Tampereen ammattikorkeakoulu

Hemuli REST-rajapinnan simulaattori

Lari Liuhamo

OPINNÄYTETYÖ
Syyskuu 2022

Tieto- ja viestintätekniikka
Ohjelmistotekniikka

TIIVISTELMÄ

Tampereen ammattikorkeakoulu
Tieto- ja viestintätekniikka
Ohjelmistotekniikka

LIUHAMO, LARI:
Hemuli REST-rajapinnan simulaattori

Opinnäytetyö 13 sivua
Syyskuu 2022

Työn aiheena on asiakkaan Hemuli-ohjelmiston rajapintaa simuloiva REST-rajapinta Wisematicin sisäisen projektin testaamista varten ilman pääsyä asiakkaan omiin työkaluihin.

Hemuli on asiakkaan bisneslogiikkaa hallinnoiva ohjelma, jolle Wisematicin tuottama projekti keskustelee selvittääkseen, mitä sen halutaan milloinkin tekevän. Lisäksi se hoitaa työt, jotka eivät ole Wisematicin osuuden vastuulla. Tämä tavoite saavutetaan simuloimalla alkuperäistä rajapintaa, ottaen huomioon yksityiskohtia kuten tuotetun projektin sekä asiakkaan palvelinten, kuormitustasaajien, että tiedontallennuksen välinen latenssi, ja asiakkaalta saatu palaute rajapinnan simuloinnin tarkkuudesta. Hemuli API tarjoaa myös graafisen hallintakäyttöliittymän tuotetun projektin ohjausta varten, mahdollistaen Wisematicin kehittäjille projektin helpomman testauksen ilman, että tähän tarvitsisi luoda uusi ohjelmisto. Tämä hallintakäyttöliittymä on saatavilla Wisematicin kehitystiimin intranetissä kaikilla verkkoselaimilla, ja se tukee myös matkapuhelinselaimia.

Asiasanat: rest, rajapinta, testaus, simulaattori, latenssi, hallinta, verkkoselain

ABSTRACT

Tampereen ammattikorkeakoulu
Tampere University of Applied Sciences
Information and communications technology
Software engineering

LIUHAMO, LARI:
Hemuli REST API Simulator

Bachelor's thesis 13 pages
September 2022

The goal of this project is to create a REST API that simulates the customer's Hemuli API to enable testing Wisematic's internal software product without being given access to the customer's API directly.

The Hemuli API manages the customer's business logic and manages interactions with Wisematic's product, telling it what to do on a high level and managing the things that aren't in the scope of Wisematic's product. This is achieved by accurately mocking the original API, taking into account latency between the product and the customer's own servers, load balancing and data storage, and through feedback from the customer to confirm the mocked API matches the real API. The Hemuli API additionally provides a graphical management interface for controlling the product, enabling easier testing for Wisematic's developers without needing to create a separate software for that purpose alone. The management interface is accessible via any web browser in Wisematic's developer intranet and is even compatible with mobile devices.

Key words: rest, api, testing, simulator, latency, management, web browser

SISÄLLYS

1 JOHDANTO 9
2 TEORIA 11
2.1 HTTP-protokolla 11
2.2 HTTP-metodit 12
2.3 REST-rajapinta 13
3 ESITUTKIMUS 15
3.1 Alkuperäisen rajapinnan speksien selvitys 15
3.2 Simulaation toteutuksen selvitys 15
3.2.1 Simulaation toteutuskelpoisuus 15
3.2.2 Vaadittujen ominaisuuksien selvitys 16
4 TOTEUTUS 17
4.1 Työkalujen valinta 17
4.2 Rajapinnan toteutus 17
4.3 Hallintaikkunan toteutus 18
4.4 Dokumentaation kirjoittaminen 19
5 POHDINTA 20
LÄHTEET 21

LYHENTEET JA TERMIT

API	Application Programming Interface; rajapinta
C#	Microsoftin luoma ohjelmointikieli.
CERN	Conseil Européen pour la Recherche Nucléaire; suomeksi Euroopan ydintutkimusneuvosto. Tunnetaan erityisesti hiukkaskiihdyttimistään.
CORS	Cross Origin Resource Sharing; mekanismi verkkoteknologioissa, joka mahdollistaa resurssien käsittelyn muista kuin omasta URL-osoitteesta hallitusti.
CRUD	Create, Read, Update, Delete; neljä tärkeintä elementtiä tallennetun tiedon käsittelyä varten.
FastAPI	Python-kielelle suunniteltu työkalu REST-rajapintojen sekä verkkosivujen luontia varten.
Flake8	Python-työkalu, jonka tehtävänä on varmistaa koodityylin yhteensopivuus Pythonin virallisen tyyliohjesäännön kanssa.
Git	Suomalaisen Linus Torvaldsin kehittämä versionhallintatyökalu, joka on laajassa käytössä ympäri maailman.
GitLab	Verkkopalvelu, joka tallettaa Git-projekteja helpottaakseen kehittäjien työskentelyä samassa projektissa.
HTML	HyperText Markup Language; tekstiformaatti, joka toimii Internetin verkkosivujen runkona.
HTTP	HyperText Transfer Protocol; määrittää miten Internet käsittelee tiedonsiirtoa verkkoselainten ja palvelinten välillä.
HTTPS	HyperText Transfer Protocol Secure; käytännössä sama kuin HTTP, mutta lisäksi tietoliikenne on salattu TLS-salaustekniikalla.
IETF	Internet Engineering Task Force; organisaatio, jonka tehtävänä on kehittää standardeja Intenet:iä varten.
Integraatiotesti	Ohjelmistotestauksen termi, jossa testataan jonkin kokonaisuuden toimintaa, kuten esimerkiksi erästä käyttöliittymän toimintoa.

IoT	Internet of Things; esineiden Internet
JavaScript	Ohjelmointikieli, joka on pääasiassa käytössä verkkosivujen selaimissa.
Kirjasto	Ohjelmistokehityksessä käytetty termi, joka tarkoittaa valmiiksi kirjoitettua ohjelmakoodia, jota käytetään muissa projekteissa kehityksen helpottamiseksi.
Markdown	Erityisesti ohjelmistokehityksessä vakiintunut tekstitiedostoformaatti, jossa tavoitteena on luoda sellaisenaan ihmisen luettava tiedosto, joka voidaan myös avata verkkosivumuodossa.
Mermaid	Työkalu, jolla voidaan luoda kuvaajia kirjoittamalla ne tekstimuodossa.
Mypy	Python-työkalu, joka analysoi ohjelmakoodin tietotyypit ja varmistaa, että eri komponentit saavat vain odottamansa kaltaista dataa.
Palvelin	Internetissä oleva laite, joka vastaanottaa sekä käsittelee pyyntöjä muilta laitteilta, ja joka on tavallisesti suunniteltu palvelemaan useita eri laitteita samanaikaisesti.
Poetry	Python-työkalu, jota käytetään yksinkertaistamaan projektien ja niiden vaatimien kirjastojen hallintaa.
Pydantic	Python-kielen kirjasto, joka mahdollistaa helpomman sekä turvallisemman tiedonkäsittelyn ulkoisista lähteistä.
Pylint	Python-työkalu, jonka tehtävänä on varmistaa ohjelmakoodin luettavuus sekä tyylin yhtenäisyys.
Pytest	Python-työkalu, jonka tehtävänä on ajaa ohjelmakoodille kirjoitettuja testitapauksia ja varmistaa, että ne reagoivat odotetulla tavalla.
Python	Ohjelmointikieli, jota käytetään muun muassa koneoppimisessa, automaatiotehtävissä, sekä palvelinpuolen verkkokehityksessä.
REST	Representational State Transfer; tilaton, vakioitu tietojärjestelmien välinen kommunikaatiorajapinta.

Rust	Ohjelmointikieli, joka on tarkoitettu matalan tason ohjelmointiin, esimerkiksi käyttöjärjestelmiin sekä laiteajureihin.
Snyk	Verkkopalvelu, jota käytetään tietoturvahaavoittuvuuksien automaattisessa havainnoinnissa ohjelmistoprojekteissa.
Speksi	Määritelmä; tässä kontekstissa erityisesti jonkin tietyn rajapinnan määritelmä.
SQLAlchemy	Python-kielelle suunniteltu kirjasto, joka yksinkertaistaa tietokantojen kanssa kommunikointia varsinkin tilanteissa, joissa sama projekti saattaa käyttää useampia eri tietokantoja.
SQLite	Yhden tiedoston kokoinen tietokanta, joka on laajassa käytössä erityisesti mobiilisovelluksissa.
Swagger	REST-rajapintojen dokumentointiin käytetty työkalualusta.
TailwindCSS	Verkkokehityskirjasto, joka tarjoaa valmiiksi suunniteltuja elementtejä sekä kuvia helpottamaan verkkosivujen suunnittelua.
TCP	Transmission Control Protocol; yleisin ja helppokäyttöisin matalan tason protokolla tiedon lähettämiseen ja vastaanottamiseen Internetin välityksellä.
Tietokanta	Tiedon pidempiaikaiseen tallennukseen käytetty ohjelmisto, jonne palvelin voi tallentaa esimerkiksi käyttäjätietoja tulevaisuuden tarpeita varten. Vastaa usein rakenteeltaan taulukoita, joissa jokainen rivi sisältää yhden yhtenäisen tietopaketin.
TLS	Transport Layer Security; salaustekniikka, jota käytetään erityisesti verkkoliikenteen salaukseen.
Uvicorn	Palvelinohjelma, jota voidaan käyttää verkkosivusovellusten ajamiseen.
Visual Studio Code	Microsoftin luoma kehitystyökalu ohjelmakoodin kirjoittamiseen, joka tukee useita eri ohjelmointikieliä.
Välityspalvelin	Palvelin, joka toimii välikätenä asiakkaan ja palvelimen välisessä tiedonsiirrossa.

Yksikkötesti	Ohjelmistotestauksen termi, jossa testataan yhden tietyn ohjelmakoodin komponentin (yleensä funktion) toimintaa eri arvoilla.

1 JOHDANTO

Ohjelmistokehitys rajapintojen välisessä kommunikoinnissa on nykyään melko yksinkertaista, tavanomaista, sekä kirjastojen kehittymisen myötä entistä helpompaa. Verkkosivut kommunikoivat palvelimiensa kanssa saadakseen reaaliaikaista dataa, niillä voidaan kerätä dataa IoT-laitteista, ja niitä voidaan käyttää esimerkiksi standardeissa yksinkertaistamaan eri sovellusten keskustelua toistensa kanssa. Tämän opinnäytetyön toimeksiantaja, Wisematic Oy, käyttää rajapintoja eräässä asiakasprojektissa, jota varten opinnäytetyön tekijä palkattiin yritykseen olemassa olevan kokemuspuutteen vuoksi rajapintoihin liittyen.

Mutta, aina kaikki ei mene ihan suunnitellusti. Wisematic ei saanut pääsyä asiakkaan omaan Hemuli-rajapintaan asiakasprojektin testaamista varten, ja tämä aiheutti päänvaivaa; miten varmistetaan projektin toimivuus ilman pääsyä tuohon rajapintaan? Tästä ongelmatilanteesta syntyi tämän opinnäytetyön aihe, jonka tarkoituksena oli selvittää:

- Millainen asiakkaan olemassa oleva Hemuli-rajapinta on spekseiltään
- Onko siitä mahdollista toteuttaa simulaatio, joka mukailee alkuperäistä rajapintaa parhaansa mukaan
- Jos mahdollista, toteuttaa kyseinen simulaatio käytännössä ohjelmistoprojektina

Opinnäytetyön onnistuminen on kriittinen osa asiakasprojektin toteutusta, eikä sitä ole mahdollista korvata olemassa olevalla valmiilla paketilla toiselta palveluntarjoajalta.

Asiakasprojektin tarkoitus on käsitellä sähkölaitteita asiakkaan toiveiden mukaisesti, joita tulee laitteeseen satunnaisesti. Robotin havaitessa uuden laitteen se lähettää Hemuli-rajapinnalle tästä tiedon, jonka jälkeen se saa vastauksen mitä tehdä, kun laite on tunnistettu. Laite menee tästä joko jatkokäsiteltäväksi, tai hylätään ja toimitetaan hylättyjen laitteiden joukkoon. Tarkemmassa käsittelyssä Hemuli-rajapinta hoitaa yksityiskohdat robotin lähettäessä sille tiedot laitteesta, ja lopuksi tulostaa siihen tarran.

Wisematic Oy on tamperelainen teollisuusrobotiikkayritys Hervannan kampusalueella. Yritys tarjoaa EPSON-robottien suoramarkkinointia, sekä mittatilaustöitä asiakkaiden tarpeisiin, tavanomaisesti tuotantolinjoille.

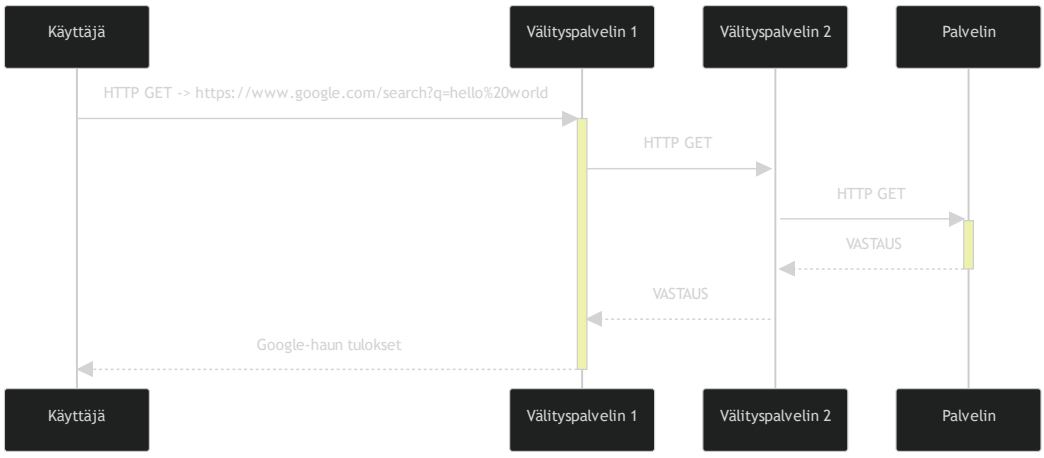
2 TEORIA

2.1 HTTP-protokolla

HTTP-protokolla on yksi yleisimmin käytetyistä kommunikaatioprotokollista, johon esimerkiksi verkkoselainten toiminta pohjautuu. Kyseessä on asiakas-palvelinprotokolla, jossa asiakas (tavallisesti verkkoselain) lähettää tietoa palvelimelle, ja odottaa vastausta. Kommunikaatio tapahtuu yksittäisillä viesteillä. Asiakkaan viestejä palvelimille kutsutaan nimellä pyyntö (engl. request), ja palvelimen viestiä asiakkaalle nimellä vastaus (engl. response). Protokollan suunnitteli alun perin Tim Berners-Lee työskennellessään CERN:illä vuonna 1989, josta sen kehitys siirtyi IETF:n käsiin. Kommunikoinnissa käytetään TCP-yhteyksiä, tai TLS-suojattuja TCP-yhteyksiä. Salattu kommunikointi toteutetaan käytännössä HTTPS-protokollalla, joka näkyy verkkoselaimilla usein vihreän lukon muodossa.

Protokollaa käytetään pääasiassa tekstin noutamiseen, mutta myös kuville ja videoille sekä sisällön lähettämiseen palvelimille.

Asiakkaan lähettämä pyyntö ohjautuu tavallisesti usean eri palvelimen kautta, joista jokainen ohjaa pyyntöä halutun palvelimen suuntaan, kunnes se saapuu perille ja käsitellään. Pyyntö sisältää sekä päätepisteen IP-osoitteen että lähettäjän IP-osoitteen, jotta vastaus saadaan ohjattua takaisin pyynnön lähettäjälle.



Kaavio 1: HTTP-pyyntö käyttäjän tietokoneelta palvelimelle ja takaisin.

HTTP-protokollan mukaisesti viestin tiedot on jaettu kahteen osaan; otsikkoon (engl. header), ja sisältöön (engl. body). Otsikko sisältää tiedot pyynnön metodista, kohteen URL-osoitteesta, sekä lähettäjän osoitteesta, ja muuta tietoa riippuen siitä, onko kyseessä käyttäjän pyyntö palvelimelta vai palvelimen vastaus.

2.2 HTTP-metodit

HTTP-protokolla määrittelee kahdeksan metodia, jotka vaikuttavat pyyntöjen käsittelyyn ja kertovat palvelimelle mitä sen halutaan tiedolle tekevän. Näistä kuusi ovat yleisessä käytössä, loput ovat asteittain harvinaisempia.

POST-, GET-, PUT-, PATCH-, ja DELETE-metodit muodostavat ns. ”CRUD”-ydinjoukon, joka on yleisessä käytössä tietokantaoperaatioissa ja nämä viisi metodia ovat sen vuoksi kaikkein tärkeimmät. Näiden lisäksi HEAD-metodi on merkittävässä roolissa.

- GET-metodi on kaikkein yleisin. Se tarkoittaa, että käyttäjä haluaa palvelimen palauttavan tietoa kohteen URL-osoitteesta. Tätä käytetään joka kerta, kun verkkoselain avaa verkkosivun, tai kun ohjelma pyytää tietoa REST-rajapinnalta.
- POST-metodia käytetään tiedon lähettämiseen käyttäjältä palvelimelle, eli esimerkiksi selainlomakkeiden täytön yhteydessä. HTML5 tukee tätä valmiiksi lomakkeissa, eli tiedon lähetykseen ei välttämättä tarvita JavaScriptiä, jos koko sivun päivitys ei ole ongelma.
- HEAD-metodi toimii samalla periaatteella kuin GET-metodi, mutta se ei palauta sisältöä, vaan pelkän otsikon. Tästä on etua, jos sisältöä ei tarvita, sillä se vähentää tiedonsiirtoon tarvittavaa aikaa, ja on hyödyllinen esimerkiksi päivitysten seuraamiseen.
- PUT-metodi on samankaltainen POST-metodin kanssa, mutta sitä voidaan käyttää useamman kerran peräkkäin ilman, että sillä on sivuvaikutuksia. Käytännössä tätä eroa ei kuitenkaan juuri huomaa, ja jotkin palvelimet saattavat kohdella niitä samoin. Jos kohteena on sisällön tietojen päivitys, tämä metodi luo uuden sisällön palvelimen tietokantaan, jos sitä ei löydy valmiiksi.

- PATCH-metodia käytetään päivittämään jonkin yksittäisen sisällön tietoja, mutta se ei luo uutta sisältöä eikä sitä tavallisesti käytetä kuin yksittäisten kenttien tietojen päivitykseen.
- DELETE-metodi käskee palvelinta poistamaan tietoa annetusta URL-osoitteesta. Esimerkiksi keskustelualustalla tätä voidaan käyttää viestin poistamiseen, tai tietokannan rivin poistamiseen.
- TRACE-metodia käytetään palvelinongelmien selvityksessä, jossa se suorittaa niin kutsutun ”loop-back” testin.
- OPTIONS-metodia käytetään kysymään palvelimelta, mitä HTTP-metodeja jokin yksittäinen URL tukee, tai vaihtoehtoisesti koko palvelimen tukemat metodit. Tätä käytetään lähinnä CORS-pyynnöissä ja testatessa palveluita käsin.

2.3 REST-rajapinta

REST-rajapinnat, joista käytetään myös nimitystä ”RESTful”, ovat arkkitehtuurityyli rajapintojen suunnitteluun. Ne ovat käytössä erityisesti verkkopalveluissa, joissa kaksi ohjelmaa kommunikoivat keskenään. Arkkitehtuurin suunnitteli tietojenkäsittelytieteilijä Roy Fielding vuonna 2000 väitöskirjassaan *Architectural Styles and the Design of Network-based Software Architectures.*

REST-rajapintojen tulee täyttää tiettyjä vaatimuksia. Rajapinnan tulee lähettää ja vastaanottaa tietoa vakioidussa muodossa, tavallisimmin JSON-formaatissa, mutta XML-formaatti on myös käytössä, lähinnä Microsoftin tuotteissa. Joskus rajapinta voi tukea useampaa formaattia, jolloin tyypin voi määritellä osana rajapinnan kutsua, mutta yleensä vain paluuarvona.

Kaikkien REST-rajapintojen on toteutettava seuraavat ominaisuudet:

- Asiakas-palvelin -arkkitehtuuri, jossa asiakkaat ottavat yhteyttä palvelimiin HTTP-protokollan kautta

- Tilavapaa kommunikaatio asiakkaiden ja palvelinten välillä, eli asiakkaan tietoja ei tallenneta palvelimelle rajapintakutsujen välillä, ja jokainen rajapintakutsu käsitellään erikseen omana erillisenä toimintonaan
- Asiakkaalle palautetut vastaukset sisältävät riittävästi tietoa kuvaamaan tapahtumaa sekä ohjeistaen asiakasta käsittelemään sen
- Välitetyn tiedon tulee olla tallennettavissa välimuistiin helpottamaan kommunikointia asiakkaiden ja palvelinten välillä
- Rajapinnan tulee olla rakenteeltaan yhtenevä, jotta informaation siirto onnistuu normalisoidussa muodossa
- Palvelimet on jaettu eri kerroksiin, esimerkiksi tietoturva-, ja kuormituksen tasapainoituskerroksiin, joista jokainen toimii näkymättömästi taustalla asiakkaan tietämättä

3 ESITUTKIMUS

3.1 Alkuperäisen rajapinnan speksien selvitys

Opinnäytetyön ensimmäisenä vaiheena oli suorittaa selvitys asiakkaan kanssa, millainen rajapinta oli kyseessä. Käytännössä tämä tarkoitti eri päätepisteiden selvitystä, mitä HTTP-metodeja ne tukivat, millaista JSON-dataa ne odottivat, mitä virhetiloja niistä voisi syntyä, sekä jonkinasteinen kuva siitä, mitä rajapinnan toisella puolella tapahtui. Asiakas onneksi antoi Wisematicille nämä tiedot, jonka jälkeen Wisematicin kehitystiimi teki annetuista tiedoista raa'at kaaviot, jotka sisälsivät tietoja eri tilanteiden käsittelystä ja seurauksista.

Käytännössä simulaattorin tulisi saada viesti robotilta, kun se käynnistyy, ja sen jälkeen ottaa järjestelmä hallintaansa. Kaikki data kulkisi simulaattorin sisällä, tietokanta mukaan lukien, ja se antaisi robotille ohjeet mitä tehdä.

Lisäksi simulaattoriin haluttiin toteuttaa hallintakäyttöliittymä, jolla robotin tilaa kyettäisiin seuraamaan ja tarvittaessa hallitsemaan, tarkoituksena helpottaa robotin testaamista.

3.2 Simulaation toteutuksen selvitys

Kun alkuperäisen rajapinnan koostumus oli selvillä, seuraava vaihe oli selvittää mahdollisuus sen toteuttamiseen sisäisenä simulaationa, ja mitä ominaisuuksia siltä vaadittaisiin. Lisäksi hallintakäyttöliittymän toteutus vaati selvitystä halutuista ominaisuuksista sekä käyttöliittymän suunnittelusta.

3.2.1 Simulaation toteutuskelpoisuus

Simulaatio haluttiin toteuttaa melko nopealla aikataululla, ja suunnitelma oli pyörittää sitä Wisematicin sisäisillä palvelimilla. Projektille ei ollut päätetty mitään tiettyä teknologiaa, eikä sillä REST-rajapintojen tapauksessa ole erityistä merkitystä. Erityisiä suorituskykyvaatimuksia ei ollut, sillä simulaattoria käyttäisi samanaikaisesti korkeintaan kaksi eri palvelua.

Alkuperäisen rajapinnan speksien perusteella simulaattori ei olisi erityisen monimutkainen, vaikkakin eri toimintojen mallintamisessa olisi paljon työtä.

Simulaattorin toteutus todettiin mahdolliseksi.

3.2.2 Vaadittujen ominaisuuksien selvitys

Simulaattorin haluttiin tekevän hieman enemmän kuin pelkästään alkuperäisen rajapinnan toteutus. Osana simulaattoriprojektia luotaisiin graafinen esitys Wisematicin laitteen senhetkisestä tilasta, sekä hallintaikkuna, jolla yksittäisiä komentoja olisi mahdollista ajaa. Näiden toteutus olisi täysin opinnäytetyön tekijän harteilla.

Halutut lisäominaisuudet todettiin toteutuskelpoisiksi, ja opinnäytetyön kehitys käynnistyi.

4 TOTEUTUS

4.1 Työkalujen valinta

Opinnäytetyön vaatimusten puitteissa sen toteuttamiskieleksi valikoitui Python. Syinä tähän valintaan olivat työntekijän olemassa oleva osaaminen, tuotetun ohjelmakoodin luettavuus tulevaisuuden jatkokehityksen varalta, sekä sopivien työkalujen helppo saatavuus. Myös muun kehitystiimin senhetkinen teknologiaosaaminen otettiin huomioon.

REST-rajapinnan luontiin valikoitui FastAPI sen tarjoamien ominaisuuksien vuoksi, jotka helpottavat kehittäjän taakkaa ja auttavat varmistamaan datan sopivuuden. Pydantic-integraatio auttaisi vähentämään ongelmia datan käsittelyssä, FastAPI:n sisäänrakennettu Swagger-tuki helpottaisi rajapinnan dokumentaation generoinnissa sekä testikomentojen ajossa, ja FastAPI:n dokumentaatio vaikutti erinomaiselta. Rajapinnan ajamiseen käytettiin Uvicorn:ia. Tietokantana käytettiin SQLite-tietokantaa, ja sen kanssa kommunikointiin käytettiin SQLAlchemy-kirjastoa. Hallintaikkunan luontiin päätettiin ottaa käyttöön saman teknologian lisäksi TailwindCSS-kirjasto helpottamaan käyttöliittymän suunnittelua. Suunnitteluvaiheessa käytettiin myös Mermaid-työkalulla luotuja kaavioita.

Kehitystyökaluiksi valikoituivat Pytest yksikkö- ja integraatiotestien luontiin, Mypy tyyppitarkistukseen, sekä Pylint ja Flake8 koodin siisteyden ylläpitämiseen. Koodieditorina käytettiin Microsoftin Visual Studio Code -sovellusta, versionhallintana Git ja Wisematicin sisäinen GitLab ajoivat asiansa. Tietoturvatarkistuksessa käytettiin apuna myös Snyk-palvelua. Projektihallinnassa käytettiin Poetryä.

4.2 Rajapinnan toteutus

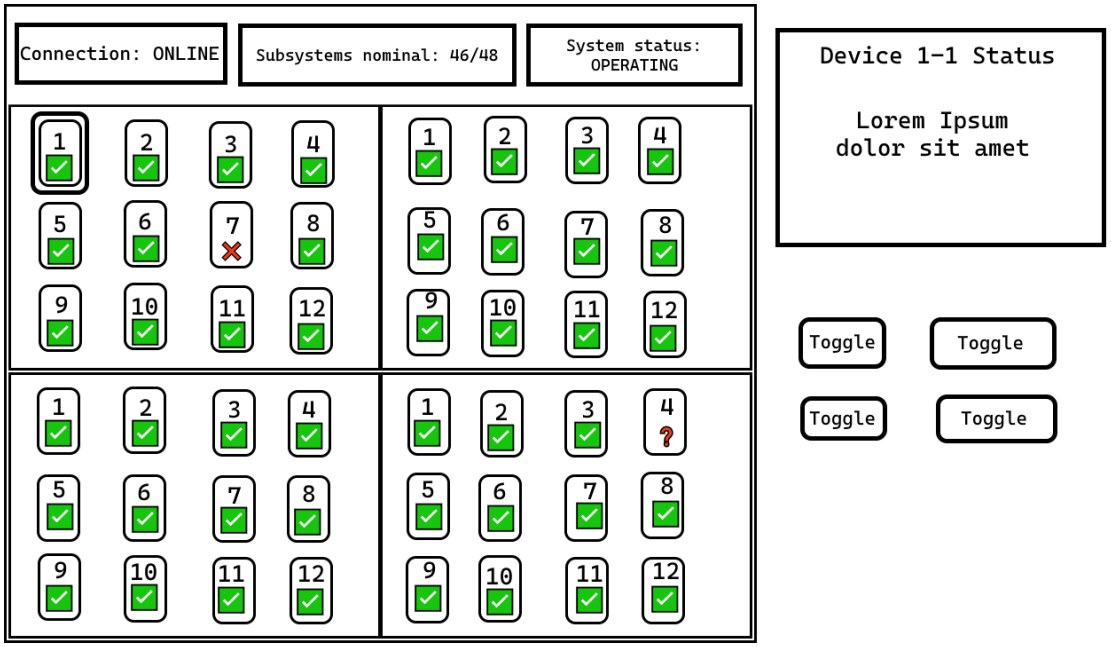
Simulaattorin rajapinta luotiin pitkälti esitutkimusvaiheen kaavioiden pohjalta. Kaikkien palveluiden toteutus onnistui lähes ongelmitta, joskin rajapintaa laajen-

nettiin hieman mahdollistamaan haluttujen lisäominaisuuksien toteutus. Jokaiselle rajapinnan osalle luotiin automaatiotestit, ja rajapinnan testikattavuus saavutti parhaimmillaan 100 %:n kattavuuden.

Eräs toteutetuista lisäominaisuuksista oli simuloidut lähtötilanteet robotille, joista annettiin konfiguroitava asetus. Mahdollisia konfiguraatioita olivat esimerkiksi tyhjä, täysi, ja satunnaisessa – mutta mahdollisessa – tilassa oleva robotti.

4.3 Hallintaikkunan toteutus

Hallintaikkuna toteutettiin simulaattorin tarjoamana verkkosivuna, joka käytti palvelinpohjaista renderöintiä eliminoidakseen tarpeen lukijan JavaScript-tuelle. Ikkunan yläpalkissa näkyi tietoa robotin yleistilasta, kuten sen nimi, senhetkinen toimintatila, ja yhteyden status. Sen alla oli näkymä, jossa vasemmalla puolella näkyi robotin neljän ”hyllykön” tila sekä yksittäisten laitteiden tila, ja oikealla näkyi tarkemmat tiedot sillä hetkellä valitusta laitteesta. Klikkaamalla eri osia vasemmalla puolella, oikealla puolella näkyvä laite vaihtui.



Kuva 2: Suuntaa antava esimerkki hallintakäyttöliittymän ulkoasusta.

Tämän ikkunan alla oli lista pikakomentoja, joilla operaattori saattoi nopeasti antaa järjestelmälle jonkin yleisen käskyn, kuten kaikkien laitteiden tilanmuutos.

4.4 Dokumentaation kirjoittaminen

Simulaattorille kirjoitettiin kattava määrä dokumentaatiota. Kaikki ohjelman funktiot ja luokat dokumentoitiin yksitellen, automaatiotestejä myöten, ja lisäksi projektin rakenteesta kirjoitettiin dokumentaatiota juurikansion docs-kansioon, joka kuvasi esimerkiksi yksittäisten tiedostojen merkitystä ja tarkoitusta, sekä miten niitä käytetään. Dokumentaatio kirjoitettiin Markdown-modossa tavallisiin tekstitiedostoihin. Swagger-ikkunan ohjeita muokattiin käsin selkeämmiksi ja siihen lisättiin Wisematicin logo, ja joihinkin rajapinnan osien kutsumiseen annettiin esimerkkejä Pythonilla, Rustilla, C#:lla ja JavaScriptillä. Simulaattorin asennus- sekä ajo-ohjeet kirjattiin README-tiedostoon. Palaute muilta työntekijöiltä oli positiivista.

5 POHDINTA

Opinnäytetyön tavoitteena oli toteuttaa simulaatio asiakkaan Hemuli-rajapinnasta ja edistää tästä riippuvan sisäisen projektin kehitystä, ja lisäksi toteuttaa hallinta-työkalu robotin tilan seuraamiseen sekä hallintaan. Osana toteutusta tavoitteena oli myös selvittää tarvitun rajapinnan yksityiskohdat, sekä ylimääräisten ominaisuuksien toteutus. Työtä tehtiin tutustumalla annettuihin spekseihin, selvittämällä tietoa työkaluista, dokumentaatiosta, ja blogipostauksista, mutta enimmäkseen omasta kokemuksesta.

Työn tuloksena kehitetty simulaattori ei sellaisenaan ole hyödynnettävissä muissa projekteissa, mutta kattavan dokumentoinnin ansiosta se toimii hyvin pohjana muihin vastaaviin tarpeisiin tulevaisuudessa, sillä olemassa olevassa koodissa on esimerkkejä kaikenlaisiin tilanteisiin ja niiden käsittelyyn.

Lopputuloksena simulaattori toteutti kaikki halutut toiminnot sekä todistetusti toimi kuten pitikin, aikataulussa, joten asiakasprojektin kehitys saattoi jatkua toivotusti.

LÄHTEET

Wisematic Oy. Luettu 03.9.2022.
https://www.wisematic.com/

Ramírez, S. 2022. FastAPI Documentation. Luettu 29.9.2022.
https://fastapi.tiangolo.com/

Mozilla Foundation. MDN Web Docs – HTTP. Luettu 08.12.2024.
https://developer.mozilla.org/en-US/docs/Web/HTTP

IETF. About IETF. Luettu 08.12.2024
https://www.ietf.org/about/

Mermaid Team. Mermaid – Docs. Luettu 08.12.2024
https://mermaid.js.org/intro/

TailwindCSS. TailwindCSS Docs. Luettu 08.12.2024
https://tailwindcss.com/docs/installation

The Poetry Team. Poetry Documentation. Luettu 08.12.2024
https://python-poetry.org/docs/

Michael Bayer. SQLAlchemy 1.4 Documentation. Luettu 08.12.2024
https://docs.sqlalchemy.org/en/14/