jaoks valiti JavaScripti raamistik SolidJS, mis on mõeldud kasutajaliideste kirjutamiseks. Kasutati versiooni 1.7.0. SolidJS lähenemine reaktiivsusele on komponentide funktsionidesse panemine, mida käivitatakse sellel juhul, kui reageerivas skoobis olevaid sõltuvusi muudetakse. Sõltuvustest peab järge SolidJS ise. [27]

Kõige põhilisem reageeriv primitiiv on CreateSignal, mis suudab jälgida ühe väärtuse muutumist. Funktsioon võtab argumendiks algse väärtuse ning tagastab kaks funktsiooni: väärtuse ligipääsemise funktsioon ning väärtuse muutmisfunktsioon. Kutsudes välja ligipääsemisfunktsiooni skoobis, mis võimaldab signaalide jälgimist, tekitatakse sõltuvust sellest signaalist. Kui signaali väärtust muudetakse, jooksutatakse kõik funktsioonid uuesti, mis sõltuvad sellest signaalist. SolidJS on võtnud inspiratsiooni ühest kõige populaarsemast JavaScripti raamistikust, Reactist. [28]

SolidJS kasutab JavaScripti laiendust JSX. See laiendus võimaldab tagastada HTML elemente funktsioonide tagastusväärtustena. Sellega, et kasutajaliidese visuaalne disain ning selle tagaolev reaktiivne loogika on tihedamini samas failis seotud, lihtsustab JSX kasutajaliidese disainimist ja loogika kirjutamist. Kombineerides JSX koos SolidJS reaktiivsusega on võimalik lisada sõltuvusi JSX elementide sisse, mida uuendatakse kui signaalide väärtused muutuvad. [27], [29]

Veebirakenduse arendamisel kasutati arenduskeskkonda Vite. See keskkond võimaldas panna ülesse rakenduse jaoks serveri, millele oli võimalik lokaalsel masinal ligi pääseda. Vite suudab näidata veebirakenduses tehtud muudatusi reaalajas. Ilma viiteta koodi muudatuste nägemine kiirendab oluliselt kasutajaliidese arendamise protsessi. Lisaks on Vite võimekus teha loodud veebirakendusest *bundle*, mida saab veebiserver kliendile teenindada [30]

3.5.1 Veebilehel liikumine

Algne vaade veebirakenduse avamisel on sisselogimise ekraan, sest andmebaasi ligipääsu küsijat on vaja autentida. Pärast kasutajanime ja parooli sisestamist kontrollitakse, kas andmebaasis on vastav kombinatsioon olemas ning seejärel genereeritakse ressursi ligipääsu küsijale token.

REST APIga suhtlus toimub üle HTTP päringute. Päringute tegemiseks JavaScriptis on võimalik kasutada Fetch API-t või XMLHttpRequest teeki. Nende kahe teegi erinevus on, et Fetch API tagastab lubaduse, mille sisu peab ootama asünkroonselt. XMLHttpRequesti puhul kutsutakse välja päringu vastuse kohale jõudmise korral funktsioon. SolidJSi poolt on olemas primitiiv CreateResource, mis on mõeldud just asünkroonsetele funktsioonidele reaktiivsuse lisamiseks. See primitiiv võtab sisenditeks käivitatava funktsiooni ning signaali, mille muutumise tulemusena kutsutakse välja funktsioon. CreateResource poolt tagastatav informatsioon on samuti signaal, mida on võimalik reaktiivselt kuvada JSX elementides ning kasutada teistes reaktsioonides.[31]

Veebisait peab olema ainult lubatud kasutajate poolt ligipääsetav. Tavaliselt toimub sisselogimise tuvastus kasutades küpsiseid, mis on serverist allalaetud failid, mida veebilehitseja hoiab mälus ning edastab HTTP päringutega. Server saab päringutega kaasnevate küpsistega kinnitada, kas päringut tegeval kliendil on õigus ressursidele ligipääseda ning samuti ka hoida meeles tegevusi mida klient teinud on. REST arhitektuuristiili jälgides, ei tohi olla serveris olekuid ja seetõttu kasutatakse küpsist, et hoiustada REST API poolt allkirjastatud tookenit, mis autendib tookeniga koos saadetuid päringuid REST API-le. [32]

Sisselogimise õnnestumisel annab SolidRouter ligipääsu otsingu vaatele. SolidRouter võimaldab veebisaidil simuleerida veebilehtede vahel liikumist ilma uusi HTML faile laadimata. Virtuaalsed lehed on defineeritud *Route* elementidena ning need viitavad funktsioonidele, mis tagastavad JSX elemente. Selle veebilehe ruuteri eesmärk on olla abstraktsiooni kiht JSX komponentide jaoks, muutes ruudi defineerimises viidatud komponendi veebilehitseja jaoks eraldi veebileheks. See vähendab tehtavaid päringuid veebiserverisse ja võimaldab teha rohkem veebirakenduse loogikat kliendi seadmes [33]. Näide SolidJS ruutimise elementide kasutusest on välja toodud joonisel 8.

```
function App() {
    return (
        <div class="grid grid-cols-1 items-center m-auto">
            <header class="text-white my-4 p-2 text-xl flex items-center gap-4">
                <A href="/search">Search</A>

            </header>
            <div >
                <Routes >
                    <Route path="/" component={RouteGuard}>
                        <Route path="/search" component={Search} />
                    </Route>
                    <Route path="/login" component={Login} />
                </Routes>
            </div>
        </div >
    );
};
```

Joonis 8. Näide ruutimise komponentide kasutusest

3.4.2 Seadme skanneerimine

Seadme skanneermise ja otsingu vaatele pääseb ligi asukohas „*/search*“. JavaScriptis QR koodi dekodeerimiseks on vaja seadme kaamerale ligipääsu. Seda on võimalik teha läbi dokumendi objekti mudeli ehk DOM (ingl. k. *Document Object Model*). DOM on veebi dokumendi andmete esitluse viis ning programmeerimisliides. See võimaldab keeltel nagu JavaScript, muuta veebilehe elemente ning ligipääseda ressurssidele. DOM-i liides *navigator* võimaldab pärida veebilehele ligipääseva veebilehitseja kohta andmeid. Kasutades *navigatori* omadust getMediaDevices on võimalik kasutada kaameraid ja mikrofone JavaScripti koodis. [34], [35]

Kaamera pilti edastatakse JavaScripti teegile jsqr, mis võimaldab dekodeerida piltidel olevaid QR koode. Kasutades SolidJS funktsiooni SetTimeout saab sättida jsqr funktsiooni perioodilist välja kutsumist ning kui kaamerapildile satub QR kood, siis jsqr dekodeerib selle ning tagastab väärtuse signaali. Tänu signaalile on võimalik kohene reaktsioon koodi skänneerimisele, mille käigus tehakse andmebaasist päring vastavalt skanneeritud seadme numbrile. [36] Lisaks skaneerimisele on võimalik ka seadme number sisestada manuaalselt, millele reageeritakse samal viisil.

Komponentide kuvamine läbi signaalide on võimalik SolidJS elemendi *show* abil. Tingimuse täitumise puhul on võimalik veebilehel reaktiivselt kuvada JSX elemente. Sättides tingimuseks andmebaasist päritud info kohale jõudmise, tekitab SolidJS *show* elemendi ja andmete vahele seose. Kui andmed muutuvad siis käivitub *show* elemendi tingimuse kontroll ning selle olemas

olul on võimalik kuvada seadme ilma veateadet andmata, sest info on lehekülje laadimise hetkel defineerimata. Kui päritud andmeid on palju siis on võimalik neid kuvada JSX väljundis kasutades elementi *for*. Selle elemendi eripära on see, et tsükli elemendid võimaldavad üksteisest sõltumatut reageerimist signaalidele. [27]

3.4.3 Kujundus

CSS (ingl. k. *Cascading style sheets*) on märgistuskeel, millega kirjeldatakse veebilehe välimust. Kõigepealt peab defineerima CSS-is reegleid, mida jälgitakse HTML komponentide kuvamiseks veebilehel. Kujunduse kirjeldused on eraldi failis ning reegel algab kujundatava HTML elemendi valikuga. Reegli sisuks on omadused, ning nendele omistatavad väärtused. Näiteks saab öelda, et HTML komponendi <p> tekst peab olema punast värvi ja kui veebilehitseja hakkab seda tüüpi komponente kujutama, siis kõikides seda tüüpi komponentides on tekst punane. Selline lähenemine rakendab disaini reeglit kõikidel <p> elementidel, kuid mõnikord on vajalik spetsiifilise elemendi muutmine. Selle jaoks saab kasutada HTML elemendi atribuuti *class*, millele saab anda CSS keeles märgitud ühe või mitme reegli nimesid. [37], [38]

Veebilehtede kujundamine on väga mahukas protsess ja nõuab tegijalt loovust. Aja säästmiseks kasutati töös raamistiku Tailwind CSS, mis võimaldab kujundada HTML elemente *html* failis kasutades selleks eeldefineeritud CSS klasse. HTML atribuuti *class* kasutus koos Tailwindi utiliseerimisega kiirendab veebilehe arenduse kulgemist, jättes vahele CSS reeglite kirjutamise protsessi. [39]

Et kujundus protsess oleks veel kiirem võeti kasutusele Flowbite. Flowbite on kasutajaliidese elementide kogum, mis on loodud kasutadesTailwind CSSi ning on vabavarana kasutatav. See võimaldab luua kasutajaliidest, mis näeb hea välja ja mida on mugav kasutada. [40] Valminud veebirakenduse lähtekood on üleval gitlabi koodirepositooriumis. [41]

3.4.4 Staatilise veebilehe teenindamine

Vite keskkonnas rakenduse ehitamise tulemusena tekkis failide kogum, mida on võimalik teenindada staatilise veebilehena serveris. Staatiline veebileht saadetakse kliendi veebilehitsejasse täpselt samal kujul nagu seda serveris hoitakse.[42] Staatilise veebilehe teenindamiseks võeti kasutusele Apache2 HTTP serveri versioon 2.4.57.[43]

Apache võimaldab teenindada klientidele veebilehti kasutades HTTP-d. Veebilehe kuvamise jaoks defineeritud Apache sätted on välja toodud joonisel 9. Veebileht on ligipääsetav läbi

edastusohje protokolli, ehk TCP (ingl. k. *Transport control protocol*) pordi 443, mis on mõeldud HTTPS suhtluse edastamiseks. HTTPS on turvaline viis kuidas luua ühendus veebiserveri ning kliendi vahel. Suhtlus üle protokolli on krüpteeritud ning tänu sellele on raskendatud pahatahtlikud tegevused ründajate poolt. Suhtlusekrüpteerimise võimaldamiseks on sätetes antud *virtuaalhost*ile TLS sertifikaat ning privaatne võti. Lisaks on ülesse seatud ka päringute edasi suunamine, mis suunab päringud sihtkohta „/api/“ edasi pordil 8000 jooksvale REST API-le. Apache server jooksutati teenusepakkuja Digital Oceani virtuaalmasinas. Selle teenusepakkuja poolt pakutavad virtuaalmasinad ehk Dropletid on tulevikus laiendatavad, nii mahult, protsessori kiiruse poolest kui ka infoedastuskiiruse poolest. [44] Virtuaalmasina operatsiooni süsteemiks on Ubuntu 20.04.

```
<VirtualHost *:443>
        ServerName siilipesa.ee
        ServerAlias *.siilipesa.ee

        ProxyPreserveHost on
        ProxyPass /api http://localhost:8000
        ProxyPassReverse /api http://localhost:8000

        DocumentRoot /var/www/frontend
        <Directory /var/www/frontend>
          Options -Indexes +FollowSymLinks
          AllowOverride All
          Require all granted
        </Directory>

        include /etc/letsencrypt/options-ssl-apache.conf
        sslcertificatefile [redacted]
        sslcertificatekeyfi[redacted]
</VirtualHost>

<VirtualHost *:80>
        ServerName siilipesa.ee

        RewriteEngine on
        RewriteCond %{SERVER_NAME} =siilipesa.ee [OR]
        RewriteCond %{SERVER_NAME} =*.siilipesa.ee
        RewriteRule ^ https://%{SERVER_NAME}%{REQUEST_URI} [END,NE,R=permanent]
</VirtualHost>
```

Joonis 9. Apache veebilehe konfiguratsiooni fail

3.4.5 Domeen ja sertifikaadid

Selleks, et oleks mugavam ligi pääseda veebirakendusele ja luua HTTPS sertifikaat, osteti teenusepakkujalt veebimajutuse domeen nimega „siilipesa.ee“. Domeen on sõne, millega on võimalik identifitseerida interneti ressursse ning lihtsustab nendele ligipääsemist [45]. Sihtpunkti IP aadressi asemel on võimalik otsida sihtkoha nime järgi. Selleks tuleb siduda IP domeeni nimega ning see peab olema esindatud domeeni nime süsteemi serveris ehk DNS-i serveris (ingl. k. *Domain Name System*) [46]. Veebimajutus pakub domeeni ostmist ning selle konfigureerimist. [47]

Domeeni nime omandamisel seati üles SSL sertifikaatide lisamine. SSL sertifikaat on allkirjastatud tunnistus, mis tõestab sertifikaadi ning selle privaatvõtme omaniku autentsust ja võimaldab osapoolte vahel krüpteeritud andmesuhtlust. Sertifikaat antakse domeenile *certificate authority* poolt, mille juursertifikaadist algab sertifikaadi ahel. Sertifikaat koosneb domeeni nimest, sertifikaadi andja nimest, andja digitaalsest allkirjast, loomise kuupäevast, aegumiskuupäevast ja avalikust võtmest. Ühendudes veebilehele kontrollitakse, kas sertifikaat veel kehtib ning kas sertifikaadi ahelas on mõni veebilehitseja poolt usaldatud *certificate authority* olemas. [48]

Veebirakenduse sertifikaadid allkirjastati Let’s Encrypt *certificate authority* poolt. „Let’s Encrypt“ on tasuta teenus mida pakub Interneti turvalisuse uurimisgrupp ehk ISRG (ingl. k. *Internet Security Research Group*). Nende poolt antavad sertifikaadid aeguvad kolm kuud pärast allkirjastamist ning pärast aegumist on vaja teha uus.[49] Sertifikaatide allkirjastamine seati ülesse Certbot tööriista abil, mis võimaldab automaatselt uuendada sertifikaate ning selle paigaldamine on triviaalne. [50]

4 Tulemused ja järeldus

Lõputöös disanitud prototüübi arendamine oli edukas ning vastas püstitatud nõuetele. Tulemusena arendati välja süsteem, millega on võimalik pärida infot reaalmaailmas olevate seadmtete kohta ning millele on võimalik ligi pääseda brauseriga. Andmebaasi kasutamiseks on võimalik sisestada läbi kasutajaliidese tuvastusandmed. Kuvatõmmis sisselogimise kuvast on leitav lisas 1.

Süsteemi testimiseks lisati seadmeid andmebaasi kasutades Pythoni programmi, mis kasutas töö raames loodud moodulit. Programm võtab sisendiks seadme IMEI , MAC või seeria koodi ning seejärel saadab need andmed üle REST API andmebaasi. Andmebaasist saadud kood prinditi välja kleepsuna ning lisati seadmetele. Näide andmebaasi lisatud seadme andmetest on toodud välja lisas 2. Ühe seadmete tabeli kirje suuruseks tuli 53 baiti ning see teeb miljoni sellise kirje puhul kokku 53 megabaiti. Võib järeldada, et andmebaasi maht ei ole piirav faktor.

Süsteem anti kasutada kolleegidele, et valideerida lahenduse kasutatavust. Neil paluti minna veebilehele ning proovida sisse logida rakendusse. Seejärel anti neile eelnevalt andmebaasis sisestatud seadmed ja paluti neilt seade andmebaasist pärida, kasutades selles seadmel olevat QR koodi. Seadme sisse skanneerimisel avanes vaade, kust oli näha seadme andmeid, logisid ning logi sisestamise lahtreid. Selle tulemus on välja toodud lisas 3.

Töö tulemusena valminud süsteemi on võimalik integreerida tootmisesse minevate seadmetega ning tänu töös kasutusele võetud tarkvara tõttu on edasised süsteemiarendused võimalikud. Kasutajaliides annab hetkel ülevaate ühest seadmest, kuid plaanis on lisada juurde funktsionaalsusi, mis võimaldavad anda ülevaadet mitme seadme kohta ning teha nendega keerulisemaid operatsioone.

Kokkuvõte

Bakalaureuse töös antakse ülevaade turul olevatest lahendustest tootmises ja prototüüpimises ning käidi üle nende positiivsed ja negatiivsed omadused. Turul olevate lahenduste puudujääkide abil püstitati nõuded süsteemile.

Loodi PostgreSQL andmebaas, mis on mõeldud seadme eluea, prototüüpimise ja tootmise tarbeks ning liidestati Pythonis implemeteeritud REST API-ga. Apache päringute edasissuunamise võimekusega ning JWT tokenitega loodi andmebaasiga turvalise suhtluse võimalus.

Rakistest seadmete lisamiseks loodi Pythoni moodul, mis teeb HTTP päringuid REST API poole. Moodul on paigaldatav kasutades Pythoni paketihaldustarkvara Pip.

Kujundati kasutajaliides kasutades JavaScripti raamistiku SolidJS. Loodud kasutajaliides pakub kasutajale ligipääsu andmebaasiga suhtlemiseks, et näha seadme ajalugu ja lisada uusi andmeid. Liidese kasutatavust testiti kolleegide abil.

Tänuavaldused

Soovin tänu avaldada:

Oma juhendajatele, kes aitasid ning suunasid mind bakalaureusetöö valmimisel.

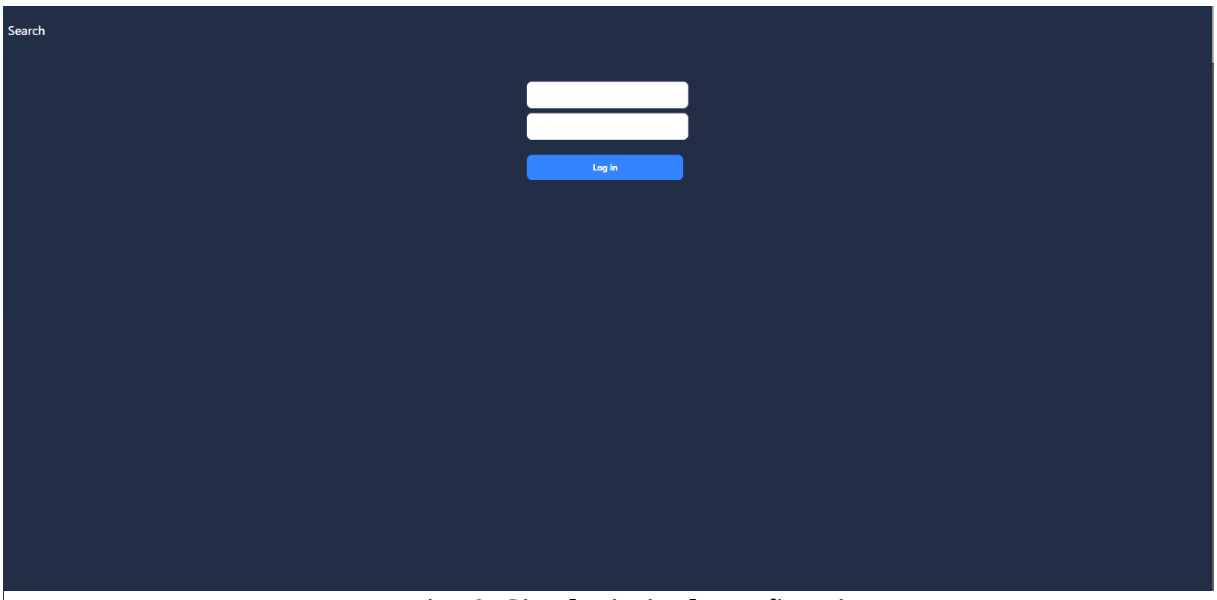
Kirjandus

[1] „Cisco Annual Internet Report (2018–2023) White Paper“. Vaadatud: 19. mai 2023. [Online]. Available at: https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html
[2] „Product Lifecycle Management“. https://www.oracle.com/scm/product-lifecycle-management/what-is-plm/ (vaadatud 16. mai 2023).
[3] „Atlassian“. https://www.atlassian.com/ (vaadatud 17. mai 2023).
[4] „Jira Service Management“. https://www.atlassian.com/software/jira/service-management (vaadatud 17. mai 2023).
[5] „Jira Assets Limit“. Vaadatud: 17. mai 2023. [Online]. Available at: https://jira.atlassian.com/browse/JSDCLOUD-8556
[6] „Asset panda“. 17. mai 2023. [Online]. Available at: https://www.assetpanda.com/
[7] Oracle, „What is a Relational Database (RDBMS)?“ https://www.oracle.com/database/what-is-a-relational-database/ (vaadatud 20. mai 2023).
[8] „PostgreSQL“. Vaadatud: 10. mai 2023. [Online]. Available at: https://www.postgresql.org/
[9] „DB-engines“. Vaadatud: 17. mai 2023. [Online]. Available at: https://db-engines.com/en/ranking
[10] „pgcrypto extension“. Vaadatud: 17. mai 2023. [Online]. Available at: https://www.postgresql.org/docs/current/pgcrypto.html
[11] „The WebSocket API (WebSockets)“. [Online]. Available at: https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API
[12] „REST API“, Vaadatud: 9. mai 2023. [Online]. Available at: https://www.ibm.com/topics/rest-apis
[13] „Django“. Vaadatud: 10. mai 2023. [Online]. Available at: https://www.djangoproject.com/
[14] „FastAPI“. 9. mai 2023. Vaadatud: 9. mai 2023. [Online]. Available at: https://fastapi.tiangolo.com/
[15] „C teek libpq“. Vaadatud: 17. mai 2023. [Online]. Available at: https://www.postgresql.org/docs/9.5/libpq.html
[16] „SQLAlchemy“. Vaadatud: 10. mai 2023. [Online]. Available at: https://www.sqlalchemy.org/
[17] D. H. Krawczyk, M. Bellare, ja R. Canetti, „HMAC: Keyed-Hashing for Message Authentication“, nr 2104. Request for Comments. RFC Editor, veebruar 1997. Vaadatud: 9. mai 2023. [Online]. Available at: https://www.rfc-editor.org/info/rfc2104
[18] „pyjwt“. Vaadatud: 18. mai 2023. [Online]. Available at: https://pyjwt.readthedocs.io/en/stable/
[19] „Json Web Token“. 17. mai 2023. [Online]. Available at: https://jwt.io/
[20] „Pydantic“. Vaadatud: 10. mai 2023. [Online]. Available at: https://docs.pydantic.dev/latest/
[21] „Gunicorn“. Vaadatud: 18. mai 2023. [Online]. Available at: https://gunicorn.org/
[22] „REST API kood“. [Online]. Available at: https://gitlab.com/KristjanLa/andmebaas_loputoo
[23] „TOML“. Vaadatud: 10. mai 2023. [Online]. Available at: https://toml.io/en/
[24] „pip“. Vaadatud: 18. mai 2023. [Online]. Available at: https://pypi.org/project/pip/

[25] „Rakise mooduli lähtekood“. [Online]. Available at: https://gitlab.com/KristjanLa/testjigadapterlib
[26] „Javascript“. Vaadatud: 9. mai 2023. [Online]. Available at: https://developer.mozilla.org/en-US/docs/Web/JavaScript
[27] „SolidJS“. Vaadatud: 18. mai 2023. [Online]. Available at: https://www.solidjs.com/docs/
[28] „SolidJS“. Vaadatud: 9. mai 2023. [Online]. Available at: https://github.com/solidjs/solid
[29] „JSX“. https://legacy.reactjs.org/docs/introducing-jsx.html
[30] „Vite“. Vaadatud: 9. mai 2023. [Online]. Available at: https://vitejs.dev/
[31] „Fetch API“. https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API/Using_Fetch (vaadatud 18. mai 2023).
[32] „Cookies“. Vaadatud: 18. mai 2023. [Online]. Available at: https://developer.mozilla.org/en-US/docs/Web/HTTP/Cookies
[33] „SolidRouter“. Vaadatud: 18. mai 2023. [Online]. Available at: https://github.com/solidjs/solid-router
[34] „DOM“. https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model/Introduction (vaadatud 18. mai 2023).
[35] „Navigator“. https://developer.mozilla.org/en-US/docs/Web/API/Navigator (vaadatud 18. mai 2023).
[36] „jsqr“. Vaadatud: 18. mai 2023. [Online]. Available at: https://github.com/cozmo/jsQR
[37] „CSS“. https://developer.mozilla.org/en-US/docs/Learn/CSS/First_steps/What_is_CSS (vaadatud 18. mai 2023).
[38] „Hypertext Transfer Protocol“. Vaadatud: 10. mai 2023. [Online]. Available at: https://developer.mozilla.org/en-US/docs/Web/HTTP
[39] „Tailwind CSS“. Vaadatud: 9. mai 2023. [Online]. Available at: https://tailwindcss.com/
[40] „Flowbite“. Vaadatud: 10. mai 2023. [Online]. Available at: https://flowbite.com/
[41] „Frontend lähtekood“. [Online]. Available at: https://gitlab.com/KristjanLa/frontend
[42] „Static web page“. Vaadatud: 18. mai 2023. [Online]. Available at: https://en.wikipedia.org/wiki/Static_web_page
[43] „Apache“. Vaadatud: 18. mai 2023. [Online]. Available at: https://httpd.apache.org/
[44] „Digital Ocean droplet“. Vaadatud: 18. mai 2023. [Online]. Available at: https://www.digitalocean.com/products/droplets
[45] „Domain“. Vaadatud: 18. mai 2023. [Online]. Available at: https://en.wikipedia.org/wiki/Domain_name
[46] „Domain name server“. [Online]. Available at: https://en.wikipedia.org/wiki/Domain_Name_System
[47] „Veebimajutus“. Vaadatud: 18. mai 2023. [Online]. Available at: https://www.veebimajutus.ee/
[48] „SSL certificates“. https://www.cloudflare.com/learning/ssl/what-is-an-ssl-certificate/
[49] „Let´s Encrypt“. https://letsencrypt.org/ (vaadatud 19. mai 2023).
[50] „Certbot“. Vaadatud: 18. mai 2023. [Online]. Available at: https://certbot.eff.org/

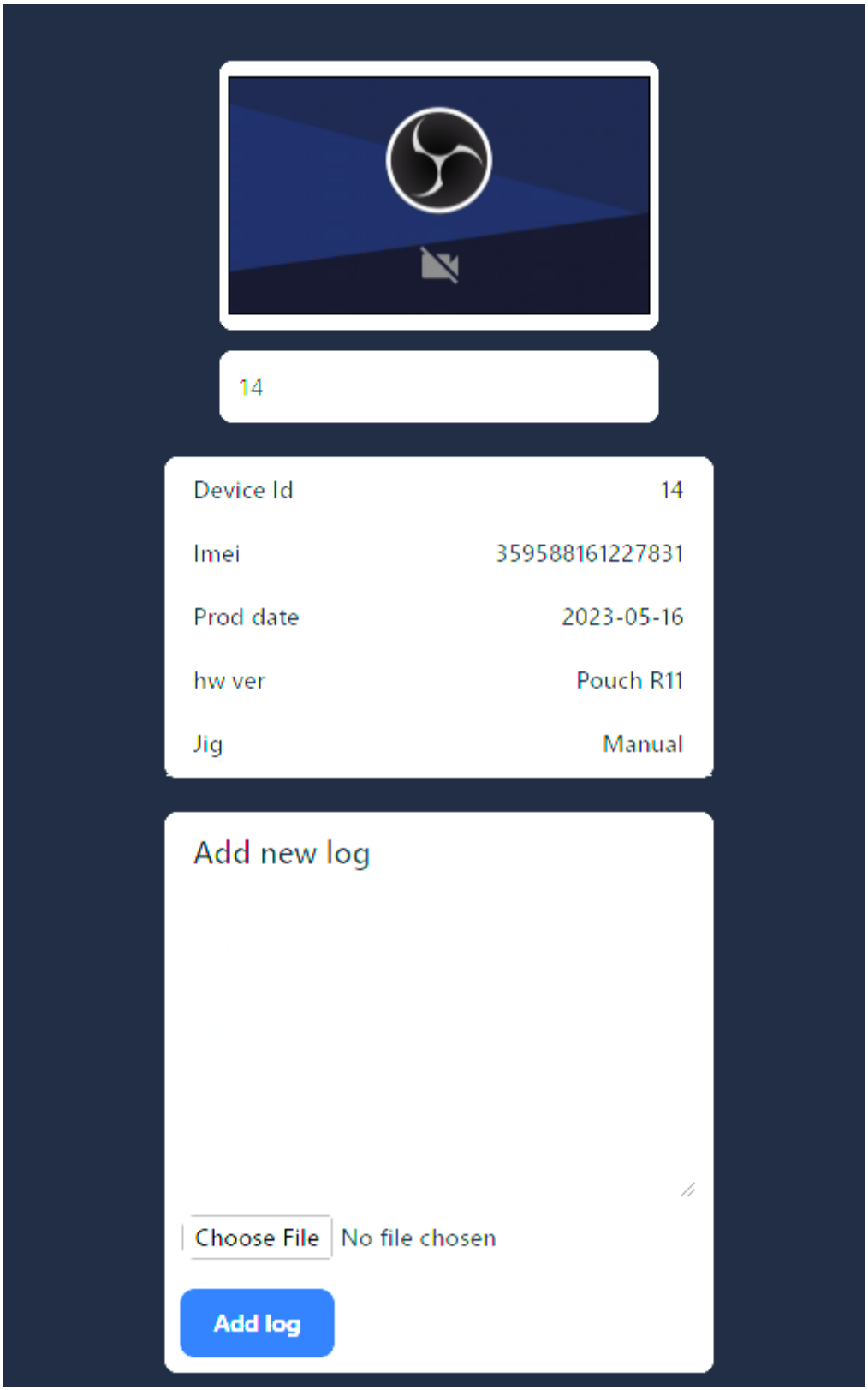
Lisad

Lisa 1 Veebirakenduse sisselogimise lehekülg



Joonis 10. Sisselogimise kuvatõmmis

Lisa 2 Seadme andmed veebirakenduses



Joonis 11. Seadme kuvatõmmis veebirakenduses

Lisa 3 Seadme logi veebirakenduses

Joonis 12. Logi kuvatõmmis veebirakenduses

Lihtlitsents lõputöö reprodutseerimiseks ja üldsusele kättesaadavaks tegemiseks

Mina, Kristjan Laugesaar,

1. annan Tartu Ülikoolile tasuta loa (lihtlitsentsi) minu loodud teose „Elektroonikainseneeria ettevõtte tarbeks toote elutsükli haldussüsteemi loomine“ , mille juhendajateks on Mihkel Heidelberg ja Renno Raudmäe, reprodutseerimiseks eesmärgiga seda säilitada, sealhulgas lisada digitaalarhiivi DSpace kuni autoriõiguse kehtivuse lõppemiseni.

2. Annan Tartu Ülikoolile loa teha punktis 1 nimetatud teos üldsusele kättesaadavaks Tartu Ülikooli veebikeskkonna, sealhulgas digitaalarhiivi DSpace kaudu Creative Commonsi litsentsiga CC BY NC ND 4.0, mis lubab autorile viidates teost reprodutseerida, levitada ja üldsusele suunata ning keelab luua tuletatud teost ja kasutada teost ärieesmärgil, kuni autoriõiguse kehtivuse lõppemiseni.

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

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

Kristjan Laugesaar

20.05.2023