TALLINNA TEHNIKAÜLIKOOL
INSENERITEADUSKOND
Virumaa kolledž

Rakenduse loomine nutimaja prototüübi jaoks

Application creation for smart house prototype

ARUKAD SÜSTEEMID JA RAKENDUSINFOTEHNOLOOGIA ÕPPEKAVA
LÕPUTÖÖ

Üliõpilane: Juri Kubinets
Üliõpilaskood: 207572EDTR
Juhendaja: Natalja Ivleva, lektor

Kohtla-Järve, 2024

AUTORIDEKLARATSIOON

Olen koostanud lõputöö iseseisvalt.

Lõputöö alusel ei ole varem kutse- või teaduskraadi või inseneridiplomit taotletud. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on viidatud.

LIHTLITSENTS LÕPUTÖÖ ÜLDSUSELE KÄTTESAADAVAKS TEGEMISEKS JA REPRODUTSEERIMISEKS[1]

Mina Juri Kubinets (sünnikuupäev: 15.08.1986)

1. Annan Tallinna Tehnikaülikoolile tasuta loa (lihtlitsentsi) enda loodud teose "Rakenduse loomine Arduino nutimaja prototüübi jaoks", mille juhendaja on Natalja Ivleva,
   1.1. reprodutseerimiseks säilitamise ja elektroonilise avaldamise eesmärgil, sealhulgas Tallinna Tehnikaülikooli raamatukogu digikogusse lisamise eesmärgil kuni autoriõiguse kehtivuse tähtaja lõppemiseni;
   1.2. üldsusele kättesaadavaks tegemiseks Tallinna Tehnikaülikooli veebikeskkonna kaudu, sealhulgas Tallinna Tehnikaülikooli raamatukogu digikogu kaudu kuni autoriõiguse kehtivuse tähtaja lõppemiseni.
2. Olen teadlik, et punktis 1 nimetatud õigused jäävad alles ka autorile.
3. Kinnitan, et lihtlitsentsi andmisega ei rikuta kolmandate isikute intellektuaalomandi ega isikuandmete kaitse seadusest ja teistest õigusaktidest tulenevaid õigusi.

---

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

SISUKORD

LÜHENDITE JA TÄHISTE LOETELU

ACID (Atomicity, Consistency, Isolation, Durability) – Andmebaaside tehingute neli põhiprintsiipi, mis tagavad andmete usaldusväärsuse ja õigsuse.

Angular – Veebirakenduste raamistik, loodud Google'i poolt, põhineb TypeScriptil.

API (Application Programming Interface) – Komplekt protseduuridest ja funktsioonidest, mis võimaldavad rakendustel suhelda teiste rakenduste või operatsioonisüsteemidega.

Arduino – Mikrokontrolleri põhine avatud lähtekoodiga elektroonikaplatvorm, mõeldud interaktiivsete projektide loomiseks.

Back-end – Tarkvara või rakenduse serveripoolne osa, tegeleb andmetöötluse, andmebaasi halduse ja muude taustaprotsessidega.

Blynk – Mobiilrakenduste arendusplatvorm IoT projektide jaoks.

Django – Pythoni-põhine kõrgetasemeline veebiraamistik, soodustab kiiret arendust ja puhta, pragmaatilise koodi disaini.

Front-end – Tarkvara või rakenduse kasutajapoolne osa, mõeldud otse suhtlemiseks kasutajaga.

IntelliJ Idea – Integreeritud arenduskeskkond (IDE) Java jaoks, loodud JetBrains'i poolt.

IoT (Internet of Things) – Kontseptsioon, mis viitab füüsiliste seadmete, sõidukite, koduseadmete ja muude esemete ühendamisele internetiga, et koguda ja jagada andmeid.

JSON (JavaScript Object Notation) – Kergekaaluline andmevahetusformaat, inimestele loetav ja masinate jaoks hõlpsasti parseeritav.

MQTT (Message Queuing Telemetry Transport) – Kergekaaluline sõnumiedastusprotokoll, kasutatakse eriti IoT lahendustes.

MySQL – Maailma populaarseim avatud lähtekoodiga relatsiooniline andmebaasisüsteem.

MVC (Model-View-Controller) – Tarkvaraarenduse mudel, jagab rakenduse kolmeks põhikomponendiks: mudel, vaade ja kontroller.

MVCC (Multi-Version Concurrency Control) – Andmebaaside lukustamismehhanism, võimaldab mitmetel kasutajatel andmeid korraga lugeda ja kirjutada ilma üksteist segamata.

PostgreSQL – Võimas avatud lähtekoodiga objekt-relatsiooniline andmebaasisüsteem.

Raspberry Pi – Ühesüsteemiline arvuti, loodud õpetamiseks ja väikesteks projektideks.

React – Facebooki poolt loodud JavaScripti raamistik kasutajaliideste loomiseks.

REST API (Representational State Transfer) – API, määratleb kogumi reegleid veebiteenuste loomisel.

SPA (Single Page Application) – Veebirakendus või veebileht, koosneb ühest lehest ja laadib kogu sisu dünaamiliselt.

Spring Boot – Java-põhine raamistik, loodud kiireks ja lihtsaks veebirakenduste arendamiseks.

Vue-chart.js – Vue.js komponent, mis võimaldab andmeid visualiseerida kasutades Chart.js raamatukogu.

Vue.js – Progressiivne JavaScripti raamistik kasutajaliideste loomiseks.

WebSocket – Protokoll, mis võimaldab avada kahesuunalise sideseansi veebiserveri ja kliendi vahel.

SISSEJUHATUS

Käesoleva lõputöö eesmärgiks on arendada rakendus Arduino-põhise prototüübi jaoks, vastates kaasaegse IoT (Internet of Things) arengu vajadustele. Prototüüp, mis varem töötas Blynk rakenduse abil, vajas oma lahendust pärast Blynki platvormilt eemaldamist. Seega tekkis vajadus luua isiklik rakendus, mis oleks kolmandate osapoolte platvormidest sõltumatu, tasuta kasutatav ja võimeline töötama võrguühenduseta režiimis.

Lõputöö esimene ülesanne oli Arduino koodi parandamine ja ajakohastamine, et tagada süsteemi stabiilsus ja funktsionaalsus. Teine oluline ülesanne keskendus andmete efektiivsele vahetusele prototüübi ja rakenduse vahel ning nende usaldusväärsele säilitamisele andmebaasis, kasutades MQTT andmeedastusprotokolli ja REST API-d. Arvestades rakenduse vajadust töötada võrguühenduseta ja olla kasutajatele tasuta, valiti lokaalseks andmebaasiks PostgreSQL.

Kolmas ülesanne hõlmas kasutajasõbraliku liidese väljatöötamist Vue.js raamistiku abil. Neljanda ülesandena oli andmete visualiseerimine. Viimasena lõi lõputöö autor konstruktori, mis võimaldab nutimaja automatiseerimist, lähtudes sensoritest saadud andmetest.

Integreerides kõik need komponendid üheks toimivaks tervikuks, kasutas autor Spring Boot'i raamistikku, mis võimaldab tõhusat andmetöötlust ja süsteemi komponentide koostööd.

Lõputöö demonstreerib, kuidas erinevad tehnoloogiad suudavad efektiivselt koostööd teha, luues optimeeritud ja kasutajasõbraliku süsteemi nutimaja prototüübi jaoks. Töö tõstab esile IoT tehnoloogiate tähtsust ja võimalusi, pakkudes uusi perspektiive nii teoreetilises kui ka praktilises plaanis.

Peatükkides käsitletakse prototüübi ülesehitust, valitud tehnoloogiate analüüsi, rakenduse arendusprotsessi ja lõpptulemuse funktsionaalsust. Lisad ja graafilised materjalid pakuvad täiendavat teavet ja visuaalseid näiteid töö käigus kasutatud lahendustest.

Võtmesõnad: Arduino, Spring Boot, Vue.js, nutimaja, bakalaureusetöö.

1. NUTIMAJA PROTOTÜÜP

TalTech Virumaa kolledži üliõpilased lõid Arduino põhjal praktilise töö raames IoT-süsteemide arendamisel nutimaja prototüübi. Lõputöös kasutatav prototüüp on reaalse maja makett ilma sisemiste ruumijaotusteta.

Kõik andurid on paigaldatud ülaossa. Nutimaja prototüübil on 11 erinevat andurit ja 4 täiturmehhanismi.

Andurid:

- heliandur;
- klaviatuuriandur;
- valgustusandur;
- tulekahjuandur;
- gaasiandur;
- veetaseme andur;
- mulla niiskuse andur;
- DHT11 (temperatuuri- ja niiskuse andur);
- DS18B20 (temperatuuriandur);
- BMP085 (temperatuuri- ja rõhuandur);
- ultraheli kaugusandur.

Täiturmehhanismid:

- pumba relee;
- piesoelektriline kõlar;
- ukse servoajam;
- lamp.

Põhifunktsioonid ja võimalused:

Nutimaja prototüüp pakub järgmisi funktsioone:

- anduritelt saadud andmete lugemine ja töötlemine;
- täiturmehhanismide reaalajas juhtimine;
- andmete kuvamine LCD-ekraanil;
- andmete saatmine kaugserverisse MQTT protokolli kasutades;

- käesoleva aja sünkroniseerimine NTP serveriga;
- seadmete automaatne ja käsitsi juhtimine;
- andmete logimine järgnevaks analüüsiks.

See nutimaja makett esindab optimaalset kombinatsiooni funktsionaalsusest, paindlikkusest ja visuaalsusest, muutes selle ideaalseks vahendiks IoT valdkonna rakenduste arendamiseks, testimiseks ja demonstreerimiseks.

1.1. Arduino

Arduino on avatud riist- ja tarkvaraplatvorm, mis on välja töötatud neile, kes on huvitatud interaktiivsete objektide või keskkondade loomisest. Arduino peamine eesmärk on pakkuda vahendeid kiireks ja lihtsaks prototüüpimiseks. [1]

1.1.1. Peamised Arduino plaadid

Arduino plaadid põhinevad Atmeli mikrokontrolleritel. Kõige populaarsemad neist on Arduino Uno, Arduino Nano, Arduino Mega ja Arduino Micro. Igal plaadil on oma omadused, näiteks erinev arv digitaalseid ja analoogporte või erinevad mälumahud. [2]

1.1.2. Arenduskeskkond (IDE)

Arduino IDE on spetsialiseeritud programmeerimiskeskkond, mis on mõeldud programmide (sketšide) kirjutamiseks ja laadimiseks Arduino plaatidele. See põhineb Processingi tarkvarakeskkonnal ja toetab keeli C ja C++. Arenduskeskkond on saadaval erinevatele operatsioonisüsteemidele, sealhulgas Windowsile, macOS-ile ja Linuxile. [3]

1.1.3. Andurid ja moodulid

Arduino suudab suhelda suure hulga anduritega, nagu temperatuuri, valguse, liikumise, rõhu ja paljude teiste anduritega. Tänu sellele saab Arduino abil luua mitmesuguseid projekte, alates lihtsatest temperatuuri indikaatoritest kuni keerukate robotite või nutimaja süsteemideni. [4]

Arduino pakub lihtsat ja kättesaadavat viisi mitmesuguste projektide loomiseks, mis on seotud automatiseerimise, robootika ja interaktiivse kunstiga. Selle avatud struktuur ja aktiivne kogukond teevad sellest platvormist ideaalse vahendi õppimiseks ja eksperimenteerimiseks.

1.1.4. Arduino ja Raspberry Pi võrdlemine

Arduino ja Raspberry Pi on kaks kõige populaarsemat seadet, mida arendajad kasutavad erinevate projektide nutikate süsteemide (prototüüpide) arendamisel. Mõlemal platvormil on oma tugevused ja nõrkused ning nende vahel valimine sõltub konkreetse

projekti nõuetest. Alljärgnevas tabelis 1.1 on toodud mõlema platvormi kõige olulisemad parameetrid.

Tabel 1.1 Arduino ja Raspberry Pi võrdlus [12]

Parameeter	Arduino	Raspberry Pi
Eesmärk	Mikrokontroller lihtsate projektide jaoks	Täisfunktsionaalne arvuti keerukate projektide jaoks
Protsessor	AVR mikrokontroller	ARM Cortex-A72 (Raspberry Pi 4 jaoks) või ARM Cortex-A53 (teiste mudelite jaoks)
Taktsagedus	16 MHz enamiku plaatide jaoks	1,5 GHz Raspberry Pi 4 jaoks, 1,2 GHz teiste mudelite jaoks
RAM	2-8 KB (sõltuvalt plaadist)	1 GB Raspberry Pi 4 jaoks, 512 MB Raspberry Pi 3 jaoks, 256 MB Raspberry Pi 2 jaoks
Operatsioonisüsteem	Lihtne IDE Arduino põhine keskkond	Täisfunktsionaalne OS nagu Linux
Digitaalsed sisendid/väljundid	14 digitaalset sisendit/väljundit	40 universaalset sisendit/väljundit (GPIO)
Analoogsisendid	6-8 analoogsisendit	8 analoogsisendit (GPIO ADC funktsiooniga)
Kogukond	Suur ja aktiivne kogukond	Suur ja aktiivne kogukond
Laiendused ja tarvikud	Palju saadaval olevaid plaate (Ethernet, WiFi jne)	Palju tarvikuid ja lisaplaate (HATs, kaamerad, displeid)

Lõputöö raames nutimaja süsteemi arendamisel pidi autor töötama juba olemasoleva prototüübiga, mis põhineb Arduino platvormil. Kuigi alternatiivse platvormi valimise võimalust ei olnud, on autor kindel, et Arduino oleks valitud ka sõltumatult sellest asjaolust. See valik on tingitud mitte ainult Arduino komponentide madalamast hinnast võrreldes teiste turul olevate mikrokontrolleritega, vaid ka programmeerimise lihtsusest ja ulatuslikust kogukonna toetusest, mis oluliselt lihtsustab IoT-projektide arendamist ja testimist. Tänu oma kogemusele selle platvormiga peab autor Arduinot kättesaadavaks ja paindlikuks tööriistaks erinevate projektide loomiseks interneti asjade valdkonnas, sealhulgas haridus- ja eksperimentaalprojektide jaoks.

2. KASUTATUD TEHNOLOOGIAD

Käesolevas osas käsitletakse põhitehnoloogiaid, mida autor kasutas nutimaja rakenduse loomisel, ning pakub alternatiivsete tehnoloogiate analüüsi, mida oleks võinud kasutada antud projekti kontekstis.

Projekti teostamiseks valitud tehnoloogiad näitavad aktiivset arengut IT-valdkonnas ja on populaarsed professionaalsete arendajate seas. Selline eelistus on tingitud mitmest eelisest, nagu stabiilsus, paindlikkus, skaleeritavus ja aktiivne toetav kogukond.

2.1. Back-end

2.1.1. Spring Boot

Spring Boot on Springi platvormi laiendus, mis pakub valmis lahendusi kiirete, Springil põhinevate ja kõrge jõudlusega rakenduste loomiseks minimaalsete seadistustega. See lihtsustab Springi-põhiste rakenduste seadistamise ja juurutamise protsessi. Spring Boot konfigureerib automaatselt rakenduse sõltuvalt klassitee (*classpath*) raamatukogudest. Samuti pakub see sisseehitatud servereid, nagu Tomcat, et rakendusi saaks kiiresti juurutada ilma väliste serverite seadistamise vajaduseta. [5]

Üks Spring Boot'i võtmeomadustest on selle võime luua iseseisvaid täidetavaid rakendusi, mis sisaldavad kõiki vajalikke sõltuvusi, muutes juurutamise protsessi lihtsamaks. [5]

2.1.2. Spring Boot ja Django võrdlemine

Tänapäeva tarkvaratoodete turul on palju Spring Boot'iga sarnaseid raamistikke. Üks silmapaistvamaid konkurente, mida tasub mainida koos autori valitud tehnoloogiaga, on Django raamistik.

Django on kõrgtasemeline veebiraamistik programmeerimiskeele Python jaoks, mis võimaldab arendajatel kiiresti luua turvalisi ja tõhusaid veebisaite ja rakendusi. Django tekkis praktilisest vajadusest veebiprojektide kiireks juurutamiseks ja on välja töötatud nii, et see tagab kasutusmugavuse funktsionaalsust kaotamata. [6]

Tabel 2.1 Võrdlus raamistike Spring Boot ja Django vahel [11]

Parameeter	Django	Spring boot
Põhikeel	Python	Java
Kirjeldus	Django on Pythonil põhinev veebiraamistik, mis järgib model-view-controller (MVC) arhitektuurimustrit.	Spring Boot on Java-põhine raamistik, mis on loodud populaarse Spring Framework'i alusel.
Põhiline rakendus	Veebirakenduste, CMS-i, veebiteenuste kiire arendamine.	Suuremahulised ettevõtte rakendused, mikroteenused, hajutatud süsteemid.
Põhifunktsioonid	Kõrgetasemeline raamistik. Kiire arendamine. Skaleeritavus. Suurepärane turvalisus. Integreeritud administraatori paneel, ORM jne.	Iseseisvate rakenduste loomine. Automaatne konfiguratsioon. Integreeritud servlet-konteiner. Integreerimine Spring'i ökosüsteemiga jne.
Ökosüsteem	Lai Pythoni teekide komplekt.	Võimas Java ökosüsteem paljude teekide ja tööriistadega.
Kogukond	Aktiivne ja elujõuline kogukond, paljud paketid saadaval läbi Python Package Index (PyPI).	Tugev Java kogukonna tugi, palju tööriistu, sealhulgas Spring Security, Spring Data ja Spring Cloud.
Filosoofia	"Batteries included" - lihtsustab intuitiivsete veebirakenduste loomist, sealhulgas võimas administraatori liides, ORM ja muud komponendid.	"Konventsioon üle konfiguratsiooni" - lihtsustab arendusprotsessi, pakkudes mõistlikke vaikeväärtusi ja vähendades boilerplate-koodi hulka.

Autori lõputöö raamistiku Spring Boot valik ei olnud juhuslik. Spring Boot on tõestanud end kui kõrgefunktsionaalset ja nõutud arenduskeskkonda, mis on eriti oluline, arvestades et autori peamine programmeerimiskeel oli Java. Oluliseks teguriks sai ka see, et Eesti tööturul on kõrge nõudlus spetsialistide järele, kes valdavad Spring Boot'i, mis avab lõpetajale laiad võimalused tööle asumiseks. Töötades projektide kallal, mis võiksid köita piirkonna juhtivate IT-ettevõtete, nagu Fujitsu-Siemens Estonia, Telia ja SEB, tähelepanu, püüdis autor koguda vajalikku praktilist kogemust edukaks karjääri alustamiseks back-end arenduse valdkonnas, kuna just Spring Boot'i tehnoloogiate tundmine oli oluline nende ettevõtete konkursside läbimiseks.

2.2. Front-end

2.2.1. Vue.js

Vue.js on progressiivne SPA raamistik kasutajaliideste loomiseks. See lasti välja 2014. aastal ja sai kiiresti populaarseks oma lihtsa liidese ja kerge õppimiskõvera tõttu. [14] Erinevalt monoliitsetest raamistikest on Vue kavandatud nii, et seda saaks järk-järgult implementeerida. Selle tuum keskendub ainult esitluskihile, mis teeb selle hõlpsasti integreeritavaks teiste projektide ja teekidega. Vue pakub ka kaasaegseid tööriistu ja toetavaid teeke, mis võimaldavad luua keerukaid üheleheküljelisi rakendusi. [15]

2.2.2. Vue.js, Angular ja React võrdlemine.

Angular, React ja Vue on kolm populaarset raamistikku veebirakenduste arendamiseks. Igal neist on oma eripärad, eelised ja puudused.

Angular on täisfunktsionaalne TypeScript-põhine veebirakenduste raamistik. See on laialdaselt kasutatav üheleheküljeliste rakenduste (SPA) loomisel ja algselt käivitas selle Google 2016. aastal. [14]

React on avatud JavaScripti teek kasutajaliideste komponentidel põhinevaks arendamiseks. Selle töötas välja Facebook 2013. aastal ja praegu toetavad seda nii kogukond kui ka Facebook. [14]

Kõigi kolme raamistiku selgemaks võrdlemiseks on esitatud tabel 2.2.

Tabel 2.2 Angular, React ja Vue.js parameetrite võrdlustabel [14]

Parameeter	Angular	React	Vue.js
Väljalaske kuupäev	2016	2011	2014
Tugi	Google	Facebook	Kogukond
Tüüp	Raamistik	Teek	Raamistik
Suurus	Keskmine	Väike	Väga väike
Programmeerimis keel	TypeScript	JavaScript	JavaScript
Jõudlus	Hea	Hea	Hea
Andmete sidumine	Kahepoolne	Ühesuunaline	Kahepoolne
Õppimiskõver	Järsk	Lihtne	Lihtne
Populaarsed veebisaidid	Paypal, Samsung, Upwork	Netflix, Twitter, Amazon	Alibaba, Grammarly, GitLab

Front-end'i raamistiku valimisel on oluline arvestada mitte ainult funktsionaalsete

võimalustega, vaid ka rakenduse jõudlusega. Võrdlevas tabelis, mis on esitatud joonisel 2.1, demonstreerib Vue.js end kui kõige kiirema toimega tööriista teiste populaarsete raamistike, nagu Angular ja React, seas.

Duration for...	angular-v13.0.0	react-v17.0.2	vue-v3.2.26
create rows	117.14.0 (1.14)	120.14.4 (1.17)	102.64.6 (1.00)
replace all rows	112.71.4 (1.11)	110.32.2 (1.08)	101.92.5 (1.00)
partial update	195.44.9 (1.00)	246.77.9 (1.27)	194.91.9 (1.00)
select row	71.41.9 (1.73)	111.91.6 (2.71)	41.30.7 (1.00)
swap rows	351.83.4 (6.58)	350.85.2 (6.56)	53.50.6 (1.00)
remove row	21.60.2 (1.00)	24.00.2 (1.11)	22.80.4 (1.06)
create many row	1,183.221.2 (1.09)	1,432.18.5 (1.32)	1,083.59.0 (1.00)
append rows to large table	239.91.3 (1.12)	263.94.2 (1.23)	214.61.9 (1.00)
clear rows	165.72.8 (2.30)	84.00.9 (1.17)	72.00.5 (1.00)
geometric mean	1.51	1.58	1.01

Joonis 2.1 Vue.js, Angulari ja Reacti jõudluse võrdlus [17]

Lihtne omandamine, väike suurus ja kõrge töökiirus on peamised eelised, mis on teinud Vue.js'ist ideaalse raamistiku kasutajaliidese loomiseks antud lõputöö projektis. Need omadused on eriti väärtuslikud kaasaegse veebiarenduse kontekstis, kus rakenduse laadimisaeg ja reageerimiskiirus võivad oluliselt mõjutada kasutajakogemust ja üldist toote tajumist.

Lihtne ja intuitiivne andmemudel hõlbustab koodi muutmist ja kiirendab arendusprotsessi, muutes Vue.js eriti atraktiivseks algajatele front-end'i arenduses.

Lisaboonusena on Vue.js aktiivne kogukond, mis töötab pidevalt raamistiku täiustamise kallal ja pakub palju õppematerjale ja tuge. Vue ökosüsteemi kaudu kättesaadavate valmis komponentide ja pluginite mitmekesisus võimaldab arendajatel kiiresti integreerida uusi funktsioone ja laiendada oma rakenduste võimalusi ilma lisakoodi nullist loomise vajaduseta.

Seega, Vue.js oma rõhuasetusega jõudlusele ja arendamise mugavusele muutub

suurepäraseks valikuks projektidele, mis nõuavad nii paindlikkust keerukate kasutajaliideste lahenduste rakendamisel kui ka rakenduse kõrget töökiirust, mida ongi autor valinud.

2.3. Andmete visualiseerimine

2.3.1. Chart.js

Chart.js on avatud JavaScripti teek andmete visualiseerimiseks, mis võimaldab luua interaktiivseid graafikuid ja diagramme veebilehtedele. Selle teegi peamine eesmärk on pakkuda lihtsat ja intuitiivset API-d erinevat tüüpi graafikute loomiseks, sealhulgas joon-, tulp- ja ringdiagramme ning palju muud. [16]

Chart.js peamised omadused ja võimalused:

- kohandatavus: graafikud saavad automaatselt skaleeruda vastavalt konteineri suurusele, milles neid kuvatakse;
- seadistatavus: ulatuslikud võimalused graafikute stiilimiseks ja seadistamiseks, sealhulgas animatsioon, värvid ja legend;
- lihtsus: intuitiivne API ja hästi struktureeritud dokumentatsioon teevad teegi integreerimise ja kasutamise lihtsaks;
- jõudlus: Chart.js on optimeeritud graafikute kiireks joonistamiseks ilma kvaliteeti kaotamata. [16]

Üheks Chart.js lisaeeliseks on selle aktiivne arendajate kogukond ja paljude pluginate tugi, mis laiendab visualiseerimisvõimalusi.

2.3.2. vue-chart.js

Vue-chartjs on Chart.js'i jaoks mõeldud ümbris, mis on ette nähtud kasutamiseks Vue keskkonnas. See on ideaalne lahendus neile, kes soovivad kiiresti ja vaevata integreerida lihtsaid graafikuid oma Vue rakendustesse. Vue-chartjs abstraheerib alusloogikat, kuid pakub Chart.js objekti, et anda maksimaalset paindlikkust. [18]

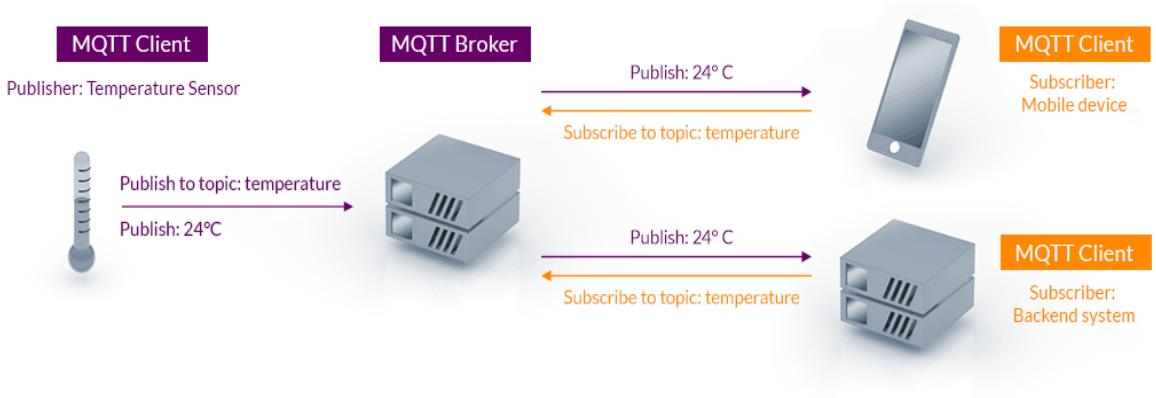
2.4. Sideprotokollid

2.4.1. MQTT

MQTT (Message Queuing Telemetry Transport) on väljaandja/tellijapõhine sõnumivahetusprotokoll, mis on loodud kasutamiseks madala ribalaiusega olukordades ja kus seadmed võivad olla piiratud ühenduvusega. See muudab selle ideaalselt sobivaks asjade interneti (IoT) jaoks. Protokolli tööpõhimõte on skemaatiliselt kujutatud joonisel 2.2. [9]

MQTT omaduste hulka kuuluvad järgmised karakteristikud:

- kergekaaluline protokoll: MQTT on kavandatud nii, et minimeerida edastatavate andmete mahtu ja ressursside kasutust, muutes selle ideaalseks paljudele IoT-rakendustele;
- QoS tasemed: MQTT toetab erinevaid teeninduskvaliteedi (QoS) tasemeid, mis võimaldavad arendajatel määratleda, kuidas sõnumid seadmete vahel edastatakse;
- turvalisus: kuigi MQTT baasprotokoll ei sisalda turvamehhanisme, on võimalik seda kombineerida turvaliste protokollidega, nagu TLS/SSL, turvaliseks sõnumivahetuseks. [9]

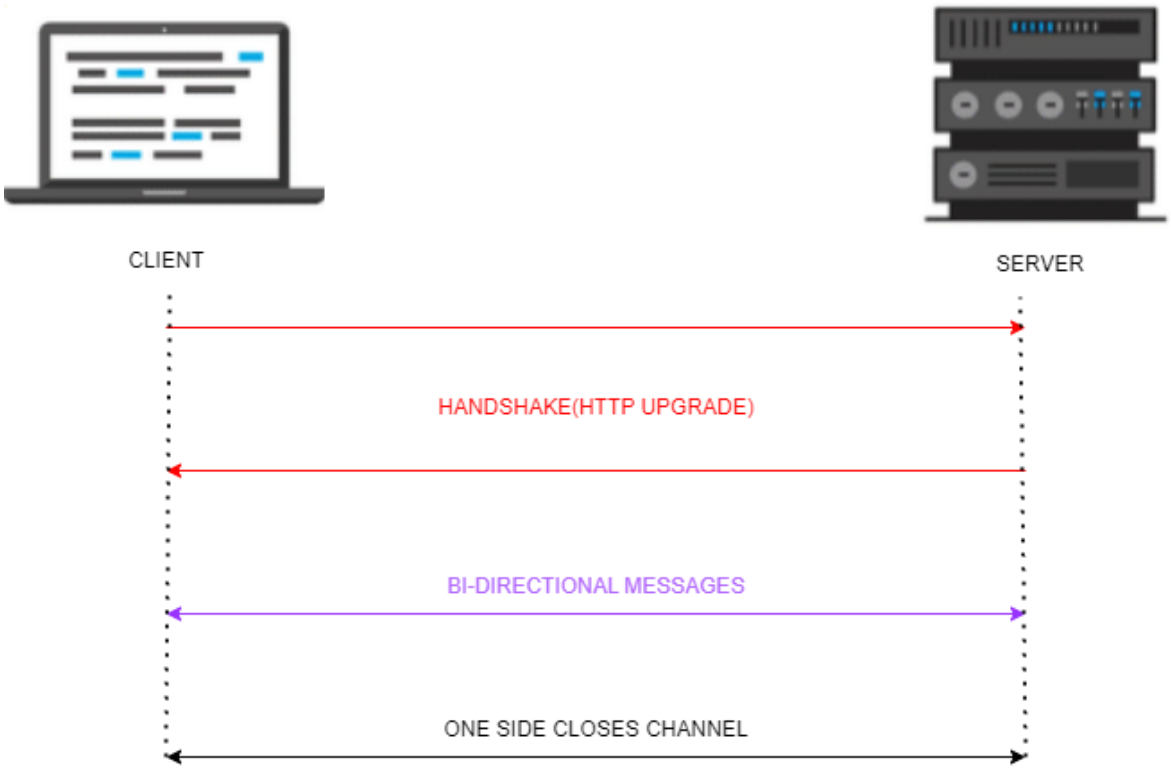


Joonis 2.2 MQTT protokolli tööpõhimõte [9]

2.4.2. MQTT ja WebSocket võrdlemine

WebSocket on kaasaegne protokoll, mis võimaldab andmevahetust brauseri ja serveri vahel reaalajas. See protokoll loob stabiilse, kahepoolse ühenduse ühe TCP-pordi kaudu. See tähendab, et pärast ühenduse loomist saab andmeid edastada samaaegselt mõlemas suunas, muutes kliendi ja serveri vahelise suhtluse kiireks ja tõhusaks. [7]

WebSocketi tööpõhimõtte skeemilist esitust võib kirjeldada järgnevalt (vt Joonis 2.3). Klient, näiteks veebibrauser, algatab ühenduse serveriga, kasutades WebSocket protokolli, saates erilise päringu 'veebisoketi' loomiseks. Pärast ühenduse loomist võib server igal ajahetkel saata andmeid otse kliendile, ja klient võib saata andmeid serverisse ilma viivitusteta, tagades sellega veebirakenduste reaktiivsuse ja interaktiivsuse.



Joonis 2.3 WebSocketi tööpõhimõte

WebSocketi tehnoloogia sobib paljude andmeedastusega seotud ülesannete jaoks, näiteks sõnumirakendused, reaalajas mitmikmängud jne. [7] Tabelis 2.3 on toodud MQTT ja WebSocketi peamised erinevused.

Tabel 2.3 MQTT ja WebSocketi võrdlus [8]

Parameeter	MQTT	WebSocket
Prioriteedi seadistamine	Võimalik	Pole võimalik
Kasutus	Klient ja server	Veebiklient
Kulud	Minimaalsed	Suured paljude IoT seadmete kasutamisel
Disain	IoT seadmetele	Kahepoolseks kommunikatsioonikanaliks
Sõnumite jaotamine	Üks-mitmele	Üks-ühele
Alus	Põhineb sõnumitel	Põhineb sessioonidel

Autori arvates on MQTT eelistatum, kuna see pakub funktsioone ja abstraktsioone, mida WebSocket ei paku. Lisaks sobib MQTT suurepäraselt sisseehitatud süsteemide jaoks, kuna see töötab tõhusalt, nõuab suhtlemiseks vähe ressursse ja võimaldab ühel seadmel n-ö rääkida teistele, mis nendega toimub. [8]

2.5. Andmebaasid

2.5.1. PostgreSQL

PostgreSQL on võimas, avatud lähtekoodiga relatsiooniliste andmebaaside haldussüsteem (RDBMS). Sellel on üle 15-aastane arendusajalugu ja palju funktsioone, mis teevad sellest ühe maailma kõige edumeelsema RDBMS-i. [10]

PostgreSQL omadused:

- laiendatavus: lisaks paljudele sisseehitatud andmetüüpidele võimaldab PostgreSQL luua oma andmetüüpe;
- JSON-i tugi: PostgreSQL pakub põhilist JSON-i tuge, võimaldades arendajatel hõlpsasti integreeruda kaasaegsete veebitehnoloogiatel põhinevate rakendustega;
- protseduuriline programmeerimine: PostgreSQL toetab salvestatud protseduure ja võimaldab luua funktsioone mitmetes programmeerimiskeeltes;
- kogukondlik tugi: oma avatud lähtekoodi staatuse tõttu omab PostgreSQL aktiivset ja pühendunud arendajate ja toe kogukonda. [10]

2.5.2. PostgreSQL ja MySQL võrdlemine

MySQL ja PostgreSQL on kaks populaarset relatsiooniliste andmebaaside haldussüsteemi. Kuigi nad on kontseptuaalselt sarnased, on neil mitmeid erinevusi (vt Tabel 2.4), mis võivad mõjutada arendajate valikuid sõltuvalt projekti konkreetsetest nõuetest.

Tabel 2.4 PostgreSQLi ja MySQLi võrdlus [13]

Parameetrid	MySQL	PostgreSQL
Paralleelne ligipääsu juhtimine (MVCC)	Ei toeta MVCC.	Toetab MVCC.
Indeksi tüübid	Toetab B- ja R-puu indekseerimist.	Toetab puuindekseid, avaldisindekseid, osalisi indekseid ja hash-indekseid.
Andmetüübid	Relatsiooniline andmebaas.	Objekt-relatsiooniline andmebaas. Toetab andmetüüpe nagu massiivid ja XML.
Vaated	Toetab vaateid.	Pakub vaadetele laiendatud võimalusi, sealhulgas materialiseeritud vaateid.
Salvestatud protseduurid	Toetab salvestatud protseduure.	Toetab erinevates keeltes, sh SQL-ist erinevates

		keeltes kirjutatud salvestatud protseduure.
Trigerid	Toetab ainult AFTER ja BEFORE trigereid.	Toetab AFTER, BEFORE ja INSTEAD OF trigereid.

PostgreSQLi valimine andmebaaside haldussüsteemina nutimaja projekti jaoks oli määratud mitmete võtmeteguritega, sealhulgas rakenduse nõuetega kõrge sagedusega päringute töötlemisele. See süsteem on tuntud oma usaldusväärsuse, laiendatavuse ja täieliku SQL-toe poolest, muutes selle ideaalseks lahenduseks keerukatele ja dünaamilistele rakendustele.

Lisaks tehnilistele eelistele oli autoril juba praktiline kogemus PostgreSQLiga tänu andmebaaside kursusele, kus peamiselt kasutati seda süsteemi. See tagas PostgreSQLi spetsiifikast ja omadustest arusaamise, võimaldades autoril efektiivselt integreerida ja optimeerida andmebaasi arendatava rakenduse kontekstis.

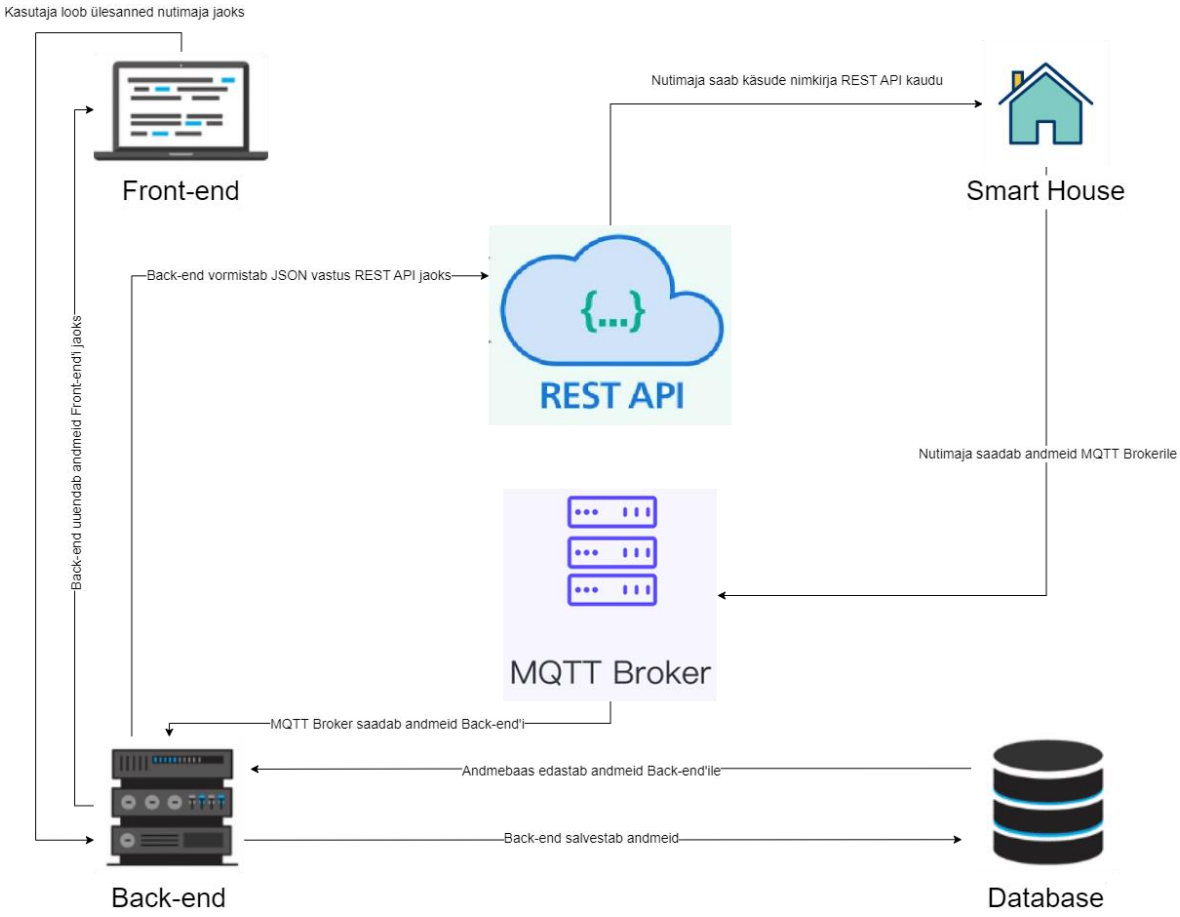
Eelnev kogemus kiirendas oluliselt arendusprotsessi ja aitas tõsta projekti andmebaasi kvaliteeti ja usaldusväärsust. Seega sai PostgreSQLi tundmaõppimine akadeemilise õppe ja praktilise arendustöö kaudu otsustavaks teguriks selle valikul lõputöö projekti jaoks.

3. RAKENDUSE REALISATSIOON

Nutimaja rakenduse arhitektuur hõlmab kuut võtmetähtsusega komponenti, mis moodustavad süsteemi aluse:

- nutimaja prototüüp, mis on süsteemi füüsiline alus ja sisaldab täideviijaid ja andureid;
- andmebaas, mille ülesanne on kogutud andmete, sealhulgas andurite näitude ja toimingute logide, hoidmine;
- Back-end, mis tagab andmete töötlemise loogika ja nutimaja juhtimise;
- Front-end, mis vastutab kasutajaliidese eest, võimaldades süsteemiga mugavalt ja visuaalselt suhelda;
- MQTT server, mis toimib keskse sõlmpunktina anduritelt tulevate sõnumite vastuvõtmiseks ja täideviijatele edastamiseks, toetades IoT põhimõtteid;
- REST API, mis pakub operatsioonide kogumit ülesannete ja stsenaariumide juhtimiseks, võimaldades paindlikult seadistada nutimaja käitumist.

Kuigi MQTT ja REST API on osa serveri loogikast, käsitletakse neid selles arhitektuuris eraldi komponentidena nende spetsiifiliste rollide tõttu süsteemi kommunikatsiooni ja juhtimise tagamisel. MQTT vastutab otsese andmeedastuse eest seadmete vahel reaalajas, samas kui REST API pakub haldusliidest, mille kaudu toimub stsenaariumide ja ülesannete konfigureerimine. Nende komponentide integratsioon loob täisfunktsionaalse süsteemi, kus iga element mängib oma rolli nutimaja terviklikkuse ja funktsionaalsuse saavutamisel. Süsteemi erinevate elementide vahelise suhtluse visuaalne esitus on kujutatud skemaatilisel kujutisel (vt Joonis 3.1).



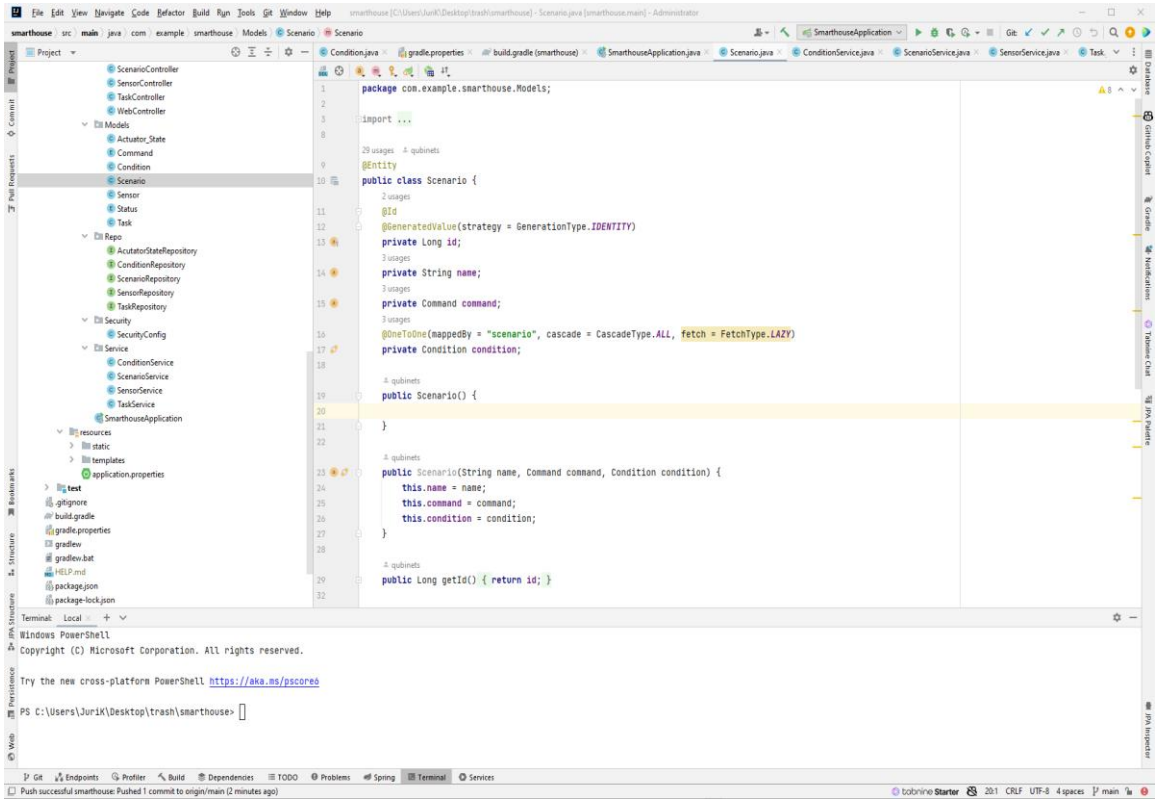
Joonis 3.1 Nutimaja rakenduse arhitektuur

3.1. Arenduskeskkond

Arenduskeskkonnana valis autor IntelliJ Idea (IDE), mis on integreeritud Java arendamiseks. Selle lõi JetBrains ja see pakub paljusid edasijõudnud funktsioone arendaja tootlikkuse suurendamiseks, nagu nutikas koodi lõpetamine, süntaksi esiletõstmine, refaktorimine, koodi staatiline analüüs ja palju muud. [19]

Üks IntelliJ IDEA unikaalsetest omadustest on selle võime analüüsida koodi ja pakkuda kasulikke nõuandeid ning lahendusi erinevatele probleemidele, mis võivad arendamisel tekkida. See mitte ainult ei aita vigu vältida, vaid õpetab ka arendajaid parimatele programmeerimistavadele. [19]

IntelliJ IDEA toetab ka paljusid pluginaid, mis võimaldavad IDE funktsionaalsust laiendada ja toetada mitmesuguseid keeli ja raamistikke alates Androidist kuni Scala ja Kotlinini. [19]



Joonis 3.2 IntelliJ Idea kasutajaliides

3.2. Back-end rakenduse struktuur

Tagamaks nutimaja süsteemi funktsionaalsust, töötas autor välja back-end'i, mille struktuur on kujutatud joonisel 3.3, kasutades Spring Boot tehnoloogiat. See osa kirjeldab back-end'i struktuuri ja komponente.

Back-end rakenduse struktuur on organiseeritud kihtidesse ja järgib MVC (Model-View-Controller) arhitektuuristiili. Siiski on oluline märkida, et antud rakenduse puhul on 'View' komponent eraldatud ja paigutatud front-end osasse, kasutades Vue.js raamistikku.

1. Config

- MqttConfiguration: Vastutab MQTT konfiguratsiooni eest, mis on protokoll seadmetevaheliste sõnumite vahetamiseks.

2. Controllers

See kiht pakub sisenemispunkte HTTP-päringute töötlemiseks.

- ConditionController: Juhib tingimustega seotud päringuid.
- ConditionRequest: DTO (Data Transfer Object) tingimuste andmete edastamiseks.

- ScenarioController: Juhib stsenaariumitega seotud päringuid.
- SensorController: Juhib sensorite päringuid.
- TaskController: Juhib ülesannetega seotud päringuid.
- ActuatorStateController: Juhib täiturmehhanismide olekutega seotud päringuid.
- WebController: Juhib üldisi veebipäringuid.

3. Models

Sisaldab üksusi klasside kujul, mis esindavad andmebaasi andmeid.

- Command: Kättesaadavate käskude loetelu.
- Condition: Üksus, mis esindab tingimust.
- Scenario: Üksus stsenaariumide jaoks.
- Sensor: Üksus sensorite jaoks.
- Status: Ülesannete olekute loetelu.
- Task: Üksus ülesannete jaoks.
- ActuatorState: Üksus täiturmehhanismide olekute jaoks.

4. Repo

See kiht sisaldab JPA repositooriume, mis võimaldavad andmetele juurdepääsu.

- ConditionRepository: Repositoorium tingimuste jaoks.
- ScenarioRepository: Repositoorium stsenaariumide jaoks.
- SensorRepository: Repositoorium sensorite jaoks.
- TaskRepository: Repositoorium ülesannete jaoks.
- ActuatorStateRepository: Repositoorium täiturmehhanismide olekute jaoks.

5. Security

- SecurityConfig: Turvalisuse konfiguratsioon, mis määrab juurdepääsu ja autentimise reeglid.

6. Service

Teenuste kiht, mis sisaldab rakenduse äri-loogikat.

- ConditionService: Teenus tingimustega töötamiseks.
- ScenarioService: Teenus stsenaariumitega töötamiseks.
- SensorService: Teenus sensoritega töötamiseks.

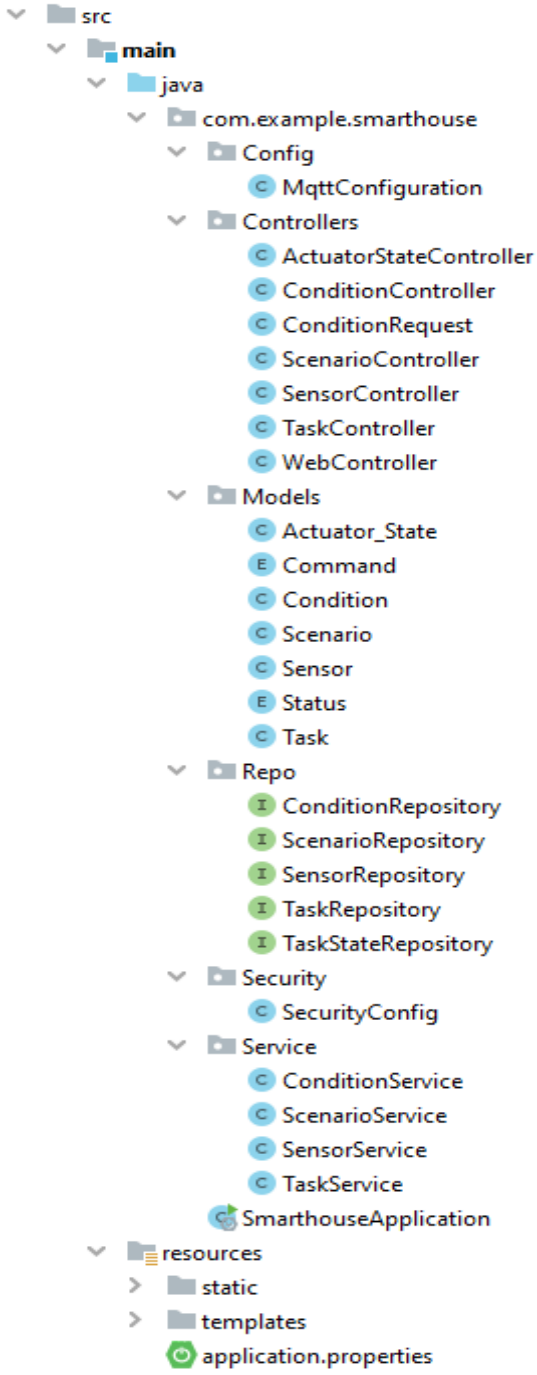
- TaskService: Teenus ülesannetega töötamiseks.

7. SmarthouseApplication

- Rakenduse peamine klass, mis käivitab Spring Boot rakenduse.

8. resources

- Sisaldab staatilisi ressursse, nagu CSS, pildid ja JavaScript, samuti konfiguratsioonifaile ja HTML-malle.

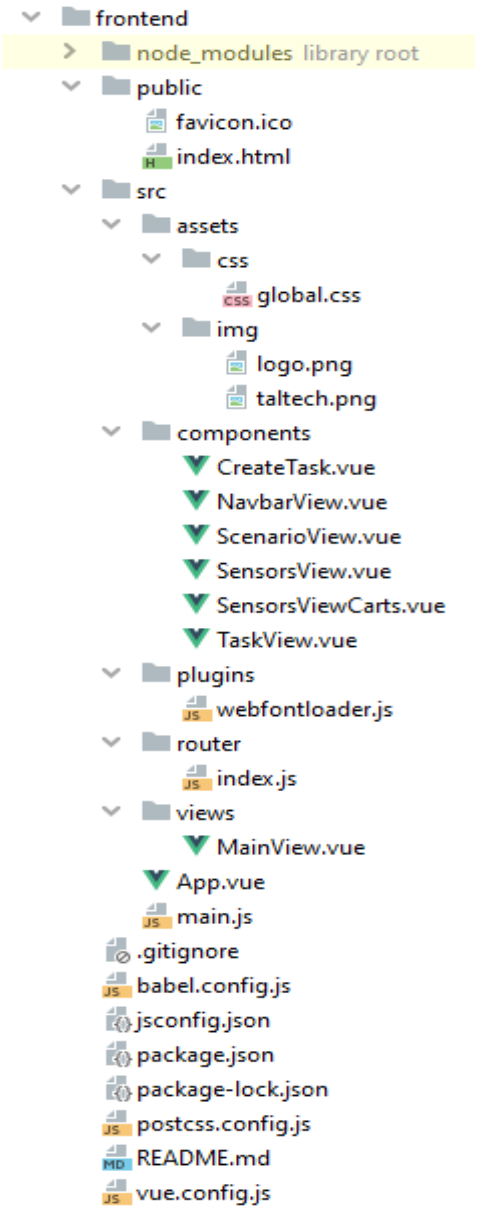


Joonis 3.3 Back-end rakenduse struktuur

Nutimaja rakenduse tagaosa esindab paindlikku ja modulaarset struktuuri, mis on jaotatud kihtideks, et tagada arendamise, toe ja mastaapsuse lihtsuse. Spring Boot tehnoloogia kasutamine võimaldab kiirelt ja usaldusväärselt arendada kaasaegseid veebirakendusi, toetades turvalisust, andmetele juurdepääsu ja muid võtmeomadusi.

3.3. Front-end rakenduse struktuur

Projekti front-end osa arendamisel valis autor Vue.js raamistiku. Projekti struktuuri ja selle haldamise korraldamiseks kasutati selle raamistiku standardseid tööriistu ja soovitusi. Joonisel 3.4 on kujutatud rakenduse front-end struktuur.



Joonis 3.4 Front-end rakenduse struktuur

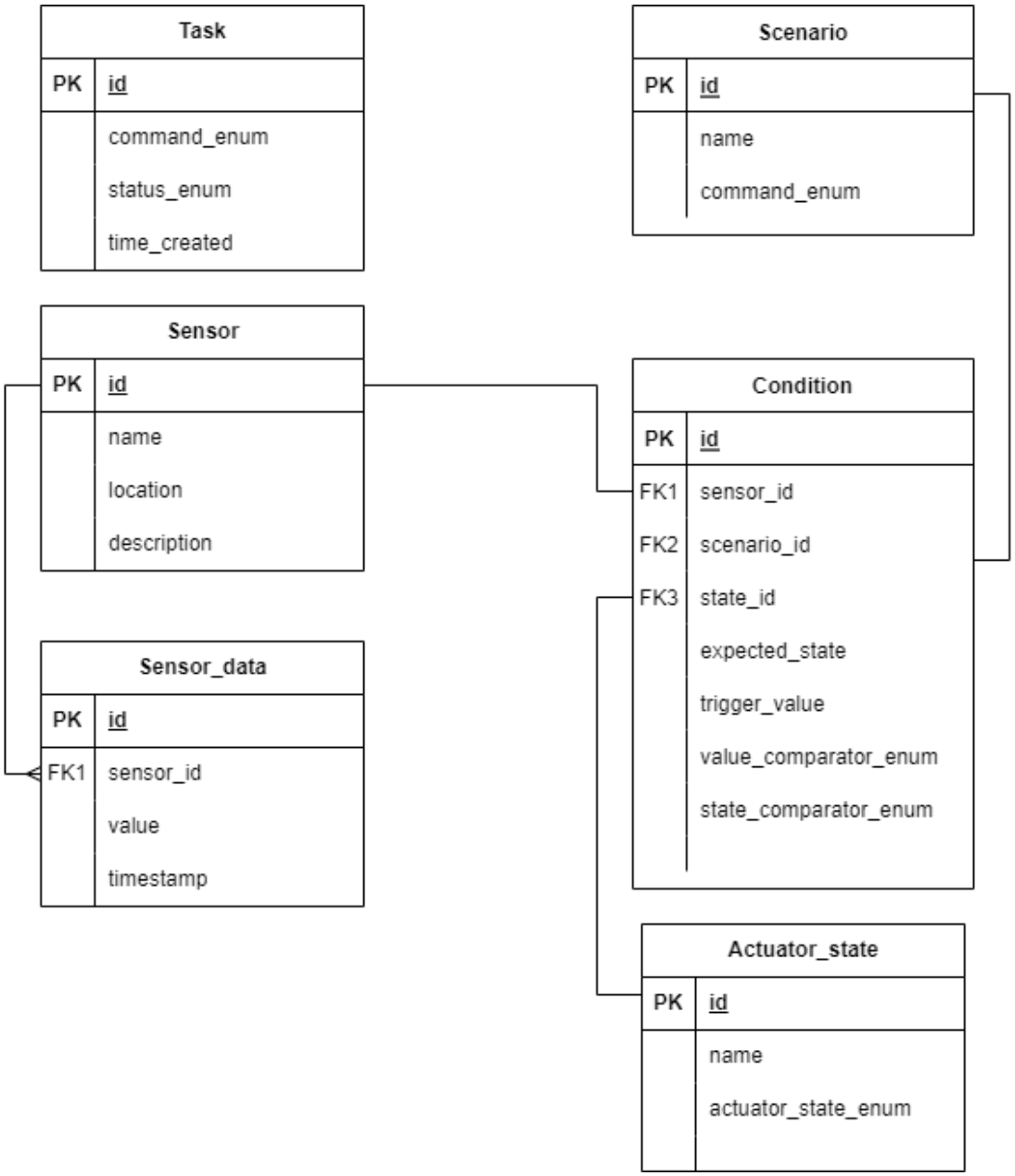
Allpool on toodud projekti peamiste kataloogide ja failide lühikirjeldus:

1. node_modules: Kataloog, mis sisaldab kõiki projektis npm paketihalduri abil installitud sõltuvusi. Selles kaustas on talletatud rakenduse töötamiseks vajalikud teegid ja tööriistad.
2. public: Selles kataloogis hoitakse staatilisi faile, mis ei vaja enne klientidele saatmist kompileerimist ega töötlemist.
3. src: Peamine kataloog, mis sisaldab rakenduse lähtekoodi.
    - assets: Ressursside kataloog, nagu pildid või stiilid.
    - components: Selles kataloogis hoitakse Vue komponente, mis on korduvkasutatavad liidese osad.
    - plugins: Kataloog Vue pistikprogrammide jaoks, näiteks webfontloader.js.
    - router: Kataloog, mis sisaldab Vue marsruuteri konfiguratsiooni.
    - Views: Kataloog komponentide jaoks, mis esindavad erinevaid vaateid (views) rakenduses.
    - App.vue: Rakenduse juurkomponent.
    - main.js: Rakenduse sisenemispunkt, kus luuakse uus Vue eksemplar.
4. .gitignore: Fail, mis annab Gitile teada, milliseid faile või katalooge tuleks commit'ides eirata.
5. babel.config.js, postcss.config.js, vue.config.js: Projektis kasutatavate tööriistade ja pistikprogrammide konfiguratsioonifailid.
6. package.json ja package-lock.json: Failid, mis sisaldavad projekti sõltuvuste ja muu metaandmete infot.
7. README.md: Dokument projektikirjelduse, juhendite ja muu kasuliku teabega.

See struktuur tagab koodi selge organiseerimise, muutes selle hõlpsasti loetavaks ja hooldatavaks.

3.4. Andmebaasi struktuur

Diplomitöö raames arendas autor PostgreSQL-i põhjal nutika kodu süsteemi andmebaasi, mille peamine eesmärk on tagada nutika kodu automatiseerimisega seotud andmete, nagu stsenaariumid, nende täitmise tingimused, ülesanded ja andurid, säilitamine ja haldamine.



Joonis 3.5 Andmebaasi ERD mudel

Andmebaas koosneb järgmistest tabelitest, mis on kujutatud joonisel 3.5 :

- Scenario - esindab nutika kodu automatiseerimise stsenaariumi.
- Condition - kirjeldab stsenaariumi aktiveerimise tingimusi.
- Sensor - andurid, mis koguvad nutika kodu näitusid.
- Sensor_data – andmed, mida andurid edastavad.
- Task - ülesanded, mis tuleb täita stsenaariumi aktiveerimisel või käsitsi juhtimisel.
- Actuator_state - täiturmehhanismide olek.

Tabelite kirjeldus:

- Scenario
    - id - stsenaariumi unikaalne identifikaator.
    - name - stsenaariumi nimetus.
    - command - stsenaariumiga seotud käsk, võib võtta väärtusi loendist Command (door, light, sound, pump).
- Condition
    - id - tingimuse unikaalne identifikaator.
    - sensor_id - anduri id, mille andmeid tingimuses kasutatakse.
    - scenario_id – stsenaariumi id, mille jaoks tingimus luuakse.
    - state_id – täiturmehhanismi praeguse oleku id, mis määrab stsenaariumi täitmise.
    - expected_state – täiturmehhanismi oodatav olek.
    - trigger_value – anduri lülitusväärtus.
    - valueComparator ja stateComparator - loendid, mis määravad, kuidas praegust ja oodatavat väärtust võrrelda.
- Sensor
    - id - anduri unikaalne identifikaator.
    - name - anduri nimi.
    - location – anduri asukoht.
    - description – anduri kirjeldus.
- Sensor_data
    - id - unikaalne identifikaator.
    - sensor_id - anduri id.
    - value - anduri praegune väärtus.
    - timestamp - viimase värskenduse aeg.
- Task
    - id - ülesande unikaalne identifikaator.
    - command - ülesandega seotud käsk.
    - created - ülesande loomise kuupäev ja kellaaeg.

- status – loendid, mis määravad ülesande oleku (PENDING või COMPLETED).

- Actuator_state
  - id - täiturmehhanismi oleku unikaalne identifikaator.
  - name - täiturmehhanismi nimi.
  - actuator_state_enum – loendid, mis määravad täiturmehhanismi oleku (ON või OFF).

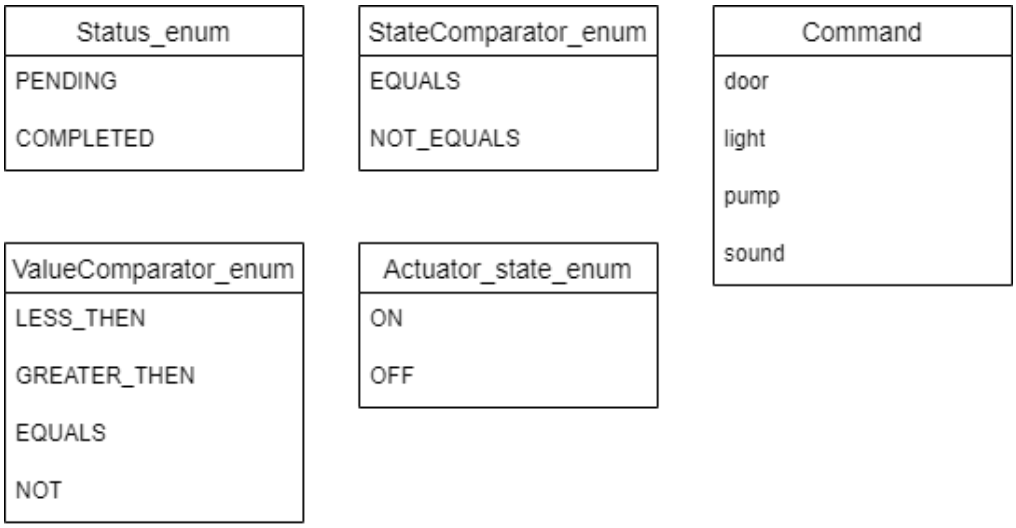
Vastastikused seosed:

Scenario ja Condition'i vahel on seos üks-ühele. Igale stsenaariumile vastab täpselt üks aktiveerimistingimus ja igale aktiveerimistingimusele täpselt üks stsenaarium.

Condition ja Actuator_state vahel on seos üks-ühele. Tingimus määratletakse konkreetse täiturmehhanismi oleku järgi.

Sensor ja Sensor_data vahel on seos üks-mitmele. Igal anduril on palju sisendandmeid.

Arendatud andmebaas tagab nutika kodu süsteemi andmete paindliku ja usaldusväärse haldamise. Loendite kasutamine (vt Joonis 3.6), käskude, ülesannete olekute ja võrdluste esindamiseks võimaldab tagada ranged tüübipiirangud ja vähendada vigade tõenäosust. Tabelitevahelised seosed tagavad andmete terviklikkuse ja võimaldavad tõhusalt välja võtta teavet automatiseerimisprotsesside kohta.



Joonis 3.6 Andmebaasi loendid

3.5. Andmevahetus

Andmete vahetamiseks Spring Boot'i baasil loodud rakenduse ja Arduino platvormil töötava seadme vahel kasutatakse MQTT protokolli ja REST API'd.

3.5.1. MQTTConfiguration

Autori loodud Spring Boot rakenduses kasutatakse org.springframework.integration teeki MQTT maakleri integreerimiseks. Andmete vastuvõtt Arduinolt rakendusse on kujutatud joonisel 3.7.

```
Received MQTT Sensor Data: SensorName=DS18B20 Temp, Value=21.5, Timestamp=2023-06-13T06:26:44
Received MQTT Sensor Data: SensorName=Light sensor, Value=5376.34, Timestamp=2023-06-13T06:26:44
Received MQTT Sensor Data: SensorName=Distance, Value=14.64, Timestamp=2023-06-13T06:26:44
Received MQTT Sensor Data: SensorName=Fire sensor, Value=0.92, Timestamp=2023-06-13T06:26:44
Received MQTT Sensor Data: SensorName=Gas sensor, Value=0.09, Timestamp=2023-06-13T06:26:44
Received MQTT Sensor Data: SensorName=Water level, Value=0.0, Timestamp=2023-06-13T06:26:44
Received MQTT Sensor Data: SensorName=Soil level, Value=0.0, Timestamp=2023-06-13T06:26:44
Received MQTT message: Topic=tasks/state/1/update, State=OFF
Updated task state: TaskId=1, NewState=OFF
Received MQTT message: Topic=tasks/state/2/update, State=OFF
Updated task state: TaskId=2, NewState=OFF
Received MQTT message: Topic=tasks/state/3/update, State=OFF
Updated task state: TaskId=3, NewState=OFF
Received MQTT message: Topic=tasks/state/4/update, State=OFF
Updated task state: TaskId=4, NewState=OFF
```

Joonis 3.7 Spring Boot rakenduse andmete vastuvõtmine MQTT kaudu

MqttConfiguration klassis (vt LISA 3) on toodud seadistused MQTT maakleriga ühenduse loomiseks:

MQTT maakleri aadress, kasutajanimi ja parool laaditakse rakenduse omadustest (mqtt.broker.url, mqtt.broker.username ja mqtt.broker.password). meetodis mqttClientFactory() luuakse MQTT klientide tehas antud ühenduse parameetritega.

MQTT teemadel (topics) tellimine: Erinevate kanalite (MessageChannel) abil tellib rakendus erinevaid teemasid:

- tööde oleku uuendamine: "tasks/state/+/update";
- lõpetatud tööd: "tasks/completed/update";
- andurite andmed: "sensors/+/value";
- tööde päringud: "tasks/request".

Sissetulevate sõnumite töötlemine: teatud teemadelt saadud sõnumite saabumisel kutsutakse välja vastavad töötlejad:

- mqttMessageHandler(): Tööde oleku muutmise sõnumite töötlemine.
- mqttCompletedTasksMessageHandler(): Lõpetatud tööde sõnumite töötlemine.
- mqttSensorDataMessageHandler(): Andurite andmete töötlemine.

REST API andmetele juurdepääsuks: TaskController kontrolleris on esitatud meetod getPendingTasks() (vt Joonis 3.8), mis tagastab kõik ootel olevad tööd.

```
@GetMapping("/tasks/pending")
public ResponseEntity<List<Task>> getPendingTasks() {
    List<Task> tasks = taskService.getPendingTasks();
    return ResponseEntity.ok().body(tasks);
}
```

Joonis 3.8 getPendigTask() meetodi kood

See meetod võimaldab Arduino baasil seadmel pärida ootel olevaid töid.

Arduino jaoks JSON formaadis tööde väljanägemine, mida Arduino saab:


```
[{"id":152,"command":"door","created":"2023-10-
01T06:11:46.443","status":"PENDING"},{"id":153,"command":"light","create
d":"2023-10-
29T05:40:23.319","status":"PENDING"},{"id":154,"command":"pump","created
":"2023-10-
29T05:40:24.12","status":"PENDING"},{"id":155,"command":"sound","created
":"2023-10-29T05:40:24.826","status":"PENDING"}]
```


Arduinos on loodud funktsioon, mis saab ja töötleb sissetulevaid ülesandeid, kasutades seda REST API meetodit.

3.5.2. checkPendingTasks()

Funktsioon checkPendingTasks() (vt LISA 1) esindab protsessi, kus Arduino küsib serverilt ootel olevate tööde nimekirja ja seejärel täidab neid töid vastavalt nende tüübile. Funktsiooni detailne analüüs:

- Kliendi ja HTTP-kliendi initsialiseerimine:
    - EthernetClient ethClient;: Loob Ethernet klient-objekti võrguoperatsioonide jaoks.
    - HttpClient httpClient(ethClient, server, port);: HTTP-kliendi initsialiseerimine Ethernet-kliendi, serveri aadressi ja pordi kasutamisega.
- HTTP-päring:
    - int result = httpClient.get("/tasks/pending");: Arduino pärib serverilt ootel olevad tööd.
- Vastuse staatuse kontrollimine:
    - Kui vastus on staatuses 200 (OK), alustab funktsioon vastuse töötlemist. Vastasel juhul kuvatakse veateade.
- Vastuse lugemine ja töötlemine:

    - Vastus loetakse HTTP-kliendist ja salvestatakse *response'i*.
    - See vastus on JSON-string, mis seejärel deserialiseeritakse doc objektiks.
    - JSON-i parsimise vea korral kuvatakse veateade.
- JSON-vastusest tulenevate tööde töötlemine:
    - Iga JSON-vastuses oleva töö jaoks kontrollib Arduino käsku ja teostab vastava toimingu.
    - door: Servomootori juhtimine ukse avamiseks või sulgemiseks.
    - pump: Relee juhtimine pumba sisse- või väljalülitamiseks.
    - light: Valguse juhtimine (MOSFETi kaudu).
    - sound: Helisignaali juhtimine (piesoelektrilise kõlari kaudu).
    - Iga töö täitmise järel saadetakse teavitus töö lõpetamisest.
- Ühenduse lõpetamine:
    - HTTP-kliendiga ühendus lõpetatakse pärast kõikide tööde töötlemist.

See funktsioon võimaldab Arduinol toimida nutiseadmena, mis suudab vastu võtta ja täita serveri kaudu saadetavaid käsklusi. See tagab võimaluse kaugjuhtida Arduino erinevaid seadmeid ja funktsioone.

Andmete uuendamiseks sensoritelt ja tööde olekust ning nende serverisse saatmiseks on samuti loodud funktsioon. Joonisel 3.9 on kujutatud protsess, kuidas sensorite näitajaid ja täiturmehhanismide olekuid serverisse MQTT kaudu readSendData() funktsiooni abil saadetakse.

```
Publishing: Topic=sensors/Sound sensor/value, Payload=6.02,2023-06-13T06:26:44
Publishing: Topic=sensors/Motion sensor/value, Payload=0.00,2023-06-13T06:26:44
Publishing: Topic=sensors/Touch sensor/value, Payload=1.00,2023-06-13T06:26:44
Publishing: Topic=sensors/BMP085 Temp/value, Payload=0.00,2023-06-13T06:26:44
Publishing: Topic=sensors/BMP085 Pressure/value, Payload=1.76,2023-06-13T06:26:44
Publishing: Topic=sensors/DHT11 Temp/value, Payload=23.00,2023-06-13T06:26:44
Publishing: Topic=sensors/DHT11 Humidity/value, Payload=48.00,2023-06-13T06:26:44
Publishing: Topic=sensors/DS18B20 Temp/value, Payload=21.50,2023-06-13T06:26:44
Publishing: Topic=sensors/Light sensor/value, Payload=5376.34,2023-06-13T06:26:44
Publishing: Topic=sensors/Distance/value, Payload=14.64,2023-06-13T06:26:44
Publishing: Topic=sensors/Fire sensor/value, Payload=0.92,2023-06-13T06:26:44
Publishing: Topic=sensors/Gas sensor/value, Payload=0.09,2023-06-13T06:26:44
Publishing: Topic=sensors/Water level/value, Payload=0.00,2023-06-13T06:26:44
Publishing: Topic=sensors/Soil level/value, Payload=0.00,2023-06-13T06:26:44
Publishing task state to topic: tasks/state/1/update
New state: OFF
Task state published.
Publishing task state to topic: tasks/state/2/update
New state: OFF
Task state published.
Publishing task state to topic: tasks/state/3/update
New state: OFF
Task state published.
Publishing task state to topic: tasks/state/4/update
New state: OFF
Task state published.
```

Joonis 3.9 Arduino andmete saatmine MQTT kaudu

3.5.3. readSendData()

Funktsioon readSendData() (vt LISA 2) teostab andmete lugemist erinevatelt anduritelt, vormib neist andmetest JSON-struktuuri ja saadab selle serverisse. Samuti uuendab funktsioon täiturmehhanismide olekuid nutikas kodus.

Funktsiooni samm-sammuline analüüs:

- Seadmete oleku uuendamine:
    - Uuendatakse ja saadetakse serverisse nelja seadme: servoajami, lambi, releede ja piesoelektrilise kõlari praegused olekud.
- Praeguse aja saamine:
    - Kasutades funktsiooni getCurrentTimestamp(), saadakse praegune ajatempel, mis lisatakse iga anduri andmetele.
- JSON-struktuuri initsialiseerimine:
    - Luua dünaamiline JSON-dokument ja andurite andmete massiiv.
- Andurite andmete lugemine ja nende JSON-i lisamine:
    - Kasutatakse tsüklit, et läbi käia kõik massiivis *sensors* määratletud andurid.
    - Iga anduri jaoks kutsutakse välja vastav andmete lugemise funktsioon

(näiteks readBMP085Temp BMP085 temperatuurianduri jaoks).

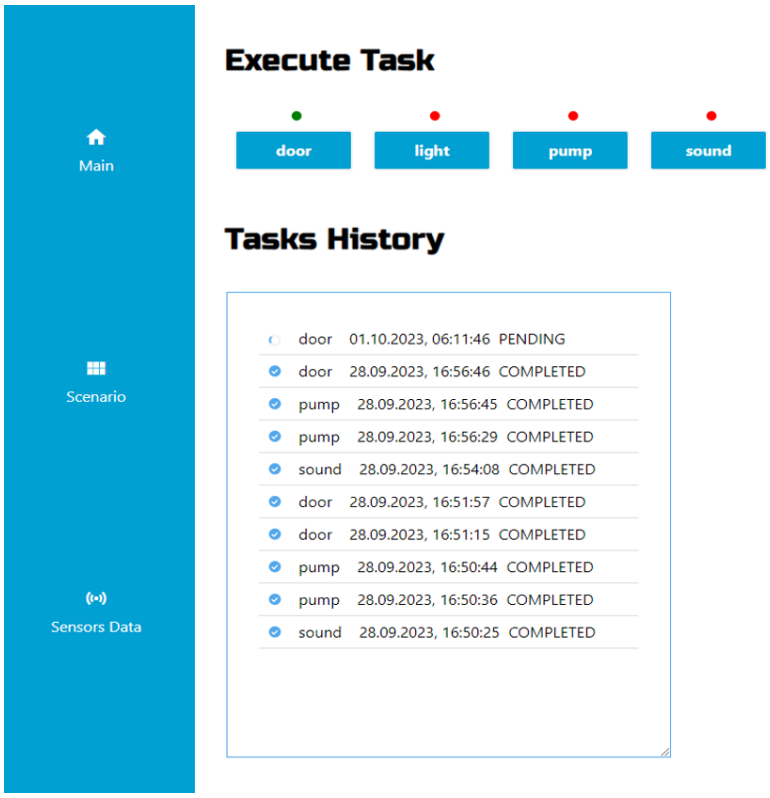
- Loetud andmed salvestatakse vastavasse muutujasse, näiteks temperatuur BMP085 puhul.
- Luua uus JSON-objekt andurite andmete massiivis ja sellesse objekti lisatakse anduri nimi, loetud väärtus ja ajatempel.

- Andmete saatmine serverisse:
  - Kasutatakse funktsiooni sendAllSensorData(sensorDataArray) kõigi andurite andmete serverisse saatmiseks. See funktsioon võtab sisendiks andurite andmete massiivi, koostab MQTT sõnumi teema ja kasuliku koormuse ning teostab andmete avaldamist serveris.

Kokkuvõttes automatiseerib funktsioon readSendData() anduritelt andmete kogumise, nende vormistamise ja serverisse saatmise protsessi. Kogu protsess toimub tsüklilisel viisil iga anduri jaoks, mis tagab süsteemi paindlikkuse ja skaleeritavuse.

3.6. Kasutajaliides

Rakenduse kasutusmugavuse tagamiseks on autor välja töötanud selge ja intuitiivselt arusaadava kasutajaliidese (vt Joonis 3.10).



Joonis 3.10 Kasutajaliides

Peamiseks liidese elemendiks on navigatsioonipaneel, mis sisaldab järgmisi jaotisi:

- Main (Pealeht): See jaotis on nutikodu süsteemi juhtimiskeskus. Siin saab kasutaja käsitsi juhtida erinevaid seadmeid, jälgides nende praegust olekut. Lisaks pakutakse viimase kümne nutikodus loodud ülesande ajalugu, et automatiseerida tegevusi.
- Scenario (Stsenaariumid): Jaotis, mis on mõeldud kasutaja loovuseks. Siin saab luua oma stsenaariume erinevate protsesside automatiseerimiseks nutikodus.
- Sensors Data (Andurite andmed): Selles jaotises saavad kasutajad jälgida erinevate andurite praeguseid näite ja vaadata nende ajalugu. See võimaldab süsteemis toimuvatele muutustele kiiresti reageerida.

Kokkuvõttes sisaldab antud liides kõiki vajalikke elemente mugavaks juhtimiseks ning on tavalisele kasutajale intuitiivselt arusaadav.

3.7. Andmete visualiseerimine

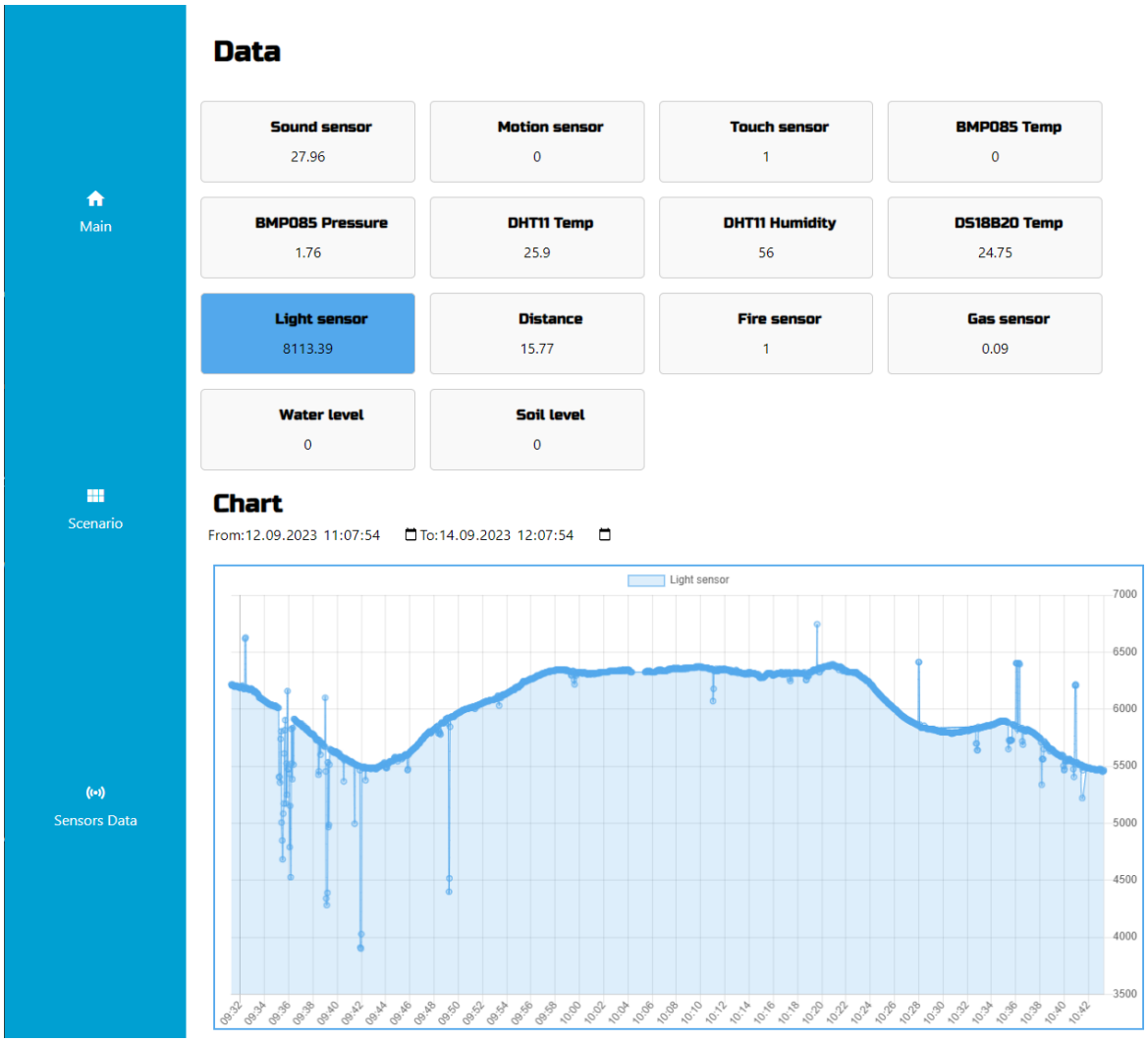
Andurite andmete visualiseerimise aluseks on front-end komponent, mida esindavad kaks malli (vt Joonis 3.11). Need mallid kuvatakse kasutajale interaktiivsete kaartidena iga anduri jaoks ja ajas muutuvate näitajate graafikutena.

3.7.1. Andurite kaardid

Komponent SensorsViewCarts loob võrgustiku (sensor-grid) andurite kaartidega (sensor-card). Iga kaart kujutab endast anduri nime ja selle praegust väärtust. Kui kasutaja klõpsab kaardil, rõhutatakse valitud andurit ja selle andmed võivad olla kuvatud graafikul.

3.7.2. Andurite näitajate muutuste graafik

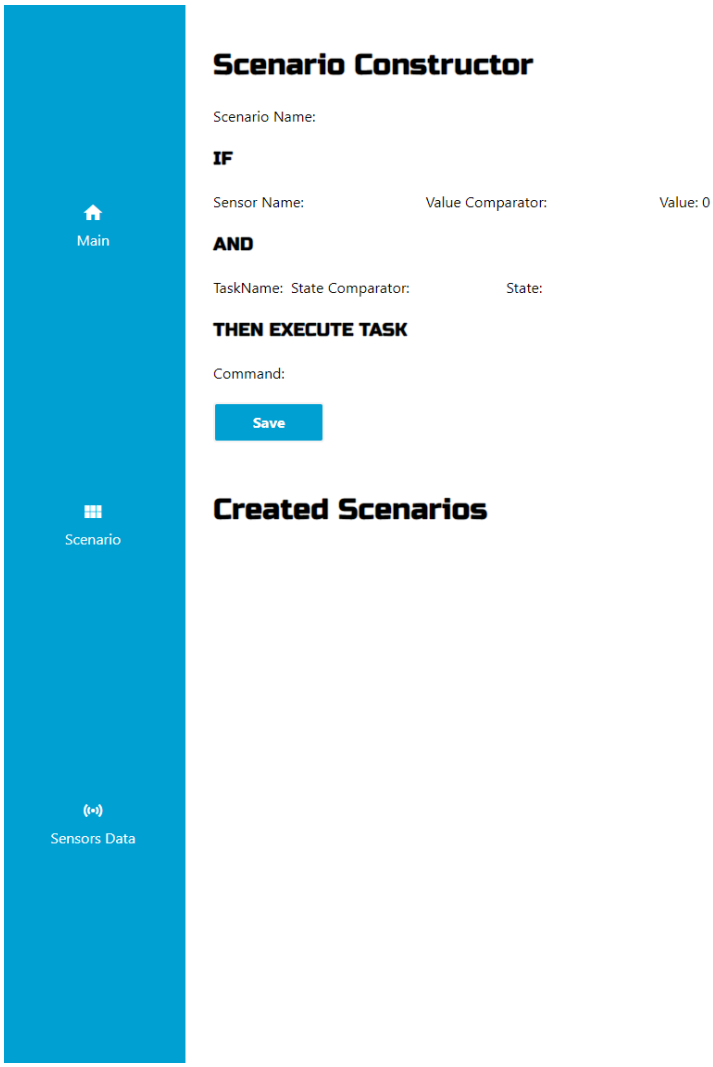
Komponent SensorChart kuvab valitud anduri näitajate muutuste graafiku. Kasutaja saab valida teda huvitava ajavahemiku graafikul kuvamiseks, kasutades *datetime-local* tüüpi sisendvälju. Graafiku renderdamiseks kasutatakse Chart.js teeki koos chartjs-plugin-zoom pistikprogrammiga, mis lisab graafiku suumimise funktsionaalsuse.



Joonis 3.11 Andmete visualiseerimine

3.8. Stsenaariumide konstruktor

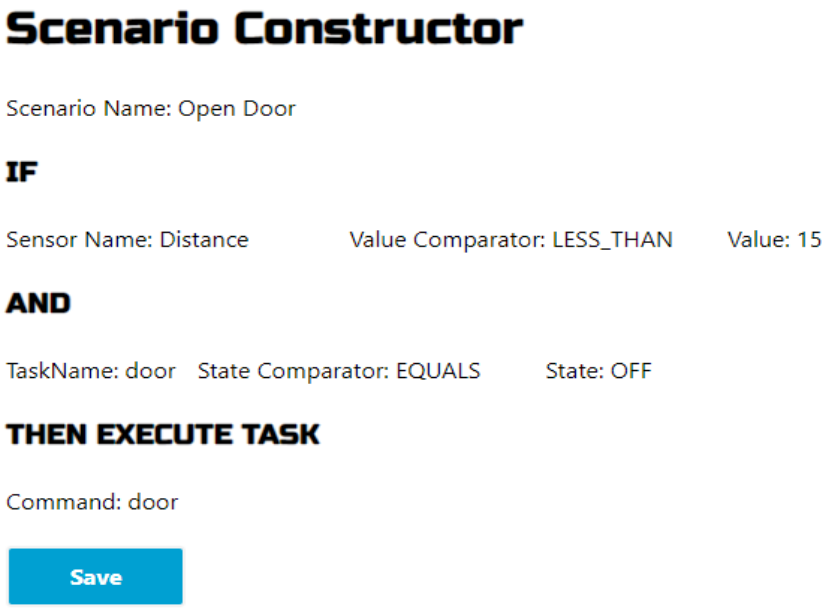
Erinevate protsesside automatiseerimiseks nutikodu süsteemis on diplomitöö autor välja töötanud stsenaariumite konstruktori. See konstruktor võimaldab kasutajal valida vajaliku anduri aktiivsete andurite loendist ja selle näitude põhjal koostada tegevusalgoritmi täiturmehhanismidele. Joonisel 3.12 on kujutatud selle konstruktori kasutajaliides.



Joonis 3.12 Stsenaariumide konstruktor

3.8.1. Stsenaariumi loomise algoritm

Diplomitöö autor otsustas stsenaariumi loomise protsessi selgitamiseks demonstratiivselt näidata ukse avamise ja sulgemise algoritmi. Joonisel 3.13 on esitatud ukse avamise algoritm.



Joonis 3.13 Open Door Algoritm

Esmalt tuleb sisestada kavandatava algoritmi nimetus.

Liides jaguneb kolmeks peamiseks jaotiseks: "IF", "AND" ja "THEN EXECUTE TASK".

Jaotises "IF":

- "Autor kasutab rippmenüüst "Sensor Name" andurit "Distance", mis näitab lähima objekti kaugust.
- Jaotises "Value Comparator" tuleb valida "LESS_THAN", mis näitab, et anduri näidud peavad olema madalamad kindlaksmääratud lävendist.
- Väljale "Value" sisestab autor väärtuse "15", eeldades, et käivituslävend on alla 15 mõõtühiku.

Jaotises "AND":

- Rippmenüüst "TaskName" valib autor "door", mis näitab "ukse" täiturmehhanismi kasutamist.
- Jaotises "State Comparator" tuleb valida "EQUALS", mis näitab, et täiturmehhanismi praegust olekut võrreldakse kasutaja poolt määratud olekuga.
- Jaotises "State" valis autor "OFF", lähtudes sellest, et algolekus peaks uks olema suletud.

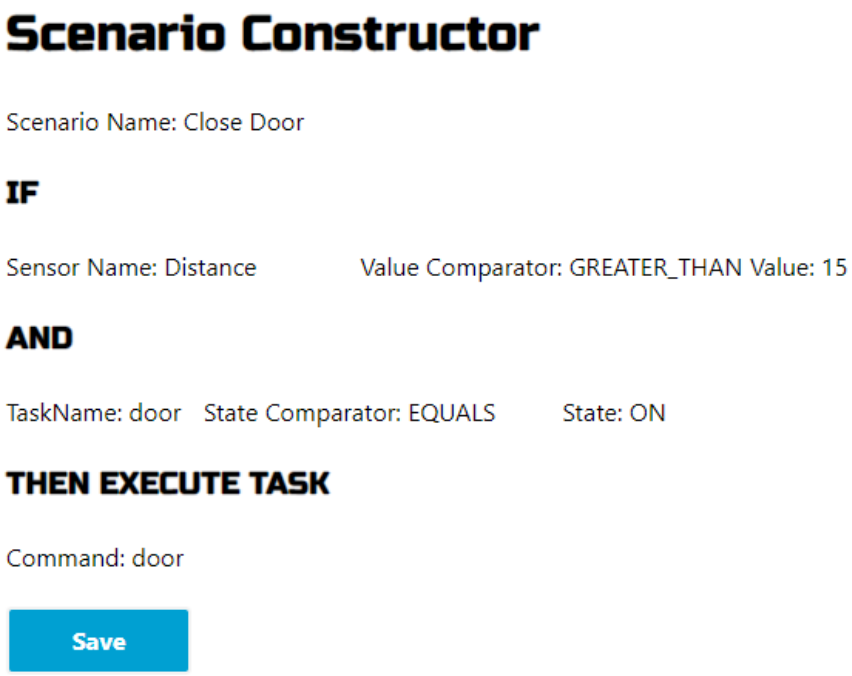
Jaotises "THEN EXECUTE TASK":

- Rippmenüüst "Command" tuleb valida ukse avamise käsk ("door"). See tähendab, et kui kõik ülalnimetatud tingimused ühtivad valitud tingimustega, saadetakse Arduinole käsk täiturmehhanismi "door" oleku "OFF" muutmiseks olekuks "ON" ja uks avaneb.

Protsessi lõpetamiseks tuleb vajutada nuppu "Save".

See algoritm toimib järgmiselt: kui mõni objekt asub lähemal kui 15 mõõtühikut ja praegu on uks suletud, aktiveerib nutikodu süsteem ukse avamise käsu.

Ukse sulgemise protsessi automatiseerimiseks tuleb luua vastupidine algoritm sellele, mis oli välja töötatud ukse avamiseks, muutes ainult kahte parameetrit (vt Joonis 3.14).



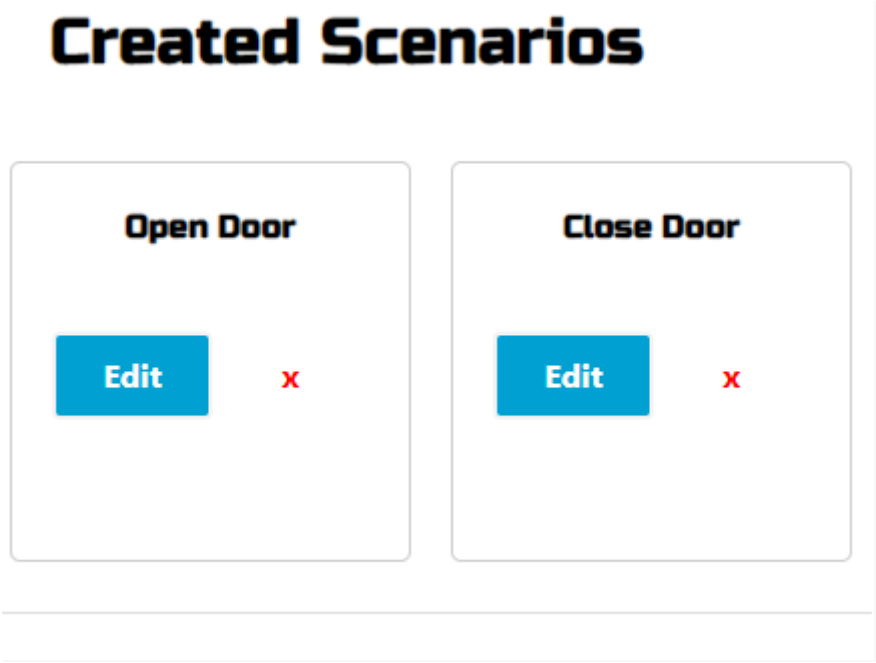
Joonis 3.14 Close Door Algoritm

Muudatused, mida on vaja teha:

- Jaotises "IF" alajaotises "Value Comparator" tuleb valida "GREATER_THAN".
- Jaotises "AND" alajaotises "State" tuleb näidata "ON".

See algoritm töötab ja sulgeb ukse, kui lähima objekti kaugus on suurem kui 15 mõõtühikut ja praegu on uks avatud.

Nii kujundatakse kaks algoritmi ukse automaatseks juhtimiseks: üks avamiseks, teine sulgemiseks, sõltuvalt kasutaja poolt määratud tingimustest.



Joonis 3.15 Loodud algoritmid

Vajadusel saab kumbagi stsenaariumit redigeerida või kustutada (vt Joonis 3.15).

3.9. Turvalisus

Kaasaegses digiajastus seisab rakenduste turvalisus esiplaanil. Rakenduses kasutatakse Spring Boot tehnoloogiaid veebiteenuste jaoks ning Mosquitto MQTT serverina. Selles jaotises käsitleb autor rakendatavaid turvameetmeid ja võimalusi süsteemi turvalisuse edasiseks tugevdamiseks.

3.9.1. Spring Security

Spring Security on võimas ja kohandatav raamistik Java-rakenduste autentimiseks ja autoriseerimiseks. See pakub laia valikut turvafunktsioone, alates autentimisest kuni erinevat tüüpi rünnakute vastase kaitse pakkumiseni. [20]

Rakendatud turvameetmed:

Nutikodu veebirakenduse turvalisuse tagamiseks on autor projekti integreerinud Spring Security kaitsemehhanismid. Selles jaotises vaadeldakse üksikasjalikult turvalisuse konfiguratsiooni põhielemente koos koodi illustreerimise ja nende funktsionaalse eesmärgiga rakenduse kontekstis. Täielik turvaseadistus on kujutatud joonisel 3

```java
@Configuration
@EnableWebSecurity
public class SecurityConfig extends WebSecurityConfigurerAdapter {

    qubinets
    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http
                .authorizeRequests() ExpressionUrlAuthorizationConfigurer<...>.ExpressionInterceptUrlRegistry
                .antMatchers( ...antPatterns: "/tasks/pending").permitAll()
                .antMatchers( ...antPatterns: "/**").authenticated()
                .and() HttpSecurity
                .httpBasic() HttpBasicConfigurer<HttpSecurity>
                .and() HttpSecurity
                .formLogin() FormLoginConfigurer<HttpSecurity>
                .and() HttpSecurity
                .csrf() CsrfConfigurer<HttpSecurity>
                .disable();
    }

    qubinets
    @Autowired
    public void configureGlobal(AuthenticationManagerBuilder auth) throws Exception {
        auth.inMemoryAuthentication() InMemoryUserDetailsManagerConfigurer<AuthenticationManagerBuilder>
                .withUser( username: "taltech") UserDetailsManagerConfigurer<...>.UserDetailsBuilder
                .password("{noop}vk_2023")
                .roles("USER");
    }
}
```

Joonis 3.16 Spring Security kood

Järgnevalt esitatakse peamised kaitsemehhanismid, mida autor on selle koodi abil rakendanud.

HTTP Basic Authentication:

- Konfiguratsioonikoodis httpBasic() aktiveerib põhilise HTTP-autentimise, kus kasutajatunnused (kasutajanimi ja parool) edastatakse iga päringuga Authorization päises. Sellist autentimise meetodit on lihtne rakendada, kuid kasutajatunnuste kaitsmiseks on tungivalt soovitatav kasutada krüpteeritud ühendust HTTPS-i kaudu.

Form-based Authentication:

- Direktiiv formLogin() võimaldab Spring Security sisseehitatud sisselogimisvormi kaudu autentimist, pakkudes kasutajatele liidest oma kasutajatunnuste sisestamiseks. See lähenemine võimaldab keerukama sisselogimisloogika

rakendamist ja pakub suuremat kontrolli autentimisprotsessi üle.

In-Memory Authentication:

- Metoodika inMemoryAuthentication() abil määratakse kasutajatunnused, mis hoitakse otse rakenduse mälus ja on testimiseks mugavad. Sel juhul määratakse kasutajale taltech parool vk_2023. Käsitletav konfiguratsioon on kasutusel ainult demonstratsiooni eesmärgil ja ei sobi reaalseks ekspluatatsiooniks.

URL-based Authorization:

- Rakenduse erinevate osade juurdepääsu kontrollimiseks kasutatakse URL-põhist autoriseerimist. Konfiguratsioon määrab, et tee /tasks/pending on kõigile kasutajatele avatud (permitAll()), samas kui kõikidele teistele URL-idele juurdepääs nõuab autentimist (authenticated()). See võimaldab kasutajaõiguste detailset haldamist, piirates või andes juurdepääsu kindlatele rakenduse ressurssidele.

CSRF Protection:

- Vaikimisi lülitab Spring Security sisse kaitse Cross-Site Request Forgery (CSRF) tüüpi rünnakute vastu. Sel juhul on CSRF-kaitse demonstratsiooni eesmärgil keelatud direktiiviga csrf().disable(). Siiski on reaalsetes rakendustes soovitatav jätta CSRF-kaitse sisse lülitatuks, et suurendada turvalisust.

Spring Security kaitsemehhanismide kasutamine võimaldab luua usaldusväärse ja turvalise süsteemi, mis on efektiivselt vastupidav enamikule standardsetele võrgurünnakutele ja haavatavustele, tagades kasutajatunnuste kaitse ja juurdepääsu kontrolli veebirakenduse ressurssidele.

3.9.2. MQTT autentimine

Autentimine kasutajanime ja parooli alusel on MQTT turvalisuse põhimeetod ja seda kasutatakse autori rakenduses täiendava kaitsemeetmena. Järgnevalt on kirjeldatud, kuidas see töötab.

Klient, üritades ühenduda MQTT-maakleriga, saadab oma kasutajanime ja parooli. Maakler seejärel kontrollib neid kasutajatunnuseid vastavuses andmebaasi või muu autentimismehhanismiga. Sellel lähenemisel on nii eeliseid kui ka märgatavaid puudusi.

Eelised:

- Lihtne rakendamine: Ei nõua keerulist infrastruktuuri ega täiendavat tarkvara.
- Ühilduvus: Enamik MQTT-maaklereid toetab seda autentimismeetodit kohe karbist välja.

Puudused:

- Avalik edastamine: Kui ühendus ei ole SSL/TLS-i abil kaitstud, võidakse kasutajatunnuseid edastada avalikult, muutes need haavatavaks pealtkuulamise suhtes.
- Staatiline olemus: Kasutajanimed ja paroolid ei muutu iga sessiooniga, mis suurendab nende kompromiteerimise riski.
- Paroolihaldus: Nõuab regulaarset paroolide uuendamist ja hoidmist, mis võib suure hulga seadmete korral muutuda keeruliseks ülesandeks.

KOKKUVÕTE

Asjade interneti (IoT) integratsiooni kiirendumine meie igapäevaelus avab uusi väljavaateid ja võimalusi nutikate tehnoloogiate jaoks. Käesoleva lõputöö peamine eesmärk oli arendada IoT-projekti jaoks veebirakendus, mis demonstreerib kaasaegseid võimalusi ja tulevikupotentsiaali nutikodu valdkonnas.

Esimeses peatükis tutvustab autor olemasoleva nutikodu prototüüpi, kirjeldab selle põhielemente ja funktsioone. Teises peatükis analüüsib projekti teostamiseks valitud tehnoloogiaid ning kaalub alternatiivseid lahendusi, mis võiksid olla kasutusel sarnaste süsteemide loomisel. Autor selgitab üksikasjalikult tehnoloogiliste vahendite valiku põhjuseid.

Töö keskmes on kolmas peatükk, kus lõputöö autor kirjeldab nutikodu funktsionaalsuste arendamist. See hõlmab käsitsijuhtimist, sensoriandmetel põhinevaid automatiseeritud stsenaariume, andmete salvestamist ja visualiseerimist ning kasutajasõbraliku liidese väljatöötamist Vue.js raamistiku abil. Spring Booti abil loodud back-end võimaldab andmete tõhusat töötlemist ning erinevate komponentide integreerimist ühtseks süsteemiks. Lisaks on autor rakendanud põhilised turvameetmed, kasutades Spring Securityt ja MQTT protokolli autentimist, mis suurendab süsteemi usaldusväärsust.

Töö lõpuosas toob autor välja projekti edasise arendamise perspektiivid, mis hõlmavad prototüübi muutmist realistlikumaks mudeliks, videoseire lisamist ja tehisintellektil põhinevate tuvastusalgoritmide integreerimist, turvalisuse tugevdamist kasutajate andmebaasi loomise ja juurdepääsuõiguste eristamisega ning SSL/TLS krüpteerimise kasutamist MQTT-kommunikatsioonis. Kuigi praeguse projekti jaoks on valitud Arduino platvorm, kaalub autor keerukamate lahenduste jaoks Raspberry Pi kasutuselevõttu.

SUMMARY

Juri Kubinets's thesis 'Application creation for smart house prototype' focuses on the development of an application for an Arduino-based smart home prototype, prompted by the removal of the Blynk application from the platform. This necessitated the creation of a personal application that is independent of third-party platforms, free for users, and capable of operating offline. The primary tasks included updating and modernizing the Arduino code, ensuring efficient data exchange and reliable storage in the system using MQTT and REST API. Considering the need for offline functionality and free user access, the author chose a local PostgreSQL-based database.

The third task was the development of a user-friendly interface and data visualization, utilizing the Vue.js framework. The fourth task involved creating a constructor for smart home automation based on data received from sensors. The integration of all these components into a functioning whole was achieved using the Spring Boot framework.

The result of this thesis is to demonstrate how various technologies can effectively collaborate to create an optimized and user-friendly system for a smart home prototype, showcasing the importance and potential of IoT technologies in both theoretical and practical aspects. The work includes an overview of the smart home prototype, its components, and characteristics. Additionally, the author discusses the technologies used to achieve the goals of the thesis, as well as alternative technologies that could be used for a similar project, justifying the choices made.

The main part of the thesis is dedicated to the implementation of the application for the smart home, covering all components created by the author for automation. All the objectives and tasks set by the author were successfully accomplished, resulting in an application that allows controlling the smart home both manually and through scenarios based on sensor data. The application also features data visualization, allowing the user to see historical data for each sensor at a selected time. The author also implemented a basic security system using Spring Security and basic MQTT protocol authentication.

For future development, the author sees the transformation of the smart home prototype into a more realistic form, the addition of a camera and AI recognition algorithms, and improving security through the creation of a user database with roles and access, using SSL/TLS encryption for MQTT. Despite considering Arduino preferable for this project, for more serious projects the author would switch from Arduino to the latest Raspberry Pi.

KASUTATUD KIRJANDUSE LOETELU

1. Arduino FAQ. [Online] https://www.arduino.cc/en/Guide/Introduction (04.09.2023).
2. Arduino Hardware. [Online] https://www.arduino.cc/en/hardware (10.09.2023).
3. Arduino Software. [Online] https://www.arduino.cc/en/software (20.10.2023).
4. Arduino Project Hub. [Online] https://projecthub.arduino.cc (20.10.2023).
5. Spring Boot Overview. [Online] https://spring.io/projects/spring-boot#overview (15.10.2023).
6. Django Documentation. [Online] https://docs.djangoproject.com/en/4.2/ (25.10.2023).
7. Max Pekarsky. WebSocket for fun and profit. [Online] https://stackoverflow.blog/2019/12/18/websockets-for-fun-and-profit/ (01.11.2023).
8. MQTT vs WebSocket Comparison. [Online] https://www.geeksforgeeks.org/mqtt-vs-websocket/ (23.10.2023).
9. MQTT FAQ. [Online] https://mqtt.org/faq/ (20.10.2023).
10. About PostgreSQL. [Online] https://www.postgresql.org/about/ (20.10.2023).
11. Cyber Success. Django vs Spring Boot 2023 – Know The Differences. [Online] https://www.cybersuccess.biz/django-vs-spring-boot (22.10.2023).
12. Lance Cassidy. Arduino Vs Raspberry Pi: Which Is the Best Board for You. [Online] https://www.flux.ai/p/blog/arduino-vs-raspberry-pi-comparison (22.10.2023).
13. The Difference Between MySQL vs PostgreSQL. [Online] https://aws.amazon.com/ru/compare/the-difference-between-mysql-vs-postgresql (22.10.2023).
14. Mohit Joshi. Angular vs React vs Vue: Core Differences. [Online] https://www.browserstack.com/guide/angular-vs-react-vs-vue (07.10.2023).
15. Vue.js Introduction. [Online] https://vuejs.org/guide/introduction.html (23.10.2023).
16. Chart.js Documentation. [Online] https://www.chartjs.org/docs/latest/ (02.09.2023).

17. Zartis Team. Angular vs React vs Vue.js: A Comparative Study – 2022. [Online] https://www.zartis.com/angular-vs-react-vs-vuejs/ (23.10.2023).
18. Vue-chartjs Guide. [Online] https://vue-chartjs.org/guide/ (23.10.2023).
19. IntelliJ IDEA Overview. [Online] https://www.jetbrains.com/idea/ (10.09.2023).
20. Spring Security Project. [Online] https://spring.io/projects/spring-security (01.11.2023).

LISA 1 ARDUINO CHECKPENDINGTASKS() FUNKTSIOON

```
void checkPendingTasks() {
  EthernetClient ethClient;
  HttpClient httpClient(ethClient, server, port);
  Serial.println("Checking for pending tasks...");
  int result = httpClient.get("/tasks/pending");
  int statusCode = httpClient.responseStatusCode();
  if (statusCode == 200) {
    // Processing server response
    httpClient.skipResponseHeaders();
    String response = "";
    while (httpClient.connected() || httpClient.available()) {
      char c = httpClient.read();
      response += c;
    }
    // Parsing JSON response and executing actions based on command
    StaticJsonDocument<1024> doc;
    DeserializationError error = deserializeJson(doc, response);
    if (error) {
      Serial.print("Error parsing JSON: ");
      Serial.println(error.c_str());
      return;
    }
    JsonArray tasks = doc.as<JsonArray>();
    for (JsonObject task : tasks) {
      int taskId = task["id"];
      const char *command = task["command"];
      // Executing actions based on command
      if (strcmp(command, "door") == 0) {
        // Perform action for "door" command
        if (!servo_open) {
          // Open door to 90 degrees
          servo_motor.write(0);
          servo_open = true;
        } else {
          // Close door (return to initial position)
          servo_motor.write(100);
          servo_open = false;
        }
        sendTaskCompleted(taskId);
      } else if (strcmp(command, "pump") == 0) {
        // Perform action for "pump" command
        relay_state = !relay_state;
        digitalWrite(RELAY_PIN, relay_state);
        sendTaskCompleted(taskId);
      } else if (strcmp(command, "light") == 0) {
```

```
        // Control light
        lamp_state = !lamp_state;
        int light_ctl = lamp_state ? 255 : 0;
        analogWrite(MOSFET_PIN, light_ctl);
        sendTaskCompleted(taskId);
      } else if (strcmp(command, "sound") == 0) {
        // Control piezo buzzer
        buzzer_state = !buzzer_state;
        if (buzzer_state) {
          tone(BUZZER_PIN, 440, 0);
        } else {
          noTone(BUZZER_PIN);
        }
        sendTaskCompleted(taskId);
      } else {
        Serial.print("Unknown command: ");
        Serial.println(command);
      }
    }
  } else {
    // Error in fetching pending tasks
  }
  httpClient.stop();
}

// Sending task completed using MQTT
void sendTaskCompleted(int taskId) {
  String topic = "tasks/completed/update";
  String payload = String(taskId) + ",COMPLETED";
  Serial.print("Publishing: Topic=");
  Serial.print(topic);
  Serial.print(", Payload=");
  Serial.println(payload);
  mqttClient.publish(topic.c_str(), payload.c_str());
}
```

LISA 2 ARDUINO READSENDDATA() FUNKTSIOON

```
void readSendData() {
  // Update the current states
  publishTaskState(1, servo_open ? "ON" : "OFF");
  publishTaskState(2, lamp_state ? "ON" : "OFF");
  publishTaskState(3, relay_state ? "ON" : "OFF");
  publishTaskState(4, buzzer_state ? "ON" : "OFF");

  String timestamp = getCurrentTimestamp();

  // Reading data from sensors and adding them to JSON
  DynamicJsonDocument doc(1024);
  JsonArray sensorDataArray = doc.to<JsonArray>();

  for (int i = 0; i < sizeof(sensors) / sizeof(sensors[0]); i++) {
    *sensors[i].value = sensors[i].readFunc();
    JsonObject sensorData = sensorDataArray.createNestedObject();
    sensorData["name"] = sensors[i].name;
    sensorData["value"] = *sensors[i].value;
    sensorData["timestamp"] = timestamp;
  }

  // Sending all sensor data at once
  sendAllSensorData(sensorDataArray);
  lcd.clear();
  lcd_printstr("SMART HOUSE!");
}
//Sending data to the server
void sendAllSensorData(JsonArray sensorDataArray) {
  for (JsonObject sensorData : sensorDataArray) {
    String name = sensorData["name"].as<String>();
    float value = sensorData["value"].as<float>();
    String timestamp = sensorData["timestamp"].as<String>();

    String topic = "sensors/" + name + "/value";
    String payload = String(value) + "," + timestamp;

    Serial.print("Publishing: Topic=");
    Serial.print(topic);
    Serial.print(", Payload=");
    Serial.println(payload);
    mqttClient.publish(topic.c_str(), payload.c_str());
  }
}
```

LISA 3 SPRING BOOT MQTTCONFIGURATION

```
@Configuration
public class MqttConfiguration {
    @Value("${mqtt.broker.url}")
    private String mqttBrokerUrl;
    @Value("${mqtt.broker.username}")
    private String mqttUsername;
    @Value("${mqtt.broker.password}")
    private String mqttPassword;
    private static final Logger LOGGER =
LoggerFactory.getLogger(MqttConfiguration.class.getName());
    // HashSet for sent tasks
    private final Set<Long> sentTasks = new HashSet<>();

    @Bean
    public MqttPahoClientFactory mqttClientFactory() {
        DefaultMqttPahoClientFactory factory = new
DefaultMqttPahoClientFactory();
        MqttConnectOptions options = new MqttConnectOptions();
        options.setServerURIs(new String[] { mqttBrokerUrl });
        options.setUserName(mqttUsername);
        options.setPassword(mqttPassword.toCharArray());
        factory.setConnectionOptions(options);
        return factory;
    }
    @Bean
    public MessageChannel mqttInputChannel() {
        return new DirectChannel();
    }
    @Bean
    public MessageChannel mqttCompletedTasksInputChannel() {
        return new DirectChannel();
    }
    @Bean
    public MessageChannel mqttSensorDataInputChannel() {
        return new DirectChannel();
    }
    @Bean
    public MessageChannel mqttTasksRequestInputChannel() {
        return new DirectChannel();
    }

    @Bean
    public MessageProducer mqttInbound() {
        MqttPahoMessageDrivenChannelAdapter adapter =
                new
MqttPahoMessageDrivenChannelAdapter("springBootClient",
mqttClientFactory(), "tasks/state/+/update");
        adapter.setCompletionTimeout(5000);
        adapter.setConverter(new DefaultPahoMessageConverter());
        adapter.setQos(1);
        adapter.setOutputChannel(mqttInputChannel());
        return adapter;
```

```
    }
    @Bean
    public MessageProducer mqttCompletedTasksInbound() {
        MqttPahoMessageDrivenChannelAdapter adapter =
                new
MqttPahoMessageDrivenChannelAdapter("springBootClientCompleted",
mqttClientFactory(), "tasks/completed/update");
        adapter.setCompletionTimeout(5000);
        adapter.setConverter(new DefaultPahoMessageConverter());
        adapter.setQos(1);
        adapter.setOutputChannel(mqttCompletedTasksInputChannel());
        return adapter;
    }
    @Bean
    public MessageProducer mqttSensorDataInbound() {
        MqttPahoMessageDrivenChannelAdapter adapter =
                new
MqttPahoMessageDrivenChannelAdapter("springBootClientSensorData",
mqttClientFactory(), "sensors/+/value");
        adapter.setCompletionTimeout(5000);
        adapter.setConverter(new DefaultPahoMessageConverter());
        adapter.setQos(1);
        adapter.setOutputChannel(mqttSensorDataInputChannel());
        return adapter;
    }
    @Bean
    public MessageProducer mqttTasksRequestInbound() {
        MqttPahoMessageDrivenChannelAdapter adapter =
                new
MqttPahoMessageDrivenChannelAdapter("springBootClientTasksRequest",
mqttClientFactory(), "tasks/request");
        adapter.setCompletionTimeout(5000);
        adapter.setConverter(new DefaultPahoMessageConverter());
        adapter.setQos(1);
        adapter.setOutputChannel(mqttTasksRequestInputChannel());
        return adapter;
    }
    @Bean
    @ServiceActivator(inputChannel = "mqttInputChannel")
    public MessageHandler mqttMessageHandler(AcutatorStateRepository
acutatorStateRepository) {
        return message -> {
            String topic = (String)
message.getHeaders().get(MqttHeaders.RECEIVED_TOPIC);
            String newState = message.getPayload().toString();
            LOGGER.info("Received MQTT message: Topic={}, State={}",
topic, newState);
            Long taskId = Long.parseLong(topic.split("/")[2]);
            Actuator_State task =
acutatorStateRepository.findById(taskId)
                    .orElseThrow(() -> new
IllegalArgumentException("Invalid task id: " + taskId));
            Actuator_State.State newTaskState =
Actuator_State.State.valueOf(newState);
```

```
            if (newTaskState != task.getTaskState()) {
                task.setTaskState(newTaskState);
            }
            acutatorStateRepository.save(task);
            LOGGER.info("Updated task state: TaskId={}, NewState={}",
taskId, newTaskState);
        };
    }
    @Bean
    @ServiceActivator(inputChannel = "mqttCompletedTasksInputChannel")
    public MessageHandler
mqttCompletedTasksMessageHandler(TaskRepository taskRepository) {
        return message -> {
            String payload = message.getPayload().toString();
            String[] parts = payload.split(",");
            Long taskId = Long.parseLong(parts[0]);
            LOGGER.info("Received MQTT Status updated: TaskId={},
COMPLETED", taskId);
            Task task = taskRepository.findById(taskId)
                    .orElseThrow(() -> new
IllegalArgumentException("Invalid task id: " + taskId));
                task.setStatus(Status.COMPLETED);
            taskRepository.save(task);
        };
    }
    @Bean
    @ServiceActivator(inputChannel = "mqttSensorDataInputChannel")
    public MessageHandler mqttSensorDataMessageHandler(SensorService
sensorService) {
        return message -> {
            try {
                String topic = (String)
message.getHeaders().get(MqttHeaders.RECEIVED_TOPIC);
                String payload = message.getPayload().toString();
                String[] parts = payload.split(",");
                String sensorName = topic.split("/")[1];
                float value;
                if (parts[0].equalsIgnoreCase("NAN")) {
                    value = Float.NaN; //NAN means that the sensor is
not available.To avoid errors converting it to Float.NaN
                } else {
                    value = Float.parseFloat(parts[0]);
                }
                String timestamp = parts[1];
                LOGGER.info("Received MQTT Sensor Data: SensorName={},
Value={}, Timestamp={}", sensorName, value, timestamp);
                Sensor sensor = new Sensor();
                sensor.setName(sensorName);
                sensor.setValue(value);
                sensor.setTimestamp(LocalDateTime.parse(timestamp));
                sensorService.save(sensor);
            } catch (NumberFormatException e) {
                //Exeptions logging
                LOGGER.error("Error parsing float value: ", e);
```

```
                }
            };
        }
    }
```