HAMK
Hämeen ammatti-
korkeakoulu

ASIAKASTUKIPYYNTÖJEN VISUALISOINTI YRITYKSEN INFONÄYTÖLLÄ

Ammattikorkeakoulututkinto
Tieto- ja viestintätekniikka, insinööri (AMK)
Kevät, 2024
Toni Heinänen

HAMK
Hämeen ammatti-
korkeakoulu

Tieto- ja viestintätekniikka, insinööri (AMK) Tiivistelmä
Tekijä Toni Heinänen Vuosi 2024
Työn nimi Asiakastukipyyntöjen visualisointi yrityksen infonäytöllä
Ohjaaja Joni Järvenpää

Tämän opinnäytetyön tavoitteena oli luoda järjestelmäkokonaisuus, jolla pystyttäisiin tehostamaan ohjelmistotukityöntekijöiden käsitystä asiakasprojektien kokonaistukitarpeesta visuaalisesti. Vastaavanlainen toteutus oli käytössä jo toisessa yksikössä toimeksiantajayrityksessä ja siitä saatujen hyötyjen perusteella lähdettiin toteuttamaan tätä projektia.

Toimeksiantajana opinnäytetyössä toimi ohjelmistokehitysyhtiö Whitelake Software Point Oy (WSP). Toimeksiantajalla on jatkuva tavoite kehittää omia työmenetelmiään tehokkaamman asiakaspalvelun tarjoamiseksi ja tämä työ toi arvoa yritykselle, sekä tehokkuutta asiakastukitiimin toimintaan.

Työssä suunniteltiin ja toteutettiin ohjelmistokokonaisuus, jonka lopullisena tarkoituksena oli näyttää WSP:n työntekijöille isolla infotaululla reaaliaikaista tietoa tukipyyntöjärjestelmän tilanteesta. Tämä antaa työntekijöille mahdollisuuden nähdä kriittiset tiketit yhdellä vilkaisulla, joutumatta hakemaan tietoa omalta päätteeltään, keskeyttäen mahdollisesti muut käynnissä olevat työtehtävät.

Toteutuksessa käytettiin yrityksen infonäyttöä, Raspberry Pi 4 Model B mikrotietokonetta, Debian käyttöjärjestelmää ja ohjelmistoja, jotka tullaan esittelemään tässä työssä. Lyhyesti Raspberry ajaa ajastetusti Python koodia, joka kyselee Jira tiketöintijärjestelmän rajapinnasta tietoja ja tallentaa ne tietokantaan. Lopulta Grafana visualisointiohjelmisto esittää tiedot infonäytöllä.

Avainsanat Tietojenkäsittely, rajapinta, visualisointi
Sivut 25 sivua ja liitteitä 0 sivua

HAMK
Hämeen ammatti-
korkeakoulu

Information and Communication Technology
Author Toni Heinänen
Subject Visualization of customer support tickets on an info monitor
Supervisors Joni Järvenpää

Abstract
Year 2024

The aim of this thesis was to create a system that would enhance the software support staff's understanding of the overall support needs of client projects visually. Similar implementation was already in use in another unit of the commissioning company, and the project was initiated based on the benefits observed there.

The client for this thesis was the software development company Whitelake Software Point Oy (WSP). The client has an ongoing goal to develop its working methods for more efficient customer service, and this work added value to the company and efficiency to the customer support team's operations.

The work involved designing and implementing a software system whose ultimate purpose was to display real-time information about the ticket system status on a large information board for WSP employees. This allows employees to see critical tickets at a glance, without having to search for information on their own terminals, potentially interrupting other ongoing tasks.

The implementation used the company's information display, a Raspberry Pi 4 Model B microcomputer, Debian operating system, and software that will be presented in this work. Briefly, the Raspberry runs Python code on a schedule, querying the Jira ticketing system interface for information and storing it in a database. Finally, Grafana visualization software displays the information on the information display.

Keywords Data processing, API, visualization
Pages 25 pages and appendices 0 pages

Sisällys

Kuvat

1 Johdanto

Tämän opinnäytetyön aiheena on tiketöintijärjestelmä Jiran tukipyyntöjen esittäminen toimeksiantajayrityksen infonäytöllä, hyödyntäen yrityksen käytössä olevaa teknologiaa ja laitteistoa. Työn tavoitteena on parantaa asiakaspalvelun tehokkuutta, antamalla työntekijöille nopea tapa nähdä reaaliaikainen tukipyyntötilanne aktiivisissa asiakasprojekteissa. Tämä projekti kattaa suunnitellusti vain osan WSP:n useista asiakaspalvelutiimeistä, mutta jatkossa tätä järjestelmää voidaan laajentaa kattamaan useampia tiimejä, heidän tukipyyntötilanteensa, sekä useampia näyttöjä yrityksen toimitiloissa.

Infonäyttö on ollut käyttämättömänä WSP:n Espoon toimiston eräässä siivessä jo pitkään. Näytön rinnalle oli myös ehditty hankkia Raspberry Pi 4 mikrotietokone, johon oli suunniteltu asennettavaksi yksinkertainen käyttöjärjestelmä, jota hyödyntämällä mahdollistettaisiin hetkittäisen tukipyyntötilanteen visualisointi Jira -tehtävänhallintajärjestelmästä. Nämä suunnitelmat oli kuitenkin toistaiseksi hyllytetty etätyön yleistyessä koronapandemian vuoksi, sekä ajankäyttöä priorisoitaessa asiakastyöhön. Pandemian jälkeinen lähitöihin palaaminen herätti keskustelun infonäytön hyödyistä ja tarpeellisuudesta uudelleen.

Työn toimeksiantajana toimii Whitelake Software Point, joka on johtava laboratoriotietojärjestelmien (LIMS) kehittäjä- ja myyjäyritys Pohjois-Euroopassa. Software Point Oy on perustettu vuonna 1992 Espoossa (Softwarepoint, n.d.) Tässä opinnäytetyössä esiteltävän projektin on tarkoitus tehostaa WSP:n asiakaspalvelun toimintaa.

Tämän opinnäytetyön tavoitteena oli saada visualisoitua yrityksen asiakaspalvelun tiketöintijärjestelmän tilanne käyttämättömänä olleelle infonäytölle, hyödyntäen jo olemassa olevia laitteita. Mikrotietokone Raspberry Pi 4 toimisi järjestelmäkokonaisuuden alustana ja hermokeskuksena datalle. Infonäyttöä suunniteltiin käytettäväksi mikrotietokoneen keräämän datan visualisointiin.

2 Toteutuksen suunnittelu ja teoria

Ennen varsinaisen toteutuksen aloittamista, suunniteltiin erilaisia tapoja toteuttaa projekti. Suunnitelmissa piti ottaa huomioon toteutuksen tehokkuus, automatisointimahdollisuudet, sekä esitettävän datan selkeys ja muokattavuus. Toteutuksen jatkokehitysmahdollisuuksille annettiin myös painoarvoa suunnitelmaa laadittaessa. Tärkeänä tekijänä pidettiin myös tietoturvaa, koska tiketöintijärjestelmä sisältää tietoa yrityksen asiakkaista, sekä asiakasyrityksien työntekijöistä. Tässä vaiheessa kartoitettiin sellaiset ohjelmistot ja palvelut, jotka olivat toimeksiantajayrityksen käytettävissä. Yksinkertaisin ja myös alkuperäisen suunnitelman mukainen toteutus olisi kattanut Raspberry Pi 4 tietokoneen käyttämisen internet-selaimena, jolla olisi esitetty suoraa kuvaa Jiran tukipyyntöjonosta infonäytön toimiessa Raspberryn monitorina.

2.1 Käytettävissä olevat laitteet

WSP:n tiloissa oli käyttämätön Samsung merkkinen infonäyttö, jonka koko ja sijainti sopivat tämän työn vaatimuksiin. Samassa tilassa oli myös Raspberry Pi 4 model B -mikrotietokone, jonka käyttöjärjestelmäksi oli asennettu Raspberry Pi Os. Mikrotietokoneeseen ja näytön väliin piti hankkia HDMI – MicroHDMI johto, jolla Raspberryn kuva saatiin esitettyä infonäytöllä.

2.2 Jira

WSP käyttää asiakastukipyyntöjensä, sekä projektiensa hallintaan Atlassian Jira projektinhallintaohjelmistoa. Jira on johtava ketterän projektinhallinnan työkalu, jota käyttävät lukuisat ohjelmistekehitystalot ympäri maailmaa (Jira, n.d.). Jira on web-pohjainen ohjelmisto, joka mahdollistaa projektien, asiakkuuksien, sekä asiakastukipyyntöjen hallinnan.

2.3 Jira Dashboard

Projektin suunnitelman ensimmäinen vaihe oli päättää käytettävä palvelu datan visualisointiin näytöllä. Yksinkertaisin mahdollinen vaihtoehto oli käyttää Jiran omaa Dashboard-näkymää, joka mahdollistaisi eri tiimien projektien tukipyyntötilanteiden näyttämisen suoraan. internet-selaimessa. Jiran omien näkymien käyttämisessä olisi monia hyviä puolia, kuten se, että näin ei olisi tarvetta muille ohjelmistoille tai palveluille, joka parantaisi järjestelmän vakautta ja helppokäyttöisyyttä.

2.3.1 Jira Dashboard tietoturvakysymys

Koska infonäyttö ja Raspberry Pi sijaitsevat tilassa, joihin on pääsy myös vierailijoilla ja esimerkiksi siivoojilla, huomattiin ongelma tietoturvan kanssa. Jira järjestelmään pääsy suoraan laitteelta, vaikkakin vain katselija (eng. viewer) oikeuksin oli tilanne, jota haluttiin välttää. Lisähuolta aiheutti näppäimistö ja hiiri, jotka ovat kytkettynä Raspberry Pi:hin pikaisia ongelmanselvitystilanteita ja nopeita huoltotoimenpiteitä varten.

2.4 Vaihtoehtoiset datan visualisointimenetelmät

Eri vaihtoehtoja datan visualisointiin infonäytöllä on lukuisia (Chapman, 2019). Vaihtoehtoja datan visualisointityökaluksi selvitettiin etsimällä tietoa eri tuotteista, sekä tutkimalla käyttäjäkokemuksia yleisimmistä ohjelmista. Tärkeimmiksi valintakriteereiksi valikoituivat helppokäyttöisyys, muokattavuus ja kustannustehokkuus. Tutkimusten ja kokeilujen päätteeksi, lopullinen valinta rajattiin kahteen tunnettuun ohjelmistoon, Grafana ja Microsoft Power Bi (Benner, 2022).

2.4.1 Grafana

Grafana on avoimen lähdekoodin analyytikka- ja visualisointiohjelmisto. Se on suunniteltu analysoimaan ja visualisoimaan laajoja datakokonaisuuksia. Grafanalla on mahdollista luoda dashboard näkymiä, joissa dataa voidaan esittää erinäisissä paneeleissa (eng. panel) hyödyntäen useita erilaisia ohjelmaan sisäänrakennettuja mittari- ja kaaviotyyppejä. Grafanaan voidaan liittää lukuisia eri tietolähteitä (eng. data source), joista dataa haetaan käyttäen kullekin tietolähteelle sopivaa kyselykieltä. (Grafana, n.d.)

2.4.2 Microsoft Power Bi

Power Bi on tehokas ja ammattimainen analytiikkatyökalu, joka sisältää lukuisia tapoja kerätä, analysoida ja visualisoida dataa. Ohjelma on helppokäyttöinen, skaalautuva ja sisältää tuen useille eri tietolähdetyypille. Power Bi myös integroituu saumattomasti Microsoftin muihin tuotteisiin, kuten Exceliin ja Azureen. Lisäksi se mahdollistaa datan reaaliaikaisen analysoinnin ja raportoinnin. (Microsoft, n.d-a)

2.4.3 Visualisointityökalun valinta

Grafanan valinta tässä kontekstissa tuntui luontevalta, ottaen huomioon työn toteuttajan aiemman kokemuksen tämän ohjelman kanssa. Tämä aiempi kokemus ei ainoastaan nopeuttaisi projektin toteutusta, vaan myös vähentäisi tarvetta syventyä uusiin ohjelmistoihin. Lisäksi, vaikka Microsoft Power BI tarjoaakin monipuolisia ominaisuuksia ja integraatioita, sen käyttöönotto vaatisi maksullisen lisenssin, etenkin kun dataa aiotaan esittää tiloissa, jossa useampi käyttäjä pääsee näkemään tuloksen. Grafana, toisaalta, on avoimen lähdekoodin ohjelmisto, joka tarjoaa kustannustehokkaan ja joustavan ratkaisun datan visualisointiin. Sen avulla voidaan luoda monipuolisia dashboard-näkymiä ilman lisäkustannuksia. Grafanasta oli valittavissa Cloud tai itse hallinnoitava (eng. self-managed). Cloud toimisi palveluna, jolloin paikallisasennuksia ei tarvittaisi, mutta ilmaisversiossa on useita rajoituksia, kuten esimerkiksi rajoitettu käyttäjämäärä ja lokitietojen maksimikoko. Tämän vuoksi päädyttiin self-managed asennukseen, jolloin Raspberry Pi:tä voitaisiin käyttää asennusalustana Grafanalle.

2.5 Datalähde visualisoinnille

Tässä vaiheessa projektin suunnittelua oli tarpeen valita visualisointityökalulle sopiva datalähde. Grafanaan on kehitetty Jira lisäosa, joka mahdollistaa datan visualisoimisen Jirasta, yhdistämällä Jiran tietolähteeksi. Tämä vaatisi kuitenkin Grafanan Cloud version, sekä Enterprise tason lisenssin. Tätä projektia lähdettiin suunnittelemaan sillä periaatteella että lisähankintoja ei tarvitse tehdä, joten vaihtoehtoinen datalähde oli tarpeen. Grafanan self-managed asennus tukee lukuisia eri ilmaisia datalähteitä, kuten Microsoft SQL Server, PostgreSQL ja InfluxDB.

2.5.1 Microsoft SQL Server

Microsoft SQL Server on laajalti käytetty relaatiotietokantaohjelmisto, joka tarjoaa vankan ja luotettavan alustan suurten tietomäärien säilyttämiseen ja hallintaan. Sen vahvuuksia ovat korkea suorituskyky, tietoturvaominaisuudet ja helppo integroitavuus muiden Microsoft-tuotteiden, kuten Power BI:n kanssa (Microsoft, n.d-b). SQL Server tukee monipuolisia kyselykieliä ja tarjoaa kehittyneitä analytiikkaominaisuuksia. Se soveltuu erityisen hyvin yritysympäristöihin, joissa käytetään muita Microsoftin ratkaisuja. SQL Serverin haasteena on sen lisenssimaksut, jotka voivat kasvaa merkittävästi suurten datamäärien ja monimutkaisten ympäristöjen hallinnassa.

2.5.2 PostgreSQL

PostgreSQL on avoimen lähdekoodin relaatiotietokantajärjestelmä, joka tunnetaan sen kestävyydestä, luotettavuudesta ja tietoturvasta (PostgreSQL, n.d.) PostgreSQL tukee laajaa valikoimaa ohjelmointikieliä ja käytäntöjä, kuten JSON ja hajautetun datan hallinta. Se on suosittu valinta monipuolisten sovellusten kehittäjien keskuudessa, sillä se tarjoaa hyvän suorituskyvyn sekä suurten että pienten datamäärien kanssa. PostgreSQL:n avoimen lähdekoodin luonne tarkoittaa myös, että sitä voidaan käyttää ilman lisenssimaksuja, mikä tekee siitä kustannustehokkaan vaihtoehdon. Sen käyttö Grafanan kanssa on suoraviivaista, minkä ansiosta se soveltuu hyvin datan visualisointiin.

2.5.3 InfluxDB

InfluxDB on avoimen lähdekoodin aikasarjadatalle optimoitu tietokantajärjestelmä (Influxdata, n.d.). Se on suunniteltu erityisesti nopeaa datan kirjoitusta ja kyselyä varten, mikä tekee siitä ihanteellisen valinnan reaaliaikaisen datan seurantaan ja analysointiin. InfluxDB:n käyttöliittymä on suunniteltu helppokäyttöiseksi, ja se integroituu hyvin erilaisten sensori- ja IoT-laitteiden kanssa. Sen rajoituksena on kuitenkin, että se on erikoistunut aikasarjadatan käsittelyyn, mikä voi rajoittaa sen soveltuvuutta yleiskäyttöisemmissä sovelluksissa.

2.5.4 Tietolähteen valinta

Kolmen tutkitun tietolähteen – Microsoft SQL Server, PostgreSQL ja InfluxDB – joukosta päädyttiin valitsemaan PostgreSQL:n. Syynä tähän valintaan oli ensisijaisesti sen perusteellinen dokumentaatio, sekä laaja käyttäjäkunta (Linster, 2023). Tämä helpotti merkittävästi projektin toteutusta, vähentäen tarvetta perehtyä uuteen tietokantajärjestelmään. Lisäksi PostgreSQL:n avoimen lähdekoodin luonne ja sen tarjoama kustannustehokkuus olivat tärkeitä tekijöitä, kun otetaan huomioon projektin budjetti ja resurssit. PostgreSQL:n kyky tallentaa dataa monipuolisesti ja sen yhteensopivuus Grafanan kanssa olivat myös avaintekijöitä valinnassa. Kun otetaan huomioon projektin tarpeet ja käytettävissä olevat resurssit, PostgreSQL tarjosi parhaan yhdistelmän joustavuutta, suorituskykyä ja kustannustehokkuutta.

2.6 Datan siirtäminen

Datan siirtämisen suunnittelussa Jirasta PostgreSQL-tietokantaan keskeiseksi osaksi valikoituivat Pythonilla kirjoitetut skriptit, jotka kutsuvat Jiran REST-rajapintaa. Pythonin

request -kirjasto soveltuu erinomaisesti korkean tason http kyselyihin (Requests, n.d.). Ensimmäisessä vaiheessa suunniteltiin skripti projektien listaukseen, jossa kysely Jiran REST-rajapinnalle hakisi käyttöoikeuksien puitteissa nähtävissä olevat projektit. Tämän jälkeen toisella skriptillä suunniteltiin haettavan avoimet tiketit listatuista projekteista, keskittyen tikettien perustietojen keräämiseen.

Kolmannessa vaiheessa suunnitelmaan sisältyi skripti, joka vastaa kerätyn datan siirtämisestä ja tallentamisesta PostgreSQL-tietokantaan. Tämä prosessi varmistaa datan järjestelmällisen ja turvallisen tallennuksen, jolloin tietokannasta saadaan ajantasainen ja hyödynnettävä tietokokoelma. Tällä tavoin suunnitellun prosessin tavoitteena on automatisoida datan siirto mahdollisimman tehokkaasti ja turvallisesti.

2.6.1 Jira REST API ja käyttäjäoikeudet

Datan siirron suunnittelussa Jirasta PostgreSQL-tietokantaan keskeinen askel oli käyttäjätilin luominen Jiraan. Tämän käyttäjätilin roolina olisi toimia datan hakemisen välikätenä, hyödyntäen Jiran REST-rajapintaa. Suunnitelmassa korostettiin, että tälle käyttäjätilille annetaan vain tarvittavat oikeudet niihin projekteihin, jotka ovat projektissa relevantteja, varmistaen näin tietoturvan ja datan eheyden.

Käyttäjätilin luomisen yhteydessä määriteltiin selkeästi, mihin projekteihin hänellä on pääsy ja millaiset toiminnot ovat sallittuja. Tämä käyttäjätili luodaan nimenomaan datan hakemista varten, ja tilin käyttäjäoikeudet rajoitetaan vain lukuoikeuksiin, jotta vältetään mahdolliset tietoturvariskit. Tilin oikeudet konfiguroidaan siten, että ne mahdollistavat pääsyn vain niihin projekteihin ja dataan, jotka ovat olennaisia tämän projektin kannalta.

2.6.2 Projektien haku ja tallennus

Datan siirron aloitusvaiheeseen suunniteltiin Python-skripti, joka on tarkoitettu tekemään REST API -kysely Jiraan. Sen tarkoituksena on kerätä lista kaikista käytettävissä olevista projekteista. Saatu data tallennetaan paikallisesti JSON-tiedostomuodossa, mikä mahdollistaa joustavan pääsyn ja käsittelyn projektien tiedoille myöhemmissä vaiheissa.

2.6.3 Avoimien tukipyyntöjen haku ja tallennus

Seuraavassa suunnitteluvaiheessa suunniteltiin toista Python-skriptiä, jonka tehtävänä olisi suorittaa tarkennettuja kutsuja Jiran REST API:lle. Tämä skripti hyödyntäisi edellisessä vaiheessa kerättyjä projektitietoja rajoittaakseen hakuja vastaamaan vain niitä projekteja,

joista avoimia tukipyyntöjä on tarve seurata. Skripti suorittaisi kyselyt systemaattisesti kullekin projektille, yksi kerrallaan, keräten tiedot avoimista tukipyynnöistä.

Kun tukipyynnöt on haettu, ne tallennettaisiin paikallisesti JSON-tiedostomuotoon. Yhteen JSON-tiedostoon tallennettaisiin yhden projektin tukipyynnöt. Tämä mahdollistaa seuraavassa vaiheessa järjestelmällisen datan siirron PostgreSQL tietokantaan käyttäen kolmatta Python skriptiä.

2.6.4 Tukipyyntöjen syöttäminen tietokantaan

Tiedonsiirron kolmannessa vaiheessa Python-skriptin tehtävänä olisi lukea aikaisemmin tallennetut avoimet tukipyynnöt JSON-muodossa paikalliselta levyltä. Tämän jälkeen skripti käsittelisi ja muuntaisi nämä tietueet vastaaviksi SQL Insert -lauseiksi. Jokainen tukipyyntö muutetaan tietokantaan sopivaksi riviksi, joka sisältää kaikki tarvittavat tiedot, kuten tukipyynnön tunnisteet, otsikot, tilat ja muut relevantit attribuutit.

Näitä SQL Insert -lauseita käytetään sitten syöttämään tukipyyntöjen tiedot PostgreSQL-tietokantaan. Tämä prosessi suoritetaan huolellisesti, varmistaen, että kaikki tieto siirtyy tietokantaan oikein ja säilyttäen tietokannan eheyden. Tämä skripti toimii keskeisenä välineenä datan siirtoketjussa, varmistaen, että tukipyynnöt rekisteröidään järjestelmällisesti ja ovat valmiita jatkokäsittelyyn ja analysointiin.

2.7 Tietokannan suunnittelu

Tietokannan rakenteen suunnitteluvaiheessa keskityttiin luomaan selkeä ja yksinkertainen rakenne, joka vastaa projektin tarpeita ja mahdollistaa tietojen tehokkaan käsittelyn. Tavoitteena oli luoda tietomalli, joka on sekä joustava että riittävän yksinkertainen ylläpidettäväksi ja laajennettavaksi tulevaisuudessa. Päätettiin luoda yksi keskeinen taulu, joka nimettiin *issues*. Tämän taulun kolumneiksi suunniteltiin seuraavat:

- **id** - Yksilöllinen sarjanumero jokaiselle tukipyynnölle, joka toimii taulun pääavaimena. Tämä on juokseva, automaattisesti inkrementoituva luku.
- **key** - Tukipyynnön yksilöllinen avain Jira-järjestelmässä, joka mahdollistaa helpon viittaamisen ja seurannan.
- **status** - Kuvaa tukipyynnön nykytilaa, kuten "Pending”, ”Waiting for customer”, “IN PROGRESS”, Waiting for support”.
- **assignee** - Tukipyynnön vastuuhenkilö.
- **project** - Projekti, johon tukipyyntö kuuluu.

- **category** - Tiimi, jolle tukipyyntö kuuluu ja pyynnön tyyppi, tukipyyntö tai palvelupyyntö.
- **organization** - Organisaatio, josta tukipyyntö on tullut.
- **issue_priority** - Tukipyynnön prioriteetin kuvaus, kuten ”None”, ”Class 3 – Normal”, “Class 2 – Severe”, “Class 1 – Critical”.
- **jira_createdat** - Aikaleima, joka osoittaa milloin tukipyyntö on luotu Jira-järjestelmässä.
- **pg_createdat** - Aikaleima, joka osoittaa milloin tietue on lisätty PostgreSQL-tietokantaan.

Tämä rakennelma mahdollistaa tärkeiden tukipyyntöjen ominaisuuksien tallentamisen ja järjestämisen tavalla, joka tukee tietojen analysointia ja raportointia. Jokainen kolumni on valittu huolella, jotta se vastaa sekä liiketoiminnallisia että teknisiä vaatimuksia, säilyttäen samalla tietokannan suorituskyvyn ja skaalautuvuuden.

2.8 Grafanan ja PostgreSQLn alusta

Suunnittelun alussa harkittiin Grafanan ja PostgreSQL:n asentamista natiivisti Raspberry Pi:n päälle. Tämä olisi ollut helppo ja yksinkertainen ratkaisu, joka olisi mahdollistanut esimerkiksi ohjelmien asetuksien manipuloimisen vain ottamalla yhteyden Raspberryyn ja tekemällä muutokset. Kuitenkin suunnittelun edetessä, päädyttiin toteuttamaan asennukset Dockerin avulla, hyödyntäen sen tarjoamia etuja.

2.8.1 Docker

Docker on avoimen lähdekoodin alusta, jonka avulla voidaan suorittaa erilaisten ohjelmistojen käyttöönotto, skaalaus ja hallinta konttiteknologian avulla. Kontit ovat pieniä ja kevyitä, virtuaalisia ympäristöjä, jotka sisältävät sovelluksen riippuvuuksineen. Docker kontteja voidaan ajaa riippumatta taustalla olevasta infrastruktuurista. (Docker, n.d.)

Valinta Dockerin käytölle perustui useaan tekijään. Dockerin avulla Grafana ja PostgreSQL saatiin kapseloitua omiin kontteihinsa, mikä lisää järjestelmän joustavuutta ja vikasietoisuutta. Konttiteknologian avulla jokainen palvelu voidaan käynnistää ja sammuttaa erillään toisistaan, mikä mahdollistaa paremman hallinnan ja yksinkertaistaa mahdollisia päivityksiä. Docker-kontit ovat myös siirrettäviä, mikä tarkoittaa, että koko palvelupaketti voidaan siirtää ja ottaa käyttöön uudessa ympäristössä ilman monimutkaisia asennusprosesseja. Tämä ominaisuus on erityisen hyödyllinen, jos projektia halutaan laajentaa tai siirtää se toiseen laitteistoon tai pilvipalveluun tulevaisuudessa.

Dockerin käyttö myös vähentää ympäristöriippuvuuksia, sillä kontit sisältävät kaikki tarvittavat riippuvuudet, mikä tekee sovellusten asetusten yhtenäistämisestä ja virheenjäljityksestä helpompaa. Lisäksi Dockerin volyymit tarjoavat tehokkaan tavan tietojen pysyvään tallennukseen ja hallintaan, mikä on tärkeää tietokantapalvelujen kuten PostgreSQL:n kannalta. Tietoturvan näkökulmasta kontit tarjoavat lisäkerroksen eristystä, sillä jokainen kontti toimii omassa virtuaalisessa tilassaan, mikä minimoi riskin, että tietoturvaongelmat leviäisivät kontista toiseen. Kaiken kaikkiaan Dockerin käyttöönotto Raspberry Pi:ssä tarjoaa modernin, modulaarisen ja skaalautuvan tavan hallita sovelluspalveluita, joka tukee projektin pitkän aikavälin kestävyyttä ja kehittymistä.

2.8.2 Docker Engine

Docker Engine on avoimen lähdekoodin konttivirtualisointialusta, joka mahdollistaa sovellusten paketoinnin kontteihin Linux-ympäristössä. Kontit ovat kevyitä, sillä ne jakavat isäntäkoneen ytimen eivätkä vaadi erillistä käyttöjärjestelmää kullekin kontille, toisin kuin perinteiset virtuaalikoneet. Docker Engine toimii välittäjänä käyttäjän ja käyttöjärjestelmän ytimen välillä, hoitaen konttien elinkaaren hallintaa, kuten niiden luontia, käynnistystä ja lopetusta.

2.8.3 Docker Compose

Projektin edetessä tehtiin päätös ottaa käyttöön docker-compose-työkalu, jonka avulla konttien luominen ja hallinta tehostuu merkittävästi. Docker-compose mahdollistaa määrittelytiedoston, yleensä nimeltään ”docker-compose.yml”, avulla kokonaisvaltaisen konttiympäristön rakentamisen. Tämä tiedosto tarjoaa keskitetyn konfiguraation, joka sisältää kaikki tarvittavat asetukset ja parametrit konttien käynnistämiseen liittyen.

Projektissa suunniteltiin docker-compose.yml-tiedoston versio, jossa määritetään ensin PostgreSQL-palvelun asetukset, jotta tietokanta saadaan pystyyn halutuilla konfiguraatioilla. Tämän jälkeen tiedostoon lisätään Grafana-palvelun konfiguraatio, varmistamaan visualisointityökalun sujuva integraatio ja toimivuus. Molemmat palvelut yhdistetään suunniteltuun erilliseen verkkoasetukseen, docker networkiin, mikä mahdollistaa konttien sisäisen kommunikaation ja tietojen vaihdon. Tämä verkkoasetus on keskeinen, sillä se mahdollistaa Grafanan suoran pääsyn PostgreSQL-tietokantaan, mikä on projektin kannalta kriittistä.

Käyttämällä docker-composea projektissa saavutetaan useita etuja: yksinkertaistetaan monimutkaisten palvelukonfiguraatioiden hallintaa ja parannetaan konttien välisen

verkkoliikenteen hallintaa. Docker-compose tiedoston käyttöönotto edistää projektin modulaarisuutta ja siirrettävyyttä, mikä tekee järjestelmän laajentamisesta tai siirtämisestä muualle saumattoman.

3 Projektin toteutus

Projekti toteutettiin suunnitelmaa noudattaen, yksi vaihe kerrallaan. Toteutus aloitettiin olemassa olevien laitteiden tarkastamisella ja Raspberry Pi 4 -mikrotietokoneen tapauksessa vanhan käyttöjärjestelmäasennuksen poistamisella. Raspberry PI 4 -mikrotietokoneelle asennettiin uusi Raspberry Pi OS, joka mahdollisti sen käytön internet-selaimena, sekä alustana Docker Enginelle. Raspberry PI OS:n asentaminen aloitettiin lataamalla Raspberry Pi Imager -ohjelmisto. Tämä asennettiin työasemalle ja käynnistettiin. Imagerissa valitaan haluttu käyttöjärjestelmäversio ja tallennusmedia, jonka jälkeen käyttöjärjestelmän lataus ja siirto tallennusmedialle aloitetaan. Prosessin valmistuttua, tallennusmedia asetetaan Raspberry Pi 4 -mikrotietokoneeseen ja käynnistetään. Varsinainen asennus on suoraviivainen ja käyttäjäystävällinen, joten se sujui ilman ongelmia. Asennuksen jälkeen Raspberry Pi 4 -mikrotietokoneen asetukset tarkistettiin ja laitteelle asetettiin esim. kieli (englanti), aikavyöhyke (Helsinki) ja luotiin käyttäjätunnukset. Tässä vaiheessa valittiin myös kiinteä IP osoite (10.20.0.29), jotta laite olisi helpommin hallittavissa ja saavutettavissa verkossa.

3.1 Docker Enginen ja docker-composen asennus

Raspberry Pi 4:lle asennettiin Docker Engine, joka mahdollistaa konttien luomisen ja hallinnan. Asennus aloitettiin päivittämällä laitteen paketit ja asennettavat ohjelmistot, jotta alustan tietoturva olisi ajan tasalla. Tämä tehtiin komennolla ”sudo apt update && sudo apt upgrade -y”. Asennus jatkui Dockerin asennuksella, joka suoritettiin Dockerin virallisilta verkkosivuilta löytyvien ohjeiden mukaisesti. Tähän vaadittiin GPG avaimen lisääminen, sekä Dockerin repositorioiden lisääminen. Tämän jälkeen Dockerin asennus suoritettiin loppuun komennolla ”sudo apt install docker-ce docker-ce-cli”. Asennuksen jälkeen Dockerin käynnistys ja automaattinen käynnistyminen käynnistyksen yhteydessä varmistettiin komennolla ”sudo systemctl start docker” ja ”sudo systemctl enable docker”. Tämä tehtiin siksi, että järjestelmä käynnistyisi itsenäisesti, esimerkiksi sähkökatkoksen jälkeen. Docker-compose asennettiin Docker Enginen päälle, jotta konttien hallinta ja konfigurointi olisi helpompaa. Asennus tehtiin komennolla ”sudo apt install docker-compose”.

3.2 Projektin kansiorakenteen ja GIT versionhallinnan alustus

Projektin kansiorakenne luotiin Raspberry Pi 4:lle, jotta projektin tiedostot ja skriptit olisivat helposti hallittavissa ja löydettävissä. Kansiorakenne luotiin komennolla ”mkdir /home/limspi/jiravisualization”. Vastaava rakenne luotiin myös tekijän omalle työasemalle, sekä alustettiin GIT versionhallintaan. Tämä mahdollisti projektin tiedostojen ja skriptien hallinnan ja versionhallinnan, sekä helpotti projektin jakamista ja ylläpitoa. Raspberry PI on varsinkin SSH:n läpi käytettäessä hankala koodin kirjoittamiseen, joten projektin koodit ja skriptit kirjoitettiin tekijän omalla työasemalla ja siirrettiin Raspberry PI:lle GIT versionhallinnan avulla.

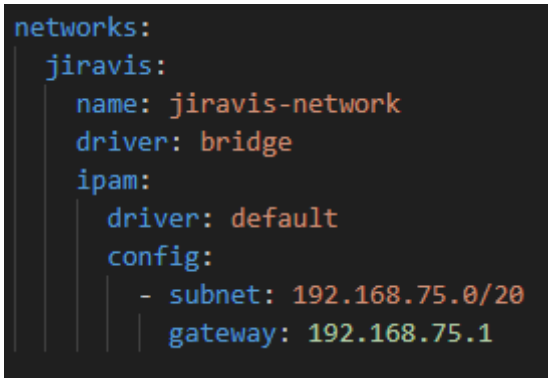
3.3 Docker-compose.yml tiedoston luonti

Projektia varten luotiin docker-compose.yml -tiedosto, joka mahdollistaa konttien luomisen ja hallinnan yhdellä tiedostolla. Tämä tiedosto sisältää kaikki tarvittavat asetukset ja parametrit konttien luomiseen ja käynnistämiseen liittyen. Tiedostoon määriteltiin PostgreSQL ja Grafana -konttien asetukset, sekä verkkoasetukset, jotka mahdollistavat konttien välisen kommunikaation ja tietojen vaihdon.

3.3.1 Docker networkin määrittely

Projektia varten docker -ympäristöön määriteltiin oma verkko, joka mahdollistaa konttien välisen kommunikaation ja tietojen vaihdon. Verkon ja konttien luonti päätettiin hoitaa yhdellä docker-compose.yaml -tiedostolla, jotta kaikki projektin asetukset olisivat keskitetysti yhdessä paikassa. Verkon nimeksi valittiin ”jiravis-network” ja verkkoalueeksi ”192.168.75.0/20”. Verkon yhdyskäytäväksi asetettiin ”192.168.75.1”. Kuva 1 näyttää Docker-verkon konfiguraation.

Kuva 1. jiravis verkon konfiguraatio docker-compose.yaml -tiedostossa



3.3.2 Grafana -kontin määrittely

Grafana määriteltiin docker-composen avulla käyttäen viimeisintä versiota virallisesta Grafana levykuvasta. Kontin nimeksi valittiin ”jiravis-grafana”, uudelleenkäynnistymis-käyttäytyminen asetettiin ”unless-stopped” -tilaa ja valittiin kontin puolelle Grafanan oletusportti 3000. Tämä yhdistettiin hostin porttiin 4999, jotta Grafana olisi helposti saavutettavissa internet-selaimella. Määritettiin myös volyymi, joka mahdollistaa Grafanan asetusten persistoimisen uuudelleenkäynnistyksen jälkeen. Verkkoasetuksiin määriteltiin IP osoitteeksi ”192.168.75.2”. Kuva 2 näyttää Grafana-kontin konfiguraation docker-compose.yaml -tiedostossa.

Kuva 2. Grafana kontin konfiguraatio docker-compose.yaml -tiedostossa

3.3.3 PostgreSQL -kontin määrittely

Projektin seuraava vaihe oli PostgreSQL-tietokannan määrittely docker-compose -tiedostoon. PostgreSQL määriteltiin käyttäen viimeisintä virallista levykuvaa. Levykuvan nimeksi valittiin ”jiravis-postgres”, uudelleenkäynnistyskäyttäytyminen asetettiin ”unless-stopped”, ympäristömuuttujat määriteltiin, sekä valittiin portti 5432, joka on PostgreSQL:n oletusportti. Määritettiin myös volyymi, joka mahdollistaa tietokannan pysyvän tallennuksen ja hallinnan. Tämä on tärkeää, jotta tietokannan tiedot säilyvät myös kontin uudelleenkäynnistyksen jälkeen. Myös verkkoasetukset määriteltiin, jotta tietokanta saadaan liitettyä jiravis-verkkoon. PostgreSQL kontin osoitteeksi valikoitui ”192.168.75.3”. Kuva 3 näyttää PostgreSQL-kontin konfiguraation docker-compose.yaml -tiedostossa.

Kuva 3.PostgreSQL kontin konfiguraatio docker-compose.yaml -tiedostossa

```yaml
services:
  postgres:
    image: postgres:latest
    container_name: jiravis-postgres
    restart: unless-stopped
    ports:
      - 5432:5432
    environment:
      - POSTGRES_USER=${USERNAME}
      - POSTGRES_PASSWORD=${PASSWORD}
      - POSTGRES_DB=${DATABASE}
    volumes:
      - ./data/postgres:/var/lib/postgresql/data
    networks:
      jiravis:
        ipv4_address: 192.168.75.3
```

3.4 Konttien käynnistys ja testaus

Kun docker-compose.yml -tiedosto oli valmis, kontit käynnistettiin komennolla ”docker-compose up -d”. Sekä PostgreSQL että Grafana -kontit käynnistyivät onnistuneesti, ja niiden tila tarkistettiin komennolla ”docker ps”. Listaus näytti, kuva 4:n mukaisesti, molempien konttien olevan käynnissä, ja niiden tila oli ”up”. Tämä tarkoitti, että kontit olivat valmiita käytettäväksi ja testattavaksi.

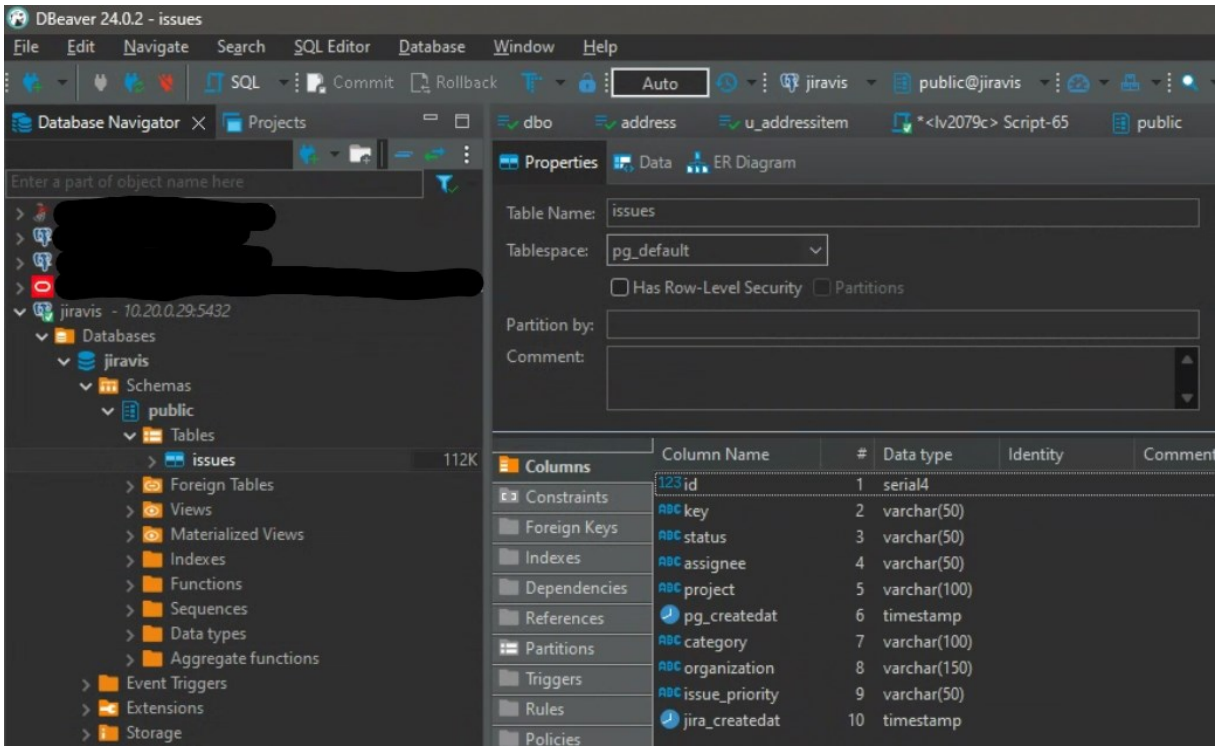
Kuva 4. docker ps -komennolla haettu listaus dockerin aktiivisista konteista

```
limspi@limsinfoberry:~ $ sudo docker ps
CONTAINER ID   IMAGE                     COMMAND                  CREATED         STATUS       PORTS                                       NAMES
b2bfc852417f   postgres:latest           "docker-entrypoint.s…"   1 weeks ago   Up 2 hours   0.0.0.0:5432->5432/tcp, :::5432->5432/tcp   jiravis-postgres
79b32441c88e   grafana/grafana:latest    "/run.sh"                1 weeks ago   Up 2 hours   0.0.0.0:4999->3000/tcp, :::4999->3000/tcp   jiravis-grafana
```

3.5 Tietokannan taulurakenteen luonti

Tietokannan taulurakenteen luonti tehtiin yhdistämällä oman koneen DBeaver -tietokantatyökalu PostgreSQL-tietokantaan. DBeaver on ilmainen ja avoimen lähdekoodin tietokantatyökalu, joka mahdollistaa tietokantojen hallinnan ja käsittelyn graafisen käyttöliittymän kautta. Docker-composella aiemmin luotuun jiravis -tietokantaan, luotiin issues-taulu, joka sisältää kaikki projektin tarvitsemat kolumnit ja attribuutit. Kuvankaappaus taulurakenteesta on alla kuvassa 5

Kuva 5. issues tietokantataulurakenne



3.6 Datan siirto Jirasta PostgreSQL-tietokantaan

Datan siirto Jirasta PostgreSQL-tietokantaan toteutettiin Python-skriptien avulla. Ensimmäisessä vaiheessa luotiin skripti, joka hakee Jirasta kaikki projektit, jotka ovat toteutukselle luodun käyttäjätilin nähtävissä. Kun projektit saatiin haettua, ne tallennettiin .JSON tiedostomuodossa omaan kansioonsa, nimeten tiedosto UUID4 -tunnisteella. Tämän jälkeen luotiin toinen skripti, joka hakee jokaisen projektin avoimet tukipyynnöt, ja tallentaa ne .JSON tiedostmuodossa omaan kansioonsa.

Kun tukipyynnöt on haettu, ne siirretään PostgreSQL-tietokantaan kolmannen skriptin avulla. Tämä skripti lukee aikaisemmin tallennetut tukipyynnöt JSON-muodossa, ja muuntaa ne SQL Insert -lauseiksi, jotka syöttävät tukipyynnöt tietokantaan. Tämä kokonaisuus, mukaan lukien useat apuskriptat mm. kansioiden luomista varten, sekä yhteyden muodostamista tietokantaan varten, nidottiin yhteen pääskriptiin, joka suorittaa kaikki vaiheet yhdellä komennolla.

3.6.1 fetch_projects.py

Projektien hakuskripta luotiin nimellä "fetch-projects.py". Koska Jiran REST-rajapinta tukee enimmillään 50 tuloksen palauttamista yhdellä kutsulla, skriptiin piti luoda silmukka, joka

hakee ensimmäiset 50 tulosta ja tekee sen jälkeen uuden kutsun, kunnes kaikki projektit on haettu. Kuva 7 näyttää fetch-projects.py -skriptan. Kun kaikki projektit on tallennettu välimuistiin, kirjoitetaan ne. JSON-tiedostoon. Tämä tiedosto sisältää kaksi elementtiä, "total" ja "projects". "total" kertoo kuinka monta projektia on haettu ja "projects" sisältää JSON taulukon, joka sisältää kaikkien projektien tunnisteet. Tämä tiedosto nimetään uuid4 -tunnisteella, joka on uniikki jokaiselle tiedostolle. Kuva 6 näyttää esimerkin projects.json-tiedoston struktuurista.

Kuva 6. projects.json-tiedoston struktuuri

Kuva 7. Python skripta, joka hakee uudet projektit REST rajapinnasta

```
data-fetch > windows > fetch_projects.py > ...
import requests
import json
import uuid
import time
import os

temp_data_dir = os.path.join(os.getcwd(), "temp_data")
temp_data_projects_dir = os.path.join(temp_data_dir, "projects")
imported_dir = os.path.join(os.getcwd(), "imported")
imported_projects_dir = os.path.join(imported_dir, "projects")

maxResults = 50
startAt = 0
total = 0

HEADERS = {
    "Content-Type": "application/json",
    "Authorization": [redacted]
}


def get_all_projects_and_save_to_temp_file():
    global startAt
    global total
    resp_json = {
        "data": []
    }

    while True:
        data = requests.get(
            f"https://lims.atlassian.net/rest/api/2/project/search?startAt={startAt}&maxResults={maxResults}", headers=HEADERS)

        length = len(data.json()["values"])
        resp_json["data"].append(data.json())
        startAt += length
        total += length
        if (length < 50):
            break

    filtered_data = {"total": total, "projects": []}

    for project in resp_json["data"]:
        for x in project["values"]:
            if x["projectTypeKey"] == "service_desk":
                filtered_data["projects"].append(x["key"])

    relevant_data = json.dumps(filtered_data, indent=4)

    for file in os.listdir(temp_data_projects_dir):
        if (filtered_data["total"] > 0):
            os.rename(os.path.join(temp_data_projects_dir, file),
                      os.path.join(imported_projects_dir, file))

    with open(f"{temp_data_projects_dir}\\{uuid.uuid4()}_{int(time.time())}.json", "w") as f:
        f.write(relevant_data)


def get_a_list_of_projects():
    projects = []
    for file in os.listdir(temp_data_projects_dir):
        with open(os.path.join(temp_data_projects_dir, file), "r") as f:
            projects = json.load(f)

    return projects
```

3.6.2 fetch_data.py

Toteutuksen toinen vaihe on hakea jokaisesta projektista avoimet tukipyynnöt. Tämä toteutettiin skriptillä "fetch-data.py". Skripti etsii projects -kansiosta .JSON tiedoston ja lukee sen sisällön välimuistiin, listaksi. Tämän jälkeen skripti suorittaa rajapintakyselyn JIRAan, hakien projektin alta löytyvät avoimet tukipyynnöt, joissa on muutoksia viimeisen puolen vuoden aikana. Tämä rajapinta seuraa samaa logiikkaa projektirajapinnan kanssa, joten yli

50 tulosta vaatii useamman kutsun. Vastauksesta parsitaan jokaisen tiketin kohdalla tarvittavat avainkentät, "key", "status", "assignee", "project", "category", "organization", "issue_priority" ja "jira_createdat" ne tallennetaan issues kansioon omaksi .JSON -tiedostokseen. Nämä tiedostot sisältävät kaksi elementtiä, "total" ja "issues". "total" kertoo kuinka monta tukipyyntöä on haettu ja "issues" sisältää JSON taulukon, joka sisältää jokaisen tukipyynnön avainkentät JSON oliona. Tämä tiedosto nimetään projektien tapaan uuid4 -tunnisteella, joka on uniikki jokaiselle tiedostolle. Kuva 8 näyttää esimerkin issues.json -tiedoston struktuurista.

Kuva 8. issues.json tiedoston struktuuri

3.6.3 save_data.py

Kolmas vaihe on tallentaa haetut tukipyynnöt PostgreSQL-tietokantaan. Tämä toteutettiin skriptillä "save-data.py". Skripti käy läpi jokaisen issues -kansiosta löytyvän .JSON tiedoston ja lukee sen sisällön välimuistiin. Tämän jälkeen skripta tarkistaa, että suljettuja / poistettuja tukipyyntöjä ei löydy paikallisesta PostgreSQL-tietokannasta. Mikäli paikallisessa tietokannassa on tukipyyntö, jota ei löydy uusimmasta haetusta datasta, se poistetaan

paikallisesta tietokannasta tässä vaiheessa. Seuraavaksi skripti vertaa haettua dataa paikalliseen tietokantaan ja lisää puuttuvat tukipyynnöt tietokantaan, sekä päivittää muuttuneet tukipyynnöt. Uusien tukipyyntöjen lisääminen paikalliseen tietokantaan toteutettiin SQL Insert -lauseilla, ja muuttuneiden tukipyyntöjen päivitys toteutettiin SQL Update -lauseilla. Kun kaikkien projektien kaikki tukipyynnöt on käyty läpi, skripti siirtää projects ja issues -kansioiden sisällöt arkisto -kansioon, jotta seuraavassa ajossa voidaan aloittaa puhtaalta pöydältä.

3.6.4 control_script.py

Skriptakokonaisuus yhdistettiin yhdeksi pääskriptiksi, joka suorittaa kaikki vaiheet yhdellä komennolla. Tämä skripti nimettiin "control-script.py". Ensimmäinen vaihe on ajaa apuskripta "create_folders.py", joka luo tarvittavat kansiot, mikäli niitä ei jo ole olemassa. Kun kansiot on luotu onnistuneesti tai niiden olemassaolo on varmistettu, skripti jatkaa seuraavaan vaiheeseen, joka on projektien ja avoimien tukipyyntöjen hakeminen JIRA:n rajapinnasta ja tallentaminen. JSON tiedostoihin. Kun tallennus on onnistuneesti suoritettu, skripta odottaa 5 sekuntia, jonka jälkeen suoritetaan haetun datan lukeminen levyltä, sekä kirjoittaminen PostgreSQL-tietokantaan. Tämä skripta myös tulostaa konsoliin tietoja suoritetuista toimenpiteistä ja haettujen tietojen lukumääristä. Kuva 9 on esimerkki skriptan tulosteesta, kun se ajetaan komennolla ”python .\control_script.py”

Kuva 9. esimerkkituloste control_script.py python skriptan ajamisesta

3.6.5 Utility skriptat ja ulkoiset kirjastot

Jotta skriptakokonaisuuden ajo sujuisi mahdollisimman sujuvasti ja virheettömästi, luotiin useita apuskriptoja, jotka hoitavat esimerkiksi kansioiden luomisen, tietokantayhteyden muodostamisen ja tietokantakyselyiden suorittamisen. Luotiin myös esimerkiksi ylläpidollisia

toimia varten skripta, joka poistaa tietokannasta kaiken datan. Tämä on hyödyllinen toiminto, koska tietokannan sisältämä data on käytännössä vain kopio Jiran tietokannasta, joten sen poistaminen ei aiheuta merkittävää haittaa. Tätä voidaan hyödyntää esimerkiksi ajastettuna toimenpiteenä, jotta kanta ei pääse kasvamaan liian suureksi. Tietokantayhteyksiin käytettiin ulkoista kirjastoa psycopg2, joka mahdollistaa PostgreSQL-tietokantayhteyden muodostamisen ja kyselyiden suorittamisen Python-skriptissä.

3.7 Skriptien ajo ja automatisointi

Skriptien ajo ja automatisointi toteutettiin käyttämällä Linuxin cron -palvelua. Cron on Unix-pohjainen ajastettu tehtävien suorituspalvelu, joka mahdollistaa komentojen suorittamisen ajastetusti. Ensin ajastettiin pääskripti "control_script.py". Tämän skriptin ajastus vaikuttaa siihen, kuinka usein tietokantaan haetaan uutta dataa Jirasta ja samalla se on siis vastuussa visualisoinnin ajantasaisuudesta. Skripta ajastettiin ajettavaksi viiden minuutin välein, jotta data infonäytöllä olisi mahdollisimman tuoretta, kuitenkaan joutumatta tilanteeseen, jossa edellinen ajo on vielä kesken. Seuraavaksi määriteltiin ajastus, joka poistaa väliaikaiskansioista, joihin rajapintakutsuilla haetaan dataa, kaikki tiedostot. Imported -kansioista data poistetaan harvemmin, koska se on dataa, joka on jo siirretty PostgreSQL-tietokantaan. Tämä ajo suoritetaan kolme minuuttia puolenyön jälkeen, päivittäin. temp_data -kansioista data poistetaan kolmen tunnin välein. Seuraava ajastus on vastuussa tietokannan tyhjentämisestä, se ajetaan joka päivä kolme minuuttia yli puolenyön. Tämä on tärkeää, jotta tietokanta ei kasva liian suureksi ja hidasta järjestelmää. Tämä on myös hyödyllinen toiminto, jos tietokantaan on tullut virheellistä dataa, joka halutaan poistaa. Lopuksi määriteltiin ajastus, joka poistaa kaikki vanhat cronin keräämät logitiedostot, jotka ovat yli viikon vanhoja. Tämä koettiin tärkeäksi lisätä, koska logitiedostojen määrät paisuvat nopeasti suureksi, ja ne hidastavat järjestelmää. Kuvassa 10, kuvankaappaus cron -ajastuksista.

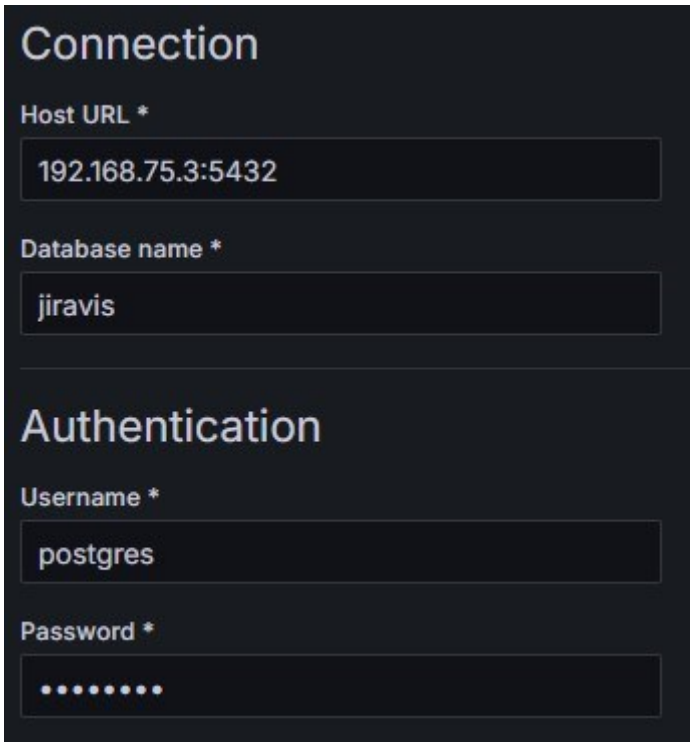
Kuva 10. tuloste crontab -e komennon ajamisesta Raspberry Pi:llä

```
*/5 * * * * /usr/bin/python /home/limspi/dev/jiravisualization/data-fetch/control_script_linux.py >> /home/limspi/dev/jiravisualization/cronlogs/`date +\%Y\%m\%d\%H\%M\%S`-cron.log 2>&1
3 */24 * * * rm -rf /home/limspi/dev/jiravisualization/data-fetch/imported/projects/*
3 */24 * * * rm -rf /home/limspi/dev/jiravisualization/data-fetch/imported/issues/*
3 */3 * * * rm -rf /home/limspi/dev/jiravisualization/data-fetch/temp_data/projects/*
3 */3 * * * rm -rf /home/limspi/dev/jiravisualization/data-fetch/temp_data/issues/*
3 */24 * * * /usr/bin/python /home/limspi/dev/jiravisualization/data-fetch/linux/refresh_linux.py >> /home/limspi/dev/jiravisualization/cronlogs/REFRESH-`date +\%Y\%m\%d\%H\%M\%S`-cron.log 2>&1
0 0 * * * /usr/bin/find dev/jiravisualization/cronlogs -type f -mtime +7 -exec rm -f {} \;
```

3.8 Grafanan konfigurointi

Grafanan konfigurointi aloitettiin avaamalla internet-selain ja kirjoittamalla osoitekenttään Raspberry Pi:n kiinteä IP-osoite ja portti 4999. Tämä avasi Grafanan kirjautumissivun, josta kirjauduttiin sisään oletuskäyttäjätunnuksilla admin/admin. Grafanan suhteen ensimmäiseksi konfiguroitiin tietokantayhteys PostgreSQL-tietokantaan. Tämä konfiguraatio on esitelty kuvassa 11. Tämä tehtiin Grafanan sivupalkista "Data sources" -linkistä ja valitsemalla "Add data source". Data sourcen lähteeksi valittiin PostgreSQL ja täydennettiin vaaditut kentät. Yhteysosoitteeksi tässä kohtaa piti käyttää PostgreSQL -kontin IP-osoitetta, joka oli määritelty docker-compose.yml -tiedostossa. Tallennuksen yhteydessä Grafana testasi yhteyden tietokantaan ja ilmoitti tämän onnistuneeksi. Kuva 12 näyttää onnistuneen yhteyden luonnin Grafanan ja PostgreSQL:n välillä.

Kuva 11. PostgreSQL tietolähteen yhteysasetukset Grafanassa



Kuva 12. Grafana ilmoitaa onnistuneesta yhteyden muodostuksesta

3.8.1 All Teams Highlight dashboard

Grafanaa varten luotiin kolme erillistä Dashboard -näkymää, jotka visualisoivat tietokannasta haettua dataa. Ensimmäinen dashboard visualisoi kaikkien tiimien avoimet tukipyynnöt, jaettuina kolmeen eri tilaan, "Unassigned", "Waiting for support" ja "Unresolved". Tämä dashboard nimettiin "All Teams Highlight" ja se sisältää 4x3 gridin, jossa jokainen solu on oma gauge -tyyppinen visualisointi. Kuva 13 näyttää All Teams Highlight -näkymän.

Kuva 13. Grafanan All Teams Highlight dashboard näkymä

3.8.2 All Teams Critical & Severe dashboard

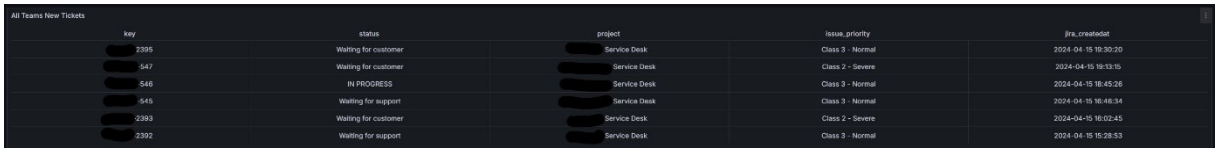
Toinen dashboard visualisoi kaikkien tiimien avoimet tukipyynnöt, jotka ovat prioriteetiltaan "Critical" tai "Severe". Tämä dashboard nimettiin "All Teams Critical & Severe" ja se sisältää kaksi erillistä gauge -tyyppistä visualisointia. Toinen näistä visualisoi "Critical" -prioriteetin tukipyynnöt ja toinen "Severe" -prioriteetin tukipyynnöt. Kuva 14 näyttää All Teams Critical & Severe -näkymän.

Kuva 14. Grafanan All Teams Critical & Severe dashboard näkymä

3.8.3 All Teams New Tickets dashboard

Kolmas ja viimeinen dashboard visualisoi kaikkien tiimien, viimeisen 24 tunnin aikana luodut tukipyynnöt. Tämä dashboard nimettiin "All Teams New Tickets" ja se sisältää yhden table -tyyppisen visualisoinnin, joka näyttää kaikki uudet tukipyynnöt, niiden tikettinumeron, tilan, projektin, prioriteetin ja luontiajan. Kuva 15 näyttää All Teams New Tickets -näkymän.

Kuva 15. Grafanan All Teams New Tickets dashboard näkymä



3.8.4 Dashboardien näyttäminen infonäytöllä

Grafanasta löytyy kätevä ominaisuus, jolla eri näkymiä voidaan pyörittää näytöllä vuoron perään. Tätä ominaisuutta käytettiin hyväksi, jotta saatiin jokainen dashboard näkymään 30 sekunnin välein. Tämän ansiosta infonäytöllä näkyy jokainen dashboard, eikä näkymiä tarvitse vaihtaa käsin. Tämän ominaisuuden nimi Grafanassa on "Playlist" ja se löytyy sivupalkista "Dashboards" -linkin alta. Playlistiin voidaan valita näytettävät dashboardit ja niiden järjestys, sekä näytettävän dashboardin vaihtoväli.

4 Projektin tulokset

Tämän projektin tavoitteena oli rakentaa järjestelmä, joka hyödyntää Jiran REST-rajapintaa asiakasyrityksen asiakaspalvelutiimien tukipyyntöjen hakemiseen, sekä mikrotietokonetta ja infonäyttöä näiden visualisointiin. Tavoite saavutettiin onnistuneesti ja infonäyttö on ollut käytössä asiakaspalvelutiimien tukena valmistumisen jälkeen.

4.1 Haasteet

Projektin aikana kohdattiin haasteita, kuten Docker Enginen verkkoasetuksien oikeaoppinen konfigurointi, sekä Jiran REST-rajapinnan rajoitteet. Näistä ongelmista kuitenkin selvittiin kunnialla. Erityisesti REST-rajapinnan rajoitukset, kuten vain viidenkymmenen tukipyynnön hakeminen yhdellä kutsulla, vaativat luovia ratkaisuja ja teknistä osaamista.

4.2 Jatkokehitys

Tällä projektilla on potentiaalia jatkokehitykseen. Esimerkiksi useampien WSP:n asiakaspalvelutiimien tuominen palvelun piiriin, monipuolisemman ja kattavamman datan hakeminen Jirasta, sekä erityisesti python scriptien optimoinnin ja virhesietoisuuden lisääminen. Koko kokonaisuuden hallintaan voisi myös jatkossa kehittää web-portaalin, josta yleisimpien ylläpitotoimenpiteiden suorittaminen onnistuisi graafisen käyttöliittymän kautta.

4.3 Yhteenveto

Projekti oli kaikkiaan opettavainen ja avasi mahdollisuuksia tulevaisuuden projekteja varten. Haasteita kohdattiin, mutta niistä päästiin johdonmukaisesti yli. Projekti toi kokonaisuutena lisäarvoa yrityksen toimintaan, sekä kehitystä asiakaspalvelutiimien työskentelyn

tehokkuuteen. Projektin tulokset ovat aktiivisessa käytöksessä asiakasyrityksessä, mikä osoittaa sen käytännön hyödyn ja arvon yritykselle.

Lähteet

Benner, E. (2022). *TOP DATA VISUALISATION TOOLS FOR 2024*. Haettu 15.5.2024 osoitteesta
https://logit.io/blog/post/data-visualisation-tools/

Chapman, C. (2019). *A Complete Overview of the Best Data Visualization Tools*. Haettu 15.5.2024 osoitteesta
https://www.toptal.com/designers/data-visualization/data-visualization-tools

Docker. (n.d.). *Docker overview*. Haettu 15.5.2024 osoitteesta
https://docs.docker.com/get-started/overview/

Grafana. (n.d.). *About Grafana*. Haettu 16.4.2024 osoitteesta
https://grafana.com/docs/grafana/latest/introduction/

Influxdata. (n.d.). *InfluxDB*. Haettu 18.4.2024 osoitteesta
https://www.influxdata.com/

Jira. (n.d.). *Welcome to Jira*. Haettu 14.5.2024 osoitteesta
https://www.atlassian.com/software/jira/guides/getting-started/introduction#what-is-jira-software

Linster, M. (2023). *Postgres is the Undisputed Most Admired and Desired Database in Stack Overflow's 2023 Developer Survey*. Haettu 15.5.2024 osoitteesta
https://www.enterprisedb.com/blog/postgres-most-admired-database-in-stack-overflow-2023

Microsoft. (n.d.-a). *What is Power BI*? Haettu 16.4.2024 osoitteesta
https://learn.microsoft.com/en-us/power-bi/fundamentals/power-bi-overview

Microsoft. (n.d.-b). *What is SQL Server*? Haettu 18.4.2024 osoitteesta
https://learn.microsoft.com/en-us/sql/sql-server/what-is-sql-server?view=sql-server-ver16

PostgreSQL. (n.d.). *PostgreSQL: The World's Most Advanced Open Source Relational Database*. Haettu 18.4.2024 osoitteesta
https://www.postgresql.org/

Requests. (n.d.). *Requests: HTTP for Humans™*. Haettu 18.4.2024 osoitteesta https://requests.readthedocs.io/en/latest/