Tampereen ammattikorkeakoulu

Lammaspaikka-sovelluksen käyttöönottokokeilu

Ville Heinämäki

OPINNÄYTETYÖ
Huhtikuu 2022

Tietojenkäsittelyn tutkinto-ohjelma
Ohjelmistotuotanto

TIIVISTELMÄ

Tampereen ammattikorkeakoulu
Tietojenkäsittelyn tutkinto-ohjelma
Ohjelmistotuotanto

HEINÄMÄKI, VILLE:
Lammaspaikka-sovelluksen käyttöönottokokeilu

Opinnäytetyö 30 sivua
Huhtikuu 2022

Tämä opinnäytetyö käsittelee lammastalouden tuotannonohjaukseen tarkoitetun Lammaspaikka-sovelluksen käyttöönottokokeilua. Työn tavoitteena on valmistella sovellus käyttöönottokokeilua varten ja tarvittaessa toimia sen uuden kehitystiimin tukena kokeilujakson ajan. Työn tarkoituksena on refaktoroida sovelluksen koodi, siirtää sovellus sovelluskonttiin käyttöönoton helpottamiseksi sekä parannella sovellusta käyttäjäpalautteen perusteella. Työn toimeksiantajana toimii AhlmanEdu, ja se on tehty osana Pientuottajat Yhteistyöllä Markkinoille -hanketta.

Työn tuloksena sovelluksesta korjattiin useita virheitä, tietoturvaa parannettiin määrittämällä käyttäjätilien oikeuksille tarkat rajat sekä optimoitiin sovelluksen suorituskykyä. Konkreettisena tuotoksena rakennettiin ajoympäristöstä riippumaton sovelluskontti, joka on testikäytön ajan ajossa Heroku-palvelussa. Käyttöönottokokeilun aikana lammastukkujen ja -teurastamoiden edustajat pääsivät kokeilemaan sovelluksen käyttöä ja antamaan siitä palautetta.

Sovellus tulee tulevaisuudessa käyttöönottokokeilun ja sitä seuraavan jatkokehityksen jälkeen siirtymään joko tuottajaorganisaation tai kolmannen osapuolen haltuun. Kontitus on tehty osittain siitä syystä, että käyttöönotto tulevassa lopullisessa ajoympäristössä olisi sujuvampaa. Jatkokehityksen kannalta olennaisimpia ominaisuuksia ovat tiedonkulkua tehostavat integraatiot muihin lammastalouden sovelluksiin kuten Sorkkaan ja Nettikatraaseen.

Asiasanat: Käyttöönotto, Lammastuotanto, Node.js, Docker

ABSTRACT

Tampereen ammattikorkeakoulu
Tampere University of Applied Sciences
Degree Programme in Business Information Systems

HEINÄMÄKI, VILLE:
Test Deployment of the Lammaspaikka Application

Bachelor's thesis 30 pages
April 2022

This thesis discusses the test deployment of the Lammaspaikka operations and management planning application used in sheep breeding. The objective of the thesis is to prepare the application for deployment and to support its new development team during the testing period when needed. The purpose of the thesis is to refactor the codebase of the application, containerize the application to ease the technical side of the deployment process, as well as improve the application according to user feedback.

As a conclusion, many errors in the application were fixed, security vulnerabilities were solved by limiting user account permissions and performance of the application was improved. A Docker container for the application was created and deployed to Heroku. The testing period was started successfully, and the target audience of the application was invited to give feedback on it.

In the future when the testing period and the following further development has been finished, the application will be possibly handed over to a third party. Developing integrations to other sheep farming applications such as Sorkka and Nettikatras is a high priority for further development.

Key words: deployment, Node.js, Docker

SISÄLLYS

ERITYISSANASTO

Node.js	Yleisesti palvelinsovelluksissa käytetty JavaScript-ajo-ympäristö.
npm	JavaScript-pakettirepositorio ja paketinhallintaohjelma.
API	Application Programming Interface, sovelluksen eri osien välinen kommunikaatiorajapinta.
Sovelluskontti	Ajoympäristöstä riippumaton sovelluskokonaisuus.

1 JOHDANTO

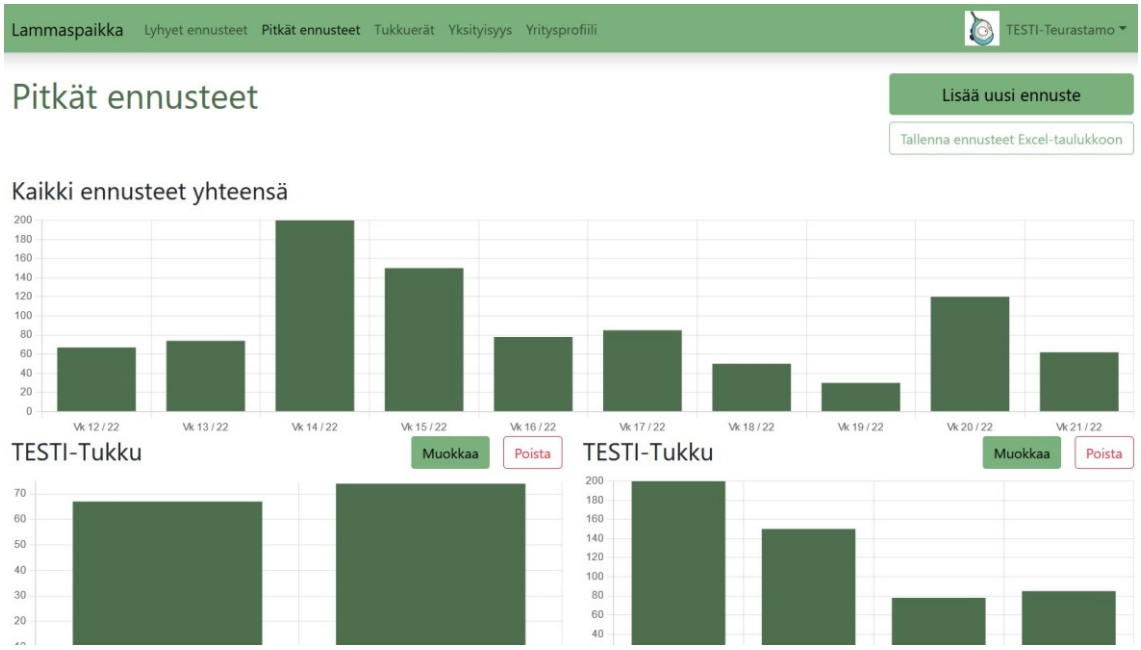
Lammaspaikka-sovelluksen toteutus on aloitettu osana Pientuottajat Yhteistyöllä Markkinoille -hanketta. Kyseessä on lammasketjun tuotannonohjauksen sovellus, jonka avulla on tarkoitus parantaa tiedonsiirron sujuvuutta eri osapuolten välillä, ja siten lisätä liiketoiminnan ennustettavuutta. Sovellus on suunnattu lammasteurastamoiden ja -tukkujen käyttöön.

Lammastalouden ketjussa on useita eri osapuolia. Alkutuotanto tapahtuu lammastilalla, josta eläimet myydään teurastamolle. Teurastamot puolestaan myyvät ruhot edelleen tukuille, jotka jalostavat ne lihatuotteiksi. Tukut myyvät tuotteet jalostamoille tai suoraan vähittäiskauppaan. Lampaanlihan kysyntä ja tarjonta vaihtelee kausittain, joten eri osapuolten on tärkeää kommunikoida tulevien lampaiden ja karitsoiden volyymi toisilleen liiketoiminnan ennakoitavuuden parantamiseksi.

Tehokas, keskitetty tiedonhallinta nouseekin tärkeäksi osaksi liiketoimintaa. Lammasketjun alkupäässä lammastilojen tiedonhallintaa sujuvoittamaan on suunniteltu Nettikatras-niminen sovellus, jolla lampaiden ja karitsojen eläintietoja saadaan seurattua ja kirjattua ylös. Teurastamoille on puolestaan olemassa Sorkka-toiminnanohjausjärjestelmä. Sorkka automatisoi teurastamon tiedonhallintatoimia, kuten eläintietojen vastaanottoa ja niiden raportoimista Ruokavirastolle sekä tarjoaa integraatioita teurastamon digitaalisiin laitteisiin (Sorkka.fi, n.d). Näiden kahden sovelluksen välille on suunniteltu tiedonsiirron integraatio, joka helpottaa lammastilojen ja teurastamoiden yhteistoimintaa.

Lammasketjun keskiosalla ei kuitenkaan vielä ole vastaavanlaista järjestelmää. Tukkujen ja teurastamoiden välinen tiedonsiirto onkin pitkälti yritysten omien, sisäisten käytäntöjen varassa. Yhteisten tietovarantojen puuttuminen johtaa huonompaan lammasvolyymien ennustettavuuteen etenkin tukkujen osalta. Käytännössä toimintaan liittyy paljon arvailua, ja ennusteet tulevista lampaista saadaan niin lyhyellä aikavälillä, ettei niihin ehditä reagoida. Lammaspaikan kehittäminen on lähtenyt liikkeelle tästä tarpeesta sujuvoittaa tukkujen ja teurastamoiden yhteistoimintaa.

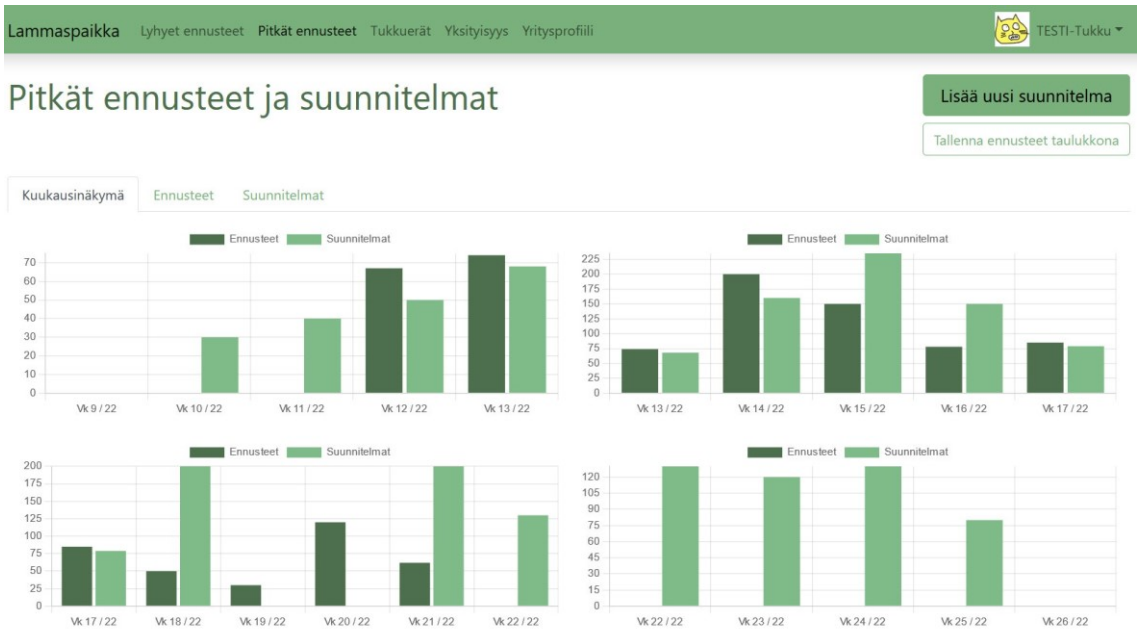
Lammaspaikka on toteutettu verkkopohjaisena sovelluksena, jossa sekä palvelinpuolen sovellus että verkkokäyttöliittymä on kirjoitettu JavaScript-ohjelmointikielellä. Käyttäjärooleja on kolmenlaisia: teurastamoiden edustajat, tukkujen edustajat sekä ylläpitäjät. Yritykset muodostavat sovelluksessa sopimuskumppanuuksia, joiden perusteella yritykset voivat lähettää tietoja toisilleen. Käyttäjät voivat lähettää yrityksille kumppanuuspyyntöjä, hyväksyä, hylätä ja perua niitä. Teurastamot voivat luoda sopimuskumppaneilleen eräennusteita tulevista volyymeistä. Ennusteet on jaettu pitkän ja lyhyen aikatähtäimen kategorioihin. Pitkät ennusteet ovat viikkokohtaisia, karkeita arvioita lammasmääristä, ja lyhyet ennusteet tarkennettuja päiväkohtaisia ilmoituksia. Teurastamo voi tarvittaessa muokata ennusteita jälkikäteen, jos arvio muuttuu. Ennusteet kohdistetaan jokaiselle sopimuskumppanille erikseen, ja vastaanottavat kumppanit näkevät vain heille kohdistetut arviot. Ennusteet visualisoidaan pylvästaulukoina (kuva 1).



KUVA 1. Pitkien ennusteiden näkymä teurastamon näkökulmasta.

Tukut voivat tarkastella niille lähetettyjä ennusteita ja myös luoda suunnitelmia tavoitelluista lammasmääristä. Tukkujen suunnitelmat toimivat samalla tavoin kuin teurastamoiden ennusteet, mutta niitä ei erikseen kohdisteta sopimuskumppaneille, vaan ne ovat täysin yksityisiä. Eri teurastamoiden lähettämät yhteenlas-

ketut ennusteet näytetään tukuille suunnitelmien rinnalla, jolloin on helppo verrata, kuinka kaukana tavoitteesta ollaan (kuva 2). Tulevaisuudessa saatetaan myös toteuttaa ominaisuus, jolla tukut voivat jakaa oman suunnitelmansa teurastamoiden kanssa. Tällöin teurastamot voisivat paremmin kohdistaa lammasvolyymit niitä tarvitseville tukuille. Ylläpitokäyttäjien roolina on valvoa käyttäjien rekisteröintiprosessia. Sovellukseen rekisteröityneet uudet yritykset tarvitsevat ylläpitäjän vahvistuksen ennen kuin he voivat aloittaa sovelluksen käytön.



KUVA 2. Pitkien ennusteiden ja suunnitelmien vertailunäkymä tukun näkökulmasta.

Pidemmän aikavälin tavoitteena on toteuttaa integraatiot Nettikatrasta ja Sorkka-järjestelmää varten. Sorkka-integraation avulla teurastamot voisivat tuoda tietonsa teurastetuista lampaista Sorkan tietokannasta Lammaspaikkaan. Toiminnallisuudesta on olemassa keskeneräinen prototyyppi sovelluksessa, joka toteutettiin viime vuoden kevään ja syksyn aikana. Nettikatras-integraatiolla teurastamot voisivat puolestaan muodostaa automatisoituja ennusteita lammastilojen eläinmääristä. Integraatiot mahdollistaisivat katkeamattoman eläintietojen kulun tuotantoketjun läpi lammastiloilta tukuille.

Lammaspaikan kehityksen aloitti vuoden 2021 keväällä TAMKin opiskelijaryhmä osana sen hetkistä projektikurssia. Sovelluksen kehitystä jatkoi saman vuoden syksynä toinen ryhmä, jossa opinnäytetyön tekijä on työskennellyt. Sovellus on

kuitenkin vielä kesken, ja sitä jatkaa tämän vuoden keväänä kolmas opiskelija-ryhmä. Sovellus otetaan maaliskuun alussa testikäyttöön, jolloin teurastamoiden ja tukkujen edustajat saavat kokeilla sitä ja antaa siitä palautetta. Sovellus on tarkoitus tulevaisuudessa siirtää erillisen yrityksen haltuun, joka vastaa lopullisen palvelinympäristön toteuttamisesta ja sovelluksen ylläpidosta.

2 REFAKTOROINTI

2.1 Riippuvuuksien hallinta

Ohjelmistoprojektit harvoin koostuvat pelkästään sovelluksen kehittäjien itse kirjoittamasta koodista, ja etenkin JavaScript-projekteissa kolmannen osapuolen riippuvuuksia käytetään yleisesti paljon, sillä kielen oma standardikirjasto on melko suppea. Lammaspaikan palvelinsovellus on toteutettu Node.js-ajoympäristöä käyttäen. Sovelluksen verkkokäyttöliittymä perustuu React-kirjastoon, jonka kanssa on käytetty enimmäkseen Bootstrap-kirjaston käyttöliittymäkomponentteja. Riippuvuuksien hallintaan käytetään npm-pakettirepositoriota ja paketinhallintaohjelmaa.

2.1.1 Käyttämättömien pakettien poistaminen

Projektissa oli käytössä muutamia riippuvuuksia, joita oli varhaisemmassa kehitysvaiheessa tarvittu, mutta myöhemmin korvattu joko muilla moduuleilla tai omalla koodilla. Käytössä oli myös sellaisia taulukoiden ja ponnahdusikkunoiden näyttämiseen käytettyjä komponentteja, jotka voitiin korvata käyttöliittymässä jo valmiiksi käytetyn Bootstrapin vastaavilla komponenteilla käyttökokemuksen yhtenäistämiseksi. Käyttämättömät riippuvuudet poistamalla projektin tiedostokokoa saatiin vähennettyä. Etenkin frontend-koodissa riippuvuuksien määrän pitäminen mahdollisimman vähäisenä on tärkeää, sillä JavaScript-bundlen koko vaikuttaa suoraan sivun latausaikaan. Refaktoroinnin yhteydessä kehitys- ja palvelinympäristössä asennettavien riippuvuuksien määrää saatiin vähennettyä n. 100 Mt edestä. Palvelinympäristössä vähennys parantaa riippuvuuksien asentamiseen tarvittavaa aikaa ja lopullista tiedostokokoa. Asennusajan lyhentäminen on kätevää etenkin sovelluksen kontittamista ajatellen, jolloin kaikki riippuvuudet saatetaan joutua asentamaan uudelleen aina kun konttiin tehdään muutoksia.

2.1.2 Pakettien päivittäminen

Refaktoroinnin yhteydessä sovelluksen riippuvuudet päivitettiin uusiin versioihin. Riippuvuudet kannattaa pitää ajan tasalla, sillä uusia haavoittuvuuksia ilmenee

jatkuvasti. Paketteja asentaessa npm ilmoittaa mahdollisista pakettien kehittäjien havaitsemista tietoturvaongelmista. Tietoturvatarkastuksen voi myös suorittaa erikseen ”npm audit” -komennolla. Yleisin ratkaisu ongelmiin on paketin päivittäminen uudempaan versioon (Npmjs.com, 2020). Potentiaalisena ongelmana on se, että uudet versiot eivät ole välttämättä yhteensopivia vanhojen versioiden kanssa, jolloin uuden version käyttöönotto vaatii muutoksia koodissa. Suurin osa haavoittuvuuksista saatiin korjattua päivittämällä kyseinen paketti uudempaan taaksepäin yhteensopivaan versioon. Muutama paketeista tuli kuitenkin päivittää uuteen taaksepäin yhteensopimattomaan pääversioon.

Ongelmia riippuvuuksien päivittämisessä aiheutti lähinnä react-scripts-paketti, joka huolehtii käyttöliittymän kokoamisprosessista. Taustalla react-scripts käyttää Webpack-nimistä pakettia, joka huolehtii tiedostojen pakkaamisesta ja optimoinnista. React-scripts päivitettiin versioon 5.0.0, joka on puolestaan riippuvainen Webpackin versiosta 5.0.0. Yhtenä Webpackin taaksepäin yhteensopimattomista muutoksista oli, ettei se enää automaattisesti tarjoa selaimessa toteuttamattomia JavaScript-rajapintoja täydentäviä polyfill-moduuleita yleisiä Node.js:n moduuleita varten. Polyfill-moduulit mahdollistivat aiemmin sen, että joitain Node.js:ää varten kirjoitettuja paketteja voitiin käyttää myös selaimessa ajettavassa koodissa täydentämällä toteuttamattomat rajapinnat vaihtoehtoisilla toteutuksilla. Webpack 5 ei tee tätä enää automaattisesti, sillä polyfill-moduulit ovat raskaita ja lisäävät sovelluksen JavaScript-bundlen kokoa, eivätkä usein ole edes tarpeellisia (Webpackjs.org, 2020). Vaihtoehtoina ongelman ratkaisemiseksi oli joko konfiguroida Webpack manuaalisesti sisällyttämään tarvitut Node.js -moduulit, tai refaktoroida frontend-koodi niin, ettei Node.js -moduuleita tarvita. Jälkimmäinen vaihtoehto osoittautui kannattavammaksi, sillä polyfill-riippuvaisia moduuleita käytettiin vain kahdessa funktiossa, joiden uudelleentoteuttaminen ei ollut hankalaa. Tämän myös laski sovelluksen bundlen kokoa, sillä kolmannen osapuolen koodin määrä väheni.

2.1.3 Riippuvuuksien hallinnan tietoturva

Kolmannen osapuolen moduulien tuoman kätevyyden kääntöpuolena on vastuun ottaminen vieraiden ihmisten kirjoittamasta koodista. Paketit tyypillisesti julkaistaan avoimen lähdekoodin lisenssien alaisena, mikä mahdollistaa koodin vapaan käytön, mutta toisaalta poistaa julkaisijan vastuun mahdollisista vioista. Asennetuilla paketeilla on yleisesti myös omia riippuvuuksiaan, ja näillä riippuvuuksilla potentiaalisesti omiaan, joista jokainen lisää uuden tahon, johon täytyy luottaa. Pakettien päivittämisen ohella kaikista tehokkain tapa välttää haavoittuvuuksia onkin pitää mahdollisimman vähän riippuvuuksia, ja suosia paketteja, joilla puolestaan on vähän tai ei yhtään riippuvuuksia.

Tahattomien haavoittuvuuksien lisäksi tietoturvariskinä on myös haittaohjelmat. Esimerkiksi viime vuonna 22. lokakuuta suositussa ua-parser-js-paketissa havaittiin haittaohjelma, joka asentui käyttäjän laitteelle pakettia asentaessa. Myöhemmin 4. marraskuuta samankaltainen haavoittuvuus löytyi myös rc- ja coa -paketeista. Haittaohjelmat pystyivät asentumaan sekä kehittäjien laitteille että palvelinympäristöihin. Jokaisessa tapauksessa pakettien kehittäjien npm-tileille oli murtauduttu, jonka jälkeen paketeista julkaistiin haittaohjelman sisältävä uusi versio. Tapahtumien jälkeen npm on pyrkinyt parantamaan tietoturvaansa kehittämällä automaattista haittaohjelmien tunnistusta sekä vaatimalla käyttäjiltä kaksivaiheista tunnistautumista (Hanley, 2021).

2.2 Virhekorjaukset ja parantelut

Tietoturva on Lammaspaikassa keskeisessä roolissa, sillä käyttäjien toisilleen lähettämien ennusteiden ja suunnitelmien tulee joko olla yksityisiä tai näkyä vain valituille sopimuskumppaneille. Aiemmassa kehitysvaiheessa sovelluksen palvelintoteutus ei kaikissa tapauksissa tarkistanut käyttäjien oikeuksia eri tietoja kuten ennusteita tai tukkueriä pyytäessä, vaan tarkistukset tehtiin käyttäjän selaimessa. Selaimessa ajettavaan koodiin ei kuitenkaan voi palvelimen näkökulmasta luottaa, sillä sen toiminta on täysin käyttäjän muokattavissa. Lisäksi palvelimelle tehtävien API-kutsujen lähettämiseen voi käyttää selaimen lisäksi mitä tahansa muutakin HTTP-asiakasohjelmaa, joten frontend-koodia ei tarvitse ajaa lainkaan. Tietoturvaa parannettiin varmistamalla palvelinpuolella, että eri API-kutsuja tehdessä käyttäjällä on oikeudet kyseisen resurssin lukemiseen tai muokkaamiseen.

Esimerkiksi ennustetietoja pyytäessä vain ennusteen tuottajalla ja vastaanottajalla on oikeus lukea tietoja, ja muokkausoikeus on vain ennusteen tuottajalla.

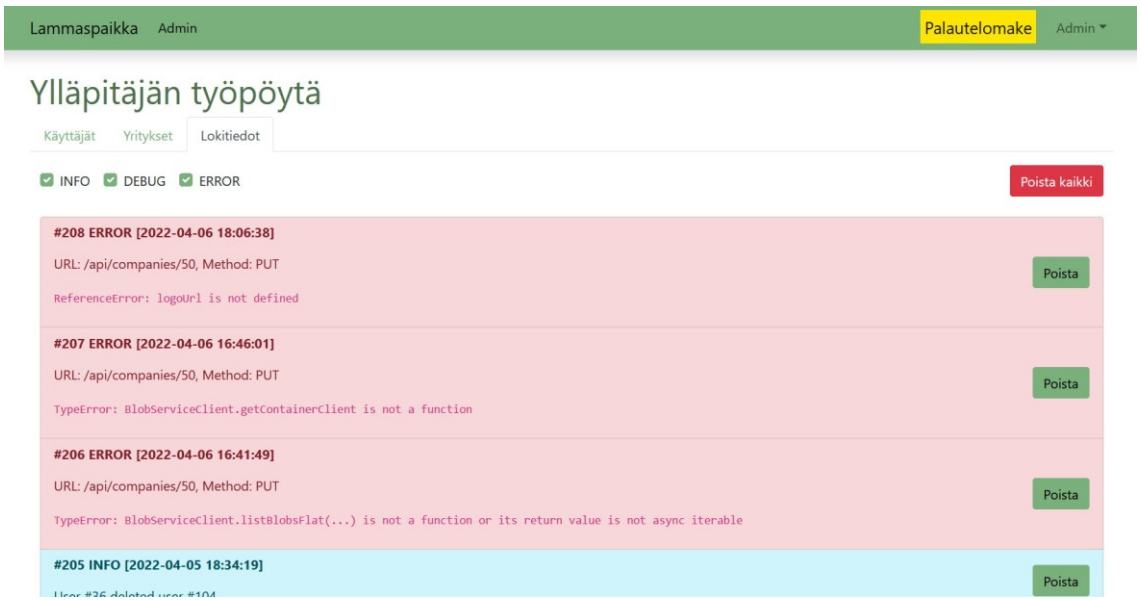
Haavoittuvuuspinta-alan minimoimiseksi eri käyttäjäroolien oikeudet pyrittiin rajaamaan vain niihin, mitä kyseiset käyttäjäryhmät sovellusta käyttäessä tarvitsevat. Ylläpitokäyttäjien oikeudet ovat kaikista laajimmat, sillä niillä pystyy luomaan ja poistamaan mitä tahansa käyttäjiä, mukaan lukien ylläpitokäyttäjiä. Yritysten pääkäyttäjät ovat oman yrityksensä ylläpitäjiä, joten he pystyvät luomaan, muokkaamaan ja poistamaan oman yrityksensä ennusteita sekä luomaan tavallisia käyttäjiä yritykseensä. Tavallisilla käyttäjillä on samat oikeudet kuin yritysten pääkäyttäjillä, uusien käyttäjien luomista lukuun ottamatta. Varmistamattomien käyttäjien oikeudet jätettiin mahdollisimman vähäisiksi, sillä käytännössä kuka tahansa pystyy luomaan sovellukseen uuden varmistamattoman yrityksen. Muutosten jälkeen varmistamattomat käyttäjät pystyvät vain muokkaamaan omia henkilötietojaan. Tämä on hyödyllistä, jos rekisteröitynyt käyttäjä teki vahingossa kirjoitusvirheen esimerkiksi sähköpostiosoitettaan kirjoittaessa.

Sovelluksen MySQL-tietokannasta korjattiin tekstin säilömiseen liittyvä ongelma vaihtamalla tietokantayhteyden oletusmerkistöksi utf8mb4. MySQL sisältää kaksi eri UTF-8-merkistöä, utf8 ja utf8mb4. Näiden merkistöjen erona on se, että utf8 ei pysty säilömään kuin korkeintaan 3 tavun suuruisia merkkejä, kun taas utf8mb4 tukee kaikkia Unicode-merkkejä. utf8-merkistön rajoitus tehtiin MySQL:ssä alun perin suorituskyvyn parantamiseksi, kun yli 3 tavun merkit olivat harvinaisia. Nykyään ne ovat kuitenkin yleisempiä, esimerkiksi emojit tyypillisesti mahtuvat neljään tavuun. Jos utf8-merkistöä käyttäessä tietokantaan yritetään tallentaa yli 3 tavun suuruisia merkkejä, MySQL ei ilmoita ongelmasta vaan katkaisee merkkijonon virheellisen merkin kohdatessaan, säilöen ainoastaan merkkijonon virheettömän alkuosan. Tämä voi johtaa tietoturvaongelmiin, jos merkkijonon sisällöllä on erityinen merkitys ja palvelinsovellus olettaa merkkijonon tallentuvan tietokantaan kokonaisena. (Bynens, 2012). MySQL käyttää versiosta 8.0 eteenpäin oletuksena utf8mb4-merkistöä, mutta tällä hetkellä Lammaspaikan käyttämä MySQL-versio on 5.7, joten merkistö täytyy ainakin vielä asettaa manuaalisesti (Oracle, n.d).

Sivun ensimmäisen latauskerran nopeutta saatiin parannettua ottamalla käyttöön HTTP-pyyntöjen kompressointi palvelimella. Kompressointiin käytetään häviötöntä gzip-algoritmia, joka on yleisesti verkkopalvelinten ja selainten tukema. (Mozilla.org, 2022). Ennen kompressoinnin käyttöönottoa ensimmäisellä latauskerralla lähetettävien tiedostojen yhteiskoko oli n. 2.18 megatavua. Kompressoinnin kanssa lähetetyn datan yhteismäärä on n. 476 kilotavua. Lähetettävän datan määrästä saatiin siis vähennettyä yhteensä n. 78%. Muutos ei vielä tässä vaiheessa vaikuta ratkaisevasti sivun latausnopeuteen käyttäjän näkökulmasta, mutta ero tulee näkyvämmäksi koodikannan ja JavaScript-bundlen koon kasvaessa.

Sovelluksen nopeutta parannettiin myös vähentämällä palvelimelle tehtävien API-kutsujen määrää. Teurastamoiden ennusteet ja tukkujen suunnitelmat koostuvat yhdestä tai useamman päivä tai -viikkoarviosta. Ennen refaktorointia API-rajapinnassa jokaista päivä ja -viikkoarviota kohden oli oma resurssinsa. Sovelluksen frontendin näkökulmasta ennusteita ja suunnitelmia käsitellään kuitenkin yksikköinä; aina suunnitelmaa katseltaessa ja muokatessa palvelimelta haetaan myös kaikki suunnitelman päivä tai -viikkoarviot. Tämä aiheutti sovelluksessa ylimääräistä viiveaikaa, sillä jokainen arvio piti hakea erillisellä HTTP-pyynnöllä. Refaktoroinnin yhteydessä API-rajapintoja muutettiin niin, että ennuste ja sen arviot voidaan hakea palvelimelta yhtenä resurssina.

Käyttöönoton aikaisten virhekorjauksien helpottamiseksi sovelluksen ylläpitokäyttäjien käyttöliittymään lisättiin näkymä lokiviestien katselua varten (kuva 3). Viestien kirjoittaminen oli jo aikaisemmin toteutettu palvelimelle, mutta niiden katselu frontendissä ei ollut mahdollista. Lokiviestien näyttäminen tarjoaa paremman lähtökohdan testikäyttäjien ilmoittamien vikojen syiden selvittämiseen.



KUVA 3. Lokiviestien näkymä ylläpitäjän käyttöliittymässä.

3 SOVELLUKSEN KONTITTAMINEN

3.1 Kontit yleisesti

3.1.1 Konttien merkitys ohjelmistokehityksessä

Viime vuosina sovelluskonteista on tullut suosittu tapa ajaa useita palvelinohjelmia samanaikaisesti yhdellä fyysisellä palvelimella. Konttien tarkoituksena on pakata sovellukset ja niiden riippuvuudet standardisoidulla tavalla, mikä sallii sovelluksen helpomman siirrettävyyden eri palvelinympäristöjen välillä, ja vähentää sovelluksen asentamiseen tarvittavaa sovelluskohtaista erityisosaamista. Kontissa ajettava sovellus on eristetty käyttöjärjestelmästä ja muista konteista, jolloin muutokset kontin sisäisessä konfiguraatiossa eivät aiheuta epätoivottuja sivuvaikutuksia muualla järjestelmässä. Sovellusten eristäminen toisistaan parantaa myös tietoturvaa, sillä se rajoittaa haavoittuvuuksien potentiaalista vaikutusaluetta. Konteille varattujen resurssien kuten muistin ja suoritinajan määrää voi myös säädellä, jolloin palvelimen resursseja voidaan priorisoida sovelluskohtaisesti.

Ennen konttiteknologioiden kehittämistä samaan tarkoitukseen käytettiin virtuaalikoneita. Virtuaalikoneet käyttävät kuitenkin paljon resursseja, sillä ne ajavat kokonaista käyttöjärjestelmää ja simuloivat laitteiston toimintaa laitteen varsinaisen käyttöjärjestelmän päällä. Kontit ovat huomattavasti kevyempiä, koska ne eivät tarvitse omaa käyttöjärjestelmäänsä vaan hyödyntävät nykyaikaisten käyttöjärjestelmäydinten ominaisuuksia sovellusten eristämiseen. Tämä sallii järjestelmän resurssien tehokkaamman hyödyntämisen, jolloin samalla fyysisellä palvelimella voidaan ajaa enemmän samanaikaisia sovelluksia kuin virtuaalikoneita käyttäessä (Poulton, 2020).

Sovelluskontit itsessään ovat yleiskäsitteinen korkean tason konsepti, joille on kuitenkin olemassa Open Container Initiative (OCI) -niminen standardi. OCI määrittelee konttien ajonaikaisen rakenteen, konttikuvien tiedostoformaatin sekä konttien ajoympäristön perustoiminnot. Standardi mahdollistaa sen, että kaikki

sen toteuttavat ajoympäristöt voivat ajaa OCI-kontteja, mikä parantaa OCI-konttien siirrettävyyttä (Opencontainers.org, n.d.).

3.1.2 Docker

Docker on Linuxilla ja Windowsilla toimiva ohjelmisto OCI-konttien hallintaan. Se toteuttaa OCI-standardin ja tarjoaa käyttäjäystävällisen tavan konttien kokoamiseen ja ajoon. Dockeria voi käyttää komentoriviltä tai Docker Desktop -sovelluksesta. Dockerin Linux-versio koostuu kolmesta pääkomponentista: dockerd, containerd ja runc. Runc on OCI:n ajonaikaisen standardin toteuttava ajoympäristö. Se huolehtii yksittäisen kontin hallinnasta. Runc hyödyntää Linux-ytimen nimiavaruuksia konttien eristämiseen. Containerd (container daemon) puolestaan on taustaprosessi, joka hallinnoi runc-prosesseja. Dockerd (Docker daemon) huolehtii dockerin omien toimintojen ylläpidosta. Dockerin komponentit on suunniteltu modulaarisiksi niin, että niitä voi korvata vaihtoehtoisilla toteutuksilla tarpeen mukaan.

Jotta Dockerilla voi ajaa sovelluskonttia, siitä tulee ensin koota konttikuva. Konttikuva on etukäteen koottu malli ajonaikaisesta kontista, josta voidaan käynnistää useita kontteja. Konttikuvissa kaikki ennen ajoa tarvittavat asennusvaiheet on valmiiksi suoritettu, jolloin itse kontin käynnistäminen on mahdollisimman nopeaa.

3.2 Kontin rakentaminen

Sovelluksen konfiguroimista kontissa ajettavaksi kutsutaan kontittamiseksi (containerization). Tähän sisältyy tyypillisesti sovelluksen refaktoroiminen konttiin sopivaksi ja konttikuvan luominen. Lammaspaikka-sovelluksen tapauksessa kontittaminen helpottaa sovelluksen käyttöönottoa mahdollisessa tulevassa ajoympäristössä. Kontitus vähentää tarvittavia asennusvaiheita, sillä ne suoritetaan osana konttikuvan koontia. Lammaspaikkaa kontittaessa pyritään käyttämään parhaita käytäntöjä, jotta kontitus toimisi samalla esimerkkinä Node.js-pohjaisen sovelluksen kontituksesta.

Dockerissa konttikuvan kokoamisohjeet määritellään Dockerfile-tiedostolla (kuva 4). Dockerfile koostuu ohjeista, jotka suoritetaan vuoron perään koottavaa konttikuvaa vasten. Dockerfilen komennoissa kirjainkoolla ei ole merkitystä, mutta ne käytännön mukaan kirjoitetaan yleensä isoilla kirjaimilla. Dockerfile toimii samalla myös dokumentaationa sovelluksen rakenteesta. Siitä käy selkeästi ilmi, mitä riippuvuuksia ja asennusvaiheita sovellus tarvitsee toimiakseen (Poulton, 2020).

```
You, 3 days ago | 1 author (You)
1  FROM node:lts-alpine
2
3  # Set correct timezone
4  RUN apk add --no-cache alpine-conf \
5      && setup-timezone -z Europe/Helsinki
6
7  WORKDIR /lammaspaikka
8
9  COPY package*.json ./
10
11 # 'ci' is faster than 'install' for clean-slate installs
12 # Don't install dev dependencies
13 RUN npm ci --only=production
14
15 # Set the api url used during frontend build process from build arguments
16 # Use localhost:8080 as default
17 ARG publicUrl=http://localhost:8080
18 ENV PUBLIC_URL=${publicUrl}
19
20 # Frontend build mode, controls visibility of test user buttons on login page
21 ARG frontendMode=development
22 ENV REACT_APP_FRONTEND_MODE=${frontendMode}
23
24 COPY public ./public
25 COPY src ./src
26 RUN npx react-scripts build
27
28 COPY . .
29
30 # Set the internal port used by the container
31 # The public-facing port can be changed to 80 or 443 at runtime
32 ENV PORT=8080
33 EXPOSE 8080
34
35 CMD ["npm", "run", "start"]
36
```

KUVA 4. Lammaspaikan Dockerfile-tiedosto.

Dockerfile alkaa FROM-komennolla, joka määrittelee kontin pohjakuvan. Pohjakuvat tyypillisesti ladataan ulkoisesta konttikuvarekisteristä, joista yleisin on Docker Hub. Lammaspaikan pohjakuvana on käytetty Node.js:n virallista konttikuvaa, joka sisältää käyttöjärjestelmän perustyökalujen lisäksi Node.js-ajoympäristön ja npm-paketinhallintaohjelman. Node.js:n vakiokuva perustuu buildpack-

deps-nimiseen kuvaan, joka on yleisesti käytetty pohja konttikuville. Tarjolla on myös Alpine Linuxiin pohjautuva versio, joka on huomattavasti kevyempi, muttei sisällä yleisiä työkaluja, jotka Linux-jakeluihin yleensä sisältyvät. Koska sovelluksen palvelintoteutus ei ole riippuvainen ulkoisista natiivikirjastoista, peruskuvana päädyttiin käyttämään Alpine Linuxia. Alpine Linuxin konttikuva on suuruudeltaan vain n. 5 megatavua verrattuna esimerkiksi Ubuntuun, joka on kooltaan 70 megatavua. Tarvittaessa työkaluja voi kuitenkin asentaa Alpine Linuxin APK-pakетinhallintaohjelmalla. Lammaspaikan Dockerfilessä rivillä 4 tätä käytetään oikean aikavyöhykkeen konfiguroimiseen kontissa, asentamalla tarvittava konfiguraatio-ohjelma. Aikavyöhykkeen konfiguroinnin yhteydessä on käytetty Dockerin RUN-komentoa, jolla voidaan ajaa komentorivikomentoja kontin sisällä. WORKDIR-komento asettaa työhakemiston seuraaville komennoille. Oletusarvoltaan työhakemisto on tiedostojärjestelmän juuressa. Työhakemisto kannattaa asettaa ennen sovelluksen tiedostojen kopiointia, ettei järjestelmätiedostoja vahingossa korvata. COPY-komennolla kopioidaan tiedostoja paikalliselta tiedostojärjestelmältä konttiin. CMD-komento määrittää kontissa ajettavan komennon. Lammaspaikan tapauksessa se käynnistää sovelluksen palvelintoteutuksen.

Kuvaa kootessa Dockerfilen sisältämät komennot suoritetaan peräkkäin. Konttikuva koostuu kerroksista, jossa jokainen uusi kerros kuvastaa tiedostojärjestelmään tehtyä muutosta. Kerrosmaisen rakenteen tarkoituksena on vähentää konttien rakentamiseen kuluvaa aikaa: kerrokset tallennetaan koonnin yhteydessä välimuistiin, (build cache) ja jos identtinen kerros on koottu jo aiemmin, sitä ei tarvitse koota uudelleen vaan valmis kerros haetaan välimuistista.

Dockerfilestä kootaan konttikuva komentorivikomennolla *docker build hakemisto*, jossa hakemisto on Dockerfilen sisältävä hakemisto eli rakennuskonteksti. Rakennuskonteksti määrittää ne paikalliset tiedostot, joita konttikuvaa rakentaessa voidaan käyttää. Rakennuskontekstiin kannattaa sisällyttää vain ne tiedostot, joita konttikuvassa tarvitaan, sillä rakennuskontekstin tiedostojen muuttumattomuus määrittelee sen, voidaanko kokoamisprosessin välimuistia hyödyntää. Jos rakennuskontekstiin sisältyy tiedosto, joka muuttuu rakennuskertojen välissä, välimuisti hylätään vaikkei tiedostoa hyödynnetä kontissa mihinkään. Ensimmäisen hylkäyksen jälkeen kaikki seuraavat rakennusvaiheet suoritetaan välimuistista riippumatta, sillä muutokset aiemmin määritetyissä kerroksissa voivat vaikuttaa

myöhempiin kerroksiin. Rakennusvaiheet kannattaa määritellä sellaisessa järjestyksessä, jossa muuttumattomat osat kootaan ensimmäisenä ja muuttuvimmat viimeisenä. Näin välimuistia saadaan hyödynnettyä tehokkaimmin. Konttikuvaa kootessa Docker merkitsee välimuistista haetut kerrokset CACHED-sanalla (kuva 5). Lammaspaikan kontin kokoamisprosessi on optimisoitu jakamalla tiedostojen kopiointi kolmeen vaiheeseen. Ensin asennetaan sovelluksen riippuvuudet kopioimalla package.json- ja package-lock.json -tiedostot ja ajamalla paketit asentava ”npm-ci”-komento. Riippuvuuksien asentaminen tehdään alussa, sillä jos riippuvuuksia on lisätty tai poistettu, on myös todennäköistä, että sovelluksen koodia on sen seurauksena muutettu. Seuraavaksi kopioidaan sovelluksen frontendin koodi ja kootaan siitä optimisoitu jakelupaketti ”npx react-scripts build” komennolla. Palvelinpuolen koodi kopioidaan vasta lopuksi, jottei frontendiä tarvitse koota uudestaan, jos vain backendin koodiin on tehty muutoksia, ja pelkässä tiedostojen kopioinnissa kestää vain vähän aikaa. Riippuvuuksien asentaminen ja frontendin kokoaminen sen sijaan voi kestää kymmenistä sekunneista useisiin minuutteihin, joten kontin koonnin optimisointi välimuistin käytön kannalta vähentää tiiviissä kehityssyklissä odottamiseen kuluvaa aikaa merkittävästi.

```
(main) ville@laptop ~/git/AhlmanPYM
$ docker build . -t lp:latest
[+] Building 119.2s (14/14) FINISHED
 => [internal] load build definition from Dockerfile                                                0.1s
 => => transferring dockerfile: 898B                                                                0.0s
 => [internal] load .dockerignore                                                                   0.1s
 => => transferring context: 34B                                                                    0.0s
 => [internal] load metadata for docker.io/library/node:lts-alpine                                 10.0s
 => [1/9] FROM docker.io/library/node:lts-alpine@sha256:2c6c59cf4d34d4f937ddfcf33bab9d8bbad8658d1b9de7b97622566a5  0.0s
 => [internal] load build context                                                                   0.1s
 => => transferring context: 15.84kB                                                                0.0s
 => CACHED [2/9] RUN apk add --no-cache alpine-conf     && setup-timezone -z Europe/Helsinki        0.0s
 => CACHED [3/9] WORKDIR /lammaspaikka                                                              0.0s
 => CACHED [4/9] COPY package*.json ./                                                              0.0s
 => CACHED [5/9] RUN npm ci --only=production                                                       0.0s
 => CACHED [6/9] COPY public ./public                                                               0.0s
 => [7/9] COPY src ./src                                                                            0.2s
 => [8/9] RUN npx react-scripts build                                                             107.4s
 => [9/9] COPY . .                                                                                  0.3s
 => exporting to image                                                                              0.8s
 => => exporting layers                                                                             0.7s
 => => writing image sha256:1360de8ebc31d76ebdf8d8c35c1f40413e4a0cb29403d08b5b9a82b0e2ad6939        0.0s
 => => naming to docker.io/library/lp:latest                                                        0.0s
```

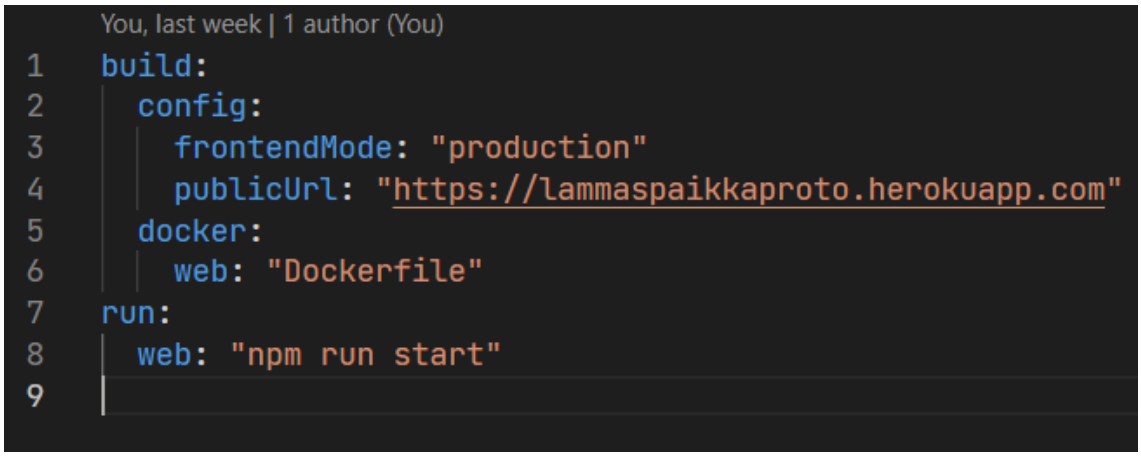
KUVA 5. Lammaspaikan kontin kokoaminen komentoriviltä.

3.3 Kontin koonnin automatisointi

Sovelluksen demoversion palveluntarjoajana käytetään Herokua. Kehitystyön sujuvoittamiseksi muutosten julkaiseminen sovelluksen versionhallintarepositorioon käynnistää automaattisesti kontin kokoamisen ja sovelluksen julkaisun Herokussa. Demoversion lisäksi Herokussa julkaistaan myös erillinen kehitysversio,

jossa sovellukseen tehtyjä uusia muutoksia voidaan ensin testata. Kehitysversio on myös kätevä tapa esitellä uusia ominaisuuksia asiakkaalle palavereissa. Tarkoituksena on, että testikäyttäjille julkaistussa demoversiossa käytettävä koodi olisi mahdollisimman virheetöntä, eli kehitysversion muutokset liitetään osaksi demoversiota vasta, kun ne on todettu teknisesti toimiviksi. Demo- ja kehitysversioilla on myös erilliset tietokannat, jotta tietokantataulujen migraatiot voidaan ensin testiajaa kehitysversiossa. Tietokantojen eriyttäminen estää myös testikäyttäjien tietojen tuhoamisen vahingossa kehitystyön yhteydessä.

Kontin koonnin automatisointia varten Heroku tarvitsee YAML-merkkausta käyttävän heroku.yml-tiedoston, jossa koottavat kontit ja konfiguraatioargumentit määritellään (kuva 6). Lammaspaikan tapauksessa tiedosto sisältää kaksi pääosiota, build ja run. Build-osio määrittelee koottavan konttikuvan sekä Dockerille annettavat koontiargumentit, ja run kontissa annettavan komennon, jolla sovellus käynnistetään. Muita heroku.yml:ssä sallittavia osioita on setup, joka määrittää sovelluksessa käytettävät Heroku-lisäosat, sekä release, jossa voidaan antaa ennen sovelluksen julkaisua suoritettavat toimet (Heroku Dev Center, 2022).



KUVA 6. Lammaspaikan demoversion heroku.yml-tiedosto.

Sovelluksen eri versioiden konfiguraatioeroja hallitaan Herokussa ympäristömuuttujien avulla. Node.js-pohjaisissa sovelluksissa vakiintuneeksi käytännöksi on noussut NODE_ENV-nimisen ympäristömuuttujan käyttö kehitys- ja tuotantoversioiden tunnistamiseen toisistaan. Kehitysversiossa muuttujan arvoksi on asetettu ”development” ja demoversiossa ”production”. Muuttujaa käytetään esimerkiksi käyttäjien varmistussähköpostien lähettämisen yhteydessä: kehitysversiossa viestit lähetetään erilliseen testipalveluun, jossa ne ovat koko kehitystiimin

nähtävissä. Demoversiossa sen sijaan sähköpostiviestit lähetetään käyttäjien oikeille sähköpostitileille. Ympäristömuuttujat sopivat hyvin sovelluksen ajonaikaisen konfiguraation määrittämiseen, mutta ne eivät ole käytettävissä konttia rakentaessa, joten rakennusvaiheessa kehitys- ja demoversion erot määritellään Dockerin argumenttien avulla. Tällaista konfiguraatiota tarvitsee sovelluksen verkkokäyttöliittymä, joka kootaan valmiiksi kontin kokoamisen aikana. Konfiguraatiossa asetetaan frontendin API-kutsuihin käyttämän palvelimen verkko-osoite sekä piilotetaan vain testikäytössä tarvittavat käyttöliittymäkomponentit. Esimerkiksi kirjautumislomakkeessa on testiversiossa napit, joita painamalla pääsee nopeasti kirjautumaan testitunnuksille ilman, että sovellukseen tarvitsee luoda käyttäjätiliä. Normaalisti Dockerin argumentit annetaan konttia kootessa komentorivillä avain-arvo-pareina, mutta Herokussa ne määritellään heroku.yml-tiedostossa. Lammaspaikan Dockerfilessä argumenteille määritetään oletusarvot ARG-komennolla ja sidotaan ympäristömuuttujiksi ENV-komennolla, jotta ne ovat frontendin kokoavan ”npx react-scripts build” -komennon käytettävissä.

3.4 Dockerin tietoturva

Koska Docker-kontissa voi ajaa mitä tahansa ohjelmia, tietoturva on syytä ottaa huomioon konttia rakentaessa ja ajaessa. Koska kontit hyödyntävät suoraan laitteen käyttöjärjestelmäydintä työkuormien eristämiseen toisistaan, ne eivät tarjoa yhtä vahvaa suojausta haittaohjelmia vastaan kuin virtuaalikoneet, jotka simuloivat koko käyttöjärjestelmää, ytimen mukaan lukien. Virtuaalikoneita kannattaakin suosia konttien sijaan, jos kontin sisällä ajettavaan koodiin ei voida täysin luottaa.

Ulkoisista konttikuvarekistereistä kuvia ladattaessa tulee olla varovainen, sillä kuka tahansa voi ladata konttikuvia Docker Hubin kaltaisiin avoimiin rekistereihin. Docker Hubissa on laaja kokoelma virallisia konttikuvia, jotka ovat Dockerin kehittäjien ylläpitämiä. Kaikki muut kuvat ovat kuitenkin kolmansien osapuolten vastuulla, eivätkä ne ole välttämättä yhtä hyvin dokumentoituja tai turvallisia kuin viralliset kuvat (Poulton, 2020). Hyvänä käytäntönä onkin käyttää ensisijaisesti vain Docker Hubin virallisia konttikuvia omien konttien pohjana.

Docker voi myös skannata paikalliset konttikuvat tunnettujen haavoittuvuuksien havaitsemiseksi käyttämällä ”docker scan” -komentoa. Skannaus tarkistaa kontin

sovelluksen lähdekoodin lisäksi sekä pohjakuvan mahdolliset ongelmat että myös tunnetuilla paketinhallintaohjelmilla, kuten npm:llä asennettujen riippuvuuksien haavoittuvuudet. Docker käyttää oletusarvoisesti Snyk-palvelun tarjoamaa skannausmoottoria, joka ylläpitää laajaa tietokantaa eri ohjelmistoissa ja pakettirepositorioissa löydetyistä haavoittuvuuksista (Docker Inc, n.d.).

4 KÄYTTÖÖNOTTOKOKEILU

Sovelluksen pilotointiin on kutsuttu testikäyttäjiksi joukko lammasteurastamoiden ja -tukkujen edustajia. Kokeilun tarkoituksena on löytää sovelluksessa piileviä ongelmakohtia ja kerätä käyttäjäpalautetta sovelluksen jatkokehityksen ohjaamiseksi oikeaan suuntaan. Käyttöönottokokeilu aloitettiin tilaisuudella, jossa testikäyttäjille ohjeistettiin sovelluksen toiminta sekä tukun että teurastamon näkökulmasta. Koska sovelluksen kehittäminen on toteutettu lammasteollisuudesta riippumattomana hankkeena, käyttökokeiluun osallistuvilla yrityksillä ei ole entuudestaan vahvaa omakohtaista motiivia sovelluksen käyttöönottoon. Testikäyttäjiä tuleekin aktiivisesti kannustaa sovelluksen testaamiseen, jotta sen käyttöönotosta koituvat hyödyt tulevat heille ilmi. Käyttäjille lähetetään ajoittain muistutuksia sähköpostilla, ja kokeilujakson aikana järjestetään välipalautetilaisuus, jolla käyttäjien aktiivisuutta pidetään yllä. Ensimmäisen viikon aikana 4 yritystä rekisteröityi Lammaspaikan käyttäjiksi, joista kaksi oli muodostanut keskenään sopimuskumppanuuden. Sovellukseen ei kuitenkaan tänä aikana lisätty uusia ennusteita, joten varsinainen käyttö oli ensimmäisen viikon osalta vähäistä. Seuraavan viikon aikana 3 muuta yritystä liittyi, ja useimmilla yrityksillä oli ainakin yksi sopimuskumppanuus.

4.1 Käyttäjien palaute

Käyttäjien palautteenanto toteutettiin verkkolomakkeena, jota varten sovellukseen lisättiin korostettu linkki. Linkki näkyy sovelluksen jokaisella sivulla, jotta käyttäjän siirtyminen kehitysehdotuksen havaitsemisesta palautelomakkeen täyttämiseen olisi kitkaton. Annettuun palautteeseen pyritään reagoimaan mahdollisimman lyhyellä viiveellä, jotta sen merkittävyys sovelluksen kehityssuunnan kannalta välittyisi myös käyttäjille.

Ensimmäinen kehitysehdotus ilmaisi huolen sovelluksen tietosuojaan liittyen. Sen etusivulla näkyy pylväskaavio, jossa on yhteenlaskettuna kaikkien teurastamojen antamat pitkät ennusteet viimeisten viikkojen ajalta. Palautetta tuli siitä, että kaaviosta voi päätellä kilpailevien teurastamoiden ennustamat eläinmäärät

vähentämällä omat ennusteet laskusta. Kaavio päätettiin ainakin tilapäisesti poistaa käytöstä, sillä se ei ole sovelluksen toiminnan kannalta välttämätön, vaan se on pikemminkin tarkoitettu esitteeksi vierailijoille tai potentiaalisille uusille käyttäjille, eikä se siten ole testikäytön aikana tarpeellinen.

Selkeämmälle kommunikoinnille sopimuskumppanien välillä kaivattiin ratkaisuja. Ennusteiden yhteyteen toivottiin kommentointimahdollisuutta, jotta niistä voitaisiin ilmaista tarkempaa vapaamuotoista tietoa. Samanlaista ominaisuutta kysyttiin jo aiemmin syksyllä sovelluksen prototyypin esittelytilaisuudessa. Sopimuskumppaneille haluttiin myös välittää muita tietoja, kuten ennusteiden puuttumisen tai vähäisyyden syy. Sovelluksen sisäistä pikaviestintäominaisuutta on ehdotettu ratkaisuksi kommunikointiongelmiin.

4.2 Havaitut ongelmat

Käyttöönottokokeilun aikana kehitystiimi havaitsi sovelluksessa myös muita ongelmia, jotka eivät tulleet suoraan testikäyttäjien palautteena. Näiden ongelmien korjaamiseen keskityttiin silloin, kun testikäyttäjien ilmoittamia ongelmia ei ollut korjattavana.

Sovellus käyttää omaa terminologiaansa, joka saattaa olla vaikeaa omaksua, jos sen toiminnot eivät ole entuudestaan tuttuja. Esimerkiksi ennusteiden ja suunnitelmien ero voi olla vaikea ymmärtää ilman erillistä ohjeistusta. Sovelluksen käytön selkeyttämiseksi sen tärkeimmille sivuille lisättiin lyhyitä ohjetekstejä, jotka selostavat ominaisuuksien käyttötarkoitusta.

Yritysten edustajat pystyvät asettamaan sovelluksessa yritykselleen profiilikuvan, joka näkyy muille käyttäjille. Asetettua profiilikuvaa ei kuitenkaan aiemmin pystynyt poistamaan ilman, että sen korvaisi toisella kuvalla. Käyttökokemuksen parantamiseksi yrityksille lisättiin ominaisuus profiilikuvan poistamista varten ilman uuden kuvan lisäämistä.

Yritystä rekisteröitäessä Y-tunnuksen vapaamuotoisuus aiheutti sen, että yksittäinen yritys pystyi rekisteröitymään monta kertaa kirjoittamalla Y-tunnuksen eri

muodossa. Testikäytön aikana yhden yrityksen kaksi eri edustajaa olivat toisistaan tietämättä molemmat rekisteröineet yrityksensä. Y-tunnuksen validointia tiukennettiin niin, ettei kahta saman Y-tunnuksen omaavaa yritystä voi rekisteröidä.

5 POHDINTA

Opinnäytetyössä käsiteltiin Lammaspaikka-sovelluksen käyttökokeilua ja sitä ennen tehtyjä valmistelutoimenpiteitä. Työn tavoitteen mukaisesti Lammaspaikka-sovellus valmisteltiin käyttöönottoa varten, ja sen potentiaaliset tulevat käyttäjät ovat päässeet kokeilemaan sen käyttöä käyttökokeilujakson aloittamisen myötä. Valmistelut saatiin ajoissa valmiiksi, eikä käyttöönotossa ei ole ollut ainakaan vielä merkittäviä teknisiä ongelmia. Työn tarkoituksena on ollut refaktoroida sovelluksen koodi, kontittaa sovellus, sekä parantaa sitä käyttäjäpalautteen perusteella. Refaktorointi ja kontittaminen onnistuivat suunnitellun mukaisesti, mutta käyttäjäpalautteen pohjalta tehtyjä muutoksia ei ole tämän opinnäytetyön puitteissa toteutettu. Palautetta on tähän mennessä tullut vähän, ja sovelluksen uusi kehitystiimi on huolehtinut niiden pohjalta tehtävien muutosten toteuttamisesta. Tästä syystä raportointi käyttökokeilun osalta on ollut vähäistä.

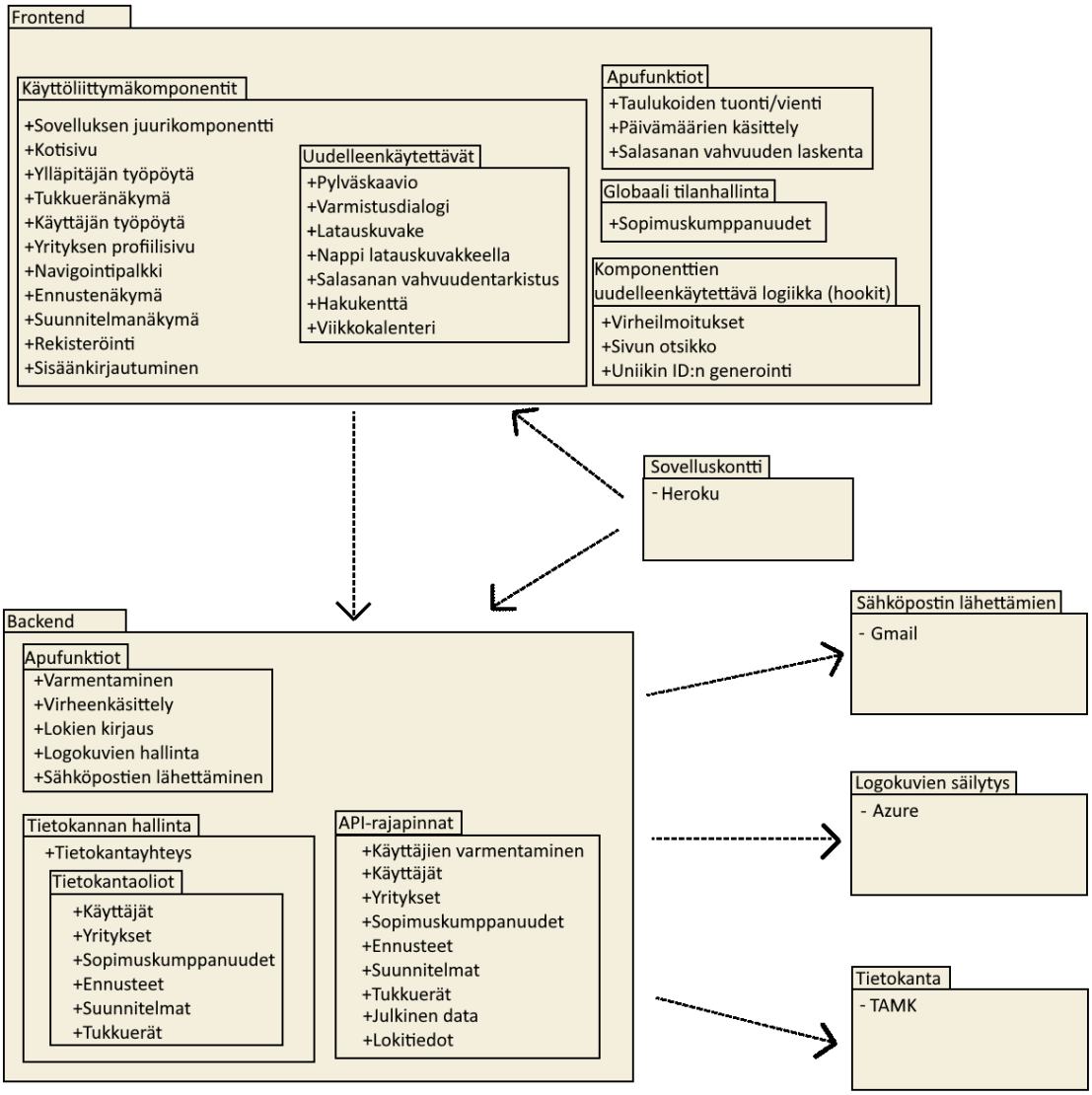
Työn alkuosassa refaktoroitiin sovelluksen koodia. Tavoitteena oli parannella sovelluksen toimintaa ja korjata sen virheitä, jotta käyttökokemus olisi testikäyttäjille jo toteutettujen ominaisuuksien osalta sujuvampi. Näin testikäytössä voitiin keskittyä korkeamman tason ongelmiin. Tarkoituksena oli korjata sovelluksen tietoturvaongelmia ja optimoida sen nopeutta. Sovelluksen kriittisimmät ongelmat saatiin korjattua, mutta muutamia vasta myöhemmin havaittuja ongelmia jouduttiin korjaamaan myös käyttökokeilun aikana.

Sovelluksen kontittamista käsittelevässä osassa tavoitteena oli siirtää sovellus konttiin käyttöönoton helpottamiseksi, sekä antaa lukijalle esimerkki Node.js-pohjaisen sovelluksen kontittamisesta. Kontittamisessa pyrittiin käyttämään parhaita käytäntöjä kontin kokoamisajan vähentämiseksi. Lopputuloksena syntyi kontti, joka julkaistiin Heroku-palvelussa testikäyttöä varten, ja jota voidaan myös hyödyntää sovelluksen lopullisella palvelimella. Samalla sovelluksesta eriytettiin erillinen kehitysversio, jonka avulla uusia muutoksia voidaan helpommin testata ja demonstroida ennen kuin ne julkaistaan testikäyttäjien käytettäviksi.

Loppuosassa tarkasteltiin käyttäjien antamaa palautetta ja niiden pohjalta tehtyjä toimenpiteitä. Käyttökokeilun tavoitteena oli antaa sovelluksen tuleville potentiaalisille käyttäjille mahdollisuus vaikuttaa sen kehityssuuntaan keräämällä palautetta nykyisistä ominaisuuksista sekä antamalla mahdollisuus ehdottaa uusia toimintoja. Käyttökokeiluun osallistuminen on tähän mennessä ollut yritysten osalta odotettua epäaktiivisempaa, mutta siitä huolimatta kokeilusta on jo saatu hyödyllistä palautetta, jonka pohjalta sovelluksen kehitystyötä voidaan jatkaa.

Tärkeimpiä ominaisuuksia jatkokehityksen kannalta ovat mahdolliset Sorkka- ja Nettikatras-integraatiot, jotka mahdollistaisivat eläintietojen katkeamattoman kulun tuotantoketjun läpi. Tulevaisuudessa sovelluksen käyttäjätyyppien vuorovaikutusta voidaan joutua muuttamaan, sillä lammastalouden tuotantoketjussa teurastamon ja tukun välissä toimii vielä lisäksi leikkaamo. Teurastamon ja leikkaamon toimia tehdään usein samassa yrityksessä, mutta leikkaamo voi myös olla kolmas osapuoli. Sovellus ei vielä tue leikkaamoa erillisenä yritystyyppinä, vaan oletuksena kaikki tiedonvaihto tapahtuu suoraan teurastamon ja tukun välillä.

Maaseudun kehittämisrahastosta rahoitetun hankkeen on määrä päättyä vuonna 2023, jolloin sovellukselle etsitään omistajaa. Tämä voi olla esimerkiksi Sorkka-järjestelmän tai Nettikatraan omistaja, tuottajaorganisaatio, alan yritys tai muu taho, joka on valmis ylläpitämään sovellusta ja huomioimaan tasapuolisesti sekä tukkuostajat että teurastamot sen jatkokehityksessä. Sovelluksen käyttämiä ulkoisia riippuvuuksia tulee harkita uudelleen omistajuutta siirrettäessä, sillä ne ovat kaikki tarkoitettu tilapäisiksi ratkaisuiksi (kuvio 1). Sovellus on kuitenkin suunniteltu niin, että ulkoisten palveluiden vaihtaminen toisiin ei ole vaikeaa.



KUVIO 1. Pakkauskaavio Lammaspaikan eri osista ja sen ulkoisista riippuvuuksista.

LÄHTEET

Bynens, M. 30.7.2012. How to support full Unicode in MySQL databases. Verkkosivu. Viitattu 28.3.2022. https://mathiasbynens.be/notes/mysql-utf8mb4

Docker Inc. n.d. Vulnerability scanning for Docker local images. Verkkosivu. Viitattu 23.3.2022. https://docs.docker.com/engine/scan/

Hanley, M. 15.11.2021. GitHub's commitment to npm ecosystem security. Verkkosivu. Viitattu 9.3.2022. https://github.blog/2021-11-15-githubs-commitment-to-npm-ecosystem-security/

Heroku Dev Center. 17.2.2022. Building Docker Images with heroku.yml. Verkkosivu. Viitattu 16.3.2022. https://devcenter.heroku.com/articles/build-docker-images-heroku-yml

Mozilla.org, 18.2.2022. Gzip compression. Verkkosivu. Viitattu 28.3.2022. https://developer.mozilla.org/en-US/docs/Glossary/GZip_compression

Npmjs.com. 23.9.2020. Auditing package dependencies for security vulnerabilities. Verkkosivu. Viitattu 9.3.2022. https://docs.npmjs.com/auditing-package-dependencies-for-security-vulnerabilities

Opencontainers.org. n.d. About the Open Container Initiative. Verkkosivu. Viitattu 9.3.2022. https://opencontainers.org/about/overview/

Oracle. n.d. MySQL 8.0 Reference Manual - Character Sets, Collations, Unicode. Verkkosivu. Viitattu 28.3.2022. https://dev.mysql.com/doc/refman/8.0/en/charset.html

Poulton, N. 2020. Docker Deep Dive. Birmingham: Packt Publishing.

Sorkka.fi. n.d. Tietoa Sorkan ominaisuuksista. Verkkosivu. Viitattu 8.3.2022. https://sorkka.fi/ominaisuudet

Webpackjs.org. 10.10.2020. Webpack 5 release. Verkkosivu. Viitattu 8.3.2022. https://webpack.js.org/blog/2020-10-10-webpack-5-release