



Yrityksen palvelinten hallintapaneeli ja sen kehitys

Lenni Korhonen

Haaga-Helia ammattikorkeakoulu
Tradenomi
Opinnäytetyö
2022

Tekijä(t) Lenni Korhonen
Tutkinto Tradenomi
Raportin/Opinnäytetyön nimi Yrityksen palvelinten hallintapaneeli ja sen kehitys
Sivu- ja liitesivumäärä 19 + 0
<p>Tämä on opinnäytetyö raporttini toiminnallisesta opinnäytetyöstä. Opinnäytetyössä kehitetään palvelinten hallintapaneeli toimeksiantaja yritykselle. Toteutuksessa käytetään Ruby on Rails frameworkia sekä PostgreSQL tietokannan hallintajärjestelmää. Toteutus kehitetään ja otetaan käyttöön hyödyntäen Docker ympäristöä.</p> <p>Teoreettisessa osuudessa käydään läpi projektissa käytettäviä teknologiota sekä DevOps kehitys menetelmää. Osiossa käydään Rubyn on Railsin taustaa ja tapoja läpi sekä sen ohjelmointi tyyliä. Myös Railsin käyttämä MVC-malli esitellään sekä mainitaan käytettävä tietokanta ja css-kirjasto.</p> <p>Teorian jälkeen määritellään projektissa toteutettavan järjestelmän vaatimuksia. Vaatimukset on eritelty neljään osa-alueeseen, joista yksi on jatkokehitys.</p> <p>Empiirisessä osuudessa kerrotaan projektin etenemisestä sekä esitellään projektin arkkitehtuuria. Osuudessa käydään läpi mitä asioita tehtiin ja miten ne tehtiin, jotta projekti onnistuisi vaatimusten mukaisesti. Kehitysmenetelmän näkyminen kehityksessä tulee myös ilmi empiiria osuudesta. Lopuksi vielä esitellään projektin arkkitehtuuria ja sen toimintaa.</p>
Asiasanat Ruby, ohjelmistokehitys, toiminnanohjausjärjestelmät

Sisällys

1	Johdanto.....	2
1.1	Käsitteet.....	3
2	Kehitykseen valitut teknologiat sekä kehitysmenetelmät.....	4
2.1	Ruby on Rails frameworkilla kehittäminen.....	4
2.2	Tietokanta sekä CSS-kirjasto.....	5
2.3	Kehitys menetelmänä DevOps.....	6
3	Järjestelmän vaatimukset.....	8
3.1	Toiminnalliset vaatimukset.....	8
3.2	Laadulliset vaatimukset.....	8
3.3	Resurssivaatimukset.....	9
3.4	Jatkokehitysideat	9
4	Empiirinen osuus	10
4.1	Projektin aloitus	10
4.2	Kehitystyö	11
4.3	DevOps menetelmien hyödyntäminen kehitystyössä	14
4.4	Kehitettävän projektin arkkitehtuuri	15
5	Pohdinta	17
5.1	Onnistumiset ja haasteet	17
5.2	Projektista opittua	17
5.3	Toimeksiantajan hyödyt	17
	Lähteet.....	19

1 Johdanto

Tämä on opinnäytetyöraportti toiminnallisesta opinnäytetyöstä. Tavoitteena opinnäytetyössä on kehittää selainpohjainen ohjelmisto, josta pystyy hallitsemaan ja ylläpitämään palvelimia. Ohjelmisto kehitetään Ruby on Rails frameworkilla ja sen tuotanto versio pyörii palvelimella Docker kontissa. Myös kehityksessä hyödynnetään Docker ympäristöä.

Raportti käsittelee kehitettävän ohjelmiston kehittämisprosessia, jonka tuloksena on hallintapaneeli, jolla hallitaan yrityksen asiakaspalvelimia. Projektin hallinnassa hyödynnetään DevOps menetelmiä ja pidetään kerran viikossa Scrumin tyyppinen status, jossa toimeksiantaja antaa palautetta sen hetkisestä versiosta.

Toimeksiantajalla on useita ylläpidettäviä palvelimia, joille on asennettu avoimen lähdekoodin toiminnanohjausjärjestelmä. Palvelinten ylläpitäminen on kriittistä ja tällä hetkellä toimeksiantaja käyttää siihen Ansible työkalua, jota käytetään komentoriviltä. Toimeksiantajalla on kuitenkin tarve graafiselle käyttöliittymälle, josta näkee selkeästi tarvittavat tiedot palvelimelta.

Projektin tavoitteena on tehdä hallintapaneeli yrityksen ylläpitämille palvelimille, josta näkee selkeästi missä versiossa toiminnanohjausjärjestelmä on. Hallintapaneelistä täytyy myös pystyä päivittämään versiota ja vaihtamaan Git -oksaa sekä näkemään palvelimella olevien ohjelmointikielien ja frameworkkien versiot. Datan paneeli saa palvelimilta, jotka lähettävät skriptillä tarvittavan datan JSON rajapintaan. Kun uudet palvelimet lähettävät dataa niiden on automaattisesti tultava näkyviin palvelimen host nimellä. Myös palvelimen käynnissä oloaika on näkyvissä, lokitiedostoja pystyy selaamaan ja palvelimen pystyy käynnistämään uudelleen. Tiedot toiminnallisuudet ovat näkyvissä vain kirjautuneille käyttäjille. Tehty hallintapaneeli asennetaan omalle palvelimelle.

Projektista on rajattu pois sellaiset palvelimen tiedot, joita tavoitteissa ei ole mainittu. Myöskään muita palvelimen ohjelmia ei pysty päivittämään tai itse palvelimen käyttöjärjestelmää tai sen versiota ei pysty päivittämään.

Projektin oppimistavoitteena on ymmärtää enemmän Ruby on Rails frameworkilla kehittämisestä sekä oppia luomaan Docker ympäristö ja käyttämään sitä tehokkaasti. Ruby on Rails on tekijälle jo tuttu työn kautta, mutta Docker ympäristön luonti on lähes uusi asia.

Toimeksiantajan hyötyjä projektista ovat muun muassa parempi ymmärrys palvelimelle asennetun ohjelmiston tilasta, joka helpottaa esimerkiksi asiakkaiden ongelmien ratkaisua. Myös uusien ominaisuuksien testaaminen on helpompaa Git -oksan vaihtamisella ja toiminnanohjausjärjestelmän päivitys ei jää asiakkaiden palvelimilla jälkeen.

1.1 Käsitteet

CSRF on hyökkäys web ohjelmistoihin. Tällä hyökkäyksellä voi pakottaa käyttöliittymän autentikoidun käyttäjän suorittamaan ei-toivottuja toimintoja (Spilca 2020, luku 10.1).

Framework on valmis ohjelmakehys, jota käytetään ohjelman vakiorakenteen toteuttamiseen

JSON on datan vaihto formaatti, jota käytetään datan siirtoon järjestelmien välillä (Basset 2015, luku 1).

Lokitiedosto tai loki on tekstitiedosto, johon kertyy tietoa ohjelmistojen tai taustaohjelmien tilasta.

MVC on ohjelmointi termi, joka tarkoittaa ohjelmiston datan, esittelyn ja näitä yhdistävän kerroksen erittelyä ohjelmistossa ja sen koodissa (Pitt 2021, etusivun sisältö What Is MVC).

Post-metodi on http-protokollan metodi, jolla lähetetään tietyssä muodossa olevaa dataa palvelimelle ja luodaan mahdollisesti uusi tietue (Tutorialspoint s.a.)

Put-metodi on http-protokollan metodi, jolla päivitetään olemassa olevaa tietuetta palvelimella (Tutorialspoint s.a.)

Skripti on teksti tiedosto, joka sisältää komentoja, joita pystyy suorittamaan ilman, että sitä käännetään binääriksi.

Ssh on protokolla, palvelimen ja käyttäjän välille luodaan salattu yhteys (SSH academy, s.a.)

Tietue on looginen kokonaisuus, joka on muodostettu yhden kohteen tiedoista. Esimerkiksi sairaalan potilas, jonka tiedot ovat tietokannassa, on tietue.

Toiminnanohjausjärjestelmä on kokoelma yrityksen toimintaa avustavia ohjelmistomoduuleja (Luomala 2010, 6)

2 Kehitykseen valitut teknologiat sekä kehitysmenetelmät

Kehityksen teknologioiksi valittiin Ruby on Rails framework, PostgreSQL tietokanta sekä Tailwindcss css-kirjasto. Projektin tuotosta kehitetään Docker-alustaa hyödyntäen ja tuotoksen valmis versio tulee myös käyttämään Docker-alustaa. Ohjelmiston kehitykseen valittiin Ruby ohjelmointikielen framework, Ruby on Rails, koska toimeksianto yrityksessä on jo paljon osaamista kyseisestä frameworkista. Sille löytyy myös hyvät ja kattavat dokumentaatiot sekä ohjelmiston rungon saa luotua vain muutamalla komennolla komentorivillä.

2.1 Ruby on Rails frameworkilla kehittäminen

Ruby on Rails, tunnetaan myös nimellä Rails, on vuonna 2004 julkaistu (David 2005) Ruby ohjelmointikielellä kirjoitettu framework. Rails:a on kehitetty selain pohjaisille ohjelmistoille ja se seuraa MVC (sanoista Model-View-Controller eli Malli-Näkymä-Käsittelijä) -mallia (Rails Guides s.a.).

Rails:lla kehittäessä suositellaan tietynlaista ohjelmointityyliä ja tämä tyyli on todella tarkka. Jotta tämän tyylin noudattaminen olisi helpompaa on Rubylle sekä Rails:lle tehty RuboCop niminen ohjelmointityylin tarkistin. RuboCoppia voi myös konfiguroida, jos haluaa itse määritellä jotain sääntöjä tai vaihtaa oletus sääntöjen asetuksia. Kun RuboCop huomaa jonkun tyylistä poikkeavan kohdan koodissa se antaa huomautuksen tekstin alle. Kun tämän huomautuksen päälle vie kursorin niin näytetään suositeltu tyyli koodille. Usein esimerkiksi if/else oikeellisuus tarkistuksen RuboCop haluaa korvata Rubysta löytyvillä guard clauseksi kutsutulla tavalla (Kuva 1.).

```
next logger.info('Id not found') if id.nil?
```

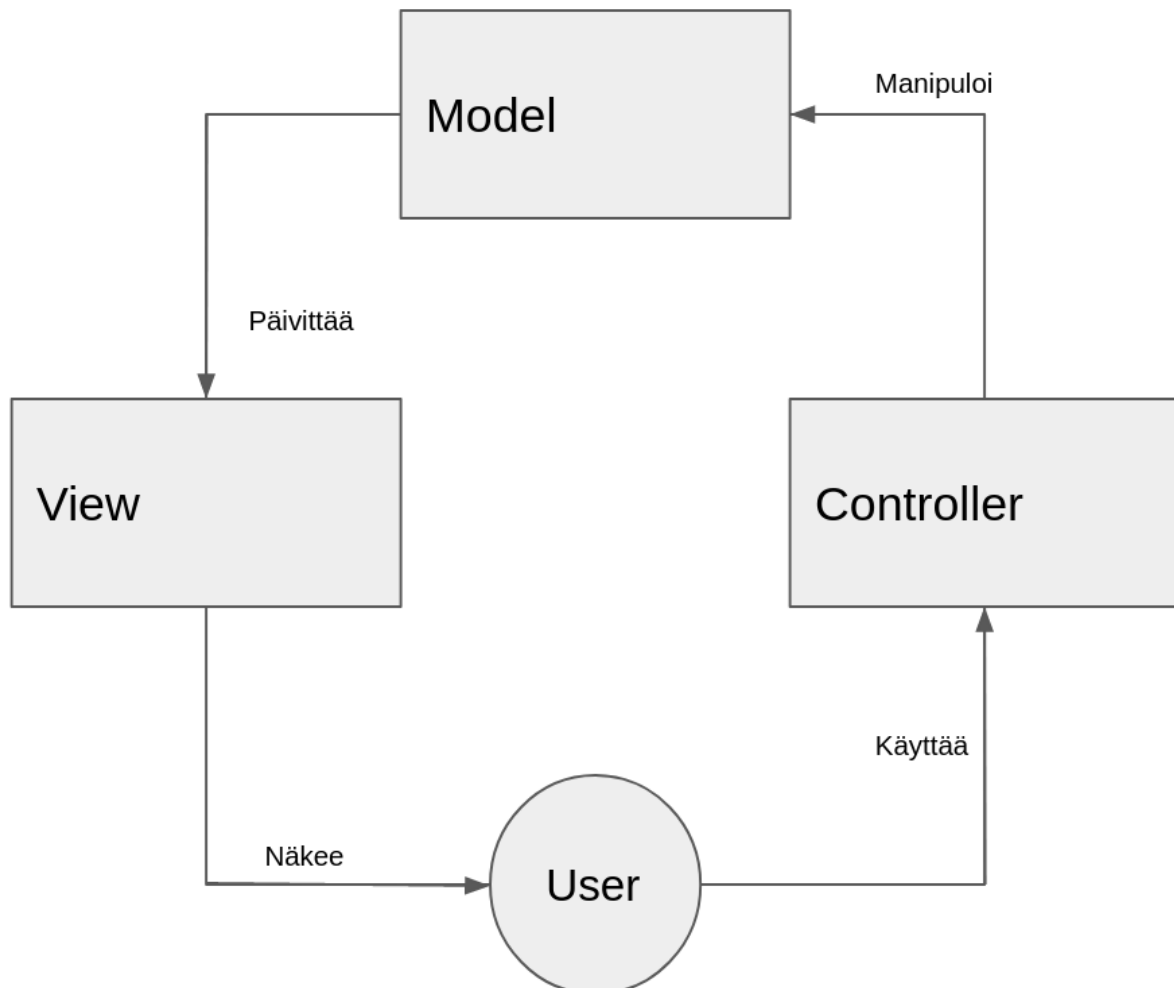
```
Server.find_by(id: id)
```

Kuva 1 Esimerkki guard clausesta

Edeltävä koodi esimerkki palauttaa lokina tekstin "Id not found" ja menee silmukan seuraavaan indeksiin, jos id muuttaja on tyhjä, muutoin etsitään se palvelin, jonka id on tällä hetkellä id muuttajaan asetettuna. Jos tämän hetken viimeinen rivi olisi metodin viimeinen rivi niin metodi palauttaisi sen palvelimen mikä sen hetkellä id:llä löytyy, Rubyssa ei vaadita siis erillistä "return" kutsua. Esimerkissä näkyvä "nil?" on Rubyn metodi, joka palauttaa arvon true(tosi) tai false(epätosi).

Railsin seuraama MVC-malli (Kuva 2.) suunniteltiin ratkaisemaan ongelma, jossa käyttäjillä oli pääsyys ohjaamaan laajaa tietoa kokonaisuutta (Reenskaug, T 2007). MVC-mallissa ohjelman käyttäjälle näkyy näkymä mutta hän ohjaa käsittelijää. Tämä käsittelijä taas manipuloi mallia ja

malli päivittää näkymää, jossa käyttäjä näkee tekemänsä muutokset. Tämän mallin hyötyjä kehittäjälle ovat mm. koodin organisointi, sekä nopeampi kehitys prosessi (Geeks for Geeks 2021). Hyvin organisoitua koodia on nopeampi ja helpompi kehittää, kun kehitettävä koodi löytyy helposti.



Kuva 2 MVC malli

2.2 Tietokanta sekä CSS-kirjasto

Projektin tietokannan hallinta ohjelmaksi valittiin PostgreSQL, koska se tukee paremmin Railsin omia ORM-metodeja kuten projektissa käytettävä upsert, eli päivitä tai lisää, metodi. Tämä metodi siis päivittää tietokannasta löytyvän riviä, mutta jos etsityllä arvolla ei löydy riviä, lisätään uusi rivi annetulla datalla tietokantaan. Valittu tietokanta on myös tekijälle tuttu, joten se oli siltä kannalta varmin valinta.

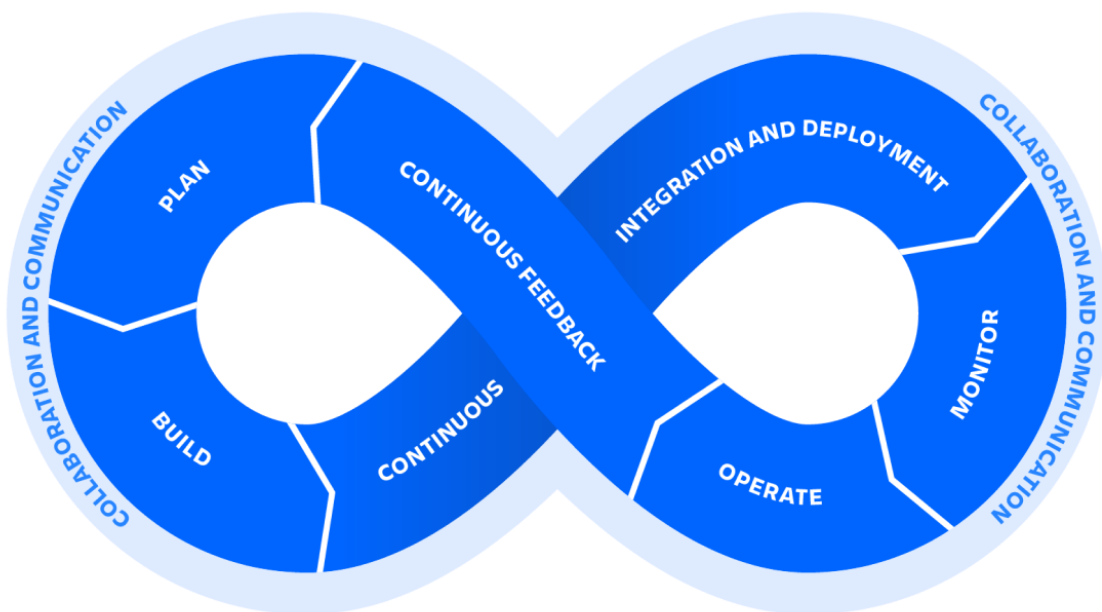
Projektissa haluttiin käyttää jotain CSS-kirjastoa, jotta voi käyttää valmiita komponentteja ja ei tarvitse itse kirjoittaa kaikkia tyylejä itse. Tähän tarkoitukseen valittiin Tailwindcss -kirjasto, koska sillä on hyvä yhteen sopivuus Railsin kanssa ja siltä löytyy hyvät dokumentaatiot.

2.3 Kehitys menetelmänä DevOps

Toimeksiantaja yrityksessä ei suoranaisesti ole mitään kehitys menetelmää, mutta heillä on käytössä heille sopivia palasia DevOps käytännöistä sekä jotain pientä Scrum käytännöistä. Tämän takia projektissa käytettiin vastaavaa kehitys menetelmää.

DevOps ei ole mikään uusi käsite ohjelmistokehityksen ympärillä mutta moni ei silti tiedä mitä se tarkoittaa. Yksinkertaisuudessaan DevOps tarkoittaa kehittäjien ja IT-operaatio tiimin välisten prosessien integraatiota sekä automaatiota (Atlassian s.a.). Osassa DevOps tiimeistä on erikseen kehittäjätiimi sekä IT-operaatio tiimi, toisissa taas nämä tiimit ovat yhdistyneet kokonaan yhdeksi tiimiksi.

Jatkuva integraatio ja julkaisu sekä palaute ovat osa DevOps menetelmää. Projektit, jotka tehdään hyödyntäen DevOps:a seuraa yleensä seuraavanlaista silmukkaa (Kuva 3):



Kuva 3 DevOps rakenne (Atlassian s.a.)

DevOps:ssa hyödynnetään automaatio työkaluja esimerkiksi versionhallinnassa ja tällä myös tuetaan testaus lähtöistä kehitystä. Kun uusi versio ohjelmasta viedään versionhallintaan, ajetaan sille ensin testit, jotta se toimii edelleen oikein ja vasta tämän jälkeen on uusi versio vietävissä tuotanto haaraan. Tuotanto versio taas viedään asiakkaiden palvelimelle esimerkiksi Ansible

työkalulla, jolla saa yhdellä komennolla vietyä muutokset kaikille asiakkaille. Ohjelmisto myös käynnistetään palvelimella samalla komennolla. Joskus taas käytetään Docker alustaa, joka on kuin luotu DevOps:a varten. Dockerilla voi luoda valmiin kehitys ympäristön, joka pyörii jokaisella laitteella ja sitä voi käyttää kaikissa kehityksen vaiheissa (KodeKloud 2020). Tällä saadaan varmistettua jo kehitys vaiheessa, että luotu ohjelmisto toimii tuotannossa.

3 Järjestelmän vaatimukset

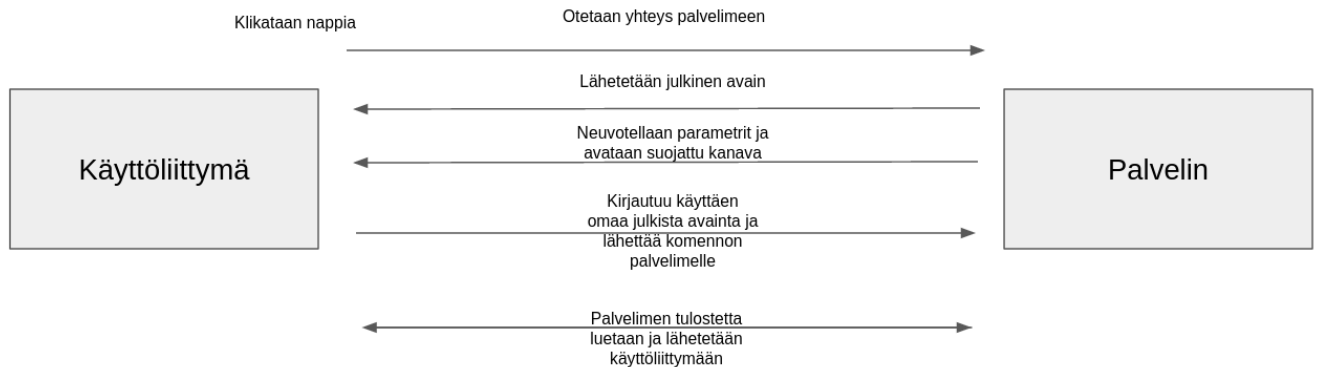
Kehitettävälle ohjelmistolle ei tehty perinteistä vaatimusmäärittely dokumenttia, mutta vaatimuksista käytiin kuitenkin palaveri ja otettiin ylös listaan mitä toiminnallisia vaatimuksia ohjelmistolle on. Laadulliset vaatimukset ja resurssivaatimukset käytiin palaverissa vain suullisesti läpi. Projektille on tullut myös joitain jatkokehitysideoita projektin alussa sekä projektin aikana.

3.1 Toiminnalliset vaatimukset

Lopullisen ohjelmiston toiminnallisuuksissa kuului olla toiminnanohjausjärjestelmän päivittäminen sekä git-oksan vaihtaminen. Nämä toiminnallisuudet olivat toimeksiantajalle tärkeitä, koska ne helpottavat heitä viemään kokeellisia toiminnallisuuksia palvelimelle sekä päivittämään toiminnanohjausjärjestelmää. Päivitys toiminnallisuudessa täytyi myös nähdä päivityksen tuloste, jos tapahtuu virheitä. Ohjelmistoon haluttiin myös kirjautuminen ja jokaiselle tietueelle täytyi olla näytä, muokkaa ja poista ominaisuudet. Uuden tietueen lisääminen oli myös yksi vaatimuksista. Kun järjestelmän ulkopuolelta tulee pyyntö lisätä uusi tietue, täytyy ensin tarkistaa, onko saman nimistä tietuetta jo olemassa ja jos on, niin silloin vain päivitetään olemassa oleva tietue. Valmiissa ohjelmistossa kuuluu myös olla käyttöliittymä sekä JSON-ohjelmointirajapinta. Käyttöliittymässä täytyy näkyä, kaikki tietueet listassa. Tietueilla kuuluu olla sarakkeina palvelimen nimi, git-oksa ja riippuvuuksien versiot sekä palvelimen käynnissä oloaika ja viimeinen päivitys päivä. Ohjelmistoa tukemaan täytyi myös tehdä skripti, joka lähettää vaadittuja tietoja ohjelmistoon.

3.2 Laadulliset vaatimukset

Toiminnanohjausjärjestelmän ja git-oksan vaihtaminen päätettiin tehdä ssh-yhteydellä, joka käyttää ssh-avaimia, koska tietoturvasyistä palvelimille ei voi kirjautua salasanalla. Ssh-yhteys ja sillä lähetettävä komento tehdään käyttöliittymästä klikkaamalla nappia, joka ottaa palvelimen nimen tietueen nimestä (Kuva 4). Ohjelmisto ottaa tiedot vastaan JSON-ohjelmointirajapintaan post-kutsulla, joka käyttää päivitä tai luo (upsert) metodia, jotta dataa lähettävä skripti olisi mahdollisimman yksinkertainen. Ohjelmisto täytyi kirjoittaa siten, että siinä olisi mahdollisimman vähän toistuvaa koodia, jotta se olisi helposti ylläpidettävä. Ohjelmiston suorituskyvyllä ei ole määritelty mitattavia vaatimuksia, mutta toiveena on, että ohjelmiston on toimittava mahdollisimman matala tehoisella raudalla.



Kuva 4: Järjestelmän päivitys ssh:lla

3.3 Resurssivaatimukset

Projektin kustannuksiin kuuluu palvelin, ja sen kuukausi kulut, jolle tuotos asennetaan.

3.4 Jatkokehitysideat

Projektille on myös suunniteltu jo jatkokehitystä. Jatkokehityksessä näkymään lisättäisiin osalla asiakkaita oleva Wordpress sisällönhallintajärjestelmän versio. Näillä näkymin tätä Wordpress versiota ei kuitenkaan voisi päivittää käyttöliittymästä. Jatkokehityksessä tehtäisiin myös datan lähetys skriptistä binääri -tiedosto, jolloin skripti voi itsessään sisältää riippuvuuksia mutta niitä ei tarvitse asentaa palvelimelle. Binääri tiedostot ovat yleensä myös pienempiä kuin teksti tiedostot ja niiden syöttö- ja tulostusnopeudet ovat nopeampia (Kjell, B s.a.). Jatkokehityksessä on myös tarkoitus tehdä yksikkötestit projektille, jolloin projektin päivittäminen esimerkiksi uuteen Ruby versioon on helpompaa. Yksikkötestien avulla pystyy myös versionhallinnassa tarkistamaan, että meneekö testit läpi ja tällä tavoin päivitykset eivät riko koodin kriittisiä kohtia.

4 Empiirinen osuus

Projektin toimeksiantaja kehittää ja ylläpitää avoimen lähdekoodin toiminnanohjausjärjestelmää Pupesoftia. Jokaiselle asiakkaalle on konfiguroitu Pupesoft heidän tarpeisiinsa sopivaksi ja asiakkaan Pupesoftia päivitetään pääasiassa silloin kun asiakas on tarvinnut lisäominaisuuksia tai järjestelmä vaatii ylläpidollisia päivityksiä. Tämän takia osan asiakkaista järjestelmät eivät pysy aina ajan tasalla, kun ei ole tietoa, milloin se on viimeksi esimerkiksi päivitetty. Asiakkaalla saattaa myös olla asetettuna versio järjestelmästä, joka eroaa pää versioista. Tämä status ei myöskään ole selkeästi nähtävissä mutta se on kriittinen tieto järjestelmän ylläpitoa varten. Projektin hyötyjä ovat mm. Pupesoft ohjelmiston ja sen riippuvuuksien, kuten ohjelmointikielien ajan tasalla pitäminen. Projekti myös helpottaa ohjelmiston Git haaran vaihtamista esimerkiksi testaus tarkoituksessa.

Projektin tuotos tulee toimeksiantajan käyttöön ja on täten projektin kohderyhmä. Toimeksiantaja on antanut melko vapaat kädet projektin tuottamiselle mutta esimerkiksi käyttöliittymän ulkonäköön he haluavat määritellä yhdessä ja projektin ohjelmointi framework:ksi sovittiin yhdessä Ruby on Rails. Projektin aikataulu on lyhyt, joka rajoittaa projektin laajuutta ja nk. "nice to have" pyyntöjä ei todennäköisesti ehdi toteuttamaan.

Lopputulos on hyvä silloin kun käyttöliittymä on toimeksiantajan mielestä selkeä ja se helpottaa palvelinten ja Pupesoftin ylläpidossa. Lopputuloksen olisi myös hyvä toimia ilman ongelmia ja olla melko nopea. Projektissa tavoiteltu lopputulos on todennettavissa esimerkiksi vaihtamalla, jonkin palvelimen Git haaraa tai tekemällä testi palvelimen, joka lähettää kerran dataa, jonka jälkeen se palvelin näkyy paneelissa.

4.1 Projektin aloitus

Jo ennen projektin alkua olimme käyneet toimeksiantajan kanssa projektin tekniset vaatimukset läpi, joten minulla oli hyvä ymmärrys alkavasta projektista. Projektin alussa aloitin pohtimalla tietokannan rakennetta ja mahdollisia tietokantatauluja, joita tässä toteutuksessa olisi. Koin tärkeäksi aloittaa tällä, jotta projektin aikana ei tarvitsisi tehdä useita tietokanta muutoksia. Tämän lisäksi toimeksiantajan puolesta minulle luotiin heidän Gitlab palveluun git-tietovarasto projektin versionhallintaa varten.

Seuraavaksi alustin projektin tekemällä Dockerfilen (Kuva 5) ja seuraamalla Dockerin omia ohjeita Ruby on Rails projektin aloitusta varten (Docker s.a.). Ohjeista muutin Ruby ohjelmointikielen versiosta 2.5 uusimpaan vakaaseen versioon eli versio 3.1.1. Ohjeiden kohtien tekemisen jälkeen minulla oli valmis Ruby on Rails projektin runko, joka on käytettävissä Docker ympäristössä, sekä valmiit käyttöliittymästä käytettävät luo, lue, päivitä ja poista metodit. Lopuksi tein vielä Railsin

komentorivi komennoilla Docker konttiin tietokanta taulun palvelimille, jonka jälkeen ohjelmistoa pystyi jo käyttämään Railsin vakio käyttöliittymästä.

```
FROM ruby:3.1.1

RUN apt-get update && apt-get install -y nodejs postgresql-client
WORKDIR /ahkio_dashboard
COPY Gemfile /ahkio_dashboard/Gemfile
COPY Gemfile.lock /ahkio_dashboard/Gemfile.lock
RUN bundle install

COPY entrypoint.sh /usr/bin/
RUN chmod +x /usr/bin/entrypoint.sh
ENTRYPOINT [ "entrypoint.sh" ]
EXPOSE 3000

CMD [ "bash", "rails", "server", "-b", "0.0.0.0" ]
```

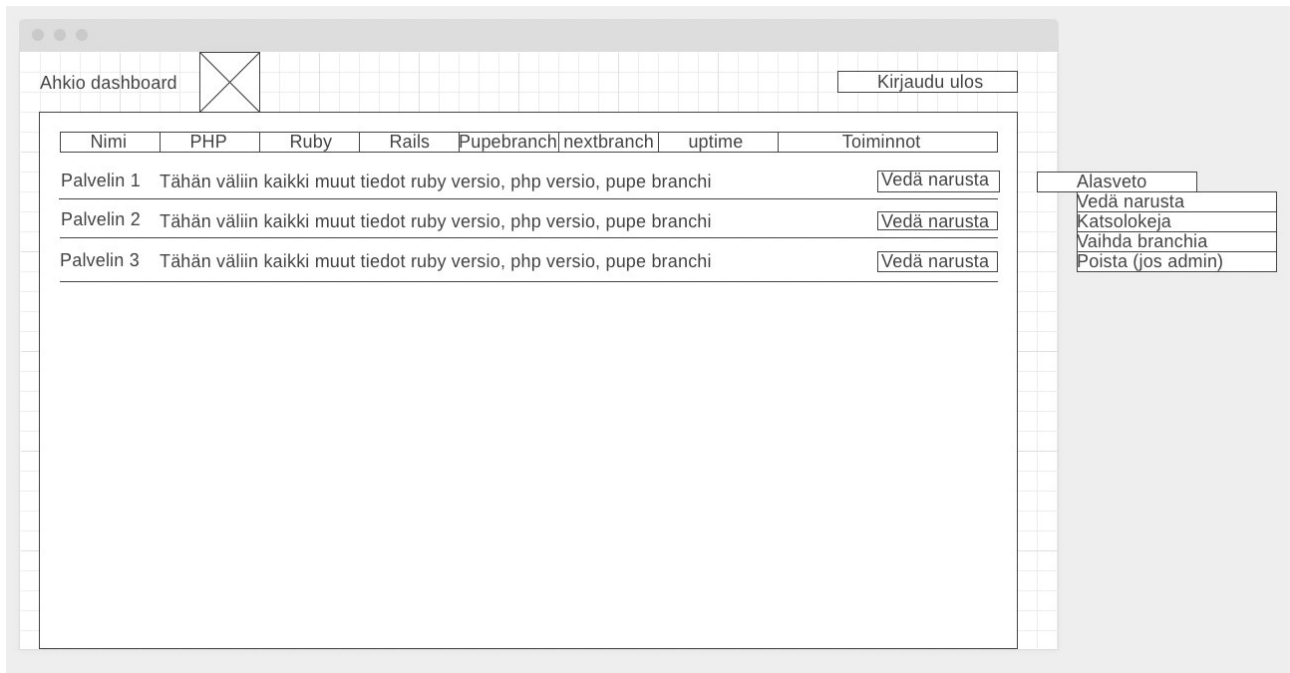
Kuva 5 Projektin Dockerfile

4.2 Kehitystyö

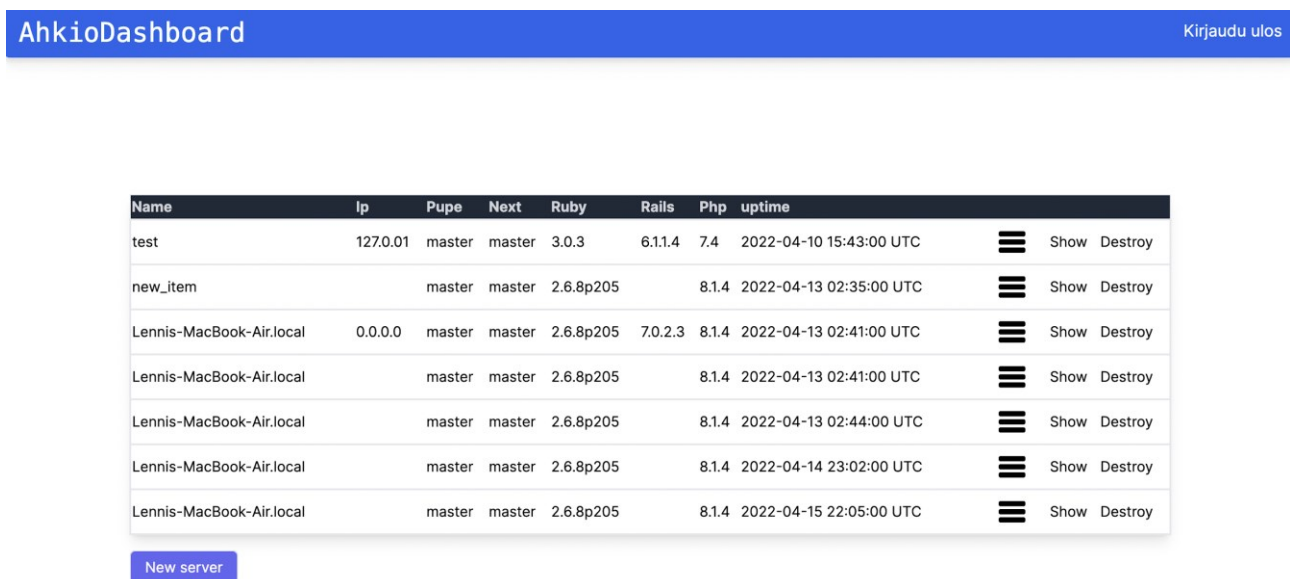
Projektin alustamisen jälkeen aloitin tekemään skriptiä ruby ohjelmointikielellä, joka asetetaan ylläpidettäville palvelimille. Tämän skriptin tarkoitus on kerätä vaadittava info palvelimelta ja lähettää se http-protokollalla hallintapaneeliin. Jotta http luonti (POST) kysely onnistuu Ruby on Rails ohjelmointirajapintaan täytyy ottaa pois käytöstä Cross Site Request Forgery (CSRF) nimisen hyökkäyksen esto. Tämä ei tietoturvan kannalta ole järkevää, joten tein ensin muutoksen, jossa jaoin ohjelmointirajapintaan menevät kyselyt omaan ohjaimeen (Controller) ja käyttöliittymästä tehtävät kyselyt omaan ohjaimeen. Seuraavaksi otin vain ohjelmointirajapinnan kyselyiltä CSRF eston pois, jotta voin jatkaa kehitystä. Ohjelmointirajapintaan on tässä vaiheessa vain suunniteltu myöhemmin lisättäväksi token pohjainen tunnistautuminen tietoturvan parantamiseksi.

Käyttöliittymän tekemisen aloitin tekemällä rautalankamallin käyttöliittymästä (Kuva 6.).

Toimeksiantaja koki mallin hyväksi ja yhdessä sovimme, että projektissa käytetään Tailwind CSS nimistä CSS-kirjastoa. Aloitin siis seuraavaksi tutkimaan, miten asennan tämän kirjaston Ruby on rails projektiin. Tästä löytyi viralliset ohjeet Tailwind CSS dokumentaatiosta (Tailwind CSS s.a.) mutta jouduin komentoja ajaessa tehdä pieniä muutoksia, jotta sain ne asennettua suoraan Docker ympäristöön. Kun olin saanut kirjaston asennettua aloin tekemään käyttöliittymän ensimmäistä versiota projektille (Kuva 7.). Seurasin jo mahdollisimman paljon tekemääni rautalankamallia mutta kokeilin vähän eri värejä ja pohdin, olisiko parempi tehdä alusvetovalikko toiminnoille tai laittaa toiminnot peräkkäin omaan sarakkeeseen.



Kuva 6 Rautalankamalli käyttöliittymästä



Kuva 7 Ensimmäinen versio käyttöliittymästä

Tuotoksen tavoitteena oli saada palvelimet lisäämään itsensä tietokantaan, jos palvelinta ei vielä löydy tietokannasta. Ensin lisäsin skriptiin tarkistuksen, jolla ensin haetaan kaikki palvelimet ja sen jälkeen tarkistetaan, jos palvelimen "hostname" -arvo löytyy palautetusta JSON-objektista. Tämän tiedon avulla skripti päätti, lähetetäänkö ohjelmointirajapintaan POST- vai PUT-kutsu. Kun kävin tätä ratkaisua toimeksiantajan kanssa läpi, sain palautteeksi muuttaa toiminto tapahtumaan rajapinnassa eikä skriptissä. Aloin siis tutkimaan onko vastaavaa tehty aiemmin ja minulle tuli

vastaan Rails:n oma metodi `upsert`, jolla päivitetään rivi tietokannassa, jos se löytyy sieltä etukäteen, muissa tapauksissa lisätään uusi rivi tietokantaan. Tämän lisääminen tuotti minulle ongelmia, sillä ensin minun piti tehdä jostain tiedosta uniikki, jonka avulla `upsert` metodi tarkistaa löytyykö kyseistä tietoa tietokannasta. Tähän toimeksiantajan kanssa yhdessä päädyttiin käyttämään `"hostname"` -arvoa, koska unix laitteissa ei ole suoraan uniikkia laite id:tä. Tämän jälkeen täytyi vielä kertoa tietokannalle, että uniikki arvo on indeksoitava. Railsissa tämä tehtiin luomalla uusi tietokanta migraatio. Tämän avulla sain loppujen lopuksi kyseisen toiminnon toimimaan halutulla tavalla.

Jotta sain muokattua jo luotua tietokanta taulua, jouduin tutkimaan miten teen muutokset Rails projektissa. Tähän tarkoitukseen Rails:lla on omat komentorivi metodit, joilla ensin tehdään migraatio tiedosto ja siihen tiedostoon kirjoitetaan dokumentaation mukaisesti millaiset muutokset halutaan tehdä. Sen jälkeen ajetaan toinen komento, joka hyödyntää ensimmäisen komennon tekemän tiedoston nimessä olevaa aika leimaa. Komentojen ajamisen jälkeen täytyy vielä itse tehdä muutokset näkymiin.

Aiemmin tehty skripti haluttiin tässä vaiheessa muuttaa linux daemoniksi. Daemon on Unix käyttöjärjestelmissä oleva apuohjelmisto, joka on koko ajan käynnissä taustalla (Kordek, A 2020). Daemonit ovat yleensä käyttöjärjestelmän toimintaan vaadittuja ohjelmia, mutta projektin tapauksessa haluttiin lisätä tehty skripti daemoniksi. Tämä haluttiin tehdä, koska Unixin `crontab` ajastin ei tue alle yhden minuutin välein ajettavia toimintoja ja skriptin kuuluu tapahtua muutaman sekunnin välein. Daemonin tekeminen Rubylla ei kuitenkaan ollut hankalaa sekä uusia riippuvuuksia ei tarvinnut asentaa vaan ne löytyivät suoraan Rubysta (Kuva 8.).

```
Process.daemon(true, true)
```

```
pid_file = File.dirname(__FILE__) + "#{__FILE__}.pid"
File.open(pid_file, 'w') { |f| f.write Process.pid}
```

Kuva 8 Daemonin luonti Rubylla

Esimerkin ensimmäinen rivi tekee jo skriptistä daemon prosessin. Metodille annetut arvot tarkoittavat, että pidetään sama hakemisto mistä skriptiä ajetaan ja hyväksytään mahdolliset syötöt, lähdöt ja virheet. Kaksi seuraavaa riviä tekevät pid tiedoston, jotta prosessia pystyy seuraamaan jostain tehtävien hallinta ohjelmasta.

Asiakkaan ohjelmiston päivittäminen ja git-oksan vaihtaminen vaatii ssh yhteyden kohde palvelimelle. Tämä onnistui rubylla `net-ssh` nimisellä riippuvuudella. Asiakkaiden palvelimet käyttävät ssh-avaimia, joten minun täytyi käydä lukemassa dokumentaatiosta, miten ssh-avain

autentikaatio tehdään mutta tämä löytyi onneksi helposti. Tein ssh yhdistämisestä oman metodin, jotta voin uudelleen käyttää sitä ilman, että kirjoitan samaa koodia useaan kertaan. Metodia hyödynsin ensin ohjelmiston päivitys funktion tekemiseksi. Tiesin etukäteen, että asiakkaan palvelimella on skripti, joka ajetaan ohjelmiston päivittämiseksi. Metodini siis täytyy pystyä ajamaan tämä skripti ja saamaan sen tuloste palautettua näkyviin käyttäjälle. Tämä onnistuikin melko helposti sillä riippuvuuden Github sivulta löytyi esimerkki siitä, miten tulostuksen saa palautettua. Vastaavaa metodia pystyin käyttämään myös git-oksen vaihtamiseen, mutta tässä minun piti vain käyttää gitin omaa komentoa oksan vaihtamiseen. Tulosteen sai tässä vaiheessa vain konsoliin palautettua ja toimeksiantaja halusi saada sen näkymään myös käyttöliittymässä. Tämä ominaisuus päätettiin kuitenkin siirtää jatkokehitykseen, koska se ei ollut toimeksiantajalle välttämätön vielä tässä vaiheessa.

Projektissa tuli myös toinen pieni muutos alkuperäiseen suunnitelmaan. Alun perin lokitiedostot haluttiin saada tutkittavaksi käyttöliittymään. Näiden tiedostojen hakeminen ja selaaminen päätettiin toimeksiantajan puolesta siirtää tässä vaiheessa myöhemmäksi tehtäväksi, joten sitä ei tehty vielä tässä vaiheessa.

Jotta tuotoksessa otettaisiin huomioon myös tietoturva, täytyi JSON-ohjelmointirajapintaan tehdä jokin todennus menetelmä. Tässä projektissa päädyttiin käyttämään token pohjaista todentamista. Aloitin tämän vaiheen tekemisen tutkimalla, miten Railsilla tehdään tällainen todennus. Autentikaation oli mahdollista käyttää Railsin omaa autentikaatiota tai jotain riippuvuutta. Aloitin vertailemalla olisiko Railsin oma autentikaatio riittävän hyvä, vai olisiko parempi valita jokin Railsin riippuvuus. Tässä päädyttiin käyttämään JSON Web Token -riippuvuutta, jossa käytetään tokenia autentikointiin. Projektille asennettiin myös bcrypt -riippuvuus, jolla salataan, tokenit siten, että ne eivät ole ihmiselle luettavissa. Luotu token on ennalta määrätyn ajan voimassa, jonka jälkeen käyttäjän täytyy hakea uusi token. Tätä varten täytyi vielä tehdä uusi tietokanta taulu käyttäjille, koska token on käyttäjä kohtainen. Railsin komentorivi komennoilla tämä onnistui helposti ja tämän jälkeen lisättiin myös käyttäjälle metodit tietueen luontiin, lukemiseen, muokkamiseen ja poistamiseen. Myös sisään kirjautumiselle ja autentikoinnille tehtiin omat metodit. Tämän jälkeen loin testi käyttäjän ja sain haettua, tokenin käyttäjälle. Tokenin kanssa pystyin hakemaan tietueen tietokannasta.

4.3 DevOps menetelmien hyödyntäminen kehitystyössä

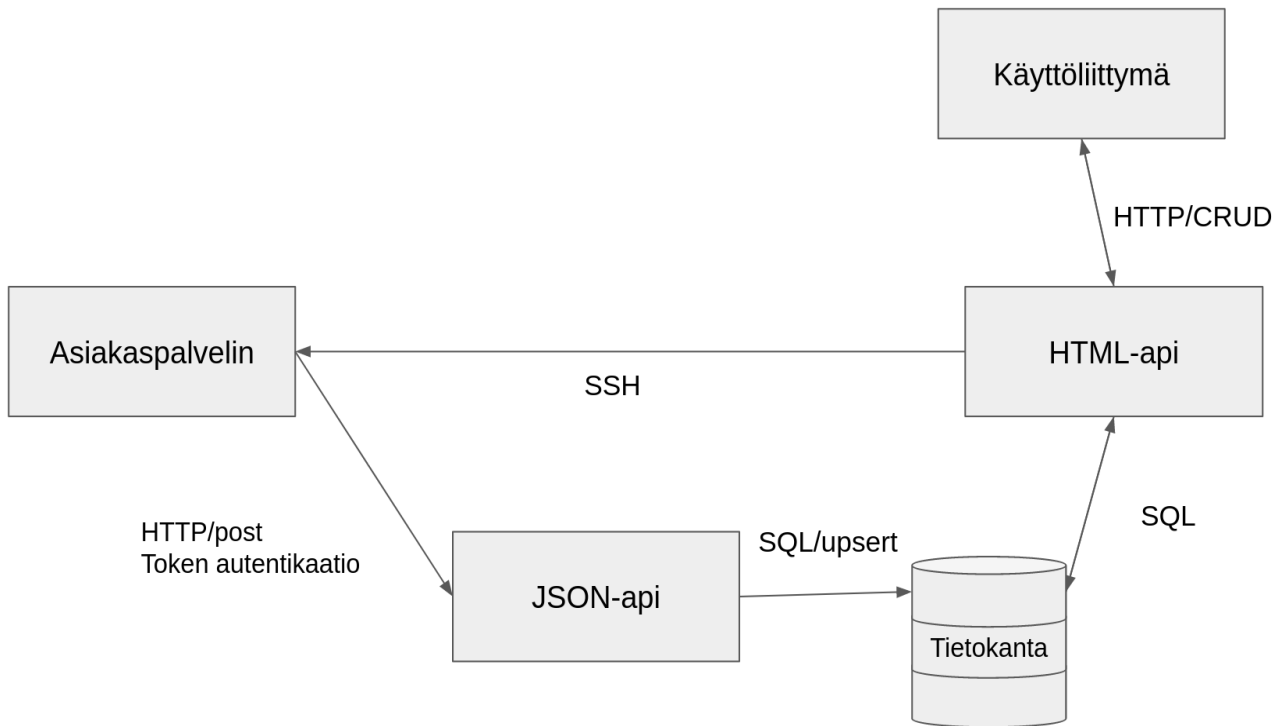
Projektissa hyödynsin DevOps menetelmiä kehittämiseen. DevOps:ssa käytettyjä työkaluja oli Git version hallinta sekä Docker. Pystyin myös kysymään milloin tahansa toimeksiantajalta, jos heillä on joku toive, miten jokin asia ratkaistaan.

Versionhallinnassa tein, jokaisesta toiminnosta oman git-oksan, jotta historia versionhallinnassa olisi mahdollisimman selkeä. Jokainen oksa myös tarkastettiin toimeksiantajan puolesta ja sain palautetta, jos jotain halutaan tehdä eri tavalla. Uuden toiminnallisuuden valmistuessa vein sen aina toimeksiantajan Gitlabiin. Käytin tätä tapaa, jotta sain hyödynnettyä DevOps:lle ominaista jatkuvaa integraatiota ja palautteen saamista.

Dockeria käytin docker-compose ohjelman kanssa, jonka avulla pystyn asentamaan Docker kuvakeelle rubyn riippuvuuksia, sekä tekemään muita muutoksia. Docker-composella siis hallitsin Docker kuvaketta. Docker oli käytössä, jotta kehittäisin tuotosta täysin samassa ympäristössä missä se julkaistaan. Tällä tavalla pystyin varmistamaan ohjelmiston toimivuuden jo kehitys vaiheessa.

4.4 Kehitettävän projektin arkkitehtuuri

Projektin arkkitehtuuria aloitettiin miettimään konkreettisesti vasta projektin loppupuolella. Tämä ei kuitenkaan ollut projektin kehittämisen kannalta ongelma, sillä Railsin ja Dockerin käyttö ohjasi arkkitehtuuria tiettyyn suuntaan. Arkkitehtuurista tein arkkitehtuurikaavion, joka käsittelee vain Dockerin sisälle tehtyä osuutta, sekä keskustelua asiakaspalvelinten kanssa (Kuva 9.). Projektin valmis tuotos viedään Ansiblen avulla palvelimelle, joka on toimeksiantajan VPN-verkon sisällä. Ansible käynnistää myös ohjelmiston Docker-kontin sisälle. Palvelimella on käynnissä Nginx verkkopalvelin, joka on ohjattu käyttöliittymään toimeksiantajan aladomainin kautta. Käyttöliittymä keskustelelee HTML-apin kanssa HTTP:n CRUD-metodien avulla. HTML-apista taas lähtee SQL kyselyitä tietokantaan ja tietokanta palauttaa tiedot HTML-apin kautta takaisin käyttöliittymään. HTML-api keskustelelee myös SSH-yhteyden avulla Asiakaspalvelinten kanssa, joka taas lähettää skriptillä tiedot JSON-api:lle. JSON-api tekee SQL upsert, eli päivitä tai lisää, kyselyn tietokantaan, jolla päivitetään muutokset tai tehdään uusi tietue.



Kuva 9: Projektin arkkitehtuurikaavio

5 Pohdinta

Projekti oli itselle erittäin mielenkiintoinen ja sitä oli mukava tehdä, vaikka projektin aikataulu oli todella kiireellinen. Tämä projekti on myös osoittanut itselleni, että pystyn työskentelemään haastavissakin ympäristöissä melko hyvin. Opin projektista paljon uutta ja sain kerrata myös jo osaamiani asioita. Mielestäni projekti onnistui aikatauluun verratessa hyvin, mutta paremmalla aikataululla olisin saanut paremman lopputuloksen.

5.1 Onnistumiset ja haasteet

Projektissa onnistuin mielestäni Docker ympäristö luonnissa projektille. Onnistuin tässä mielestäni sen takia, koska se onnistui erittäin helposti ilman aikaisempaa kokemusta vaativammasta ympäristöstä. Ympäristö myös toimi toivotulla tavalla jo ensimmäisellä iteraatiolla. Onnistuin myös mielestäni käyttöliittymän luonnissa ja myös toimeksiantaja oli tyytyväinen käyttöliittymään. Koen, että rautalankamallilla suunnittelu etukäteen auttoi paljon käyttöliittymän onnistumisessa.

Projektin isoin haaste oli ehdottomasti aikataulu. Projektin sijoittaminen samalle ajalle opinnäytetyön kanssa aiheutti haasteita ja jälkikäteen ajateltuna olisi ollut kannattavampaa varata enemmän aikaa tähän. Haasteita myös tuotti metodi, jolla päivitetään tai lisätään tietue tietokantaan. Osittain koin tämän johtuvan huonosta dokumentoinnista tämän metodin osalta mutta koin myös tiukan aikataulun vaikeuttavan tarkkaa lukemista joidenkin ohjeiden osalta. Tulosteiden näyttäminen käyttäjälle oli myös vaikeaa ja tämän takia siirrettiin jatkokehitykseen. Tämänkin ongelman olisin saanut selvitettyä, jos en olisi ollut niin intohimoinen aikataulun kanssa.

5.2 Projektista opittua

Vaikka projektia tehtiin todella kiireisellä aikataululla, ehdin silti oppia uusia asioita sekä ohjelmoinnista että työelämän ohjelmointi käytännöistä. Projektissa opin ohjelmoimaan sujuvasti Rubylla ja Railsilla sekä hyödyntämään niiden dokumentaatiota. Docker tuli myös todella tutuksi projektin aikana ja opin kuinka Dockeria käytetään. Projektissa luin todella usein dokumentaatioita ja huomasinkin projektin aika, että olen oppinut hyödyntämään teknologioiden dokumentointeja huomattavasti paremmin verrattuna projektin alkuun.

5.3 Toimeksiantajan hyödyt

Projekti alkoi samoihin aikoihin, kun opinnäytetyön kirjoittaminen alkoi ja projektin loppuminen oli samalla viikolla kuin opinnäytetyön palautus. Tämän takia toimeksiantajalta ei ole kuin alustavia ajatuksia tulevista hyödyistä. Esimerkiksi toiminnanohjausjärjestelmän viimeinen päivitys päivä ja palvelimen päällä oloaika olivat toimeksiantajan mielestä erittäin hyödyllisiä tietoja, koska ne helpottavat ongelmia ratkottaessa ja muistuttaa toiminnanohjausjärjestelmän päivittämisestä. Myös

päivitys nappi koettiin hyödylliseksi, koska päivittämistä varten ei tarvitse erikseen ajaa komentoja komentoriviltä.

Lähteet

Atlassian s.a. DevOps: Breaking the development-operations barrier. Luettavissa:

<https://www.atlassian.com/devops>. Luettu: 24.4.2022.

Basset, L. 2015. Introduction to JavaScript Object Notation. O'Reilly Media, Inc. Sebastopol. E-

kirja. Luettu: 13.5.2022.

David 2005. Rails 1.0: Party like it's one oh oh! Luettavissa:

<https://rubyonrails.org/2005/12/13/rails-1-0-party-like-its-one-oh-oh>. Luettu: 13.4.2022.

Docker s.a. Quickstart: Compose and Rails Luettavissa: <https://docs.docker.com/samples/rails/>.

Luettu: 5.4.2022.

Geeks for Geeks 2021. Benefit of using MVC. Luettavissa: <https://www.geeksforgeeks.org/benefit-of-using-mvc/>.

Luettu: 22.4.2022.

Kjell, B s.a. Why Binary Files are Needed. Luettavissa:

https://chortle.ccsu.edu/java5/Notes/chap86/ch86_6.html. Luettu: 11.5.2022.

KodeKloud 2020. The Role of Docker in DevOps. Luettavissa: <https://dev.to/kodekloud/the-role-of-docker-in-devops-1con>.

Luettu: 24.4.2022.

Kordek, A 2020. Understanding Linux Daemons. Luettavissa:

<https://www.inmotionhosting.com/support/server/linux/understanding-linux-daemons/>. Luettu: 23.4.2022.

Luomala, S. 2010. Toiminnanohjausjärjestelmä – sähköistyvät palvelut yrityksen

tietojärjestelmässä. Ylempi AMK-opinnäytetyä. Turun ammattikorkeakoulu, Bioalat ja liiketalous |

Yrittäjyyden ja liiketoimintaosaamisen koulutusohjelma. Luettavissa:

https://www.theseus.fi/bitstream/handle/10024/23098/Luomala_Sari.pdf?sequence=1. Luettu: 11.5.2022.

Pitt, C. 2021. Pro PHP 8 MVC. 2. painos. Apress. New York. E-kirja. Luettu: 14.5.2022.

Ruby on Rails s.a. Getting Started with Rails. Luettavissa:

https://guides.rubyonrails.org/getting_started.html#mvc-and-you. Luettu: 13.4.2022.

Reenskaug, T 2007. The original MVC reports Luettavissa:

https://folk.universitetetioslo.no/trygver/2007/MVC_Originals.pdf. Luettu: 22.4.2022.

Spilca, L. 2020. Spring Security in Action. Manning Publications. Shelter Island. E-kirja. Luettu: 13.5.2022.

SSH Academy s.a. SSH (Secure Shell) Home Page. Luettavissa:
<https://www.ssh.com/academy/ssh>. Luettu: 14.5.2022.

Tailwind CSS s.a. Install Tailwind CSS with Ruby on Rails Luettavissa:
<https://tailwindcss.com/docs/guides/ruby-on-rails>. Luettu 12.4.2022.

Tutorialspoint s.a. HTTP – Methods Luettavissa:
https://www.tutorialspoint.com/http/http_methods.htm. Luettu: 5.5.2022.