SAVONIA

OPINNÄYTETYÖ - **AMMATTIKORKEAKOULUTUTKINTO**
TEKNIIKAN JA LIIKENTEEN ALA

MAINOSINVENTAARION HALLINTAJÄRJESTELMÄ

Palveluun rekisteröityneitten yritysten käyttöön

TEKIJÄ: Elias Laitinen

SAVONIA-AMMATTIKORKEAKOULU

OPINNÄYTETYÖ
Tiivistelmä

Koulutusala Tekniikan ja liikenteen ala	
Koulutusohjelma/Tutkinto-ohjelma Tietotekniikan koulutusohjelma	
Työn tekijä(t) Elias Laitinen	
Työn nimi Mainosinventaarion hallintajärjestelmä	
Päiväys 8.4.2019	Sivumäärä/Liitteet 34
Ohjaaja(t) Jussi Koistinen, Keijo Kuosmanen	
Toimeksiantaja/Yhteistyökumppani(t) Mediani Group Oy	

Tiivistelmä

Työn tavoitteena oli suunnitella ja toteuttaa Mediani.fi-internetpalvelun käyttöön inventaarionhallintajärjestelmä, joka soveltuu erilaisten mainospintojen varaustilanteen hallintaan. Useat yritykset hallitsevat avoimia pintojaan tehottomasti ja monimutkaisesti, esimerkiksi Excel-taulukossa joka saattaa olla vain yhden ihmisen tietokoneen kovalevyllä. Yksi yrityksen tavoitteista on tarjota tälläisiin ongelmiin parempia ratkaisuja.

Lopullisen sovelluksen täytyi pystyä muun muassa luomaan ja hallitsemaan mainospintoja, luomaan ja hallitsemaan varauksia, näyttämään varaustilanne helpolla tavalla sekä mahdollistaa palvelussa tehdyn tilauksen liittäminen varaukseen.

Työssä käsitellään muun muassa kannattaako tilaajan toteuttaa inventaarionhallinta erillisenä mikropalveluna, vai monoliittisena palveluna muun tilaajan ydintoiminnan kanssa samassa sovelluksessa. Tehtyjen päätösten pohjalta toteutettiin palvelua beetavaiheeseen asti.

Työn lopputuloksena päätettiin sovelluksen arkkitehtuurista sekä luotiin beetaversio sovelluksesta. Sovellus toteutettiin Pythonilla käyttäen hyväksi Django-sovelluskehystä. Sen lisäksi käytettiin AngularJS-, jQuery- sekä Javascript-kieliä käyttöliittymän tekoa varten.

Avainsanat
Mainos, Inventaarionhallinta, Django, Python

SAVONIA UNIVERSITY OF APPLIED SCIENCES

THESIS
Abstract

Field of Study Technology, Communication and Transport			
Degree Programme Degree Programme in Information Technology			
Author(s) Elias Laitinen			
Title of Thesis Advertisement Inventory Management Software			
Date	8 April 2019	Pages/Appendices	34
Supervisor(s) Mr. Jussi Koistinen, Senior Lecturer, Mr. Keijo Kuosmanen, Senior Lecturer			
Client Organisation /Partners Mediani Group Oy			

Abstract

The purpose of this thesis was to research different software architectures for Mediani.fi webservice's advertisement inventory management solution, and after deciding on the architecture, to build a beta version of the feature. Many companies selling advertisement space manage their inventory in inefficient, complicated and unsafe ways, for example only in an Excel spreadsheet on one employee's laptop. The objective of the software was to offer better and safer solution, while being easy to use, and having features such as creating and managing advertisement spaces, showing reservation statuses, creating and managing reservations and connecting the reservations to orders done through the company's platform.

The planning involved reading a lot of web sources and books, and also having conversations with more experienced software developers. The application was created using Python programming language and Django web framework, and the user interface was done with AngularJS, jQuery and Javascript languages.

As a result of the thesis, software architecture was decided and beta version of the inventory management solution was created. The thesis includes topics such as positive and negative sides of microservice and monolithic architectures in software development. From these observations, the company decided the architecture and built the beta version.

Keywords
advertising, advertisement, inventory management, Django, Python

ESIPUHE

Kiitos vanhemmilleni sekä veljilleni, Savonian opettajille sekä kavereille jotka ovat minua tavalla tai toisella vuosien saatossa auttaneet.

Kuopiossa 8.4.2019
Elias Laitinen

TERMIT JA LYHENTEET

Ohjelmallinen ostaminen on mainonnan ostamista automaation avulla. Ostaminen tapahtuu reaaliajassa, ja tarkoituksena on tehostaa ostamista.

Ohjelmistokehys (software framework / web framework) muodostaa rungon sen päälle rakennettavalle sovellukselle, oli kyseessä sitten tietokoneohjelma tai verkkopalvelu. Ohjelmistokehyksen tarkoituksena on nopeuttaa ohjelmointia tarjoamalla ohjelmiston osia, joita ei siten tarvitse ohjelmoida uudelleen.

Frontendillä tarkoitetaan web-sovelluksessa käyttäjän näkemää, selaimen renderoimaa osaa, esimerkiksi nettisivun ulkoasu sekä painikkeen toiminnot, jotka on toteutettu Javascriptillä.

Backendillä tarkoitetaan web-sovelluksen osaa, joka on käynnissä palvelimella, esimerkiksi tietokannat ja rajapinnat, joita frontend käyttää hyväkseen datan näyttämisessä käyttäjälle.

Yksikkötestaamisella tarkoitetaan testaamista ja laadunvarmistamista jossa lähdekoodin osia testataan yhdessä tai erikseen. Tämä tehdään tietokoneohjelmissa yleensä automaattiseksi.

Selenium-testaamisella tarkoitetaan tässä yhteydessä selaimen automaatiota niin, että sillä voidaan toteuttaa erilaisia testejä, joilla nähdään, että sovellus toimii kuten pitääkin.

ACID (Atomicity, Consistency, Isolation, Durability, suomeksi atomisuus, eheys, eristyneisyys, pysyvyys) tarkoittaa periaatteita, joilla turvataan järjestelmän eheys kaikissa tilanteissa.

DOM (Document Object Model, dokumenttioliomalli) kuvaa esimerkiksi HTML-dokumentin rakennetta puuna. Olioita voi manipuloida, vaikka Javascriptin avulla. DOM määrittää miten dokumentin sisällä välitetään tietoa.

SISÄLTÖ

1 JOHDANTO

Mediani.fi on suomalainen mainospaikkojen osto- ja myyntipalvelu. Palvelun avulla voit etsiä, vertailla sekä ostaa mainospaikkoja kohderyhmäperustaisesti. Mediakortit on tuotteistettu helposti vertailtavaan, standardiin muotoon, oli kyse mistä tahansa mediasta tai mainospaikasta. (Mediani Group Oy, 2018)

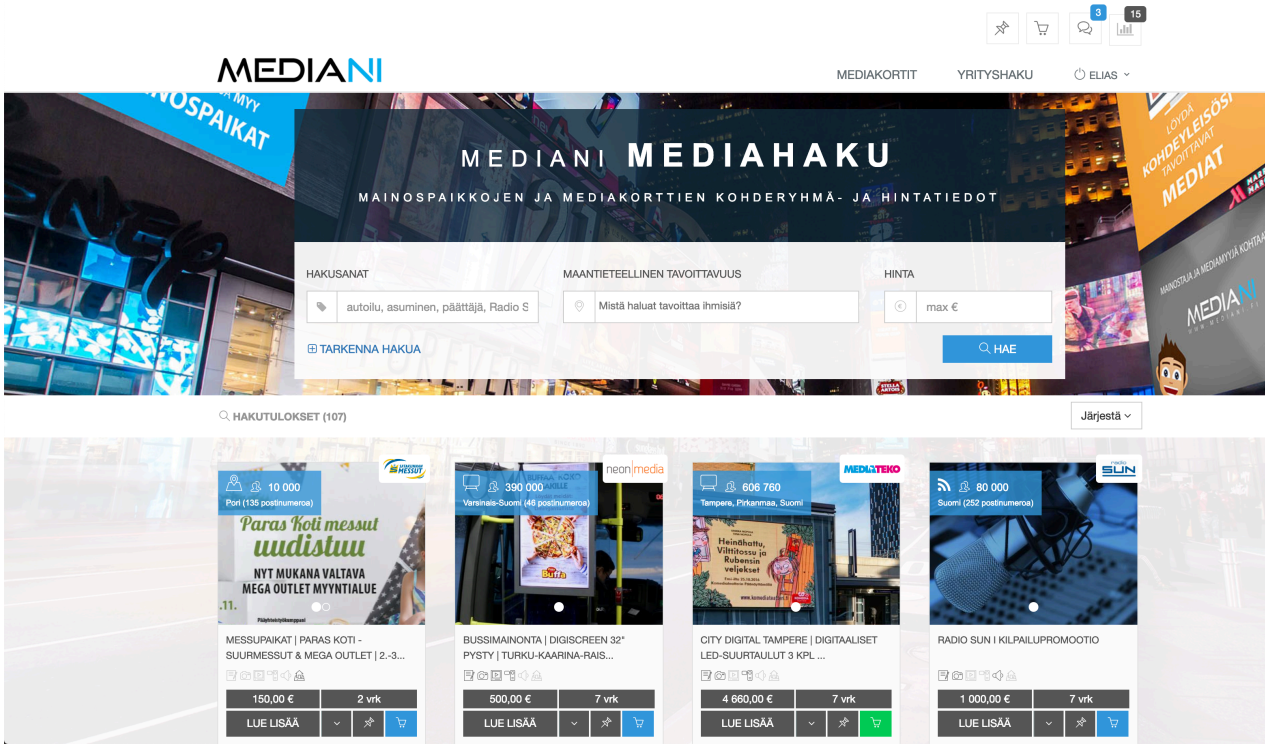
Ohjelmallinen ostaminen tulee olemaan mainosalalla kasvava trendi, ja sen mahdollistaminen tilaajan alustalla vaatii inventaarionhallinnan toteuttamista. Ohjelmallisen ostamisen mahdollistamiseksi järjestelmän täytyy tietää, onko valittu mainospaikka vapaana.

Opinnäytetyöksi valikoitui siten sen selvittäminen, kannattaako tilaajan inventaarionhallinta toteuttaa nykyisen palvelun yhteyteen, vai eriyttää omaksi mikropalvelukseen, ja sen jälkeen suunnitella sen järjestelmäarkkitehtuuri. Koska tämä on aikaa vievä prosessi, sovittiin että ohjelmointia tehdään sen verran mitä keretään ja pääpaino on suunnittelussa.

Tässä opinnäytetyössä on kuvattuna järjestelmän vaatimukset, monoliittisen- sekä mikropalvelun edut ja huonot puolet sekä inventaariohallinnan beetaversio.

2 MEDIANI.FI

Mediani.fi-palvelun tavoitteena on tehdä mainosten ostamisesta helppoa mainostajille sekä mediatoimistoille, mutta myös helpottaa mediamyyjien elämää. Palvelu tarjoaa helpon, standardoidun tavan ilmoittaa mainospaikkoja myyntiin. Palvelussa voi tehdä tilauksen mainospaikasta suoraan, koko yritykselle läpinäkyvällä tavalla, jolloin jokainen myyjä näkee tilaukset ja keskustelut, joten asioita ei jää yhden myyjän taakse niin, ettei muut yrityksen työntekijät tiedä tietyn myyjän asiakkaista mitään.



Kuva 1: Mediahaku (Laitinen, 2019)

Palvelun avulla mediamyyjä voi hallita mediakorttejaan, keskusteluitaan, sekä myyntejään helposti samassa paikassa, sekä asettaa alennuksia eri perusteilla toisille kirjautuneille käyttäjille. Sen lisäksi mediamyyjä voi luoda useita eri profiileita tapauksissa missä sama yritys omistaa esimerkiksi useita eri sanomalehtiä.

3 LÄHTÖKOHTA JA TAVOITTEET

Ohjelmallinen ostaminen vaatii sen, että yrityksen eri mainospaikkojen kapasiteettitieto on tiedossa jatkuvasti. Inventaariotiedon hallinta on asiakkaitten kanssa neuvoteltaessa tullut usein esille muutenkin, koska nykyään se saattaa pahimmassa tapauksessa olla pelkästään yhden henkilön tietokoneen kovalevyllä yhdessä Excel-tiedostossa. Yrityksen kannalta se, että tärkeä informaatio on yhden henkilön salasanojen takana, puhumattakaan siitä, ettei sitä ole edes kunnolla varmuuskopioitu minnekään, on erittäin suuri riski. Olisi tärkeää, että tämä informaatio olisi yrityksen jokaisen työntekijän saatavissa keskitetysti ja tallennettuna turvallisesti.

3.1 Tavoite

Tavoitteena oli suunnitella ja luoda helppokäyttöinen, yrityksen kaikille työntekijöille läpinäkyvä inventaarionhallintaratkaisu. Ensin oli tutkittava, tehdäänkö inventaarionhallinnasta mikropalvelu, ja sen jälkeen toteutettiin palvelusta beetaversio.

3.2 Vaatimukset

Tutkitaan mikä on tilaajan tilanteessa kaikista järkevin tapa arkkitehtuurillisesti lähteä toteuttamaan inventaarionhallintaa. Tilaaja on aloitteleva startup-yritys, jolla ei ole montaa työntekijää.
Työssä käytetään mahdollisimman paljon avoimen lähdekoodin ratkaisuja, näistä lisää myöhemmin.

Inventaarionhallinnan beetaversion tärkeimmiksi ominaisuuksiksi rajattiin resurssien luonti ja hallinta, varauksien tekeminen ja hallinta, resurssin varaustilanteen seuranta ja visuaalinen esittäminen, resurssin yksityiskohtainen tarkastelu, sekä varauksien listaaminen. Näihin täytyi suunnitella ja toteuttaa sekä front- että backend, sen jälkeen, kun yleisarkkitehtuuri oli suunniteltu. Muita ominaisuuksia toteutettiin opinnäytetyön aikarajojen puitteissa niin paljon kuin mahdollista.

Kaikkien toimintojen tuli olla mahdollisimman helppokäyttöisiä sekä ymmärrettäviä. Isojakin määriä resursseja pitää pystyä hallitsemaan mahdollisimman yksinkertaisesti. Varausten tekeminen manuaalisesti tulee olla intuitiivista, ja koneellisesti varausten tekeminen täytyy tulevaisuudessa olla mahdollista käyttäen Medianin tilaus-toimitusprosessia sekä muita myöhemmin toteutettavia tapoja hyödyntäen.

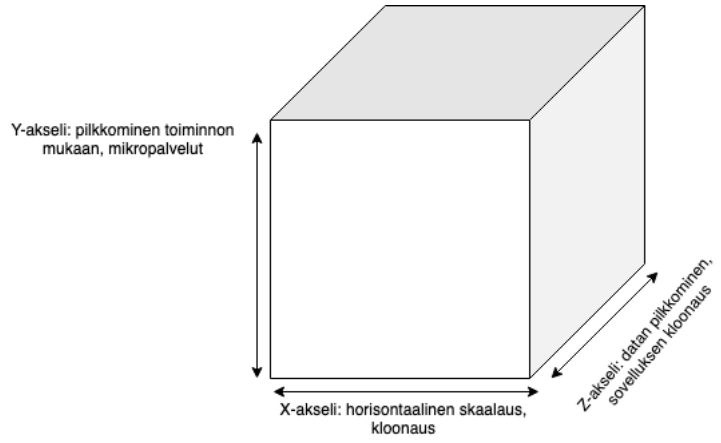
Kaikille ominaisuuksille toteutettiin automatisoidut yksikkötestit sekä Selenium-testit, jotta voidaan varmistaa koodin toimivuus ja muutoksia tehtäessä huomataan heti, jos jotain menee rikki. Testit ajetaan aina automatisoidusti tilaajan CI/CD-putkessa.

4 VAIHTOEHDOT ARKKITEHTUURILLE

Eri vaihtoehtoja mietittäessä täytyi ottaa huomioon sovelluksen vaatimukset, yrityksen resurssit, sekä eri vaihtoehtojen hyvät ja huonot puolet. Ei ole olemassa universaalia oikeaa vaihtoehtoa, joka sopisi yritykselle kuin yritykselle. Vaihtoehtoina olivat monoliittinen tai inventaarionhallinnan eriyttäminen mikropalveluksi. Lisäksi täytyi miettiä, olisiko ollut mahdollista tehdä ominaisuudesta monoliittinen sillä tavalla, että se on tulevaisuudessa helppo irrottaa Mediani.fi-palvelusta omaksi mikropalvelukseen.

4.1 Scale Cube-malli

Scale cube-mallissa sovelluksen skaalaamista verrataan kolmiulotteiseen kuutioon, jossa on X-, Y- sekä Z-akselit. Se auttaa hahmottamaan eri tapoja, joilla sovellusta voi skaalata. X-akseli edustaa horisontaalista skaalaamista eli tässä tapauksessa monoliittista ratkaisua, jossa load balancer jakaa kuormaa kahdennetulle sovellukselle. Y-akseli puolestaan edustaa sovelluksen pilkkomista esimerkiksi funktion tai palvelun perusteella, eli mikropalveluita. Z-akseli taas edustaa samanlaisten palvelujen jakamista eri osiin, vaikka ne ajaisivat samaa koodia, mutta jokainen osa käsittelee tiettyä osaa datasta. (Abbott;ym., 2015)



Kuva 2: Skaalauskuutio

4.1.1 X-akseli: Horisontaalinen skaalaaminen

X-akselilla skaalaaminen on yleisin tapa skaalata sovelluksia. Tämä tarkoittaa käytännössä sitä, että lisätään servereitä, joissa kaikissa pyörii sovelluksen identtinen koodi tai monoliittinen sovellus, joille tulevan kuorman load balancer ohjaa. Skaalaaminen tällä tavalla on helppoa ja nopeaa toteuttaa. Merkittävinä haittapuolina kustannukset voivat kasvaa nopeasti, sekä dataa joudutaan monistamaan useaan paikkaan. Myös välimuistittaminen vaatii enemmän muistia ollakseen tehokasta. (Abbott;ym., 2015)

4.1.2 Y-akseli: Pilkkominen toiminnon mukaan

Y-akselilla skaalaaminen tarkoittaa sitä, että sovelluksen toiminnot pilkotaan useiksi, erillisiksi palveluiksi. Tämä poikkeaa X- ja Z-akselilla skaalaamisesta, joissa ajetaan useita kopioita täysin samasta sovelluksesta. (Abbott;ym., 2015)

On olemassa muutamia tapoja, joilla pilkkominen yleensä toteutetaan; yleensä joko tekemisen mukaan (verbipohjainen erittely), tai asian tai esineen mukaan (substantiivipohjainen erittely). Joskus myös käytetään kummankin yhdistelmää. (Abbott;ym., 2015)

Verbipohjaisessa erittelyssä palvelut muodostetaan tekemisen mukaan, kuten esimerkiksi kirjautuminen. Substantiivipohjaisessa erittelyssä palvelut eritellään asian tai esineen mukaan, kuten esimerkiksi käyttäjien hallinta. (Abbott;ym., 2015)

Y-akseli käytännössä tarkoittaa mikropalveluja.

4.1.3 Z-akseli: Datan jakaminen eri palvelimille

Z-akselilla skaalatessa jokainen palvelin ajaa samaa koodia, kuten X-akselilla, mutta jokainen palvelin on vastuussa vain tietystä osasta dataa. Jotkut järjestelmän komponentit ovat vastuussa siitä, että pyynnöt ohjataan oikealle palvelimelle. Pyynnöt voisivat mennä eri palvelimelle esimerkiksi asiakastyypin mukaan; maksavat asiakkaat ohjataan palvelimelle, jolla on korkeampi SLA ja ilmaisasiakkaat ohjataan palvelimelle, jolla on enemmän kapasiteettia mutta on ehkä hitaampi tai pienemmällä SLA:lla varustettu. (Abbott;ym., 2015)

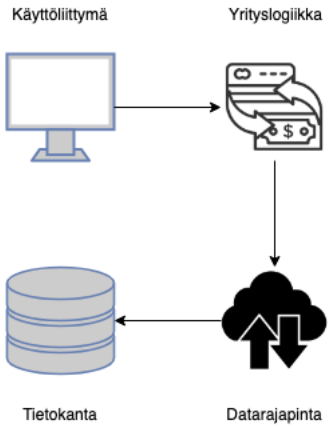
Z-akselin skaalausta käytetään usein tietokantoja skaalatessa. Z-akselin skaalauksen etuja ovat esimerkiksi välimuistin käytön parantaminen ja muistin käytön väheneminen, kun jokainen palvelin tallentaa vain osan data setistä. Transaktiot skaalautuvat paremmin, kun pyynnöt jakautuvat eri palvelimien kesken ja se että data on jaettuna useiden palvelinten kesken, parantaa eristäytymistä esimerkiksi vikatilanteissa. Suurimpia huonoja puolia ovat kompleksisuuden lisääntyminen sekä datan osituksen järkevästi suunnittelemisen vaikeus. (Abbott;ym., 2015)

Z-akselin skaalaus ei tässä vaiheessa palvelun elinkaarta ole kovin tärkeä miettimisen aihe, mutta se on tulevaisuutta ajatellen hyvä ottaa huomioon. Esimerkiksi tiettyjen lisäpalvelujen käyttäjät voisi siirtää isommat SLA:n palvelimille tulevaisuudessa niiden palvelujen osalta.

4.2 Monoliittinen sovellus

Monoliittinen sovellus tarkoittaa sitä, että yksi sovelluskerros tukee käyttöliittymää, yrityslogiikkaa sekä datan manipulaatiota. Data itsessään voi olla tallennettuna muualle, mutta logiikka sen käsittelyyn on osa sovellusta. (Microsoft, 2006) Tilaajan inventaariohallinnan tapauksessa tämä tarkoittaisi sitä, että käyttöliittymään, joka jo nyt on olemassa (Mediani dashboard) lisättäisiin uusi toiminto ”Inventaario”, joka sisältäisi yrityslogiikan kuten esimerkiksi mainospaikan lisääminen ja siihen liittyvät

toiminnot. Tämä data tallennettaisiin jo olemassa olevaan PostGres-tietokantaan, tai sitä varten voitaisiin tehdä uusi tietokanta.



Kuva 3: Yksinkertainen esimerkki monoliittisesta arkkitehtuurista (Laitinen, 2019)

Useat startup-yritykset aloittavat monoliittisestä vaihtoehdosta. Yksi tärkeimmistä syistä siihen on resurssien vähyys ja kiireinen aikataulu. Monoliittisen sovelluksen tekeminen on helpompaa, varsinkin kun kyseessä on pieni tiimi. Kun kyseessä on uusi konsepti, joka luultavasti muuttuu ja elää matkan varrella, on monoliittinen sovellus sopiva nopeaan iterointiin ja proof of concept-tyyppisten sovellusten rakentamiseen. (Lumetta, 2018) Sovelluksen ensimmäisessä vaiheessa on tärkeää todistaa sovelluksen potentiaali, joten yksinkertaisesti ja nopeasti toteutetut versiot ilman mikropalvelujen tuomaa lisätaakkaa ovat siihen tärkeitä.

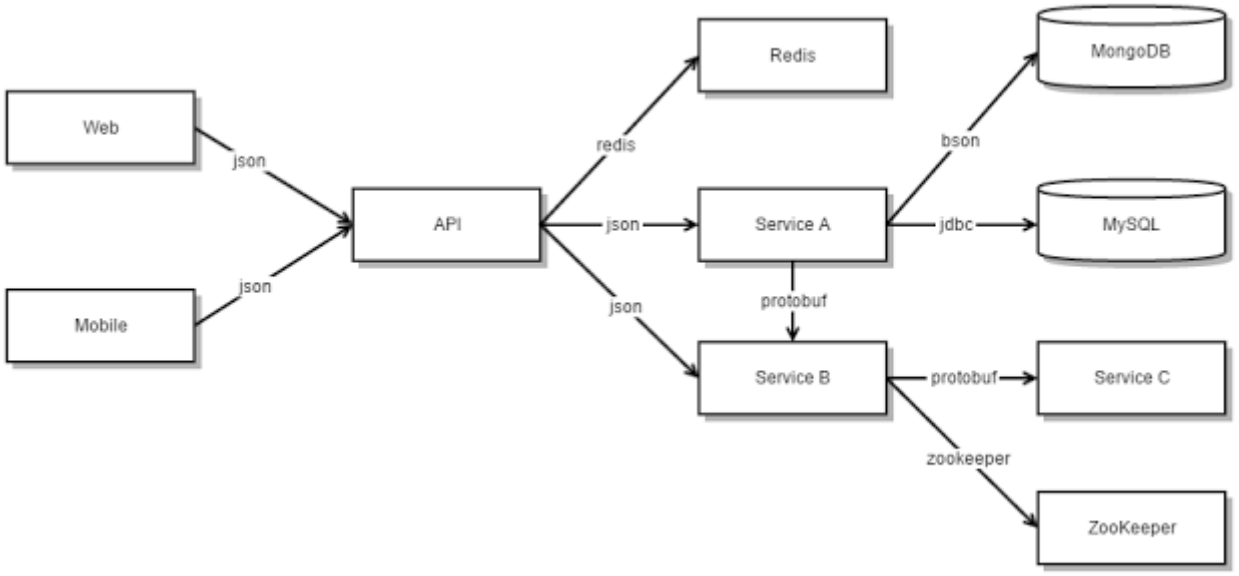
Monoliittiseen sovellukseen kattavan testauksen toteuttaminen on helpompaa. Testaamista varten sovelluksen täytyy vain olla käynnissä, ja sen jälkeen esimerkiksi käyttöliittymää on helppo testata Selenium-testeillä. Myös sovelluksen käyttöönotto on helpompaa, koska sovellus kokonaisuudessaan kopioidaan palvelimelle. Horisontaalisesti skaalaaminen on myös helpompaa, kun sovelluksesta voidaan ajaa useita kopioita load balancerin takana.

Monoliittisella sovelluksella on myös paljon haittapuolia. Sovelluksen koko ja monimutkaisuus kasvavat helposti todella vaikeiksi hahmottaa, ja siitä aiheutuu helposti se, että muutoksien tekeminen hidastuu ja niissä tehtävien virheiden todennäköisyys kasvaa. Tästä aiheutuu se, että manuaalista testaamista tarvitaan paljon, koska muutoksien vaikutuksia on vaikea hahmottaa. Vaikka monoliittinen sovellus onkin helppo ottaa käyttöön, tarkoittaa se myös sitä, että kun muutoksia tehdään, täytyy koko sovellus aina kopioida palvelimelle uudestaan, ja se on hitaampaa kuin vain tietyn osan sovellusta kopioiminen. Jatkuva käyttöönotto onkin tästä johtuen vaikeampaa.

Sovelluksen skaalaamista voi vaikeuttaa se, että eri moduuleilla saattaa olla ristiriitaisia resurssivaatimuksia. Virhe yhdessä moduulissa voi tarkoittaa sitä, että koko sovellus kaatuu. Uusien teknologioiden käyttöönotto on haastavaa. (Kharenko, 2015)

4.3 Mikropalvelu

Mikropalvelulla tarkoitetaan sitä, että joku tietty osa sovelluksesta irtaannutetaan omaksi sovellukseksen, joka kommunikoi sovelluksen tai toisten mikropalvelujen kanssa. Mikropalvelussa on oma arkkitehtuurinsa ja yrityslogiikkansa. Mikropalvelu voi esimerkiksi julkaista REST-rajapinnan, jota toinen mikropalvelu, jossa on www-käyttöliittymä käyttää hyväkseen. Yleensä kun halutaan ottaa kaikki hyöty irti mikropalvelusta, jokaisella mikropalvelulla on oma tietokantaskeemansa, jotta palvelut ovat irtonaisia toisistaan. Tämä johtaa usein siihen, että osa datasta kahdentuu. Lisähyötynä tästä seuraa se, että voidaan käyttää useita erilaisia tietokantoja monikielisen pysyvyyden arkkitehtuurin (”polyglot persistance architecture”) mukaan, eli kun joku mikropalvelu hyötyisi esimerkiksi MongoDB:n tyylisestä NoSQL-tietokannasta, voidaan käyttää sitä, kun taas joku toinen mikropalvelu käyttää PostgreSQL-tietokantaa.



Kuva 4: Esimerkki mikropalvelun arkkitehtuurikaavasta (Kharenko, 2015)

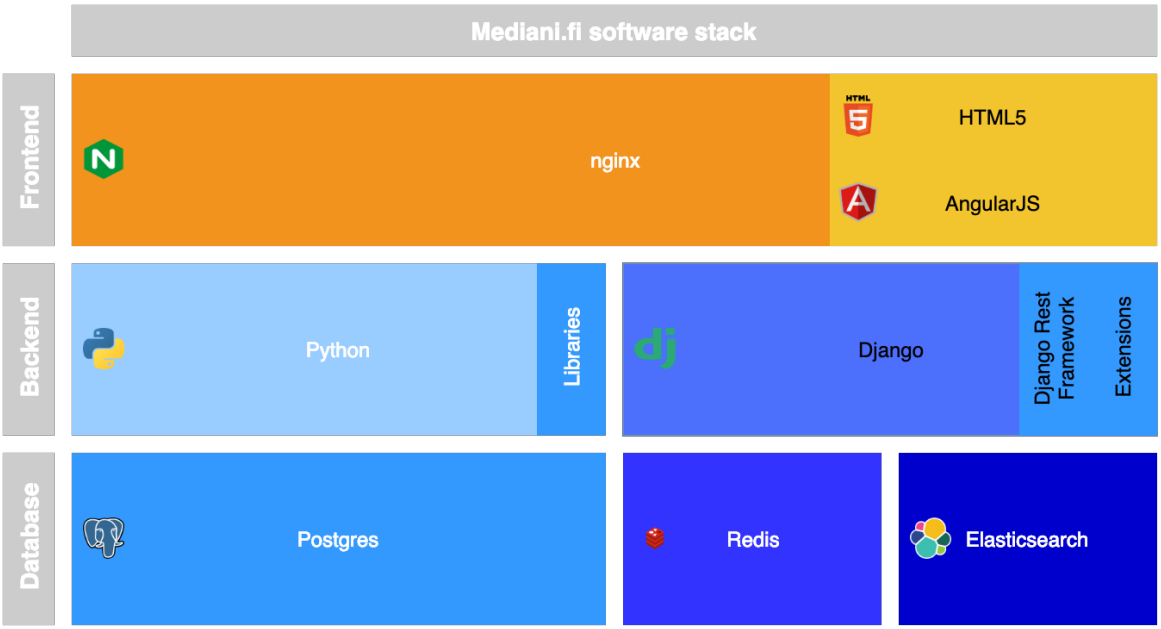
Mikropalvelun arkkitehtuuri skaalautuu ”Scale Cube”-mallin mukaan Y-akselilla, eli toiminnollisuuksia pilkkomalla. Jotkut rajapinnat voivat olla avoimia vain mobiililaitteille, toiset web-sovelluksille, ja toiset työpöytäsivustoille sekä sovelluksille. Sovellukset kommunikoivat API Gatewayn kautta, joka hoitaa esimerkiksi load balancing, välimuisti sekä monitorointitehtäviä.

Yksi mikropalveluiden parhaita puolia on se, että sovelluksen hallinta ja huolto helpottuu, kun sovellus on pilkottu mikropalveluihin. On helpompaa ymmärtää, mitä sisäänkirjautumispalvelu tekee, kuin etsiä sisäänkirjautumistoiminnallisuutta monoliittisen sovelluksen syövereistä. Koska jokaista toimintoa vastaa oma mikropalvelunsa, voidaan yhtä mikropalvelua kehittää itsenäisesti helpommin, ja sille on helpompi määrätä oma kehitystiiminsä. Uusien teknologioiden käyttöönotto helpottuu, kun niitä ei tarvitse tuoda käyttöön heti koko sovelluksen laajuisesti, vaan yksi mikropalvelu voi käyttää jotain tiettyä teknologiaa. Avoimen lähdekoodin ympäristöissä tämä on erittäin toivottua, koska teknologiat kehittyvät nopeasti ja uusia tekniikoita syntyy vuosittain lisää. Jatkuva deployaaminen helpottuu, kun yksittäisiä mikropalveluita voidaan deployata silloin kun on tarve, eikä koko sovellusta tarvitse

päivittää kerralla. Koko sovelluksen päivitys voi viedä esimerkiksi 20 minuuttia, mutta pelkän kirjautumispalvelun päivittämisessä voi mennä vain pari minuuttia. Koska mikropalvelu skaalautuu scale cube-mallin Y-akselilla, siitä seuraa myös se, että voidaan helposti skaalata vain yhtä palvelua kerrallaan.

Mikropalvelu arkkitehtuurin huonoja puolia on esimerkiksi se, että koska se on hajautettu järjestelmä, se lisää merkittävästi kompleksisuutta kokonaisuuteen. Tietokantojen päivittämiset voivat olla hajautettuja transaktioita, joka lisää kehittämisen haastavuutta. Testien kirjoittaminen vaikeutuu, koska testaamisessa täytyy ottaa huomioon se, että yhtä palvelua testattaessa saatetaan tarvita useita muita palveluita, joiden täytyy olla käytössä testien suorittamisen aikana. Isoja haasteita aiheuttaa myös sellaisten muutosten teko, jotka vaikuttavat useampaan palveluun. Mikropalveluarkkitehtuuria käytettäessä on suunniteltava ja koordinoitava muutokset jokaiseen palveluun tarkasti. Kommunikaatio eri tiimien välillä on toimittava hyvin, ettei pääse tapahtumaan virheitä. Tämä pätee myös sovelluksen käyttöönottoon, joka on monimutkaisempaa monoliittiseen palveluun verrattuna. Käytännössä kaikki käyttöönottoon liittyvä on syytä automatisoida, koska manuaalisesti asioiden tekeminen mikropalveluarkkitehtuurissa ei ole käytännöllistä.

5 KÄYTETYT TEKNIIKAT



Kuva 5: Mediani.fi sovelluspino (Laitinen, 2019)

5.1 Python

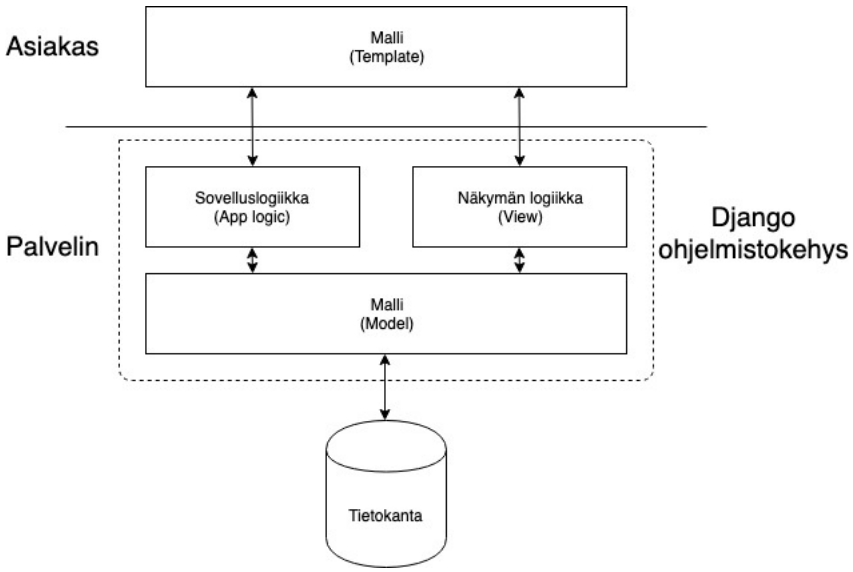
Python on tulkattava kieli, joka tarkoittaa, ettei sitä tarvitse kääntää ennen suorittamista. Tämä tarkoittaa sitä, että ohjelmointia ja testaamista voi tehdä lyhyissä sykleissä nopeasti mutta haittapuolena suorituskyky kärsii. Python tukee useita eri ohjelmointiparadigmoja, mukaan lukien olio-ohjelmointi, funktionaalinen ohjelmointi sekä proseduraalinen ohjelmointi. Python on avoimen lähdekoodin ohjelmisto, jolla on yhteisöpohjainen kehitysmalli. Pythonin kehitystä ohjaa voittoa tavoittelematon Python Software Foundation. (Python Software Foundation)

Pythonista käytettiin versiota 2.7, koska se on vakiintunut yrityksen käyttöön. Jotkut käytössä olevat paketit eivät ilmeisesti täysin ole yhteensopivia Python 3:n kanssa, ja koska resurssit ovat rajalliset niitä ei ruvettu itse kääntämään Python 3:lle. Lisäksi tähän vaikutti paljon se, että Python 2 on ohjelmoijille paljon tutumpi.

5.2 Django

Django on ohjelmistokehys, joka kannustaa nopeaan kehitykseen ja selkeään, pragmaattiseen suunnitteluun. Djangon tarkoitus on yksinkertaistaa verkkosovellusten kehittämistä, jotta ohjelmoija voi keskittyä enemmän yrityslogiikan tekemiseen. (Django Software Foundation)

Django perustuu ”Model-Template-View”-malliin (”MTV”), joka on käytännössä sama asia kuin Model-View-Controller-malli (MVC) mutta kontrolleria (Controller) kutsutaan näkymäksi (View) ja näkymää (View) kutsutaan malliksi (Template). Tätä perustellaan sillä, että Django Software Foundationin mielestä MVC-mallin ”view” kuvaa sitä miten data esitetään käyttäjälle, eikä se välttämättä tarkoita sitä miltä data oikeasti näyttää. (Django Software Foundation)



Kuva 6: Djangon MTV-arkkitehtuuri (The Django Book)

Django oli vuonna 2018 Python-ohjelmoijien keskuudessa toiseksi suosituin verkko-ohjelmistokehys, häviten Flaskille kahdella prosentilla (47% Flask, 45% Django) (Jetbrains)
Django valittiin projektiin, koska se oli vakiintunut käyttöön yrityksessä.

5.3 Django REST Framework

Django REST Framework (DRF) on työkalu verkkorajapintojen rakentamiseen. Nimensä mukaisesti se on erityisesti suunniteltu Djangon kanssa käytettäväksi. DRF yksinkertaistaa rajapintojen kehittämistä huomattavasti ja tarjoaa muun muassa selaimella selattavat rajapinnat, autentikaation toteutuksen, jossa on esimerkiksi OAuth2-tuki, serialisoinnin, joka tukee sekä ORM- ja ei-ORM datalähteitä, sekä paljon muuta. (Encode OSS Ltd.)

5.4 PostgreSQL tietokanta

PostgreSQL on ilmainen, avoimen lähdekoodin olio-relaatiotietokanta, joka käyttää laajennettua SQL-kieltä. PostgreSQL on ollut ACID-yhteensopiva (Atomicity, Consistency, Isolation, Durability) jo pitkään. (PostgreSQL Global Development Group)
PostgreSQL valittiin, koska siitä oli yrityksen työntekijöillä eniten kokemusta. Ei ole oikeastaan syytä, miksei tietokanta olisi voinut olla esimerkiksi MySQL.

5.5 AngularJS

AngularJS on avoimen lähdekoodin Javascript-ohjelmistokehys, jonka tavoitteena oli tuoda selainpohjaisiin sovelluksiin MVC-mallin tuki. Kirjasto perustuu siihen, että HTML-elementteihin laitetaan ylimääräisiä attribuutteja, jotka määrittelevät sen mitä toiminnallisuutta elementiltä halutaan. Attribuutin mukainen operaatio liitetään mallin mukaisesti muuttujiin liitettynä. AngularJS:llä ei tarvita päivittää DOM:ia, rekisteröidä kutsuja (callback) tai seurata mallin muutoksia. (Google)

AngularJS on nykyään LTS-periodissa (Long Term Support) ja sen tuki loppuu kokonaan vuonna 2021. (Google) AngularJS:n korvaaja on Angular (angular.io). Syy siihen, miksi päädyimme käyttämään AngularJS:ää on se, että muu sovellus käyttää myös AngularJS:ää. Tarkoituksena on siirtyä myöhemmin käyttämään Angularia, tai jotain muuta vastaavaa Javascript-kirjastoa, mutta päätöstä uudesta kirjastosta ei ole vielä saatu tehtyä. Koko sovellus siirretään luultavasti kerralla käyttämään uutta tekniikkaa, kunhan sen aika on.

5.6 Muut tekniikat

5.6.1 Ansible

Ansible on avoimen lähdekoodin ohjelmisto, jonka tarkoituksena on automatisoida sovellusten käyttöönottoa ja siten nopeuttaa iterointia. Ideana on muuntaa tehtävät ”pelikirjoiksi” (playbook) joita voidaan toistaa. Ansiblella on oma deklaratiivinen kielensä, jolla pelikirjat kirjoitetaan. (Red Hat Inc.)

5.6.2 Redis

Redis on avoimen lähdekoodin muistissa sijaitseva datastruktuuri varasto. Sitä käytetään tietokantana, välimuistina (cache) tai viestin välittäjänä (message broker). (Sanfilippo) Sovellus käyttää Redistä välimuistina, jonne tallennetaan pääasiassa merkkijonoja ja avain-arvo-pareja.

5.6.3 Gitlab CI/CD

CI/CD:llä tarkoitetaan jatkuvaa integraatiota, toimitusta ja käyttöönottoa (Continuous Integration, Continuous Delivery, Continuous Deployment). Käytännössä se tarkoittaa siis sitä, että kun koodi ladataan säilöön (Gitlab CI/CD tapauksessa Gitlabiin), tehdään koodin kääntäminen ja testaus auto-

maattisesti heti. Sen jälkeen sovellus voidaan julkaista automaattisesti tuotantoon milloin vain (jatkuva toimitus). Jos halutaan, voidaan sovellus pistää automaattisesti tuotantoon asti (jatkuva käyttöönotto).

Gitlabin omia CI/CD työkaluja voi käyttää ilmaiseksi kuka vain ja se on osa Gitlabin sovellusta. Gitlab tarjoaa käyttöön ilmaisia koneita, joilla työt voidaan ajaa, tai voidaan käyttää myös omia palvelimia kuten tilaaja käyttää. (Gitlab)

5.7 Työkalut

Pääasiassa työkalut, joita projektissa ovat olleet käytössä rajoittuvat ilmaisiin tekstieditoreihin ja joihinkin verkkopohjaisiin työkaluihin kuten draw.io. Chromen sisäänrakennetut kehittäjätyökalut ovat olleet kovassa käytössä. Postman, joka on sovellus rajapintojen testaamiseen on myös osoittautunut toimivaksi työkaluksi rajapintoja rakennellessa.

5.7.1 Visual Studio Code

Visual Studio Code on Microsoftin kehittämä, ilmainen, avoimen lähdekoodin tekstieditori. Se tukee yli 50:tä eri ohjelmointi-, merkintä- ja määrityskieltä suoraan ilman laajennuksia, muun muassa C, C++, C#, Java, Python, ja niin edelleen. Lisäksi laajennuksilla voidaan lisätä tuettuja kieliä. (Microsoft)

Visual Studio Code tukee muun muassa virheenkorjausta, Git-versiohallintaa, automaattista koodin täydennystä (IntelliSense), refaktorointia sekä syntaksin korostusta. Laajennuksilla on helppo lisätä puuttuvia ominaisuuksia. (Microsoft)
Visual Studio Code on saatavilla MacOS:lle, Linuxille ja Windowsille.

6 LOPPUTULOS

Lopputuloksena syntyi suunnitelma siitä, miten inventaarionhallintapalvelua lähdettiin toteuttamaan. Lisäksi saatiin tehtyä ensimmäinen beetaversio inventaarionhallinnasta, joka on nyt jo asiakkaitten testikäytössä tuotantopalvelimella.

6.1 Valittu arkkitehtuuri

Valinta kohdistui lopulta monoliittiseen arkkitehtuuriin, joka tuntui luonnolliselta valinnalta. Seuraavissa kappaleissa tarkemmat perustelut asialle.

6.1.1 Aikarajoitukset

Mikropalvelun suunnitteleminen ja toteuttaminen olisi vienyt enemmän aikaa, ja se olisi tehnyt testaamisesta vaikeampaa. Olisi tullut potentiaalisesti tarve eriyttää nykyisiä sovelluksen toimintoja, kuten kirjautuminen ja tunnusten hallinta omaksi palvelukseen, joka olisi vienyt taas lisää aikaa suunnittelun, toteutuksen ja nykyisen koodin refaktoroinnin muodossa.

Tiukoista aikatauluista johtuen, päädyttiin aikataulujen suhteen suosimaan monoliittista arkkitehtuuria.

6.1.2 Tiimin koko, resurssit

Mikropalvelu arkkitehtuuri tuo hyötyjä kehittämiseen paljon, jos kyseessä on iso joukko kehittäjiä, jotka voidaan jakaa selkeisiin tiimeihin, ja tiimi kehittää pääasiassa yhtä kokonaisuutta, tilaajan tapauksessa esimerkiksi yksi tiimi voisi kehittää hakua, tuotteita sekä käyttäjähallintaa ja toinen tiimi voisi kehittää pelkästään inventaarionhallintaa. Tämän lisäksi, kun aikataulut ovat tiukat, tiimien kaikilla jäsenillä olisi jo hyvä ennestään olla kokemusta mikropalveluiden kehittämisestä, koska ne tuovat monimutkaisuutta kokonaisuuteen.

Resurssien ja työntekijöiden vähyyden, sekä sen takia että puolella ohjelmoijista ei ole mikropalveluista kokemusta, päädyttiin tässä kategoriassa suosimaan monoliittista arkkitehtuuria. Mikropalvelut eivät toisi kehitykseen selkeää jakoa tiimeihin tässä vaiheessa, ja vain hidastaisivat kehitystä.

6.1.3 Kompleksisuus

Mikropalvelu arkkitehtuuri tuo sovellukseen kompleksisuutta, joka tässä vaiheessa vain hidastaisi kehitystyötä.

6.1.4 Nopea iterointi

Monoliittisen arkkitehtuurin sovellusta on nopeampi iteroida ja tuoda uusia ominaisuuksia testattavaksi. Tässä vaiheessa yrityksen elämää sovellus muuttuu ja kehittyy, kun yritys hakee vielä identiteettiään. Esimerkiksi inventaarionhallinta ylipäätään ydintoimintona tuli melko nopeasti eteen,

vaikka sitä olikin jo mietitty, että sellainen voisi olla hyödyllistä tehdä jossain vaiheessa. Kuitenkin nopeasti huomattiin, että tämä voisi olla sellainen ominaisuus, josta yritykset ovat valmiita maksamaan ja joka tukee palvelun muuta toiminnallisuutta ja tulevaisuuden visiota, joten se päätettiin ottaa työn alle.

6.1.5 Skaalautuvuus

Scale cube-mallin mukainen, X-akselilla tapahtuva horisontaalinen skaalaaminen ei ole optimaalisin mahdollinen tapa tehdä asioita, kun on kyseessä Medianin tyyppinen palvelu. Kuitenkin skaalaaminen tulevaisuudessa on mahdollista niinkin. Y-akselin skaalaaminen olisi ollut todella paljon tehokkaampi tapa toimia, koska on mahdollista, että inventaarionhallinta tarvitsee enemmän kapasiteettia, ja olisi ollut todella hyvä, jos olisi ollut mahdollista skaalata vain inventaarionhallintamikropalvelua.

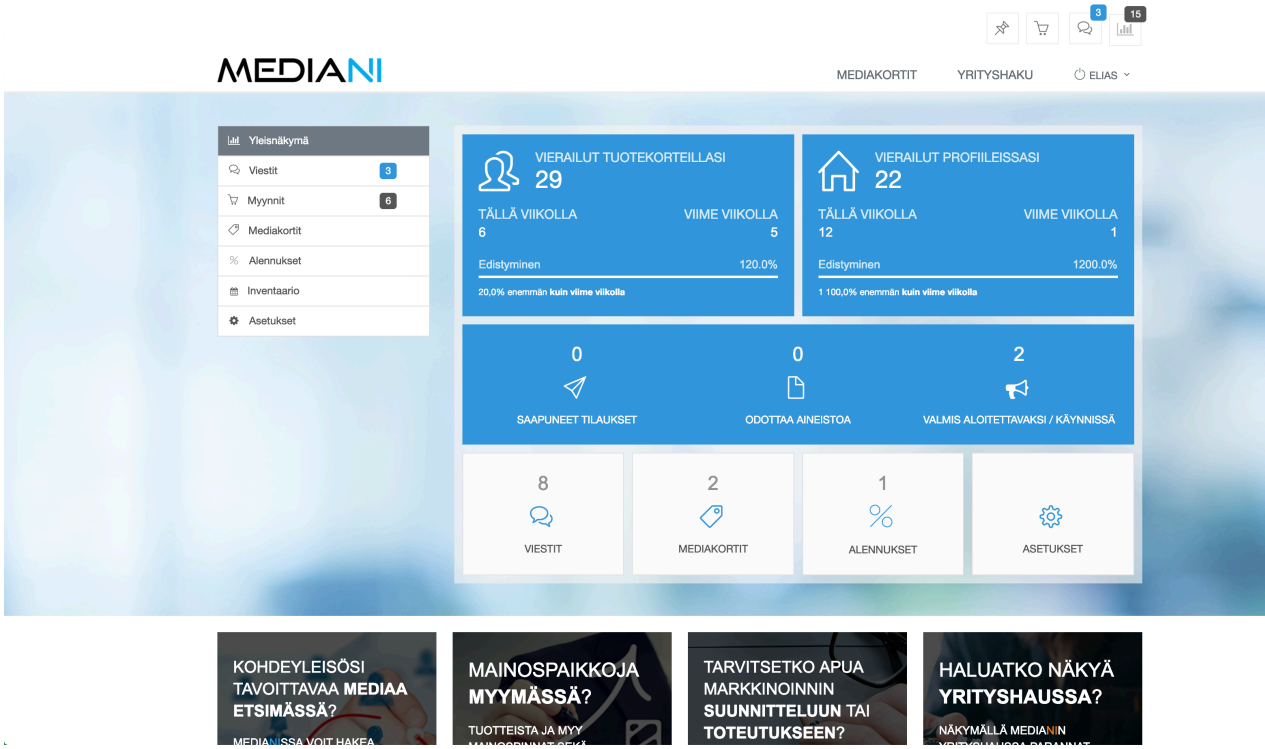
Z-akselilla skaalaaminen on myös potentiaalinen vaihtoehto tulevaisuudessa. Tietokantoja inventaarionhallinnan osalta voitaisiin siirtää uusille palvelimille. Käytännössä alussa katsotaan miten palvelimet ja skaalautuminen riittävät, ja sitten mietitään, olisiko tarvetta joillekin muille toimille.

6.1.6 Miten jatkossa?

Jatkossa sovellusta kehitetään monoliittisena, ja mitä todennäköisemmin kaikki uudet ominaisuudet toistaiseksi tehdään monoliittisella arkkitehtuurilla. Kuitenkin toiminnallisuus inventaarionhallinnan, kuten tulevien ominaisuuksienkin, tapauksessa koitetaan tehdä siten että tulevaisuudessa ne ovat helppoja irrottaa omiksi mikropalveluikseen – tämä siitä syystä, että tulevaisuudessa jos ja kun yritys kasvaa, on työnjako ja tiimien muodostaminen helpompaa.

6.2 Mediani.fi Dashboard

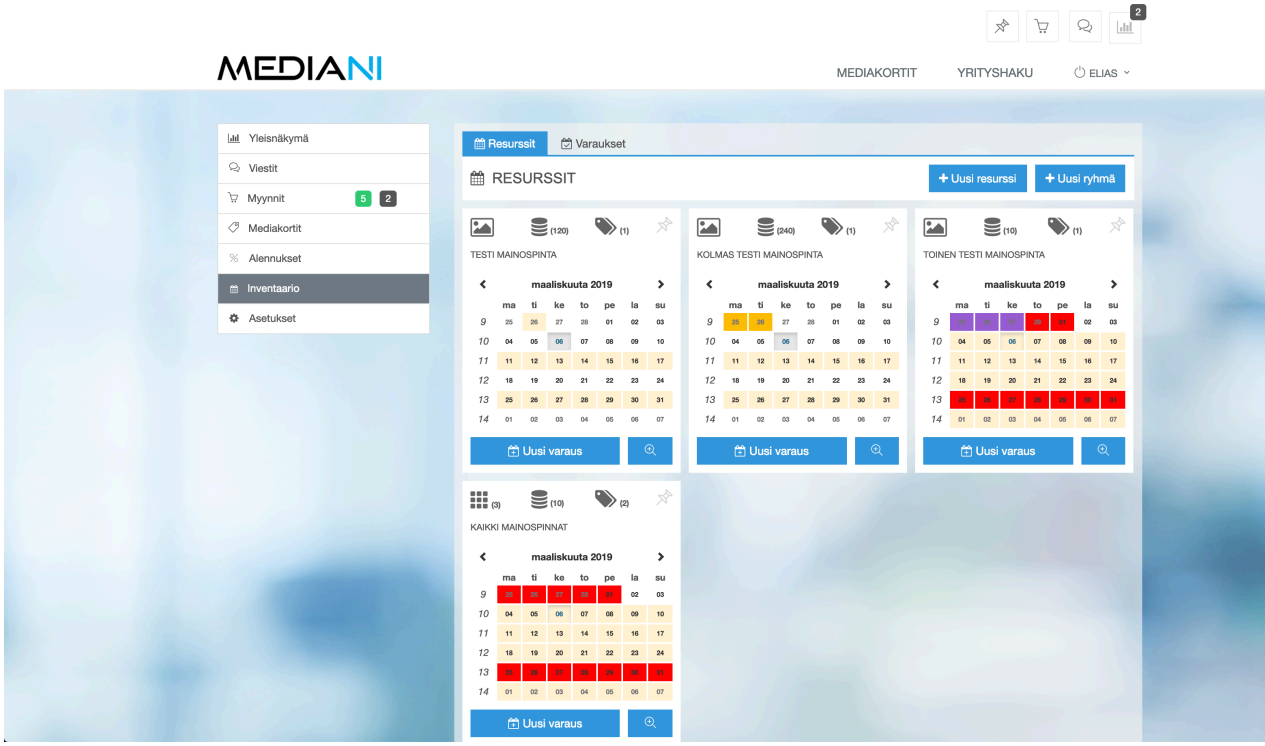
Mediani Dashboard on kirjautuneen käyttäjän yleisnäkymä, josta käyttäjä näkee yhdellä vilkaisulla perustietoja, sekä löytää linkit erilaisiin hallintatyökaluihinsa kuten myynnit, mediakortit, alennukset sekä inventaario. Näkymä vaihtelee sen mukaan, onko käyttäjä ostaja, mainostoimisto vai mediamyyjä.



Kuva 7: Dashboard, kirjautuneen mediamyyjän näkymä

Myöhemmin tähän näkymään on tarkoitus lisätä jotain perustietoja inventaarion tilanteesta, kuten esimerkiksi koko inventaarion keskimääräinen varaustilanne prosentuaalisesti.

6.3 Inventaarionhallinta yleisnäkymä



Kuva 8: Inventaarionhallinnan yleisnäkymä, kaikki resurssit

Inventaarionhallinnan yleisnäkymässä näytetään kaikki luodut resurssit. Beeta-versiossa resursseja ei vielä sivutettu näkymässä, vaikka rajapinta sitä jo tukeekin, mutta sivutus tulee olemaan tarpeen koska yrityksillä voi olla tuhansia resursseja.

Resurssilla tarkoitetaan esimerkiksi Led-näyttöä kaupungin keskustassa, bannerimainosta verkkosivustolla tai vaikka lehden etusivua. Resursseista lisää kappaleessa 6.4.

Resurssien lisäksi tässä näkymässä näytetään myös ryhmät. Ryhmiä ei ollut tarkoitus toteuttaa vielä tähän versioon, ja ne eivät olekaan vielä edes testipalvelimella, mutta ominaisuus on käytännössä viimeistelyä vaille valmis. Myös ryhmistä lisää kappaleessa 6.4.

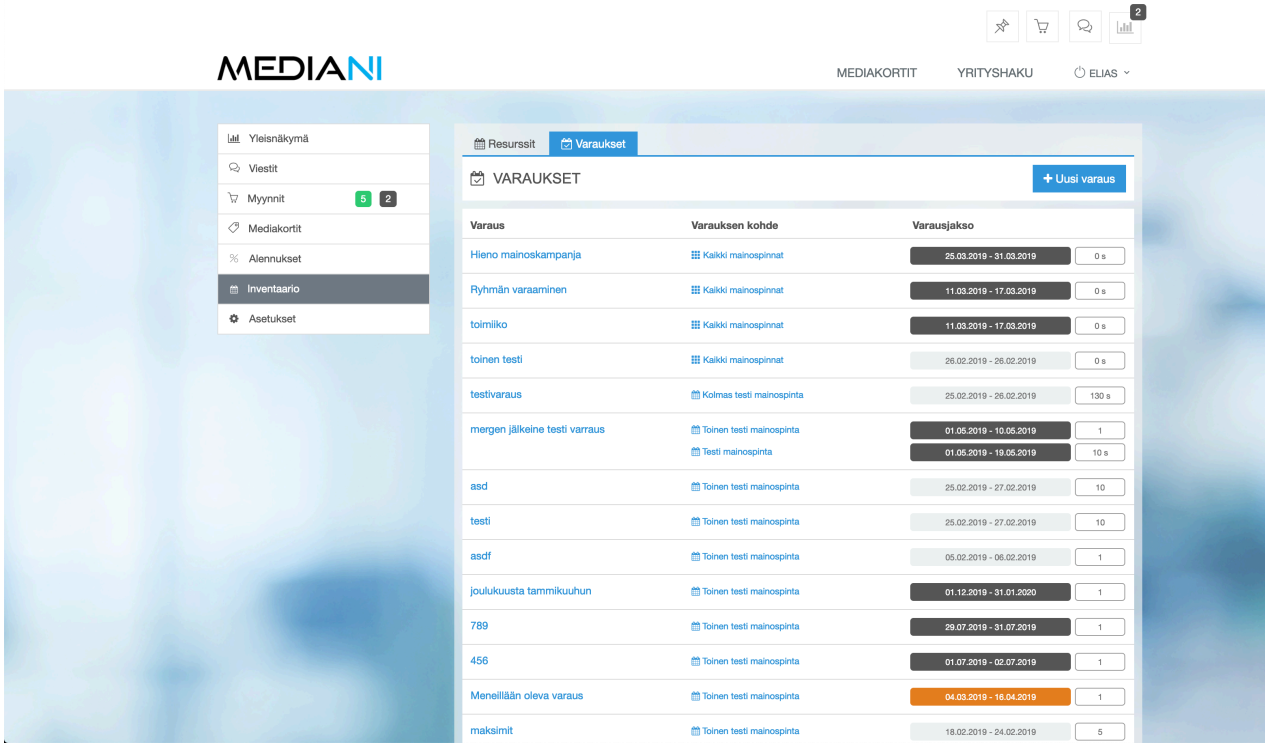
Kalentereissa näkyvät värit antavat indikaatiota päivittäisestä varaustilanteesta. Punainen tarkoittaa 100% varausta, ja violetti tarkoittaa yli 100% varausta, niin sanottua ”ylibuukkausta”. Tätä ominaisuutta toivottiin asiakkaitten suunnasta, koska joissain tilanteissa ylibuukkaus on mahdollistettava.

Koska varaustilanteiden lataaminen on suhteellisen kuormittava operaatiopalvelimelle, toteutettiin se siten, että kun varauksia tai resursseja luodaan, tallennetaan se Redikseen välimuistiin, jottei varaustilanteita ja prosentteja tarvitse laskea joka kerta kalenteria näytettäessä uudelleen vaan se voidaan hakea suoraan Rediksestä valmiiksi laskettuna. Tällä saatiin nopeutettua resurssien yleisnäkymää huomattavan paljon, kuten myöskin kappaleessa 6.6 käsiteltävää resurssin tietojen tarkastelusivua.

Seuraava kehitysvaihe tällä sivulla sisältää hakutoiminnallisuuden resursseista sekä muita resurssien selaamiseen liittyviä parannuksia.

6.3.1 Varauslista

Varauslista-näkymässä näytetään yksinkertaisesti kaikki varaukset listana.



Kuva 9: Varauslista-näkymä

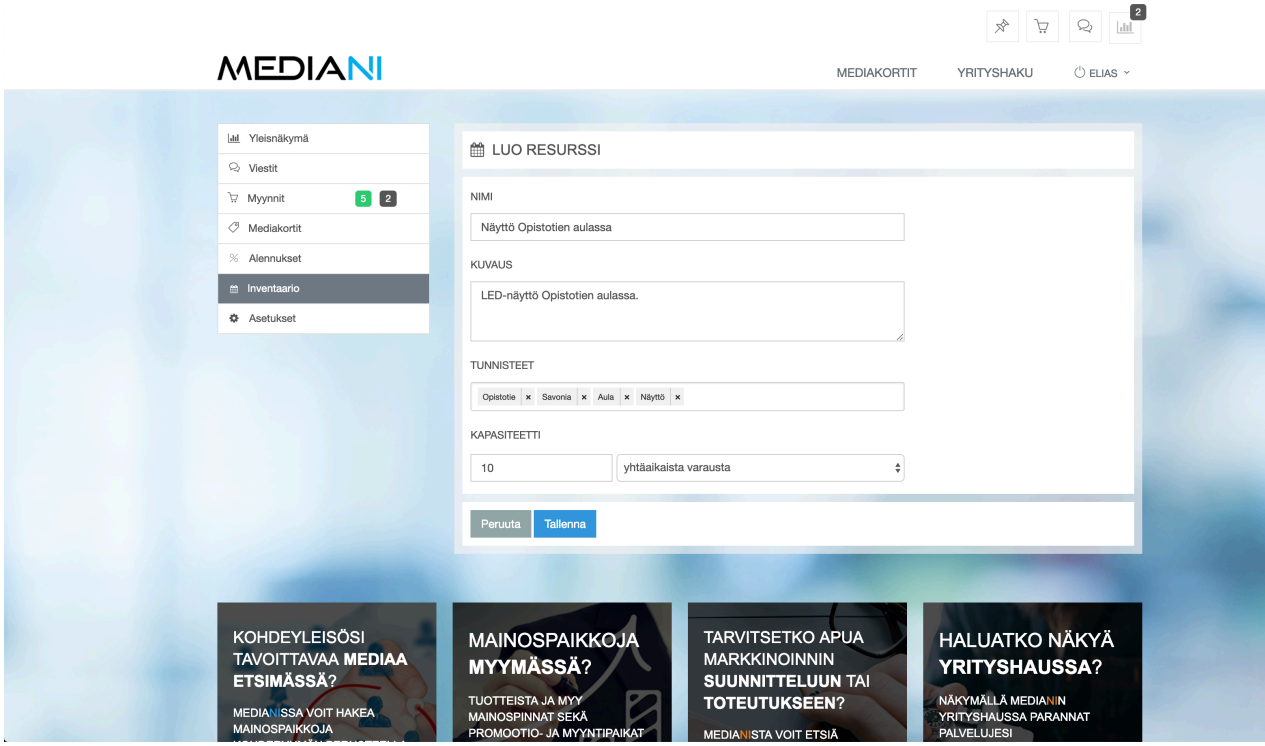
Näkymässä näytetään varauksen nimi, varatut kohteet (ryhmä ja/tai resurssit), varattujen kohteiden varausjaksot sekä kokonaiskapasiteetti, jota kohteesta on varattu. Nopean tilannekatsauksen saamiseksi tulossa olevat, meneillään olevat sekä jo menneet varatut kohteet näkyvät eri väreillä; tumma on tulossa oleva, oranssi meneillään oleva ja vaalea on jo mennyt.

Jatkossa tähänkin näkymään tulee parannuksia ainakin haun ja suodattimien muodossa. Näistä lisää kappaleessa 6.7.

6.4 Uuden resurssin luonti

Uuden resurssin luontiin pääsee inventaarion yleisnäkymästä. Uutta resurssia luotaessa sille annetaan nimi, kuvaus, tunnisteita (tulevia ominaisuuksia kuten haku ja suodattaminen varten), kapasiteetti sekä kapasiteetin yksikkö. Kapasiteetin yksikkö on joko sekunti tai ”yhtäaikaista varausta”. Esimerkiksi sanomalehdessä voi olla viisi tietyn kokoista mainospaikkaa, jolloin myyjä voi asettaa kapasiteetiksi viisi ja yksiköksi yhtäaikaista varausta, tai Led-näytössä voi olla 360 sekunnin looppi, jota toistetaan jatkuvasti, jolloin voidaan asettaa kapasiteetiksi 360 ja yksiköksi sekunti.

Tunnisteet näkyvät tuotteella, mutta niiden varsinaiset toiminnallisuudet lisätään myöhemmin. Niiden avulla voidaan esimerkiksi suodattaa tai hakea resursseja yleisnäkymässä. Tunnisteet tallennetaan tietokantaan, kun ne kirjoitetaan kenttään ensimmäistä kertaa, ja sen jälkeen ne voidaan tulevissa tuotteissa valita alasvetovalikosta, joka aukeaa, kun kenttää napsautetaan.



Kuva 10: Uuden resurssin luominen

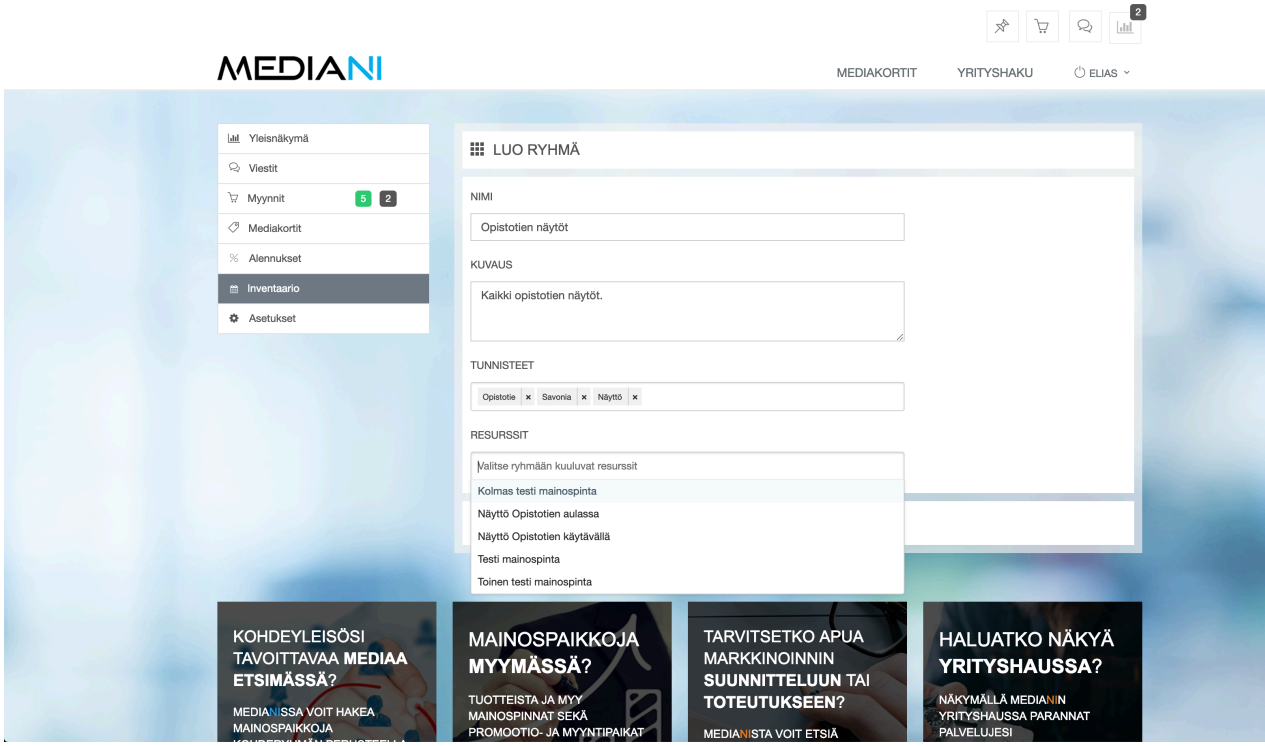
6.4.1 Uuden ryhmän luonti

Uuden ryhmän luonti käyttää hyväkseen samaa templatea kuin uuden resurssin luontikin, mutta kapasiteetin valinta korvataan resurssien valinnalla. Resurssit-kenttää napsauttamalla aukeaa lista kaikista yrityksen resursseista, joita voidaan lisätä niitä klikkaamalla. Kenttään kirjoittaminen etsii kirjoitettua tekstiä resurssien nimien joukosta.

Ryhmä käyttäytyy samalla tavalla kuin yksittäinen resurssi, ja se näkyy esimerkiksi resurssien yleisnäkymässä samaan tapaan kuin resurssi. Kun ryhmään tehdään varauksia, ne tehdään jokaiseen ryhmälle määritettyyn resurssiin. Varauksien tekemisestä lisää kappaleessa 6.5.

Koska asiakkailla saattaa olla tuhansia mainospintoja, ja ne ovat saatavilla usein ryhminä, oli ryhmien ja ryhmävarausten tekeminen mahdollisimman nopeasti tärkeää. Esimerkiksi yrityksellä saattaisi olla 5 näyttöä Savonialla Opistotiellä, 5 Savonialla Varkaudessa, 5 Microkadulla. Tällöin yritys voisi tehdä ryhmän ”Kaikki Savonian näytöt”, jossa olisi kaikki 15 näyttöä, ja yritys voisi myös tehdä ryhmät ”Opistotien näytöt”, ”Varkauden näytöt”, sekä ”Microkadun näytöt”, joitten kaikkien vapaa kapasiteetti ja saatavuus määräytyvät ryhmissä olevien resurssien mukaan. Edelleen myös yksittäisen resurssin varauksen tekeminen on täysin mahdollista.

Ryhmien käyttäminen ei ollut vielä ensimmäisen beetaversion tavoitteissa, mutta ominaisuus tehtiin koska se täytyi saada valmiiksi nopealla aikataululla. Opinnäytetyön kirjoitushetkellä (7.3.2019) ryhmät eivät ole vielä testi- eikä tuotantopalvelimella, vaikka toiminnallisuus on kirjoitettu, johtuen siitä, että kattavien testien kirjoittaminen on vielä kesken ja yhdistämispyyntö Gitlabissa odottaa hyväksyntää.



Kuva 11: Uuden ryhmän luonti, resurssien lisääminen

6.4.2 Resurssin tai ryhmän muokkaus

Resurssin muokkaus käyttää samaa templatea kuin uuden resurssin luonti sekä uuden ryhmän luonti. Myös ryhmän muokkaus tapahtuu samassa templatessa. Sivulle ohjataan käyttäen erilaista url-polkua, jossa annetaan ensin resurssin tyyppi (”resource” tai ”group”) ja sen jälkeen avainarvo UUID:nä. Alla oleva rivi inventaarionhallinnan Django-moduulin url-tiedostossa hoitaa asian.

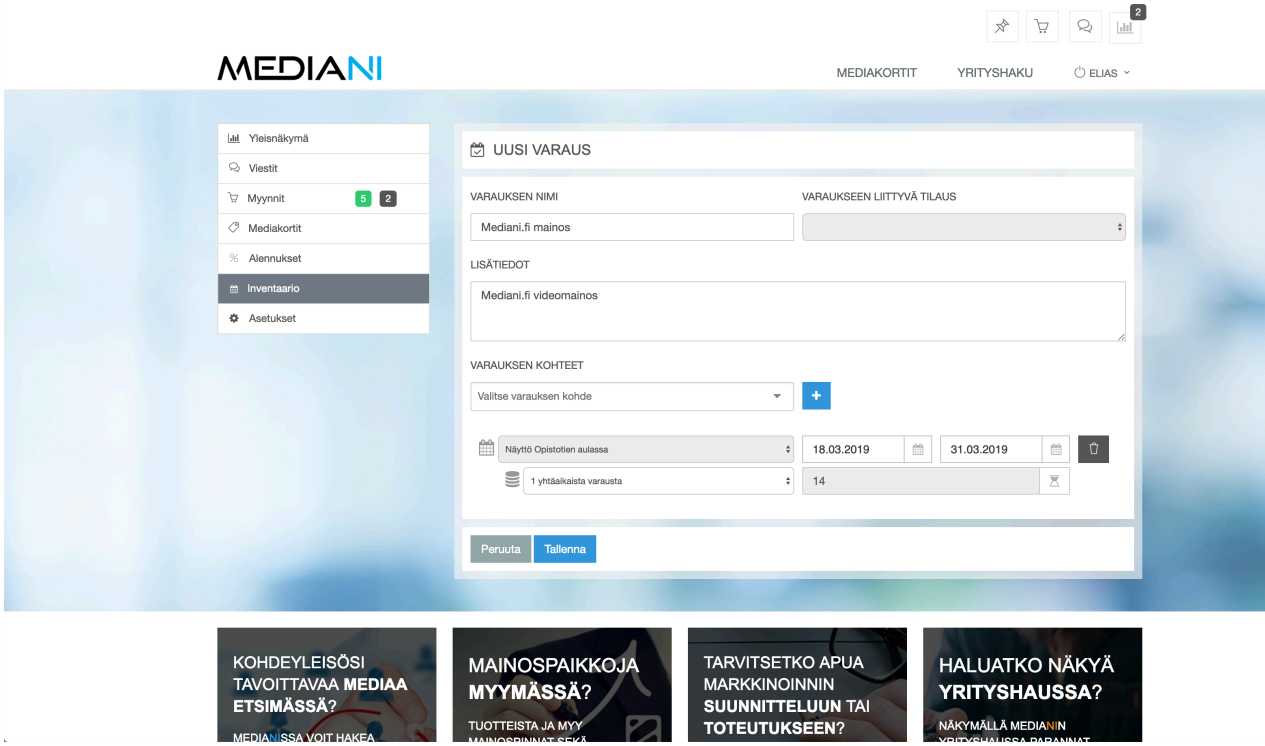
```
url(r'^dashboard/inventory/resource/(?P<type>[-\w\d]+)/(?P<pk>[0-9a-f-]+)/$',
views.InventoryResourceCreateEditView.as_view(), name="inventory_resource_edit"),
```

Kuten nähdään, rivillä määritellään ensin osoitteen alku (*”dashboard/inventory/resource/”*), sitten tyyppi (*”?P<type>”*), sallitaan käyttää kirjaimia säännöllisellä lausekkeella (*”[-|w|d]+”*), määritellään UUID (*”?P<pk>”*) jonka jälkeen määritellään, että UUID saa olla sekä numeroita 0-9 kuin myös kirjaimia (*”[0-9-f-]+”*). Lopulta osoitteeksi muodostuu esimerkiksi *”/dashboard/inventory/resource/resource/0b3548ba-9037-45de-be57-57321f077227/”*.

6.5 Uuden varauksen luonti

Uuden varauksen voi tehdä joko yksittäiselle resurssille, ryhmälle, tai useille resursseille kerrallaan. Varaukselle annetaan nimi, lisätietoja halutessaan, sekä kohteet. Kohteille annetaan aloitus- ja lopetuspäivä. Varaukseen liittyvä tilaus on valmiiksi lisätty kenttä, joka tulee käyttöön myöhemmässä

vaiheessa, kun varauksen voi liittää tilaukseen manuaalisesti ja tulevaisuudessa myös automaattisesti.



Kuva 12: Uuden varauksen tekeminen

Varauksen kohde voi olla joko resurssi tai ryhmä. Kaikki yrityksen resurssit ja ryhmät luetellaan alasvetovalikossa ja valittu kohde lisätään varaukselle plusnappia painamalla. Päivämäärävalintaa painamalla aukeaa kalenterinäkymä, jossa näytetään myös värikoodattuna varaustilanne valitulla resurssilla tai ryhmällä. Resurssin varattavaa kapasiteettia valitessa tarkistetaan suoraan onko kyseisellä resurssilla vapaana valittua määrä kapasiteettia, ja jos kapasiteettia ei ole vapaana ilmaistaan se muuttamalla valinta punaiseksi.

Kun tilaus tallennetaan, serialisoinnin yhteydessä tarkastetaan tietojen oikeellisuus ja varmistetaan myös se, että tilauksen varmasti pystyy tekemään kapasiteetin puolesta. Jos kapasiteetti on täynnä, näytetään valinta, jolla mahdollistetaan ylibookkaaminen. Tämä toiminnallisuus tarjotaan vain manuaalisesti varauksia kirjattaessa, ja sen toteuttaminen oli toivomus erään yrityksen suunnalta, joka pilotoi inventaariota kanssamme.

Tilaukset tallennetaan atomisesti ja resurssit lukitaan tietokannassa tallentamisen ajaksi. Tämä tehdään sen vuoksi, koska yrityksillä voi olla useita henkilöitä tekemässä ja muokkaamassa varauksia, ja halutaan estää se, että useampi henkilö koittaa samaan aikaan muokata tai varata samoja tuotteita. Lisäksi myös tulevaisuudessa, kun tämä kaikki toimii automaattisesti, täytyy varmistaa se, ettei virhetilanteita pääse syntymään. Jos transaktiossa tulee ongelmia kesken kaiken, kaikki siihen asti tehdyt muutokset perutaan ja käyttäjälle näytetään virheilmoitus.

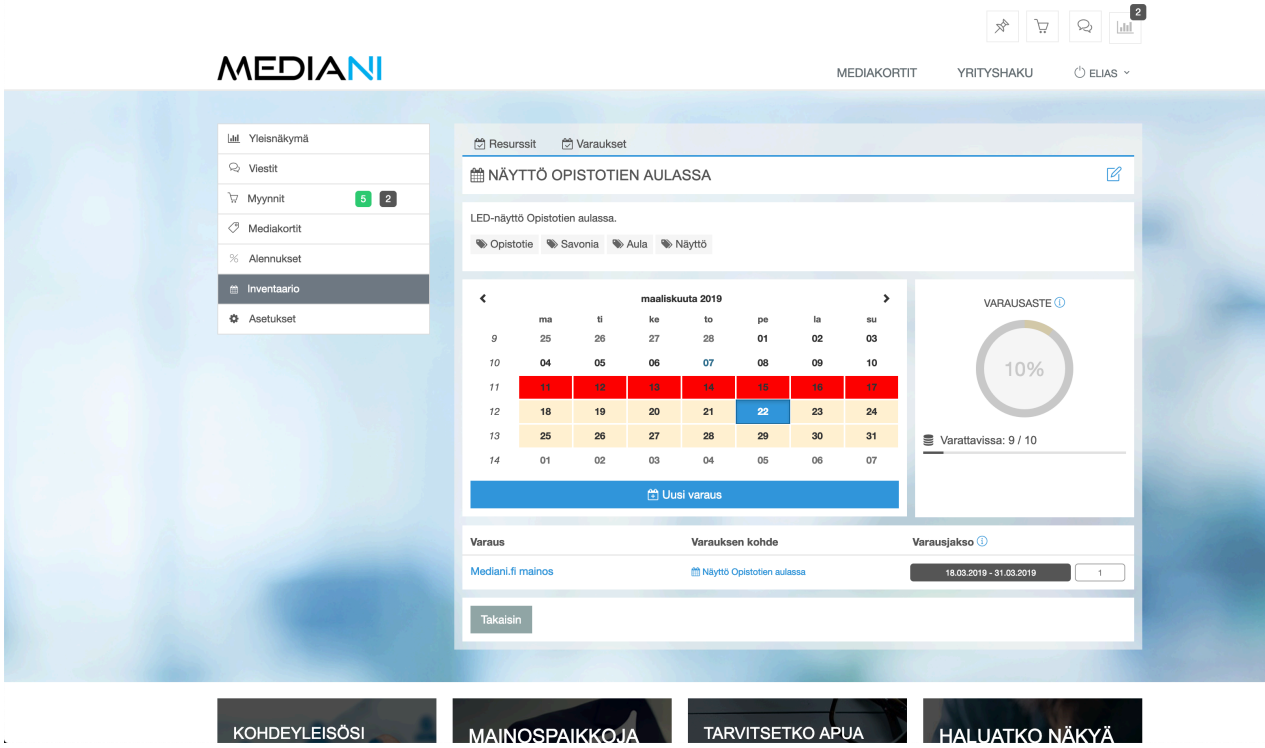
6.5.1 Varauksen muokkaus

Varausta muokatessa käytetään hyväksi samaa templatea kuin uutta varausta luodessa. Kaikki toimii käytännössä samalla tavalla, ja osoitteen parametreista saadaan tarvittava tieto siihen mitä varausta ollaan muokkaamassa. Lisänä näytetään poista-nappi, ja tallentaessa käytetään eri staattista metodia, joka päivittää olemassa olevaa tietoa. Muokkauksia tehdessä kaikki tehdään taas atomisesti. Tämä on erityisen tärkeää siksi, että itse resurssin varattua kapasiteettitietoa ei päivitetä, vaan jo valmiiksi olleet varaukset poistetaan ja tilalle luodaan uudet. Mutta koska transaktio tehdään atomisesti, jos varaus epäonnistuukin, ei tietoja menetetä vaan palataan takaisin lähtötilanteeseen.

6.6 Resurssin tietojen tarkastelu

Yksittäisen resurssin tietoja tarkastellessa pyritään käyttäjälle näyttämään mahdollisimman selkeästi resurssin varauksien kokonaistilanne sekä päiväkohtainen yksinkertaistettu näkymä. Kalenterissa näytetään päiväkohtainen tilanne yksinkertaistettuna ja päivää klikkaamalla näytetään varausaste prosentteina sekä se, paljonko kapasiteettia on tarkalleen jäljellä. Lisäksi kalenterin alla listataan kaikki valitulle päivälle osuvat varaukset, niiden kohteet, varausjaksot sekä varattu kapasiteetti.

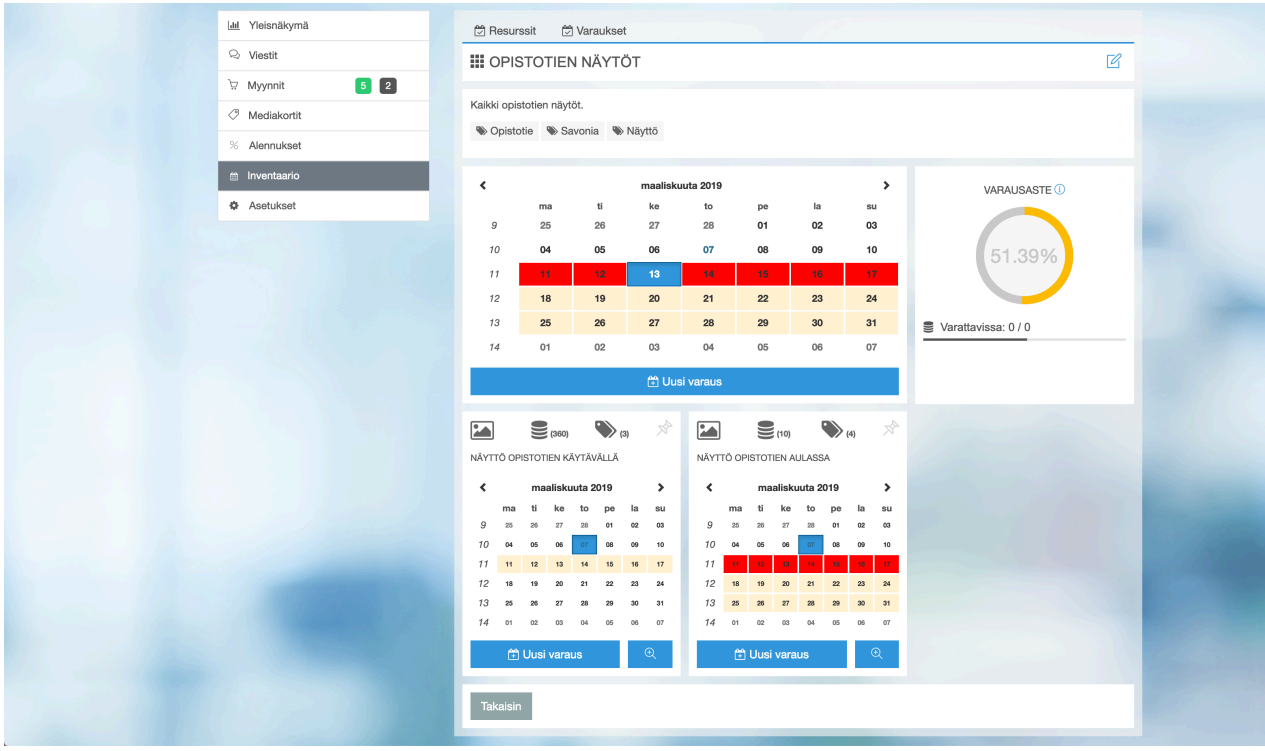
Näkymässä käytetään hyväksi samoja Redikseen tallennettuja tietoja varausasteen ja vapaan kapasiteetin näyttämisessä, kuin resurssien yleisnäkymässä.



Kuva 13: Resurssin tietojen tarkastelu

6.6.1 Ryhmän tietojen tarkastelu

Ryhmän tietoja tarkastellessa käytetään samaa templatea kuin resurssin tietoja tarkastellessa. Osoiteparametrista saadaan tieto, että kyseessä on ryhmä, joka vaikuttaa siihen mitä tietoja näytetään.



Kuva 14: Ryhmän tietojen tarkastelu

Kun kyseessä on ryhmä, näytetään varausasteessa keskiarvo prosentuaalisesti kaikista kyseisen päivän varauksista. Huomattavaa on, että vaikka keskimääräinen varausaste kuvassa 14 onkin vain 51.39%, on silti varattavissa oleva kapasiteetti nolla ja kalenterissa näytetään punainen väri sen merkiksi, että päivän varattu kapasiteetti on 100%. Tämä johtuu siitä, että ryhmän maksimikapasiteetti on aina sen mukainen, mitä pienimmän resurssin kapasiteetti on. Ryhmän varaamisessa kapasiteetin määrä riippuu siis siitä, millä resurssilla on pienin kapasiteetti. Myös tässä tilanteessa ylibookkausnappi tulee hyödylliseksi, koska joskus voi olla tilanne, jossa yhden resurssin ylibookkaaminen ei haittaa.

Ryhmää tarkasteltaessa näytetään varauslistan sijasta kaikkien ryhmän resurssien kalenterit, samaan tapaan kuin resurssien yleisnäkymässä. Näin käyttäjä saa suoraan hyvän katsauksen kaikkien ryhmän resurssien varaustilanteesta ja voi tarvittaessa helposti navigoida tutkimaan yksittäistä resurssia tarkemmin.

6.7 Rajapinta

Rajapintaan tehtiin kolme pääasiallista uutta reittiä käyttäen Django REST Frameworkin DefaultRouter-reititintä. DefaultRouter, kuten myös yksinkertaisempi SimpleRouter, sisältää standardit list-, create-, retrieve-, update-, partial_update- sekä destroy-toiminnot.

```
from django.conf.urls import url, include
from rest_framework import routers
from inventory_management import views


router = routers.DefaultRouter()

router.register(r'inventory-resources', views.InventoryResourceViewSet)
router.register(r'inventory-resource-tags', views.InventoryResourceTagViewSet)
router.register(r'inventory-reservations', views.InventoryReservationViewSet)

urlpatterns = [
    # api
    url(r"^api/", include(router.urls)),
]
```

Yllä yksinkertainen url.py-tiedosto, jossa reititin määritellään inventory_management-moduulissa. Näillä asetuksilla saadaan luotua ”/api/inventory-resources/”-, ”/api/inventory-resource-tags/”- sekä ”/api/inventory-reservations/”-endpointit. Tämän lisäksi määriteltiin muutamia erilaisia endpointteja eri käyttötarkoituksia varten käyttämällä inventory_management-moduulin views.py-tiedostossa määritellyille luokille detail_route-dekoraaattoreita. Alla esimerkki tällaisesta metodista:

```
@detail_route(methods=['get'], url_path='reservations')
    def get_reservations(self, request, pk=None):
        obj = self.get_object()
        reservations = obj.get_reservations().order_by('created_at')

        page = self.paginate_queryset(reservations)
        if page is not None:
            serializer = serializers.InventoryReservationSerializer(page, many=True,
context={'request': request})
            return self.get_paginated_response(serializer.data)

        serializer = serializers.InventoryReservationSerializer(reservations, many=True
context={'request': request})
        return Response(serializer.data)
```

Tällä metodilla saadaan aikaan uusi endpoint *”/api/inventory-resources/<avain>/reservations/”*, jolla listataan kaikki yhdelle resurssille tehdyt varaukset.

6.8 Testit

Koska tarkoituksena on automatisoida käyttöönotto, on syytä tehdä paljon testejä, joita voidaan ajaa automaattisesti. Tarkoituksena olisi, että kaikki toiminnallisuus, tietokantamallit sekä rajapinnat

olisi testattuna kattavasti. Valitettavasti tässä vaiheessa Selenium-testejä ei vielä saatu tehtyä aikataulun asettamien ehtojen takia, varsinkin koska katsottiin että koska ulkoasu muuttunee vielä jossain määrin, ja joitain asioita siirrellään vielä eri paikkoihin, olisi luotettavia Selenium-testejä vaikea kirjoittaa.

Tähän mennessä mallin ja näkymien testejä on kirjoitettu jo lähes 2000 riviä. Mallien testeissä testataan, että tiedot saadaan tallennettua oikeassa muodossa ja palautuvat objektit sisältävät sen tiedon mitä pitikin, ja sen lisäksi testataan erilaisia metodeita, joita mallien yhteyteen on rakennettu. Testejä varten käytetään niin sanottuja tehtaita, joilla generoidaan automaattisesti dataa määrättyjen ehtojen mukaan.

```
class InventoryResourceFactory(factory.django.DjangoModelFactory):
    class Meta:
        model = models.InventoryResource

    company = factory.SubFactory(user_factories.CompanyFactory)

    name = fuzzy.FuzzyText(prefix='Inventory resource ', length=20)
    description = fuzzy.FuzzyText(prefix='Inventory resource description', length=50)

    capacity = fuzzy.FuzzyInteger(1, 5)
```

Yllä esimerkki factory_boy-kirjastolla luodusta tehtaasta, jolla generoidaan tietokantaan Inventory-Resource-objekteja.

Näkymien testeissä testataan sitä, että rajapinta sekä serialisointi toimii kuten halutaan. Näin on helppo nähdä, jos joku muutos aiheuttaa sen, että rajapinnasta palautuva tai rajapinnan kautta tallennettava data muuttuu virheelliseksi.

7 YHTEENVETO

Kun opinnäytetyötä aloitettiin, oli aiheena pelkästään arkkitehtuurisuunnitelman laatiminen sen jälkeen, kun tutkittu, kannattaako tilaajan ruveta tekemään inventaarionhallintaa mikropalveluna vai toteuttaa se monoliittisena jo olemassa olevan sovelluksen kylkeen. Kävi kuitenkin nopeasti ilmi, että päätöksen tekeminen on melko yksiselitteistä yrityksen koosta johtuen, joten päätettiin tehdä nopeita päätöksiä ja aloittaa ohjelmointi mahdollisimman pian.

Sovelluksen toteutus aloitettiin joulukuun lopulla ja lopputuloksena saatiin aikaan ensimmäinen beetaversio, joka on jo päätynyt tuotantopalvelimelle asti ja on testikäytössä useilla oikeilla loppukäyttäjillä. Testikäyttäjien piiriä pyritään laajentamaan mahdollisimman nopeasti, kunhan seuraava versio, josta tämän opinnäytetyön kuvankaappaukset ovat otettu, saadaan testattua perusteellisesti ja siirrettyä tuotantoon. Inventaarionhallinnan avaaminen kaikille maksullisena laajennuksena tapahtunee näillä näkymin viimeistään kesällä.

Sovellukseen saatiin vaadittujen ominaisuuksien lisäksi tehtyä opinnäytetyöjakson aikana muun muassa varaustilanteiden visualisointia, jota toivottiin pilotoivan yrityksen suunnasta, sekä ryhmien luonti ja hallinnointi. Tämän lisäksi ylipäänsä pilotointi ei ollut alun perin suunnitelmissa vielä alkuvuoden aikana, mutta koska sovelluksen beetaversio saatiin valmiiksi nopealla aikataululla, voitiin pilotointi aloittaa aikaistetusti.

Toteutuksen aikana vaikeuksia aiheutti pääasiassa tiettyjen toimintojen hitaus. Rajapinta tekee esimerkiksi tarkistuksia, voiko tiettyä päivää varata, kun ollaan luomassa uutta varausta. Varauksessa saattaa olla kymmeniä resursseja, joita voidaan varata sadoiksi päiviksi. Tälläisissä tapauksissa varauksen luominen rupeaa kestämään yli 10 sekuntia, joka on edelleen liian pitkä aika, vaikka alkutilanteessa tuon varauksen tekeminen kesti 5 kertaa kauemmin. Sovelluksen optimointia on siis paljon edessä.

8 JATKOKEHITYS

Seuraava askel kehityksessä on hakutoimintojen tekeminen resursseille, ryhmille sekä varauksille, sekä suodattamistoiminnot. Yrityksillä voi olla tuhansia resursseja, joten näiden tekeminen isojen kokonaisuuksien hallitsemiseksi tulee olemaan todella tärkeää.

Haku- ja suodattamistoimintojen jälkeen täytyy tehdä varaussääntöjen luominen mediakorttien ja inventaarion väille. Lisäksi mediakorteilla täytyy halutessaan voida näyttää reaaliaikainen varaustilanne. Tässä vaiheessa voidaan myös toteuttaa kapasiteetin automaattinen varaaminen tilausta tehdessä.

Viimeiseksi lisätään muutamia nice to have-toiminnallisuuksia, kuten karttapisteiden lisääminen resurssille (esimerkiksi ulkomainostauluissa todella hyödyllinen ominaisuus) sekä kuvien lisääminen. Karttapisteet voidaan myös näyttää mediakortin julkisella sivulla, ja voidaan mahdollisesti toteuttaa karttahaku, jossa näytetään kartalla kaikki palvelusta löytyvät tuotteet.

Vaikka nämä askeleet ovat selkeät ja tiedossa olevat, voi asiakaspalautteen seurauksena prioriteetit ja toteutusjärjestys muuttua. On myös selvää, että tämän tyyppistä palvelua tehdessä se ei ole koskaan lopullisesti valmis vaan sitä kehitetään eteenpäin jatkuvasti ja uusia ominaisuuksia voidaan keksiä ja toteuttaa milloin tahansa.

9 LÄHDELUETTELO

Abbott, Martin L. ja Fisher, Michael T. 2015. *The Art of Scalability.* s.l. : Addison-Wesley Professional, 2015. 978-0134032801.

Django Software Foundation. Django overview. [Online] [Viitattu: 25. Tammikuu 2019.] https://www.djangoproject.com/start/overview/.

—. FAQ. [Online] [Viitattu: 20. Helmikuu 2019.] https://docs.djangoproject.com/en/dev/faq/general/#django-appears-to-be-a-mvc-framework-but-you-call-the-controller-the-view-and-the-view-the-template-how-come-you-don-t-use-the-standard-names.

Encode OSS Ltd. *Django Rest Framework kotisivu.* [Online] [Viitattu: 27. Helmikuu 2019.] https://www.django-rest-framework.org/.

Gitlab. Gitlab Continuous Integration & Delivery. *Gitlab.* [Online] [Viitattu: 1. Maaliskuu 2019.] https://about.gitlab.com/product/continuous-integration/.

Google. AngularJS. *AngularJS.* [Online] [Viitattu: 1. Maaliskuu 2019.] https://angularjs.org/.

—. Version Support Status. *AngularJS.* [Online] [Viitattu: 1. Maaliskuu 2019.] https://docs.angularjs.org/misc/version-support-status.

Jetbrains. Python Developer Survey 2018 Results. [Online] [Viitattu: 20. Helmikuu 2019.] https://www.jetbrains.com/research/python-developers-survey-2018/.

Kharenko, Anton. 2015. Microservices Practitioner Articles. [Online] 9. Lokakuu 2015. [Viitattu: 21. Joulukuu 2018.] https://articles.microservices.com/monolithic-vs-microservices-architecture-5c4848858f59.

Laitinen, Elias. 2019. 2019.

Lumetta, Jake. 2018. freeCodeCamp. [Online] 17. Tammikuu 2018. [Viitattu: 21. Joulukuu 2018.] https://medium.freecodecamp.org/monolith-vs-microservices-which-architecture-is-right-for-your-team-bb840319d531.

Mediani Group Oy. 2018. Mediani.fi. [Online] 2018. [Viitattu: 19. 12 2018.] https://mediani.fi/.

Microsoft. *Visual Studio Code.* [Online] [Viitattu: 1. Maaliskuu 2019.] https://code.visualstudio.com/.

—. 2006. Microsoft Docs. [Online] 2006. [Viitattu: 20. Joulukuu 2018.] https://docs.microsoft.com/en-us/previous-versions/office/developer/server-technologies/aa480455(v=msdn.10).

PostgreSQL Global Development Group. About. *PostgreSQL.* [Online] [Viitattu: 1. Maaliskuu 2019.] https://www.postgresql.org/about/.

Python Software Foundation. About Python. *Python.org.* [Online] [Viitattu: 25. Tammikuu 2019.] https://www.python.org/about/.

Red Hat Inc. *Ansible.* [Online] [Viitattu: 1. Maaliskuu 2019.] https://www.ansible.com/.

Sanfilippo, Salvatore. *Redis.* [Online] [Viitattu: 1. Maaliskuu 2019.] https://redis.io/.

The Django Book. Django Structure. *The Django Book.* [Online] [Viitattu: 20. Helmikuu 2019.] https://djangobook.com/mdj2-django-structure/.