Topi Matikainen, Suvi Rannisto

Web-asuntomarkkinapalvelu suosittelujärjestelmällä

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tieto- ja viestintätekniikka

Insinöörityö

3.5.2021

Tiivistelmä

Tekijä:	Topi Matikainen, Suvi Rannisto
Otsikko:	Web-asuntomarkkinapalvelu suosittelujärjestelmällä
Sivumäärä:	57 sivua
Aika:	3.5.2021
Tutkinto:	Insinööri (AMK)
Tutkinto-ohjelma:	Tieto- ja viestintätekniikka
Ammatillinen pääaine:	Ohjelmistotuotanto
Ohjaajat:	Ohjaava opettaja Juha Kämäri

Tämä insinöörityö käsittelee web-asuntomarkkinapalvelun suunnittelua ja toteuttamista nykyaikaisia web-teknologioita käyttäen. Työssä painotetaan erityisesti palvelun taustaa, konseptointia sekä palvelun hyödyntämää suosittelujärjestelmää. Web-sovelluksen selaintoteutus on kehitetty Vue.js-sovelluskehyksellä ja palvelinpuoli Node.js- ja Express.js-kehyksillä. Tietokantana toimii MongoDB, ja tietokannan sekä Node.js-kehyksen välisestä kommunikoinnista vastaa Mongoose. Web-sovelluksen käsittelemät kuvat tallennetaan Google Drive -pilvipalveluun.

Insinöörityössä tehdään taustatutkimusta Suomen ja Euroopan asuntomarkkinoista sekä käydään läpi asuntosijoittamista erityisesti aloittelevan asuntosijoittajan näkökulmasta. Lisäksi työssä tutkitaan suosittelujärjestelmiä ja suosittelualgoritmeja, jotka kuuluvat oleellisena osana nykyaikaisiin sovelluksiin.

Myös yksittäisen palvelun käyttäjän tietosuoja-asioihin perehdytään, sillä useimmat web-sovellukset keräävät ihmisistä tietoja, jotka ovat osittain tunnistettavia. Henkilötietojen säilyttämisestä määrätään laissa, jotta jokaisen oikeus omiin tietoihinsa säilyy ja tietoturva voidaan taata.

Insinöörityön tuloksena syntyi prototyyppi palvelusta, jonka pääasialliset toiminnallisuudet ja erikoispiirteet on tutkittu ja käyttövalmiita. Palvelun julkaisemista varten on kuitenkin vielä toteutettava joitakin sivutoiminnallisuuksia ja viimeisteltävä sivuston toimintaa.

Avainsanat: Web-kehitys, sovelluskehykset, moderni ohjelmistokehitys, suosittelujärjestelmät, algoritmit

Abstract

Author:	Topi Matikainen, Suvi Rannisto
Title:	Housing Market Web Application with Recommendation System
Number of Pages:	57 pages
Date:	3 May 2021
Degree:	Bachelor of Engineering
Degree Programme:	Information Technology
Professional Major:	Software Development
Instructors:	Juha Kämäri, Senior Lecturer

The aim of this Bachelor´s thesis was to design and develop a web-based housing market application with modern web technologies. The thesis especially emphasizes the background, concept and recommendation system used by the service. The browser implementation of the web application was developed with Vue.js framework and the server side with Node.js and Express.js frameworks. The database is kept in a MongoDB cloud cluster, with Mongoose being responsible for communication between the database and Node.js framework. Images processed by the web application are stored in Google Drive cloud service.

The thesis includes background research on Finnish and European real estate markets and goes through real estate investing, especially from the perspective of a novice real estate investor. In addition, the thesis investigates recommendation systems and algorithms, which are an integral part of modern applications. The privacy issues of an individual service user are also addressed, as most web applications collect information about people that is partially identifiable.

As a result of the study, a prototype of the service was created, the main functionalities and special features were researched and are ready for use. However, in order to publish the service, some further functionalities still need to be implemented and the overall functionality of the service has to be finalized.

Keywords: Web development, frameworks, modern software development, recommendation systems, algorithms

Sisällys

Lyhenteet

API: *Application Programming Interface*. Ohjelmointirajapinta, jonka avulla kaksi verkossa toimivaa sovellusta voivat kommunikoida keskenään.

CSS: *Cascading Style Sheets*. Tyylikieli, jota käytetään HTML-elementtien ulkoasun määrittämiseen.

DOM: *Document Object Model*. Käsittelee XML- ja HTML-dokumentteja puurakenteena, jossa jokainen solmu on olio.

GDPR: *General Data Protection Requlation*. EU:n vuonna 2018 voimaan asettama tietosuoja-asetus, jossa määritellään yksilön tietosuojaa koskevia säännöksiä.

HTML: *Hypertext Markup Language*. Standartoitu merkintäkieli web-sivujen luomiseen. Kuvaa verkkosivun rakennetta koostumalla elementeistä, jotka kertovat selaimelle, miten sisältö näytetään.

HTTP: *Hypertext Transfer Protocol*. Selaimen ja palvelimen väliseen tiedonsiirtoon käytettävä protokolla.

I/O: *Input/output*. Viittaa ohjelman vuorovaikutukseen järjestelmän levyn ja verkon kanssa. Näitä ovat esimerkiksi tietojen lukeminen ja kirjoittaminen, HTTP-pyyntöjen tekeminen tai tietokannan kanssa kommunikoiminen.

JSON: *JavaScript Object Notation*. JavaScriptissä usein etenkin tiedonsiirrossa käytettävä oliomainen tietorakenne.

MVVM: *Model-View-ViewModel*. Ohjelmistoissa käytettävä suunnittelumalli, joka erottaa graafisen käyttöliittymän kehityksen ohjelman logiikasta.

NoSQL: *Not only SQL* (*Structured Query Language*). Tietokantamalli, jossa tietojen tallentaminen ja hakeminen mallinnetaan muilla keinoilla kuin relaatiotietokannoissa käytetyillä taulukkosuhteilla.

npm: *Node Package Manager*. Pakkauksenhallintatyökalu Node.js-pakkauksille. Helpottaa JavaScript-kehittäjiä jakamaan, käyttämään ja päivittämään koodia.

REST: *Representational state transfer*. Usein käytetty arkkitehtuurityyli palvelintoiminnallisuutta omaavien web-sovellusten kehityksessä.

Sass: *Syntactically Awesome Style Sheets*. CSS-esiprosessori eli komentosarjakieli, joka mahdollistaa koodin kirjoittamisen erillisellä ohjelmointikielellä ennen sen kääntämistä CSS-tyylitiedostoksi.

TTL: *Time to live*. MongoDB:n elinaikaindeksi. Indeksi asetetaan tarkkailemaan tietueen aikamäärettä kuvastavaa kenttää. Jos tietokantapalvelimen kellonaika ylittää kenttään annetun aikamääreen, tietokannan hallintajärjestelmä poistaa tietueen

URL: *Uniform Resource Locator*. Merkkijono, jolla osoitetaan verkkosivuston tai tiedoston sijainti Internetissä.

1 Johdanto

Insinöörityömme tavoitteena oli suunnitella ja toteuttaa Vue-pohjainen web-sovellus asuntomarkkinoille. Perusominaisuuksien, kuten asuntojen hakemisen ja välittämisen lisäksi, sivustolla voi etsiä kämppäkaveria yhteisasumiseen. Palvelu käyttää algoritmia, jonka tarkoituksena on suositella käyttäjille tarpeen mukaan joko sopivaa kämppäkaveria tai asuntoa. Sivustoon kuuluvalla viestintäkanavalla potentiaaliset asunnonhakijat ja asunnonvälittäjät sekä kämppäkaveria itselleen etsivät käyttäjät voivat keskustella ja tutustua asuntokohteisiin sekä toisiinsa paremmin.

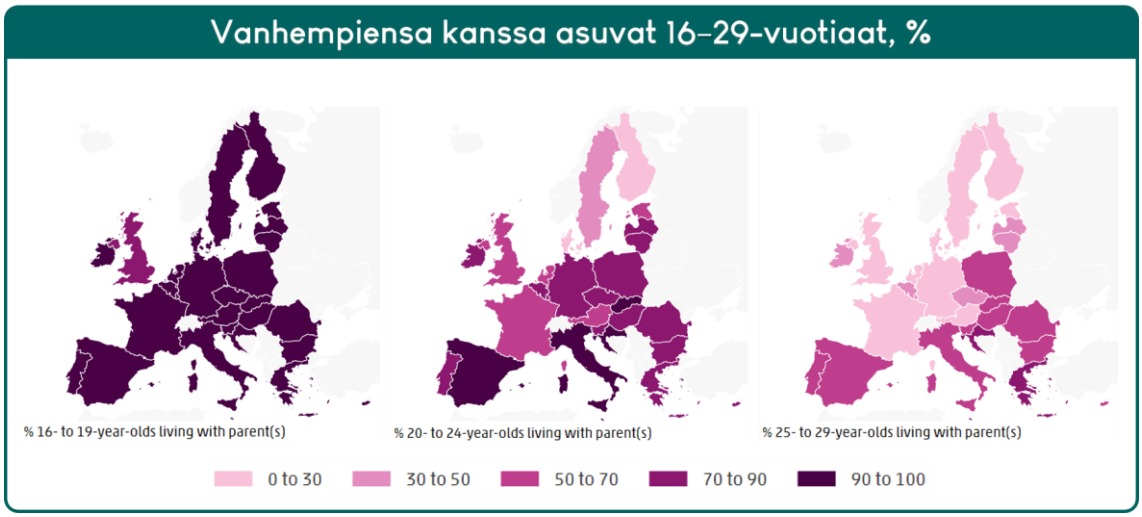
Sivuston idea lähti liikkeelle pohdittuamme asumisen kalleutta ja sitä, kuinka moni ihminen asuu yksin. Asumiskustannukset ovat nimenomaan yksinasuville suhteettoman suuret verrattuna talouksiin, joissa asumiskustannusten vastuu jakautuu useammalle ihmiselle. Palvelullamme haluammekin kannustaa ihmisiä löytämään yhteisöllisen asumisen sekä kämppisasumisen hyödyt, joita ovat pienempien asumiskustannusten lisäksi uusiin ihmisiin tutustuminen, sosiaalisuus sekä uusien ystävyyssuhteiden muodostuminen.

2 Taustatutkimus asuntomarkkinoista Suomessa ja muualla Euroopassa

Viimeisen kahdenkymmenen vuoden aikana yhteiskuntamme on kehittynyt nopeasti. Suurin vaikutus on ollut moderni teknologia, jonka luomien mahdollisuuksien myötä yhteiskunnan elintavat, kuten asuminen, työskentely ja matkustaminen, ovat muuttuneet. Globalisaatio on tuonut eri kulttuureja ja paikkoja lähemmäs kotiamme, mutta samalla meillä on kasvava tarve lähteä kotimaisemistamme kauas maailmalle. Haluamme kokea, minkälaista elämä muualla on. Varsinkin Pohjoismaissa nuoret muuttavat varhain pois lapsuudenkodistaan. Useimmat meistä valitsevat yksin asumisen helppouden, kun ei tarvitse huomioida muiden samassa taloudessa elävien tarpeita. Asuntomarkkinat eivät kuitenkaan tue yksin asuntoa etsiviä ihmisiä, joita varsinkin Pohjoismaissa on nyt enemmän kuin koskaan.

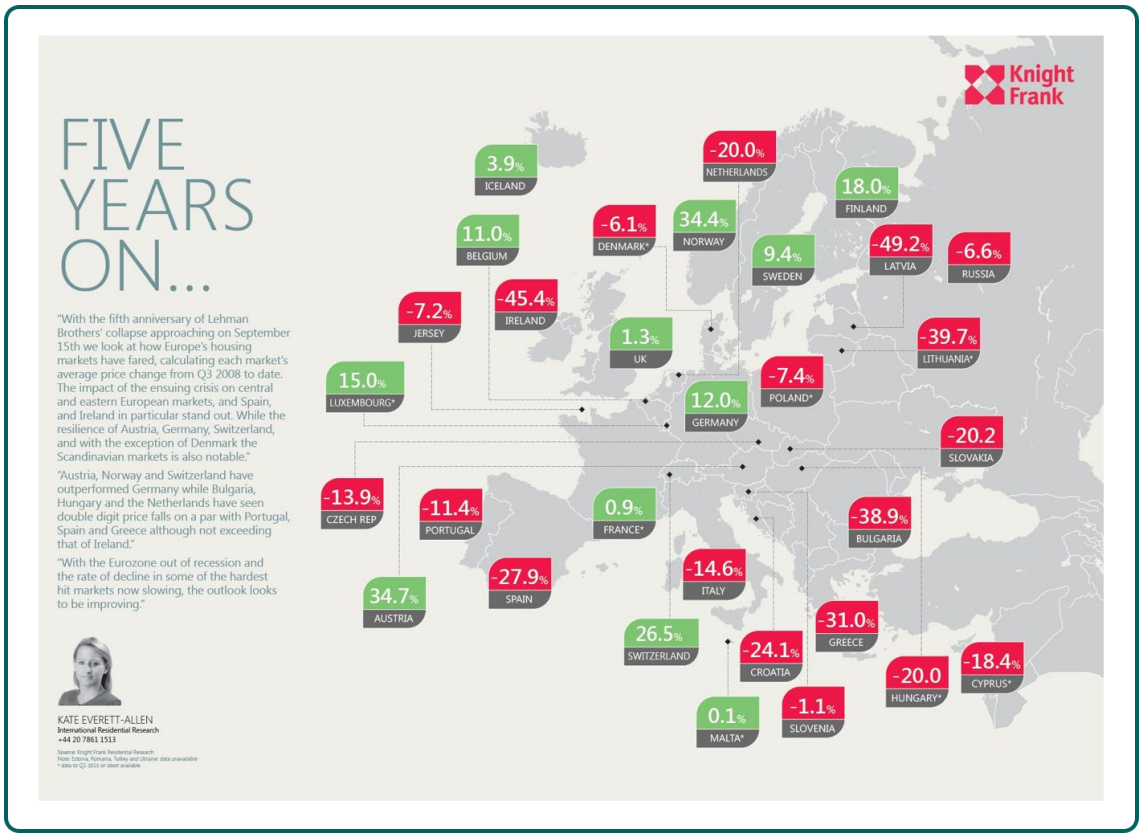
Kun Pohjoismaissa nuoret aikuiset muuttavat tyypillisesti mahdollisimman varhain pois kotoa, ei muualla Euroopassa ole tavatonta asua vielä kolmekymmentävuotiaana vanhempiensa luona. Euroopan unionin tilastotoimisto Eurostatin mukaan Suomessa lapsuudenkodista muutetaan pois keskimäärin alle 22-vuotiaina ja noin 45 prosenttia 16–

29-vuotiaista asuu vielä vanhempiensa luona. Muualla Euroopassa vastaavasti lapsuudenkodista muutetaan keskimäärin vasta yli 26-vuotiaina ja noin 67 prosenttia 16–29-vuotiaista asuu edelleen vanhempiensa luona (kuva 1). Ilmiötä selittää osaltaan erilaiset kulttuurilliset tekijät. [1.]



Kuva 1. Vuoden 2017 tilastoja vanhempiensa luona asuvista nuorista aikuisista. [2]

Pohjoismaissa nuoret oppivat jo varhain kotona itsenäistä elämää ja saavat tukea elämänvalintoihinsa. Suuri halu itsenäistyä ja nuorena omilleen muuttaminen koetaan eräänlaisena statussymbolina, josta ollaan ylpeitä. Hyvä keskimääräinen tulotaso ja valtion rahallinen tuki jatko-opiskeluissa edesauttavat varhain omilleen muuttoa. Toisaalla Välimeren alueen maissa, kuten Espanjassa, Italiassa ja Kreikassa, vaikuttaa perhekeskeisyys eikä vanhempien luona asumiseen yhdistetä samanlaisia negatiivisia asioita, kuten Pohjoismaissa. Suurin syy myöhäiseen omilleen muuttoon on kuitenkin vähävaraisuus ja asuntojen kalleus. Maissa, joihin finanssikriisi vuosina 2007–2009 iski pahiten (kuva 2), voidaan nähdä selkeää nousua nuorten aikuisten asumisessa vanhempiensa luona. [3.]



Kuva 2. Asuntomarkkinat Euroopassa viisi vuotta finanssikriisin jälkeen vuonna 2013, missä näkyy erityisesti Välimeren maiden ja Itä-Euroopan maiden heikko asuntomarkkinatilanne. Pohjoismaissa ja Keski-Euroopan-maissa puolestaan voidaan nähdä erittäin voimakasta hintojen nousua, mikä heijastuu asumiskustannusten nousuun. [4.]

2.1 Asunnonhankinta

Asunnon hankkiminen on kenelle tahansa iso päätös: oli sitten nuorempi tai vanhempi, ensiasunnon ostaja tai jo kokenut tekijä. Suurimmalle osalle ihmisistä asunnon hankkiminen määrittää elämäntyylin sekä sen, mihin paikkakunnalle ihminen haluaa pysähtyä. Asunnon ostamisessa liikkuvatkin suurien tunteiden lisäksi myös iso määrä rahaa. Useimmille ihmisille tämä tarkoittaa suurta asuntolainaa, jota maksetaan takaisin useita vuosikymmeniä. Poikkeuksena on länsinaapurimme Ruotsi, missä koko asuntolainaa ei välttämättä makseta takaisin koskaan, vaan odotetaan asunnon arvon nousua. Euroopassa on kuitenkin hyvin yleistä se, etteivät nuoret ihmiset enää edes haaveile omistusasunnon hankinnasta. Esimerkiksi Saksassa nuorille aikuisille omistusasunnon hankinta on täysin mahdoton ajatus kalliiden hintojen vuoksi. Suurin osa heistä ei edes tunne ketään, joka omistaisi asunnon [5].

Useissa Euroopan maissa myöskään vuokra-asunnon hankkiminen ei ole itsestään selvyys kalliiden asumiskustannusten vuoksi. Tämän takia yhdessä asunnossa voi hyvinkin asua enemmän ihmisiä, kuin mille se on tarkoitettu. Lisäksi vuokra-asuntoja on ylipäätään vaikeaa saada isojen metropolien suosituilta alueilta, sillä kysyntää on enemmän kuin tarjontaa.

Euroopassa asuntojen kallistuminen huippuunsa johtuu pitkälti rakennusalan kutistumisesta, joka ei ole palautunut finanssikriisiä edeltäneelle tasolle [6]. Lisäksi rakennusmateriaalien, kuten puutavaran kallistuminen, on nostanut hintoja entisestään. Asumiskustannukset nousevat nopeammin kuin kotitalouksien tulot, minkä takia asumiseen joudutaan käyttämään yhä enemmän rahaa samalla, kun asumispuitteista joudutaan tinkimään.

2.2 Asuntomarkkinat tulevaisuudessa

Pandemia ja sen myötä lisääntynyt etätyöskentely ja -opiskelu ovat johtaneet siihen, etteivät esimerkiksi opiskelijat muuta enää heti opiskelupaikkakunnalleen, vaan jäävät kotiin ja säästävät rahaa. Etätyöskentelyn puolestaan uskotaan lisääntyvän, sillä siihen pakotetusti siirtyminen on todistanut monille yrityksille, että etätyöskentely voi toimia yritykselle edullisesti. Useimmat etätyöskentelijät ovat jopa sitä mieltä, että työn tehokkuus on kasvanut, kun ihmisiä ei ole koko ajan ympärillä juttelemassa ja aikaa säästyy työmatkustuksen jäädessä pois. Toisaalta työskentely kotona lisää myös työn ja vapaa-ajan rajan häilymistä sekä henkistä työtaakkaa, mikäli ei ole ketään, kenelle puhua kasvotusten, kun työasiat painavat mieltä. Sama pätee myös etäopiskeluun.

Niin etätyöskentelyssä kuin myös etäopiskelussa kodin toimivuus ja sijainti ovat nostaneet merkitystään. Ihmiset kaipaavat nyt enemmän tilaa, lisää huoneita sekä omaa pihaa. Monet ahtaasti asuvat ihmiset ovat huomanneet haaveilevansa mökilleen luonnon keskelle tai haluavansa muuttaa omakotitaloon kaupungin ulkopuolelle. Ne, joilla on ollut tähän mahdollisuus, ovat haaveensa ja aikeensa myös toteuttaneet. Suuret kaupungit, kuten Helsinki ja Espoo, ovat pudonneet Suomen vetovoimaisimpien kuntien kärjestä [7], kun kehyskuntiin muutetaan luonnon, väljempien tilojen ja edullisempien asuntojen perässä.

Tänä päivänä keskustassa asuminen ei siis ole enää niin suosittua kuin aiempina vuosina on ollut. Ihmiset suosivat nyt asumista keskustan laitamilla uudiskohteissa tai

kauempana omakotitaloasuinalueilla. Tilan arvostaminen niin kodin sisällä kuin ulkona on kasvanut ja itse asunnon kriteerien lisäksi asetetaan kriteerejä myös ulkona harrastamiseen. Mitä lähempänä hyvät lenkkipolut ovat ja mitä monipuolisempi harrastustarjonta lähistöllä on, sitä parempi.

Kun muuttoaalto on alkanut kääntymään vastavirtaan maaseudun suuntaan, ovat varsinkin pienten kaupunkien keskustat alkaneet autioitumaan. Keskustoissa on yhä enemmän liike- ja toimistotiloja, pieniä päivittäistavarakauppoja sekä ruokailupaikkoja. Kaupunkien keskustat asuinympäristönä ovatkin vuosien varrella muuttuneet enemmän ihmisten kohtauspaikoiksi. Jotta keskustat olisivat jälleen vetovoimaisia asuinympäristöjä, on asiantuntijoilla ja kaupunkisuunnittelijoilla edessään uusi haaste. [8.]

Ratkaisuja keskustojen viihtyvyyteen on haettu kaupunkien vihertämisellä ja autoliikenteen rajoittamisella. Toisin sanoen, tulevaisuuden keskustoissa maaseutuasumisen hyvät puolet yhdistyvät kaupunkiasumisen hyötyihin. Vihertämisellä sekä moottoriäänien ja päästöjen vähentämisellä keskustoista halutaan stressittömämpiä ja viihtyisämpiä elinympäristöjä (kuva 3).

Kuva 3. Keilaniemeen avattiin vuonna 2019 Kehä I:n tunneli, jonka tarkoituksena oli tehdä keskustasta yhtenäisempi ja viihtyisämpi. Autoliikenne ohjattiin piiloon tunneliin, jotta kaupunkiympäristöä saatiin elävöitettyä tunnelin kannen päälle rakennetulla puistoalueella. [9.]

Varsinkin yksityisautoilua rajoittavaan keskustaan siirtyminen on kuitenkin herättänyt vankkaa vastustusta autoilevilta ihmisiltä. Keskustoissa oleellista onkin erityisesti kulkuväylien ja muun infrastruktuurin suunnittelu niin, että julkinen liikenne tarjoaisi

kilpailukykyisen tavan liikkua keskustassa sekä keskustan ja reuna-alueiden väliä. Tällä hetkellä haasteena on erityisesti pidempi matkustusaika verrattuna autolla liikkumiseen. Lisäksi julkisen liikenteen pysähdyspaikat eivät voi sijaita jokaisen palvelun välittömässä läheisyydessä.

Näiden asioiden lisäksi keskustoissa pyritään ottamaan yhä enemmän huomioon lapsiperheet. Kaupunkien tulevaisuuden strategiassa pohditaan, missä keskustassa on paikkoja, joissa lapset voivat leikkiä ja nuoret käydä ostoksilla, viettää vapaa-aikaa sekä toteuttaa harrastuksiaan luovuutta ja mieltä virkistävässä ympäristössä. Monipuolinen tarjonta lisäisi keskustojen viihtyvyyttä ja toisi tekemistä kaikille ikäryhmille ja eri asioita harrastaville ihmisille (kuva 4).

Kuva 4. Garden Helsinki -hankkeen suunnitelma osana Helsingin keskustan uudistamista. Gardeniin tulee tapahtuma-areenan lisäksi oheispalveluja sekä jokapäiväisiä palveluita kaupunkilaisten käyttöön, kuten kuntoilukeskus, hotelli ja ruokakauppa sekä harrastus- ja kilpapaikkoja noin kolmellekymmenelle urheilulajille. [10, 11.]

Merkittävä tekijä keskustojen autioitumisessa on myös kaupan- ja palvelualan muuttuminen. Kivijalkakaupat ovat häviämässä katukuvasta ja tilalle on tullut muita palveluita. Isot kauppakeskukset ja kauppaketjut ovat rakentaneet toimintojaan kaupunkien ulkopuolelle, jossa tilaa on enemmän. Kaupunkien tavoitteena onkin saada isot kaupat mukaan keskustojen suunnitteluun niiden vetovoiman takia.

Euroopassa ja muualla maailmassa on törmätty samaan ongelmaan kuin Suomessakin. Erityisesti muualla Euroopassa ollaan kuitenkin jo huomattavasti edellä kaupunkien keskustojen viihtyvyyden parantamisessa, kuten viherryttämisessä, ympäristöystävällisyydessä sekä liikkumisvaihtoehdoissa. Wien, München, Berliini ja Madrid pitävät neljän kärkeä maailman vihreimpien kaupunkien listauksessa, jossa on otettu huomioon esimerkiksi viheralueiden määrä, ilmanlaatu, julkinen liikenne sekä kävely- ja pyöräilymahdollisuudet (kuva 5). [12.]

Kuva 5. Resonancen listaus maailman vihreimmistä kaupungeista perustuu maailman 50 eniten vierailtuun kaupunkiin, ja niiden saamiin arvosteluihin Tripadvisor-sivustolla.

Jos tulevaisuudessa keskustojen suunnitelmat ja odotukset täyttyvät, voi hyvin olla, että keskustoista tulee jälleen vetovoimaisia myös asuinympäristöinä. Tällä hetkellä ne kuitenkin tarjoavat asunnon niille, joille on tärkeää asua opiskelu- tai työpaikkansa lähellä ja jotka ovat valmiita tinkimään asuinympäristöstään ja tilanmäärästään.

2.3 Asuntosijoittaminen

Asunnot ovat nykyään hyvin suosittu ja turvallinen sijoituskohde. Asuntosijoittamista voi tehdä kahdella eri tavalla, joko sijoittaa asuntosijoitusrahastoihin tai sijoittaa suoraan sijoitusasuntoon. Asuntosijoitusrahastot ovat helppoja kohteita, joissa alkupääomaa

tarvitaan vähän ja rahasto-osuudet ovat nopeasti realisoitavissa rahaksi. Huonona puolena on kuitenkin isot hallinnointikulut, jotka vievät suuren osan siitä tuotosta, jonka puolestaan saisi omistamalla sijoitusasunnon suoraan itse. Pidemmän päälle asunnon ostaminen ja vuokranantajaksi ryhtyminen on näin ollen tuottoisampaa. Toisaalta asunnon omistaminen vaatii enemmän aikaa ja vaivaa, sillä ostaminen ja vuokraaminen teettävät töitä. Lisäksi alkupääomaa tarvitaan enemmän verrattuna rahastoihin. [13.]

2.3.1 Sijoitusasunnon hankkiminen

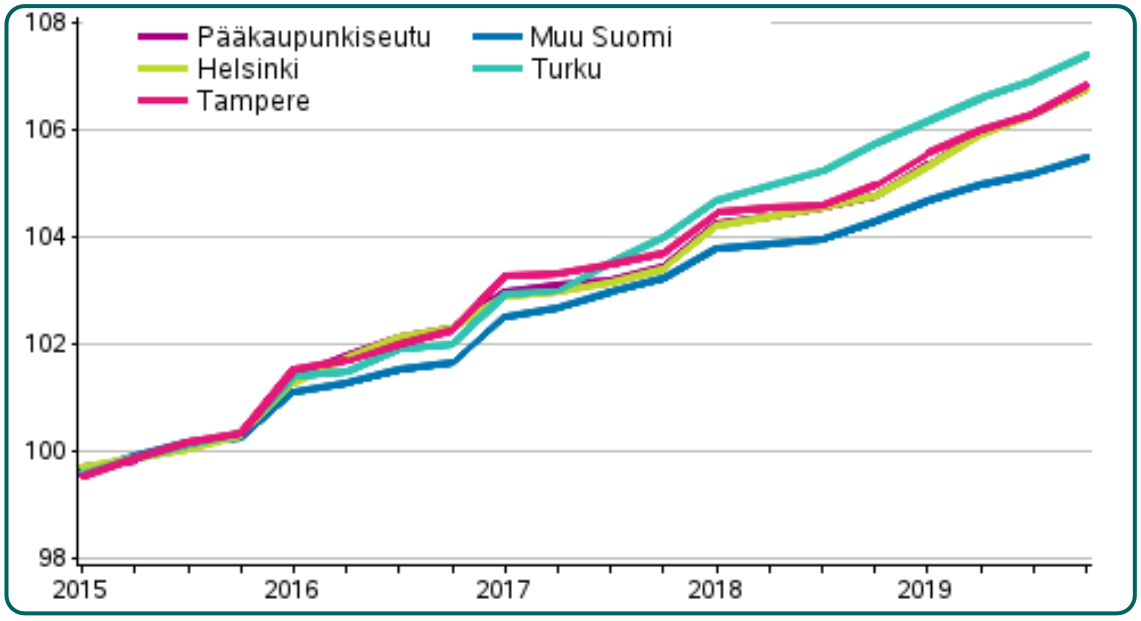
Jos asunnon haluaa ostaa sijoitusmielessä, tulisi sijoittajan miettiä, millainen on hyvä asuntosijoituskohde. Sijoitusasuntoa ei voi lähteä etsimään sellaisilla kriteereillä, joita itsellä on asumisen suhteen. Lähtökohtaisesti asuntosijoittajan tulisi miettiä, mikä on alueen potentiaalinen vuokralaiskohderyhmä, ja lähteä etsimään sopivaa asuntoa heidän tarpeisiinsa. Palveluiden sekä oppilaitosten ja koulujen läheisyys on yksi tärkeimmistä tekijöistä, joita sijoitusasunnon sijainnissa tulisi ottaa huomioon. Vaikka asunto ei olisikaan remontoitu kaikilla mukavuuksilla, esimerkiksi oppilaitoksen lähellä sijaitseva edullisempi asunto menee varmemmin heti vuokralle, kun taas moderni ja näin ollen kalliimpi asunto voi helposti jäädä vuokraamatta pidemmäksikin aikaa. Lisäksi yksiöillä ja kaksioilla on yleensä suurempia perheasuntoja enemmän kysyntää vuokramarkkinoilla. Pienten asuntojen etuna on myös suurempi vuokratuotto suhteessa neliöihin, vaikka toisaalta pienten asuntojen neliöhinnat ovat kalliimpia verrattuna suurempiin asuntoihin.

Sijoitusasuntoa ja ylipäätänsä asuntoa hankittaessa tulisi ottaa myös huomioon taloyhtiön kunto, sillä se vaikuttaa olennaisesti asunnon arvoon. Ennen asunnon ostoa on hyvä tarkistaa taloyhtiössä jo tehdyt sekä tekemättömät remontit. Isot remontit, kuten viemäri- ja julkisivuremontit, voivat tulla kalliiksi, ja yleensä ne kasvattavat merkittävästi yhtiövastiketta.

2.3.2 Sijoitusasunnot kaupungeissa

Suosituin ja turvallisin tapa on ostaa pieniä asuntoja suurempien ja kehittyvien kaupunkien keskustoista, sillä näissä paikoissa yksiöillä ja kaksioilla on jatkuvasti suurempaa kysyntää kuin tarjontaa. Tämä asetelma pitää vuokrahinnat korkealla ja nostaa vuokrahintoja entisestään (kuva 6). Vallitsevassa taloustilanteessa, jossa talouskasvu on heikkoa, vuokra-asuntojen kysyntä vain kasvaa, sillä ihmiset eivät uskalla hankkia

omistusasuntoa. Asuntojen kallis hinta sekä hintojen nousu esimerkiksi pääkaupunkiseudulla vähentävät ihmisten halua ostaa omaa asuntoa. He mieluummin säästävät rahaa tulevaisuuden varalle kuin sitoisivat kymmeniä- tai satojatuhansia euroja asuntoon. Toinen merkittävä syy vuokrahintojen nousuun on erilaiset valtion myöntämät rahalliset tuet. Esimerkiksi opiskelijoilla on enemmän varaa laittaa rahaa vuokraan, kun useimmat saavat sekä opinto- että asumistukea. Lisäksi esimerkiksi toimeentulotuki kannustaa joitakin vuokranantajia nostamaan vuokrahintaa tukien enimmäisrajaan asti. [14.]



Kuva 6. Vapaarahoitteisten vuokra-asuntojen vuokrien kehitys, indeksi 2015=100. Vuokrat ovat nousseet vuodesta 2015 pääkaupunkiseudulla 6,8 prosenttia ja muualla Suomessa 5,5 prosenttia. [15.]

Muun Euroopan vuokra-asuntomarkkinatilanne on hyvin samanlainen kuin Suomessakin. Euroopalla on suuri kasvupotentiaali, ja väestön odotetaan kasvavan tasaisesti, mikä kasvattaa asuntojen kysyntää tulevina vuosina. Vuokrahinnat nousevat edelleen useimmissa Euroopan kaupungeissa, kuten Berliinissä, Brysselissä, Lontoossa ja Pariisissa. Erityisesti kansainvälisyys ja kulttuurilliset tekijät nostavat ihmisten kiinnostusta asua suurissa Euroopan kaupungeissa. [16.]

3 Palvelu

Web-sovelluksemme tarkoitus on tarjota vaihtoehto edullisempaan asumiseen sekä tarjota matala kynnys asuntosijoittamiseen. Haluamme kannustaa ihmisiä olemaan

luovempia, mitä tulee erilaisiin asumismuotoihin ja asuntojen omistamiseen. Kimppa-asuminen on kaikin puolin edullisempaa verrattuna yksin asumiseen. Jos puolestaan haluaa aloittaa asuntosijoittamisen, kannattaa oma asunto laittaa vuokralle. Näin itselle saa konkreettisesti tuloja eikä käteen jää ainoastaan asuntoa.

3.1 Sivuston liikeidea ja kohderyhmät

Varsinkin Suomessa kimpassa asumista kuitenkin vältellään, sillä olemme itsenäisiä, emmekä halua toisia ihmisiä henkilökohtaiselle alueellemme. Sivustomme tehtävänä onkin auttaa käyttäjää löytämään eri elämätilanteisiin juuri se oikea asuintoveri, jonka seurassa ei tarvitse murehtia asumiseen ja elämiseen liittyvistä asioista.

Tämän lisäksi sivustomme tarjoaa kohtauspaikan asunnon hakijoille ja asunnon tarjoajille. Haluamme tarjota yksityishenkilöille helpon tavan välittää tai vuokrata asuntoja toisille yksityishenkilöille ilman välikäsiä ja näin madaltaa kynnystä vuokrata oma asunto ulkopuolisten käyttöön. Tämä vähentäisi tyhjillään olevien asuntojen määrää, jolloin ne tuottaisivat omistajilleen lisäarvoa sekä toisivat etenkin suurkaupunkeihin lisää erittäin kysyttyjä vuokra-asuntoja madaltaen samalla yleistä vuokratasoa.

3.2 Palvelun eroavaisuudet muuhun tarjontaan nähden

Palvelumme eroaa muiden asunnonvälityspalveluiden tarjonnasta sillä, että tarjoamme monipuolisempaa vuorovaikutusta eri osapuolten kesken. Yleensä asunnonvälityspalvelun käyttäjän tulee olla hyvin aktiivinen ja tietoinen siitä, minkälaista esimerkiksi asuntoa tai vuokranantajaa hän etsii. Useimmat käyttäjät eivät kuitenkaan koe, että heillä olisi riittävästi tietoa tai tuntemusta kohteesta, jota etsii, tai tarkkaa sijaintia, jonne haluaisi muuttaa. Palvelumme tarkoituksena onkin suositella kohteita tai mahdollisia kämppäkavereita käyttäjälle tämän tekemän profiilin perusteella. Käyttäjä itse määrittelee, mitä tietoja hän profiiliinsa laittaa. Parhaimman tuloksen kuitenkin saa kertomalla mahdollisimman paljon itsestään ja hakutoiveistaan.

Lopulliseen sovellukseen toteutetaan myös käyttäjien välinen chat-palvelu. Kun potentiaalinen asunto tai kämppäkaveri on löytynyt, voivat myyjät ja ostajat, vuokranantajat ja vuokralaiset sekä kämppisehdokkaat tutustua toisiinsa paremmin chat-palvelun kautta, jotta sopivat ihmiset ja asunnot kohtaavat. Tällä tavoin pienennetään riskiä esimerkiksi

saada häiritsevä vuokralainen tai kämppäkaveri. Asunnonostotilanteessa puolestaan myytävän talon puutteet tai mielenkiintoiset yksityiskohdat tulevat suuremmalla todennäköisyydellä esiin, kun ostajalla on mahdollisuus käydä tarkempia keskusteluja suoraan myyjän kanssa.

Kämppäkaverit voivat puolestaan hakea chat-palvelun kautta yhdessä asuntoa. Kun käyttäjä aloittaa viestittelyn toisen käyttäjän kanssa ja he päättävät hakea yhdessä asuntoa, he voivat muodostaa ryhmän. Kun tulevat kämppäkaverit löytävät mieleisensä asunnon, he voivat joko yksin tai ryhmänä ilmoittaa chatissä olevansa kiinnostuneita kyseisestä asunnosta. Tällöin asunnontarjoaja näkee, ketkä ovat kiinnostuneita kohteesta. Kiinnostuneiden käyttäjien profiilien näkeminen lisää asunnontarjoajan turvallisuudentunnetta ja laskee kynnystä vuokrata oma asunto ennestään tuntemattomille ihmisille.

4 Suunnitteluprosessi

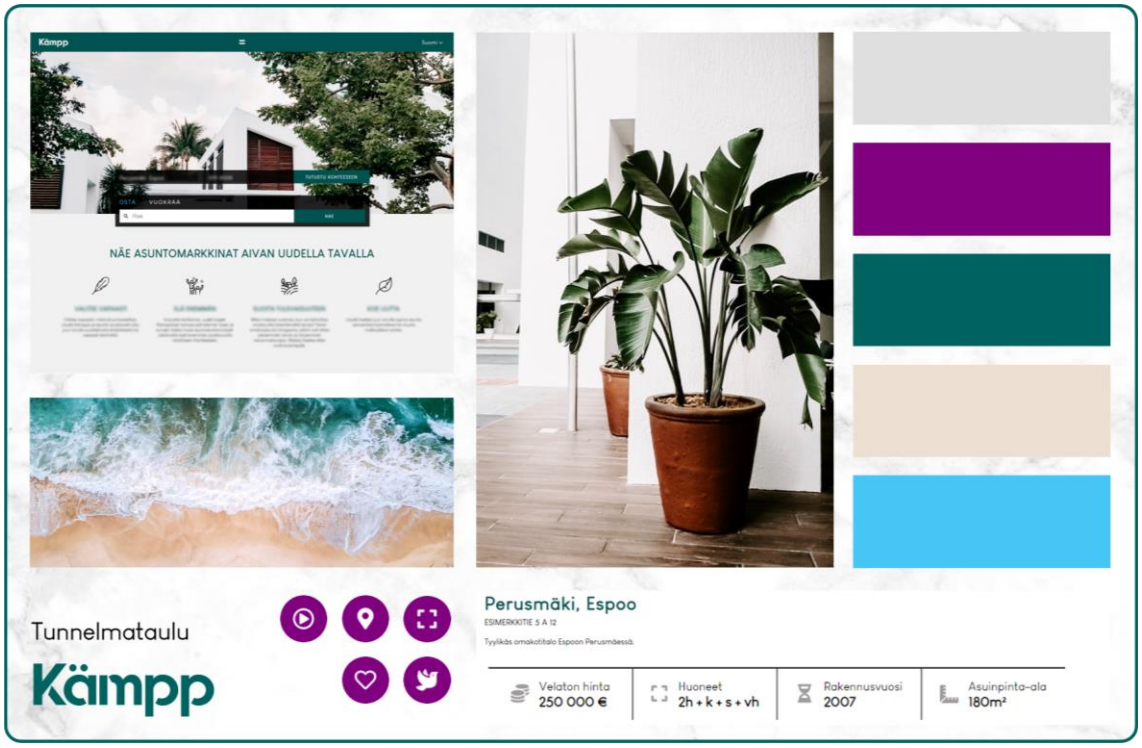
Asuntomarkkinapalvelumme suunnittelun aloitimme konseptin luomisella, johon kuuluivat visuaalinen ilme, kohderyhmät sekä palvelun toiminnot, kuten käyttäjien ja asuntojen käsittely. Lisäksi halusimme sivustomme tarjoavan keskustelualustan, jonka suunnittelussa huomioimme kevyen sosiaalisen median ominaisuudet. Web-sovelluksen suunnittelu lähti liikkeelle erilaisiin internetsivustoihin tutustumalla.

4.1 Visuaalinen ilme ja sivujen asettelu

Ensimmäisessä vaiheessa kartoitimme erilaisia visuaalisia ilmeitä ja elementtejä, joista pidimme. Näiden myötä meille alkoi pikkuhiljaa muodostua kuvaa siitä, minkä tyylisen sivuston haluamme visuaalisesti sekä toiminnallisesti.

Lähtökohtana visuaalisessa suunnittelussa oli voimakas värimaailma, jolla haluamme luoda käyttäjälle mieleenpainuvan ensivaikutelman (kuva 7). Voimakkaalla ja harvinaisemmalla värimaailmalla haluamme myös viestiä sivuston olevan erilainen kuin, mitä normaalit asunnonvälityssivustot ovat. Sivuston pääväriksi valitsimme tummanvihreän ja sen tehosteväreiksi violetin, sinisen ja beigen. Värien valintaan vaikuttivat luonnonläheisyys. Haluamme sivuston viestivän rauhallisuutta samalla tavalla, kuin luonnossa liikkuminen lievittää stressiä ja parantaa mielialaa. Pääväri tummanvihreä onkin poimittu metsän väreistä. Tehoste värit sininen ja beige puolestaan tulevat veden, taivaan sekä

hiekan ja valon väreistä. Näitä elementtejä voidaan mieltää rentouttaviksi, mutta saman aikaisesti myös mieltä virkistäviksi elementeiksi. Violetti väri puolestaan edustaa kaikenlaista luonnon väriloistoa.



Kuva 7. Kämpp-palvelun tunnelmataulu, jossa esillä värimaailmaa ja sivuston visuaalista ilmettä.

Värimaailman suunnittelun jälkeen aloimme miettiä sivujen asettelua, jolla viestitään sivuston kontekstista. Asettelun tuli kuvastaa sivuston olevan asuntomarkkinoille suunnattu. Toisessa vaiheessa kävimmekin läpi erilaisia asunnonvälitys- ja kämppishakusivustoja. Sivustoja tutkimalla pyrimme kartoittamaan, mitä erilaisia toimintoja sivustollemme tarvitsemme ja haluamme sekä minkälaisia tietoja tyypillisessä asuntoilmoituksessa kerrotaan.

Useat asunnonvälitys- ja kämppishakusivustot ovat pohjimmiltaan samannäköisiä. Suomalaisissa asunnonvälityssivustoissa toistuvat kirkkaat värit sekä elementtien kerroksellisuus ja tietynlainen elementtien ruuhkaisuus näytöllä. Tällaisella asettelulla mitä luultavammin halutaan viestiä käyttäjälle nopeaa päätöksentekoa ja kirkkailla väreillä pitää käyttäjä aktivoituneena. Koska halusimme sivuston erottuvan tavallisista asunnonvälityssivustoista kämppishakuominaisuuden vuoksi, pyrimme suunnittelemaan sivujen asettelun hyvin yksinkertaiseksi. Yksinkertaisella asettelulla tuemme sivuston värien

valintaa, mutta halusimme myös viestiä sivuston helposta käytöstä. Yksinkertaisuutta sekä helppoutta pyrimme tuomaan esiin myös käyttämällä isoja, selkeitä elementtejä sekä vahvoja, mutta minimalistisia fontteja.

4.2 Toiminnot

Palvelussamme on kaksi käyttäjäryhmää, asunnontarjoajat sekä muut käyttäjät, jotka etsivät joko kämppäkaveria tai asuntoa. Useimmat käyttäjistä tulevat käyttämään palvelua melko lyhyen aikaa eli siihen saakka, kunnes he löytävät kämppiksen tai asunnon. Näistä asioista johtuen erittelemme kaksi erilaista käyttäjätyyppiä rekisteröitymisen yhteydessä. Kahdelle täysin erilaiselle käyttäjäryhmälle on järkevää tarjota kohdistettua sisältöä eli vain sellaista tietoa, joita he tarvitsevat. Käyttäjätyyppien määrittämisellä karsitaan esimerkiksi se, että asuntoa etsivä käyttäjä ei näe asunnon myynti-ilmoituslomaketta eikä asunnontarjoaja kämppishakuun tarvittavaa profiililomaketta tai kämppisehdokaslistausta. Käyttäjätyyppi valitaan kirjautumisikkunan yhteydessä olevaa painiketta painamalla.

Palvelumme on laaja, joten sivustoomme tarvitaan kattava kokoelma erilaisia tiedonkäsittelytoimintoja. Kun käyttäjä etsii kämppäkaveria tai asuntoa, hänen tulee luoda profiili etsimästään kohteesta. Sopivan kohteen löytyessä, kämppisehdokkaiden sekä asunnonhakijoiden ja -välittäjien välinen vuorovaikutus tapahtuu chat-palvelun kautta.

Olennaisena osana palveluamme on myös suosittelualgoritmi, joka toivekohdetietojen perusteella käy läpi muita käyttäjiä tai asuntoja käyttäjän puolesta ja suosittelee niitä samankaltaisuuden mukaan lajiteltuna. Suosittelualgoritmin myötä hakijan ei tarvitse itse suodattaa tuloksia tai käydä läpi useita kohteita satunnaisesti.

4.3 Tietoturva

Tietoturva ja käyttäjän tietosuoja ovat nykypäivänä oleellisia osia Internetissä toimivan sovelluksen kehityskaarta. Palvelua suunnitellessa meidän on otettava huomioon kehitysratkaisuissa asioita myös tietoturvallisuuden kannalta. Kehityspalvelimen URL-osoitteet sekä tietokanta- ja kuvapankkitunnukset tulee salata versionhallinnan ulkopuolelle jätettyyn .env-ympäristömuuttujatiedostoon. Lisäksi käyttäjälle tulee luoda autentikoitu istunto tämän sisäänkirjautumisen yhteydessä, jotta palvelimelle ei voi lähettää mielin

määrin HTTP-pyyntöjä. Käyttäjien salasanat suojataan tallentamalla ne sovellustietokantaan salatussa muodossa.

Profiilia täytettäessä keräämme paljon tietoa käyttäjästä, joten projektin edetessä meidän olisi hyvä arvioida uudelleen joidenkin tietojen tarpeellisuutta sekä näkyvyyttä. Esimerkiksi käyttäjän sukunimen pyytäminen kämppishakuprofiilissa ei ole oleellista kämppistä haettaessa. Toisaalta asuntoa haettaessa ja asuntoa välitettäessä se olisi tärkeä tieto, sillä koko nimellä luodaan luotettavuutta. Tästä johtuen sukunimeä ei tulla näyttämään kämppisehdokkaiden profiileissa, mutta asunnonvälittäjät näkevät asunnosta kiinnostuneen käyttäjän koko nimen sekä käyttäjä näkee asunnonvälittäjän koko nimen. Kämppishakuprofiilin muutto-osiossa puolestaan tarkan osoitteen kysyminen ei ole sekään välttämätön esimerkiksi suosittelualgoritmillemme, joten mitä luultavammin poistamme sen kokonaan, jotta käyttäjät eivät hämmenny siitä. Asunnonvälityspuolella tietenkin tarkat osoitetiedot ovat oleellisia.

Sovellukseemme siis kerätään käyttöä varten käyttäjästä paljon osin tunnistettavia tietoja. Sovelluksen käyttäjät syöttävät tietonsa sovellukseen itse, emmekä vastaanota sovelluksen käyttäjistä mitään tietoja kolmansilta osapuolilta. Koska tietoja kuitenkin kerätään, on laadittava EU:n yleisen tietosuoja-asetuksen eli GDPR:n mukaiset rekisteri- ja tietosuojaselosteet. [17., 18.]

GDPR:n mukaan yksityishenkilöllä on seuraavat oikeudet omiin tietoihinsa liittyen [19]:

- Oikeus tietää, mitä tietoa organisaatio on hänestä kerännyt.
- Tietää, missä tarkoituksissa ja miten rekisterinhaltija käsittelee hänen tietojaan.
- Pyytää virheellisten, epätarkkojen ja puutteellisten tietojen korjaamista.
- Pyytää kaikkien omien tietojen poistamista organisaation rekisteristä.
- Vastustaa omien tietojensa käsittelyä.
- Pyytää henkilötietojensa käsittelyn rajoittamista.
- Siirtää tietonsa toisen organisaation haltuun.
- Olla joutumatta perusteetta automaattisen päätöksenteon kohteeksi.

Lopullisessa sovelluksessa tulemme tarjoamaan käyttäjälle mahdollisuuden tarkastella, muokata ja poistaa kaikkia henkilökohtaisia tietojaan, mitä hän on sovellukseen tallentanut. Sovelluksen julkistamisvaiheessa olisi kuitenkin vielä hyvä palkata erillinen asiantuntija arvioimaan sovellusta käyttäjän tietosuojan kannalta sekä avustamaan rekisteri- ja tietosuojaselosteiden kirjoittamisessa, jotta palvelumme varmasti noudattaa ajanmukaisia säädöksiä käyttäjätietojen käsittelyyn liittyen [20].

5 Teknologiavalinnat

Web-palvelu koostuu tyypillisesti selain- ja palvelinohjelmistoista, jotka kommunikoivat keskenään välittäen tietoa HTTP-pyynnöin. Palvelinohjelmisto käsittelee tietoa ja varastoi sitä tyypillisesti palvelinohjelmistoa pyörittävällä tietokoneella toimivaan tietokantaan. Koska meillä ei ole yhteistä erillistä palvelinkonetta, sekä palvelin- että selainohjelmistoja ajetaan rinnakkain kehitystietokoneilla.

Sovelluksen mahdollisesta julkaisusta ja tuotantoversion palvelinlaitteistosta ei ole vielä varmuutta, joten päätimme pitää tietokannan ja kuvapankin ratkaisujen osalta melko avoimena, ja itse kehitysprojektin meille kustannustehokkaana. Versionhallintatyökaluksi valikoitui GitHub, johon laadimme erillisen organisaatioprofiilin insinöörityöllemme.

5.1 Teknologiavalinnat selaintoteutuksessa

Projektin suunnitteluvaiheessa mietimme, millä sovelluskehyksellä alamme palveluamme toteuttamaan, React- vai Vue-kirjastolla. Päädyimme lopulta valitsemaan Vue 3:n, sillä Vuen suosio on ollut tasaisessa kasvussa. Reactilla kehittäminen oli jo entuudestaan meille molemmille tuttua, mutta vastaavaa kokemusta Vuesta ei ollut. Insinöörityön yksi tärkeimmistä tavoitteista oman osaamisen esille tuomisen lisäksi on myös oppia täysin uusia asioita, joten Vueen tutustuminen oli meille erittäin mielenkiintoinen haaste. Vuen lisäksi valitsimme selaintoteutukseen Vuex-tilanhallintakirjaston, Axiosin HTTP-pyyntöihin ja Sassin tyylien määrittämiseen.

5.1.1 Vue.js

Vue.js on progressiivinen JavaScript-kehys reaktiivisten käyttöliittymien ja *single page* -sovellusten rakentamiseen. Ohjelmistoinsinööri Evan You aloitti sen kehittämisen

vuonna 2013, kun hän huomasi, ettei saatavilla ole yhtään kevyttä ja joustavaa nopeisiin prototyyppeihin soveltuvaa JavaScript-kehystä [21]. Huolimatta helposta käyttöönotosta ja yksinkertaisuudesta se on erittäin tehokas, suorituskykyinen ja monipuolinen. Progressiivisuus tarjoaa sen, että Vue voidaan integroida jo olemassa olevaan web-sovellukseen muiden kirjastojen kanssa, sillä sen ydinkirjasto on keskittynyt ainoastaan näkymäkerrokseen. Monimutkaisempiin sovelluksiin, kuten reititysten, tilanhallinnan ja koontityökalujen toteuttamiseen, on saatavilla monia lisäominaisuuksia, joista Nuxt.js on yksi suosituimmista tukikirjastoista.

Vue noudattaa MVVM-arkkitehtuuria, joka keskittyy deklaratiiviseen renderöintiin ja uudelleenkäytettävien komponenttien kokoonpanoon. MVVM-arkkitehtuuri helpottaa graafisen käyttöliittymän kehityksen erottamista ohjelman muusta toimintalogiikasta tai palvelinpuolen logiikasta siten, että näkymä ei ole riippuvainen mistään tietystä alustasta. Tämä helpottaa esimerkiksi ohjelman ylläpitoa. Deklaratiivinen renderöinti mahdollistaa tietojen renderöimisen DOM:in käyttäen *template*-syntaksia. Kaksoisaaltosulkeita käytetään paikkamerkkeinä tarvittavien tietojen interpolointiin DOM:ssa.

5.1.2 Vuex

Vuex on tilanhallintamalli ja kirjasto, jota käytetään Vue-sovelluksissa. React-sovelluksissa käytettävän Reduxin tapaan Vuex on saanut inspiraationsa Fluxista. Flux on sovellusarkkitehtuurimalli, jota käytetään selainpuolen web-sovellusten rakentamiseen. Se on suunniteltu ratkaisemaan ongelmaa, joka syntyy, kun useat komponentit jakavat tietoa, ja niiden välisten yhteyksien monimutkaisuus kasvaa niin, ettei tietojen tilaa voida enää ennustaa tai ymmärtää [22]. Tästä seuraa se, että sovelluksen laajentaminen tai ylläpitäminen tulee mahdottomaksi. Fluxin ideana on luoda joukko selkeitä ja helposti ymmärrettäviä sääntöjä, jotka kuvaavat skaalautuvaa selainarkkitehtuuria ja näin lieventävät aiemmin mainittua ongelmaa.

Vuex toimii keskitettynä varastona (*store*), johon kaikki komponentit pääsevät sinne säilytettyihin tietoihin käsiksi. Kun komponentit jakavat sovelluksen tietoja, niiden tulee lukea niitä tässä sijainnissa eikä säilyttää omia kopioita ristiriitojen ja erimielisyyksien estämiseksi. Sääntö vähentää merkittävästi propsien eli tietojen siirtämisen tarvetta komponenttien välillä. Lisäksi se tekee muutosten jäljittämisestä yksinkertaisempaa ja virheiden etsimisestä ja korjaamisesta helpompaa, kun tilaa muutetaan ainoastaan ennalta oletetulla tavalla.

Komponentit voivat vapaasti lukea tietoa varastosta, mutta eivät muuttaa sitä. Jotta komponentit voivat muuttaa tietoja, niiden tulee ilmoittaa aikeistaan varastolle, joka tekee muutokset itse käyttäen mutaatioita. Mutaatiot (*mutations*) ovat aina synkronisia, mikä varmistaa sen, ettei tila (*state*) ole riippuvainen arvaamattomien tapahtumien järjestyksestä tai ajoituksesta [23].

Mutaatioiden lisäksi muutoksia voidaan tehdä toiminnoilla (*actions*), jotka kutsuvat mutaatioita. Koska mutaatioiden tulee olla synkronisia, toiminnoilla voidaan suorittaa myös asynkronisia operaatioita, kuten API-kutsuja. Varaston tietojen lukemiseen komponentit käyttävät gettereitä. Getterit (*getters*) ovat funktioita, joilla voidaan muuttaa tiedon muotoa, kuten suodattaa tietoa ennen tiedon palauttamista sovellukselle.

5.1.3 Sass

Sass on CSS-esiprosessori, joka sisältää enemmän ominaisuuksia kuin mikään muu CSS-laajennuskieli (kuva 8). Lisäksi se on yhteensopiva kaikkien CSS-versioiden kanssa. Sass tarjoaa useita ominaisuuksia helpottamaan ja nopeuttamaan sivuston tyylien määrittelyä, kuten muuttujat, sisäkkäisyys, miksaus ja perintä, funktiot, matemaattiset operaatiot sekä silmukat [24].

Kuva 8. Sass muuttaa ohjelmointikielen CSS-tyylitiedostoksi, jota selain tukee.

Päätimme valita projektiin Sassin CSS:n tilalle sen monipuolisuuden takia. Isoissa projekteissa tyylitiedostot kasvavat helposti suuriksi ja monimutkaisiksi, jolloin niiden käsittely on hankalampaa. Ensisijaisesti valintaamme vaikutti muuttujien käyttö. Halusimme luoda sivustolle teematiedoston, johon on määritelty sivustollamme käytettävät fontit sekä värit. Kun nämä on määritelty muuttujiin, ei esimerkiksi yksittäisen värin RGB- tai HEX-värikoodia tarvitse muistaa ulkoa, riittää, kun muistaa muuttajan nimen. Muuttujien

käyttö helpottaa myös silloin, kun huomataan esimerkiksi tietyn värin olevan vääränlainen, jolloin riittää, että kyseisen muuttujan arvoa muutetaan teematiedostosta.

Halusimme käyttää muuttujien ohella myös sisäkkäisyyttä. Sisäkkäisyys auttaa tyylikokonaisuuksien hallitsemisessa, kun visuaalisesti näkee koodin sisäkkäin eikä rinnakkain. Tämän lisäksi se vähentää huomattavasti tyyleihin käytettävää koodin määrää. Sisäkkäisyyttä käytimme erityisesti globaaleissa tyyleissä ja isojen elementtikokonaisuuksien määrittelyissä.

5.2 Teknologiavalinnat palvelintoteutuksessa

Palvelintoteutuksen osalta päätimme pitäytyä meille tutussa ja turvallisessa. Node.js ja sen Express.js-kehys ovat olleet aikaisemmin mukana monissa eri projekteissa opiskelujemme aikana, ja niillä kehittäminen on meille luontevaa. Yhtenäisen tietokannan käsittelemiseksi valitsimme kehitysprosessin ajaksi ilmaisen MongoDB Atlas -pilvipalvelun. Itse tietokannan tarkastelu ja kehitystarkoituksissa tietojen muokkaaminen tapahtuu MongoDB Compass -tietokantatulkilla. Palvelimen kuvapankiksi valikoitui ilmainen Google Drive.

5.2.1 Node.js ja npm

Node.js on suosittu avoimen lähdekoodin ajoympäristö, joka suorittaa JavaScript-koodia palvelinpuolella ja tarjoaa näin ollen tehokkaan palvelinkehityksen. Node on pakattu kokoelma V8 JavaScript -moottorista, libuv-alustan abstraktiokerroksesta ja ydinkirjastosta. Noden tuleminen markkinoille mahdollisti reaaliaikaiset ja kahdensuuntaisen yhteyden omaavat web-sovellukset, joissa sekä selain että palvelin voivat aloittaa viestinnän, jolloin ne voivat vaihtaa tietoja vapaasti. Muutos oli merkittävä ero siihen, missä tyypillisesti selain aina aloittaa yhteyden. [25.]

Node on erittäin nopea koodin suorittamisessa V8-moottorinsa ansiosta. Lisäksi se on tehokas ja skaalautuva, sillä se suorittaa yksisäikeistä, estämätöntä ja asynkronista ohjelmointia [26]. Kaikki API-rajapinnat Node-kirjastossa ovat asynkronisia eli estämättömiä, ja tapahtumien ilmoittamismekanismi auttaa palvelinta saamaan vastauksen edellisestä API-kutsusta.

Estämättömällä (*non-blocking*) tarkoitetaan sitä, että Node eliminoi odottamisen. Estävä (*blocking*) viittaa operaatioihin, jotka estävät jatkokäsittelyn siihen asti, kunnes operaatio päättyy, kun taas estämätön ei estä seuraavaa käsittelyä, vaan on heti valmis seuraavaan pyyntöön. Nodessa, estämättömyys viittaa ensisijaisesti I/O-operaatioihin. Estävät menetelmät suoritetaan synkronisesti, kun taas estämättömät menetelmät suoritetaan asynkronisesti. Asynkronisuudella tarkoitetaan ajasta riippumatonta kommunikointia. Asynkroninen ohjelmointi on rinnakkaisen ohjelmoinnin muoto, joka mahdollistaa funktion suorittamisen ilman, että toisen funktion täytyy odottaa edellisen funktion toiminnan päättymistä.

Kun Node asennetaan, asennetaan myös npm automaattisesti. Npm-pakkauksenhallintatyökalu on erittäin suosittu, sillä siihen on saatavilla lukematon määrä erilaisia JavaScript-kehittäjien tekemiä lisäosia, jotka tarjoavat työkaluja aina koodin laadun ylläpidosta valmiisiin ohjelmatoiminnallisuuksiin. Lisäosat eli paketit sisältävät kaikki tiedostot, joita moduuliin eli JavaScript-kirjastoon tarvitaan. Pakettien asentaminen käy yksinkertaisesti komentoriviltä, ja niiden käyttöönotto tapahtuu *require*-funktiolla.

5.2.2 Express.js

Node.js-kehyksillä on vankka viittaus REST API -sovellusten nopeampaan rakentamiseen eli verkkosovellusten arkkitehtuurityyleistä ei tarvitse huolehtia. Palvelumme REST-rajapinta on toteutettu Noden suosituimmalla HTTP-palvelinkirjastolla Express.js-kehyksellä. Express kohdistuu Noden nopeisiin I/O-operaatioihin ja yksisäikeiseen luonteeseen, mikä tekee siitä oletusvaatimuksen sovelluksille, jotka on rakennettu Node.js-alustalla [27].

Express on minimalistinen ja joustava web-sovelluskehys, joka tarjoaa perustoiminnallisuuksien lisäksi monia muita ominaisuuksia web- ja mobiilisovelluksien sekä API-rajapintojen rakentamiseen. Expressin joustavuus viittaa sen muokattavuuteen. REST API:n rakentamisen lisäksi sitä voidaan käyttää esimerkiksi staattisten tiedostojen isännöintiin, palvelinpuolen renderöitiin tai välityspalvelimena [28]. Express on niin sanotusti *middleware* eli väliohjelmisto, joka toimii kahden eri ohjelman välissä liitoksena.

5.2.3 MongoDB ja Mongoose

MongoDB on dokumenttipohjainen NoSQL-tietokantaohjelmisto, jossa tietokanta rakentuu dokumenteista koostuvista kokoelmista (*collection*). MongoDB varastoi tietoa JSON-muodon kaltaisiin dokumentteihin, mikä tekee tietojen käsittelystä tehokkaampaa verrattuna perinteisen relaatiotietokannan rivi-kolumni-malliin. Luettavuuden lisäksi MongoDB:n etuna on sen joustavuus. Dokumenttien (*document*) ei tarvitse olla rakenteeltaan samanlaisia, vaan jokaisella dokumentilla voi olla erilainen skeema eli tietomalli. Dokumenttiskeeman (*schema*) avulla voi määrittää dokumenttien ja upotettujen dokumenttien rakenteen ja sisällön. Skeemalla voi myös vaatia tiettyjä kenttäjoukkoja, määrittää kentän sisällön tai vahvistaa dokumentin muutokset. Kentät (*field*) voivat sisältää normaalien arvojen lisäksi myös taulukoita ja sisäkkäisiä olioita.

Monipuolisuutensa vuoksi totesimme MongoDB:n sopivan web-sovellukseemme hyvin. Asuntojen vuokra- ja myynti-ilmoitusten tyypillisesti väliaikaisen luonteen vuoksi on mahdollista pitää tietokannan rivimäärät kohtuullisen alhaisena sekä säilyttää dokumenttitietokannan paras mahdollinen suorituskyky. MongoDB sekä Atlas-palvelu mahdollistavat myös TTL-indeksien luomisen dokumenteille.

Koska tietokannaksi valittiin MongoDB, on luonnollista, että palvelinsovelluksemme käyttää tietokantaa Mongoose-tietomallinnuskirjaston avulla. Mongoose käsittelee tietokannan dokumentteja oliomuotoisten skeemojen avulla. Skeemojen avulla tietorakenteisiin on helppoa luoda tyypitykseen ja pakollisiin kenttiin perustuvia validaatiosääntöjä. Niillä varmistutaan siitä, että kantaan ei pääse sisällöltään vajavaisia tai väärin muodostuneita tietueita.

Lisäksi Mongoose mahdollistaa automaattisen poiston lapsidokumenteille sekä emodokumentin viittauksille muissa dokumenteissa emodokumentin poiston yhteydessä. Tämä tapahtuu hyödyntämällä Mongoosen *pre-middlewarea* eli esiväliohjelmistoa. Esiväliohjelmisto (esimerkkikoodi 1) suoritetaan ennen varsinaisen tietokantamuutoksen aloittamista, ja sen voi määrittää muillekin kuin poistotyyppisille operaatioille.

```
userSchema.pre("deleteOne", function (next) {
     var query = this
     mongoose
          .model("targetProfile")
          .deleteOne({ user: query._conditions._id }, function (err) {
          if (err) {
               console.log(err)
          } else {
               return next()
          }
     })
})
```

Esimerkkikoodi 1. Esiväliohjelmiston määrittely, joka on kirjoitettu skeemamatiedoston yhteyteen. Kyseinen funktio poistaa käyttäjään liitetyn kohdeprofiilin itse käyttäjän poiston yhteydessä.

5.2.4 Google Drive ja Google Drive API

Google Drive on Googlen tarjoama pilvipalvelu, johon voi tallentaa ilmaiseksi 15 gigatavua vapaamuotoista tietoa. Driven avulla voi tallennuksen lisäksi jakaa ja yhteiskäyttää tiedostoja ja kansioita millä tahansa laitteella. Drive integroituu yli sataan eri tiedostotyyppiin, kuten PDF-, CAD- ja Microsoft Office -tiedostoihin. Driven integroituminen eri työkaluihin ja sovelluksiin mahdollistaa tiimin reaaliaikaisen yhteistyön vaivattomasti.

Google tarjoaa Drivelle myös rajapintapalvelua, jolla kehittäjät voivat hyödyntää Drive-tallennustilaa omassa sovelluksessaan ohjelmallisesti. Google Drive API:n avulla sovellukset integroituvat Driveen, ja niihin voi luoda toiminnallisuuksia, jotka hyödyntävät pilvipalvelun tarjoamia ominaisuuksia. Päätimme hyödyntää palvelussamme Google Drive API:a kuvien säilömiseen. Kuvat tallennettaisiin käyttäjä- ja asuntokohtaisiin kansioihin, ja kuvien id:t vastaavasti MongoDB-tietokantaan. Kuvien id-arvojen avulla oikeat kuvat pystyttäisiin hakemaan jälleen selaimeen.

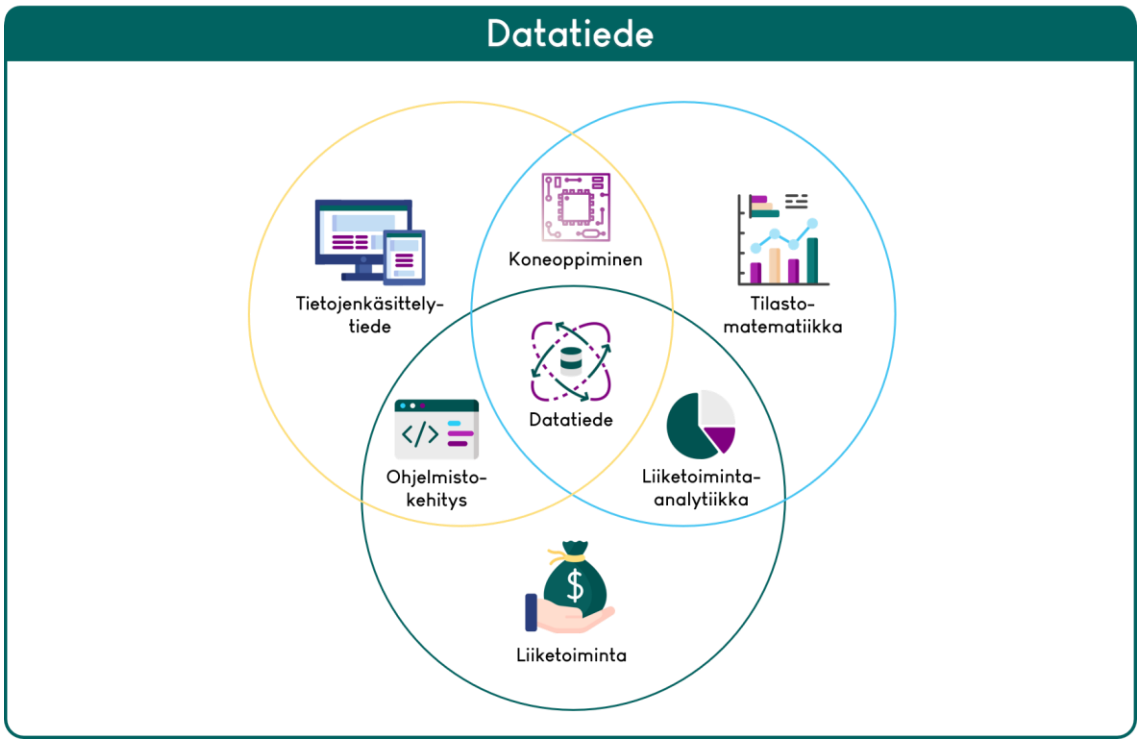
5.3 Valinnat suosittelujärjestelmän toteutuksessa

Suosittelujärjestelmissä käytettävät suosittelualgoritmit ovat osa data-analytiikan menetelmiin kuuluvaa tiedonlouhintaa. Niiden tehtävä on löytää verrattavista kohteista yhdenmukaisuuksia vertaajan tärkeiksi määrittelemien tai verrattavien yhteisten parametrien välillä. Lopputuloksena ennen satunnaisessa järjestyksessä oleva data on lajiteltu algoritmin vaatimusten mukaiseen, joko suoraan tai kääntäen suositeltavuusjärjestyksessä olevaan muotoon.

Tutustuimme projektin aikana erilaisiin suosittelutekniikoihin ja siihen, miten ne soveltuisivat omiin käyttötarkoituksiimme. Lisäksi pohdimme tekniikoiden toteuttamista omassa sovelluksessamme, ja miten pystyisimme luomaan yksinkertaisen algoritmin lajittelemaan käyttäjiä ja asuntoja käyttäjien antamien tietojen perusteella.

5.3.1 Data-analytiikka ja tiedonlouhinta

Data-analytiikka on osa datatieteen alaa, johon kuuluvat myös tekoäly ja koneoppiminen (kuva 9). Data-analytiikassa suuresta tietomäärästä pyritään tuottamaan sellaista informaatiota, jolla on arvoa ja jonka avulla voidaan tehdä hyödyllisiä johtopäätöksiä. Data-analyysiin kuuluu keskeisenä osana tiedon esittäminen visuaalisessa muodossa sekä erilaisten ennusteiden, todennäköisyyksien ja mallinnusten muodostaminen [29]. Digitalisaation myötä data-analytiikka on kasvanut merkittäväksi tekijäksi esimerkiksi liiketoiminnassa, jossa sitä hyödynnetään liiketoiminnan optimoinnissa, asiakaskokemusten kehittämisessä ja uusien palvelujen rakentamisessa. Tulevaisuuden menestyjiä ja edelläkävijöitä ovat ne, ketkä ymmärtävät tiedon ja edistyneen analytiikan arvon ja osaavat hyödyntää niitä tehokkaasti.



Kuva 9. Datatiede yhdistää tietojenkäsittelytiedettä, matematiikkaa ja liiketoimintaa. [30]

Tiedonlouhinta on puolestaan data-analytiikan osa-alue. Se on prosessi, jossa tavoitteena on erilaisin tiedonlouhinnan menetelmin löytää poikkeavuuksia, malleja ja korrelaatioita suurista tietojoukoista. Tuloksilla pyritään ennakointiin, jolla voidaan kasvattaa esimerkiksi liike-elämässä voittoja, vähentää kuluja ja parantaa asiakassuhteita. Tiedonlouhintamenetelmiä on useita, mutta yleisempiä niistä ovat assosiaatio, luokittelu, seurantamallit, regressio ja klusterointi. Tiedonlouhintaa voidaan käyttää koneoppimismallien rakentamiseen, joita käytetään edelleen tekoälyssä.

Tiedonlouhinta on tänä päivänä välttämätöntä, jotta valtavasta tietomäärästä saadaan kerättyä sellaista tietoa, joka on merkityksellistä tiedon käsittelijälle. Tiedonlouhinnan avulla ylimääräiset kaoottiset ja toistuvat kohinat suodattuvat pois, ja jäljelle jäävä merkityksellinen tieto jää hyödynnettäväksi.

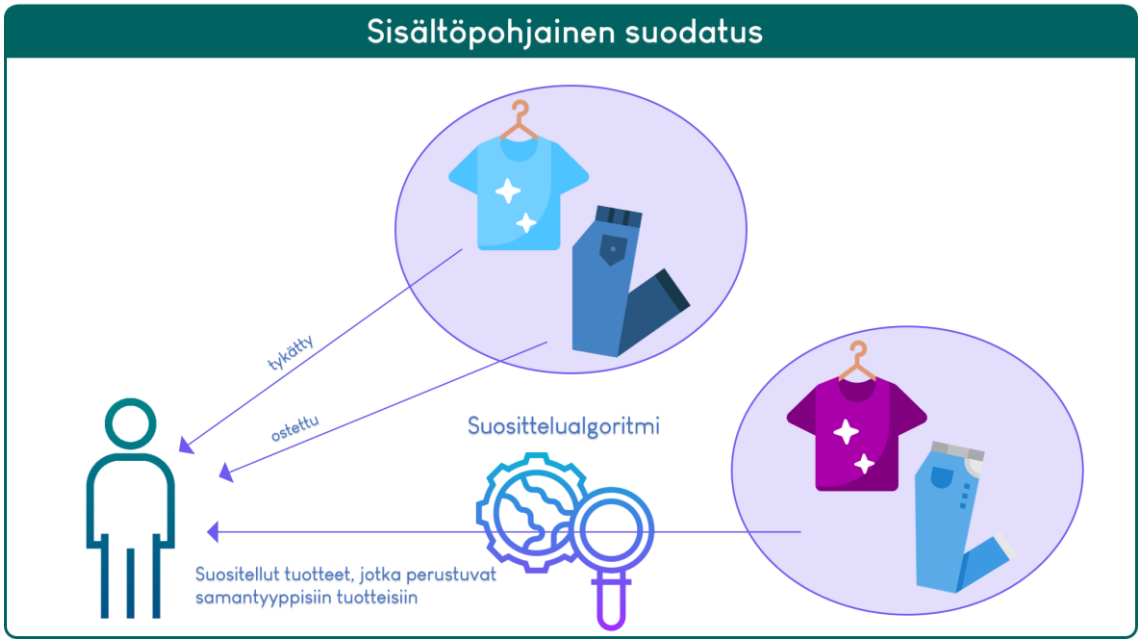
5.3.2 Suosittelujärjestelmät

Suosittelujärjestelmien käyttö on yksi tämän päivän megatrendeistä, joilla internetsivustot ja sovellukset ehdottavat käyttäjille sisältöjä, joista he saattaisivat pitää. Suureen suosioon nousseet suosittelujärjestelmät ovat tarpeellisia ja melkeinpä välttämättömiä liike-elämässä, kun Internetin lisääntyvä tiedon ja käyttäjien määrä kasvaa ja valinnanvaraa on liian paljon. Suosittelujärjestelmä tarjoaa erittäin tehokkaan tavan yrityksille tarjota käyttäjilleen kiinnostavaa sisältöä, ja näin personoida käyttäjille palveluitaan ja ratkaisujaan tuote- ja informaatioylikuormitetussa maailmassa.

Suosittelujärjestelmien ajatuksena on huomioida eri käyttäjäryhmien erityistarpeet ja -halut, jotka huomioidaan palveluita suunnitellessa. Suosittelujärjestelmässä käytettävä suosittelualgoritmi tai toiselta nimeltään suositusmoottori perustuu tiedon analysointiin. Suosittelualgoritmit keräävät käyttäjistä tietoja, kuten sijaintitietoja, profiilitietoja, tykkäyksiä ja katseluhistoriaa. Näiden avulla ne tarkkailevat, missä vietämme aikaa ja millaisista sivustoista, tuotteista tai postauksista pidämme. Tietojen perusteella algoritmi suodattaa turhan informaation pois.

Suosittelujärjestelmät voidaan jakaa karkeasti kolmeen eri tekniikkaan: sisältöpohjaiseen suodatukseen, yhteistoiminnalliseen suodatukseen ja tietopohjaiseen järjestelmään [31]. Sisältöpohjainen suodatus (*content-based filtering*) perustuu yhden käyttäjän vuorovaikutukseen ja mieltymyksiin (kuva 10). Suositukset perustuvat metatietoihin, jotka on kerätty käyttäjän Internet- tai sovellushistoriasta sekä sovelluksen sisällön

kuvauksesta. Historiaa analysoimalla algoritmi tarkastelee vakiintuneita malleja käyttäjän valinnoissa ja käyttäytymisessä ja vertailee niitä eri sisältöjen kuvauksiin. Esimerkiksi verkkokaupat käyttävät laajasti tätä tekniikkaa suositellakseen tuotteita ja palveluita, jotka liittyvät käyttäjän tykkäyksiin ja katseluhistoriaan. Mitä enemmän tietoa käyttäjä antaa, sitä suurempi tarkkuus suosituksilla on.

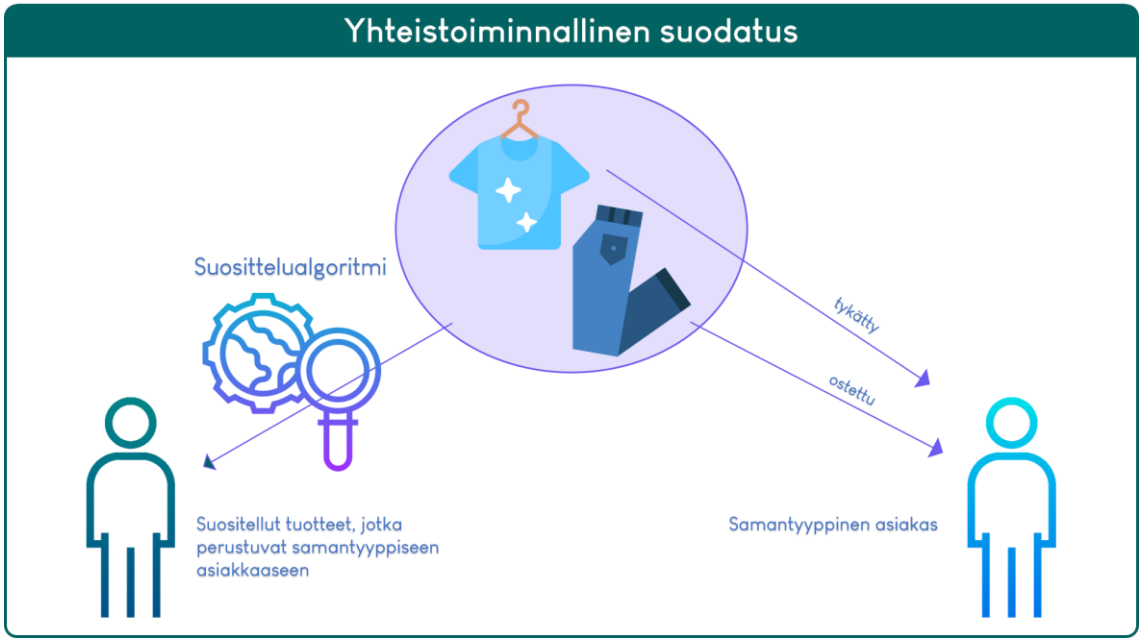


Kuva 10. Sisältöpohjaisessa suodatuksessa huomioidaan käyttäjän aiemmat toiminnat.

Yhteistoiminnallinen suodatus (*collaborative filtering*) perustuu yhden käyttäjän tietojen sijaan monien muiden käyttäjien vuorovaikutukseen, joiden perusteella yksittäinen käyttäjä saa ehdotuksia (kuva 11). Analysoitavasta tiedosta haetaan käyttäjiä, joilla on samanlainen maku tai tilanne. Tämän perusteella yhdelle käyttäjälle suositellaan muiden saman mieltymyksen omaavien käyttäjien käyttäytymisen perusteella sisältöä. Yhteistoiminnallista suodatusta käytetään esimerkiksi silloin, kun käyttäjälle suositellaan tuotetta, joka sopii hyvin toisen tuotteen kanssa. Tällöin suodatus ennakoi ja suosittelee muita tuotteita, joita muut saman maun omaavat käyttäjät ovat valinneet, eli kun käyttäjä ostaa tuotteen, hän tulee suurella todennäköisyydellä ostamaan myös tietyn toisen tuotteen.

Yhteistoiminnallisen suodatuksen tarkkuus on yleensä parempi kuin sisältöpohjaisen suodatuksen, sillä se perustuu suurempaan tietomäärään. Toisaalta tiedon määrän kasvaessa liian suureksi suodatuksen tehokkuus heikkenee. Jos kerättyä tietoa on vähän, on yhteistoiminnallinen suodatus erityisen heikko. Ilman merkityksellistä tietoa toisista

käyttäjistä, on algoritmin luonnollisesti vaikeampi suositella yksittäiselle käyttäjälle sisältöä.



Kuva 11. Yhteistoiminnallinen suodatus perustuu muihin käyttäjiin, joilla on samanlainen maku tai tilanne kuin kohdekäyttäjällä.

Tietopohjaisissa järjestelmissä (*knowledge-based system*) ehdotukset perustuvat asiantuntemukseen ja tietoon sekä käyttäjän tarpeisiin, eli tietyt säännöt määrittelevät kontekstin jokaiselle suositukselle. Sen ei oletusarvoisesti tarvitse käyttää käyttäjän historiaa, mutta se voi sisältää näitä sekä tuotteita, palvelumääritteitä ja muita asiantuntijatietoja. Tietopohjaisen järjestelmän rakentaminen on yleensä kallista, ja se sopiikin paremmin monimutkaisille verkkotunnuksille, joissa saatavilla olevaa tietoa ei ole paljon. Järjestelmän etuna onkin nimenomaan se, ettei se kärsi kylmäkäynnistysongelmista eli tiedon vähyydestä samalla tavalla kuin muut yllä mainitut tekniikat.

Huonona puolena suosittelualgoritmeissa voidaan pitää sitä, että emme itse pysty päättämään, mitä näemme tai mitä emme näe. Syntyy informaatiokupla (*the digital information bubble*), jossa tieto on hyvin yksipuolista ja käyttäjälle mieluisaa. Ristiriitaiset tiedot ja muiden ihmisten eriävät näkökulmat ja mielipiteet jäävät kuplan ulkopuolelle, jolloin lisääntyvän tiedon sijasta lisäämmekin tietämättömyyttämme. Vastakkaisten näkökulmien puute lisää tutkitusti esimerkiksi ennakkoluuloja sekä heikentää luovaa ajattelua ja motivaatiota oppia uusia asioita. [32.]

Suosittelujärjestelmiin liittyy lisäksi muita eettisiä kysymyksiä, kuten käyttäjien huomion saaminen hinnalla millä hyvänsä, riippuvuuden aiheuttaminen sekä yksityisyydensuojan loukkaaminen [33]. Suosittelujärjestelmien perimmäinen tavoite on saada käyttäjän huomio sekä pitää käyttäjä palvelussa pitkään. Mitä pidempään käyttäjä saadaan pysymään kiinnostuneena, sitä enemmän yritys saa rahaa esimerkiksi ostotapahtumista sekä mainoksista. Kiinnostusta voidaan ylläpitää esimerkiksi jatkuvilla tuotesuosituksilla, automaattisen videotoiston avulla sekä esittämällä yhä radikaalimpaa sisältöä.

Personoidut suositukset vaativat tyypillisesti henkilökohtaisten tietojen keräämistä analysointia varten, mikä tekee käyttäjistä alttiita yksityisyydensuojan rikkomiseen liittyville ongelmille. Useimmat tiedot paljastavat käyttäjien henkilökohtaisia tietoja sekä kiinnostuksen kohteita suosittelijalle. Lisäksi kerättyjä tietoja voidaan myydä kolmansille osapuolille ilman käyttäjän suostumusta. Myös tietojen vuotaminen on iso ongelma, kun eri alustoja hakkeroidaan. Alustojen onkin välttämätöntä kehittää käytäntöjä rikkomusten välttämiseksi. Erilaiset salaus- ja satunnaistamistekniikat yksityisten tietojen suojaamiseksi ja säilyttämiseksi sekä käyttäjien ryhmittely samanlaisten piirteiden mukaan poistaen henkilötiedot, olisi suositeltavaa. Tällä tavalla alustoille ja mainostajille tärkeät ominaisuudet säilyisivät tinkimättä kuitenkaan käyttäjien yksityisyydestä.

Huolimatta huonoista puolista, suosittelualgoritmit ovat erittäin yleisiä. Niiden käyttö on yrityksille tuottoisaa, sillä ne voivat merkittävästi lisätä käyntejä sivustolla sekä tuloja, klikkauksia, keskusteluja ja muita tärkeitä mittareita. Käyttäjät sitoutuvat sivustoon paremmin, kun suosittelualgoritmit auttavat käyttäjiä löytämään tehokkaasti heille sopivia palveluita ja tuotteita valikoimasta, joka olisi laajuudeltaan muuten hyvin vaikea hallita. Personoidut sisällöt esimerkiksi verkkokaupassa johtavat huomattavasti useammin ostopäätökseen verrattuna personoimattomaan sisältöön. Suosittelualgoritmi muuttaa käyttäjät asiakkaiksi, minkä lisäksi se kasvattaa asiakkaiden tilauksen arvoa sekä ostotapahtumien tuotteiden lukumäärää. Onnistuneella personalisoinnilla, suosittelualgoritmilla, on erittäin myönteisiä vaikutuksia käyttäjäkokemukseen, mikä lisää asiakastyytyväisyyttä ja asiakasuskollisuutta.

Asunnonvälityssovelluksen tyypillisen käytön määräaikaisen luonteen vuoksi sovelluksemme suosittelutoiminnallisuuden täytyy olla aggressiivinen. Tätä varten käyttäjälle tarjotaan tilaisuus antaa itsestään mahdollisimman paljon luokiteltavissa olevaa tietoa sovelluksen suosittelualgoritmin käyttöön. Tyypillinen käyttäjäelinkaari kämppistä tai asuntoa etsiessä päättyy sopivan kämppiksen tai asunnon löytymiseen. Tämä voi tapahtua

nopeastikin, joten emme voi luottaa sisältöpohjaiseen suodatukseen. Myöskään yhteistoiminnallista suodatusta ei voida hyödyntää, sillä asuntotarjonta sekä yksittäisen käyttäjän toiveet ja tarpeet kämppiksestä tai asunnolta voivat vaihdella suuresti.

5.3.3 Jaccard-indeksi

Halusimme kokeilla erilaisia algoritmeja, jotka laskevat vertailtavien otosten samankaltaisuutta. Asiaa tutkiessamme, vastaan tulivat esimerkiksi kosinin samankaltaisuus ja Jaccard-indeksi. Tarkemman tarkastelun tuloksena päädyimme siihen, että Jaccard-indeksi (kaava 1) tulisi sopimaan palveluumme paremmin.

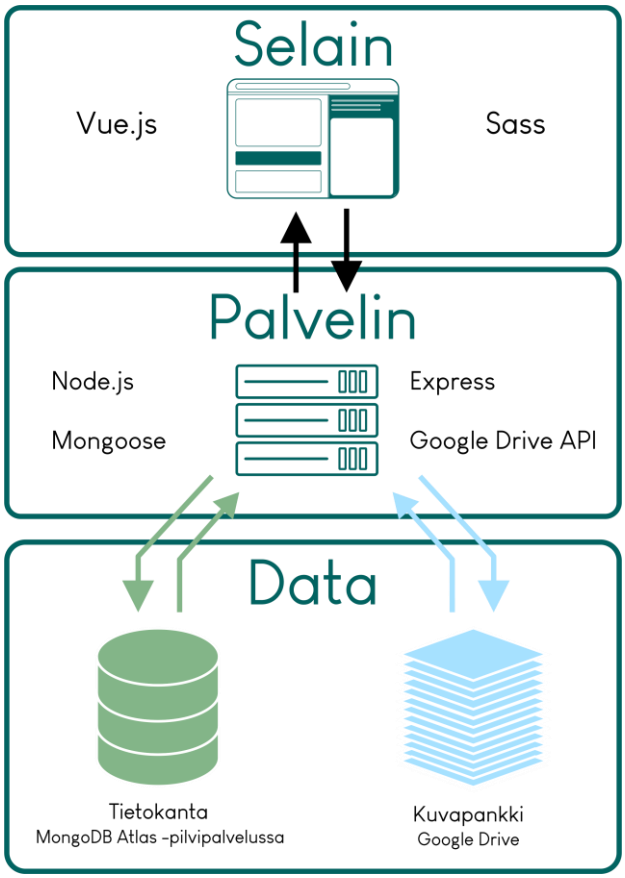
$$J(A,B) = \frac{|A \cap B|}{|A \cup B|}$$

Kaava 1. Jaccard-indeksi. Vertailtavien arvojoukkojen leikkaus jaettuna vertailtavien arvojoukkojen unionilla.

Jaccard-indeksiä, joka tunnetaan myös nimellä Jaccardin samankaltaisuuskerroin, käytetään erilaisten näytesarjojen samankaltaisuuden mittaamiseen. Jaccard-indeksi laskee leikkauksen suhteen unioniin. Indeksi on maksimissaan yksi, jolloin otokset ovat täysin samanlaiset, ja minimissään nolla, jos samankaltaisuutta ei ole. Jaccard-indeksiä käytetään paljon tietotekniikassa sekä muilla tieteen aloilla tietokokonaisuuksien samankaltaisuuden määrittämiseen. [34.]

6 Toteutus

Palvelumme koostuu selainkäyttöliittymästä, palvelinsovelluksesta sekä palvelimen hallinnoimasta tietokannasta ja kuvapankista (kuva 12). Prototyyppitoteutuksessamme poikkeamme hieman normaalista tavasta pitää sovelluksen tietokanta sekä kuvatiedostot itse palvelimella. Erillistä palvelinkonetta hyödyntäen toteutus olisi luultavasti huomattavasti suoraviivaisempi ja suorituskykyisempi, mutta tässä vaiheessa projektia vaadittavat toiminnot tai niiden luonnokset saatiin toteutettua hyvin näinkin. Lisäksi on ollut mielenkiintoista nähdä, miten pitkälle tämänkaltaisessa kehitysprojektissa pääsee hyödyntämällä ilmaisia pilvipalveluita.



Kuva 12. Palvelun arkkitehtuuri.

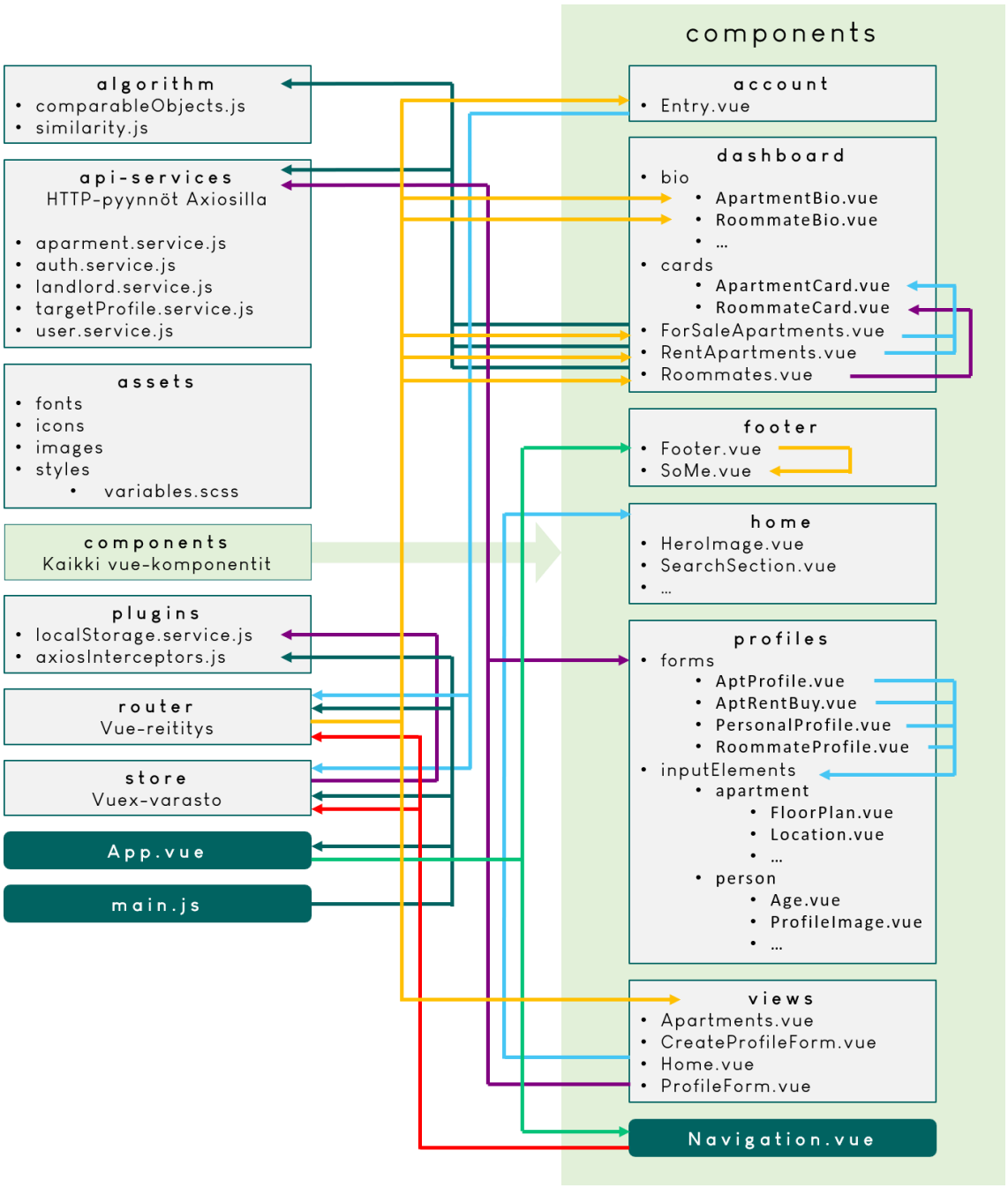
6.1 Selaintoteutus

Selaintoteutuksen tekeminen alkoi huomattavasti ennen palvelinpään toimintojen pystyttämistä. Sivustolle laadittiin väripaletti, johon kerättiin kokoelma sivustolla käytettäviä värejä. Myös sivuston typografiaan otettiin kantaa jo tässä vaiheessa. Peruslaatuiset osat sivuston visuaalisessa ilmeessä sekä sivuston arkkitehtuurissa ja sisällössä on hyvä lyödä lukkoon mahdollisimman aikaisessa vaiheessa, jotta aikaa ei myöhemmin kulu jo olemassa olevan designin ja sisällön muuttamiseen tai uudelleenmäärittämiseen.

6.1.1 Vue-komponenttirakenne

Vuen komponenttirakenne osoittautui projektin aikana miellyttäväksi. On luontevaa, että komponentin HTML-runko, toiminnalliset skriptit ja tyylimäärittelyt löytyvät samasta tiedostosta. Tämä kannustaa kehitystyössä ajattelemaan toimintokokonaisuuksia eri komponentteina. Yksinkertainen sovellusrakenne auttoi pitämään tiedostorakenteen (kuva

13) selkeänä ja se vähensi komponenteissa yksittäisten muutoksia kaipaavien rivien etsimiseen kuluvaa aikaa, joka on arvokasta etenkin rivimäärien kasvaessa.



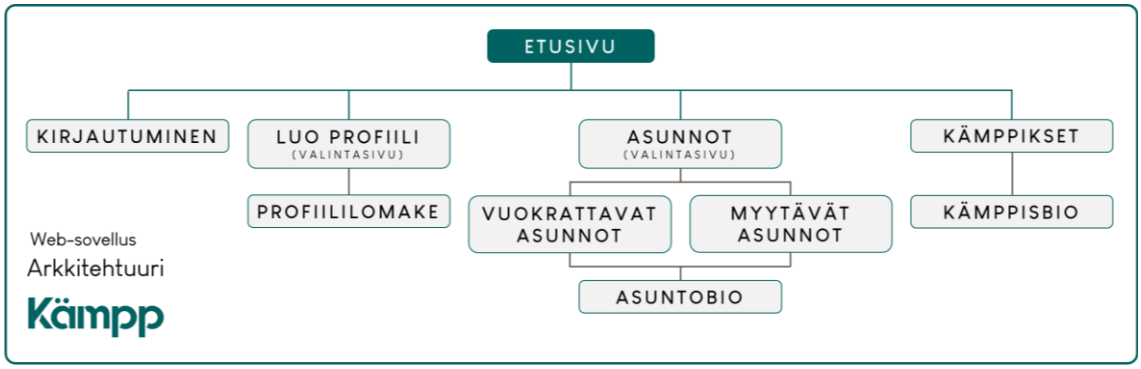
Kuva 13. Kämpp-palvelun JavaScript-tiedostojen ja Vue-komponenttien suhde toisiinsa nähden. Harmaat neliöt kuvaavat kansioita, ja niiden sisällä olevat luettelomerkityt nimet alikansioita ja tiedostoja.

Sovelluksen tämänhetkinen Vue-komponenttirakenne pysyi mielestämme hyvin järjestyksessä ja helposti hallittavissa. Jatkokehitystä varten olisi varmasti suositeltavaa

käyttää vielä enemmän esimerkiksi Vuex-varaston palvelua, jotta propsien välittämistä voitaisiin vähentää ja näin selkeyttää tietojen jakamista sovelluksen sisällä.

6.1.2 Sivuston arkkitehtuuri ja sisältö

Palvelumme web-sivusto koostuu tällä hetkellä etusivun lisäksi neljästä alasivusta (kuva 14). Navigointivalikossa on vaihtoehdot kirjautumiselle, profiilin luomiselle sekä asuntojen ja kämppisehdokkaiden listausnäkymälle. Vaihtoehtojen näkyvyys riippuu siitä, onko käyttäjä kirjautunut sisään vai ei. Kirjautumaton käyttäjä ei voi selailla kämppisehdokkaita eikä luoda profiilia. Vielä toteuttamattomien alasivujen linkit, kuten tietosuojaseloste ja yhteystiedot, sijoitimme sivuston alatunnisteeseen.

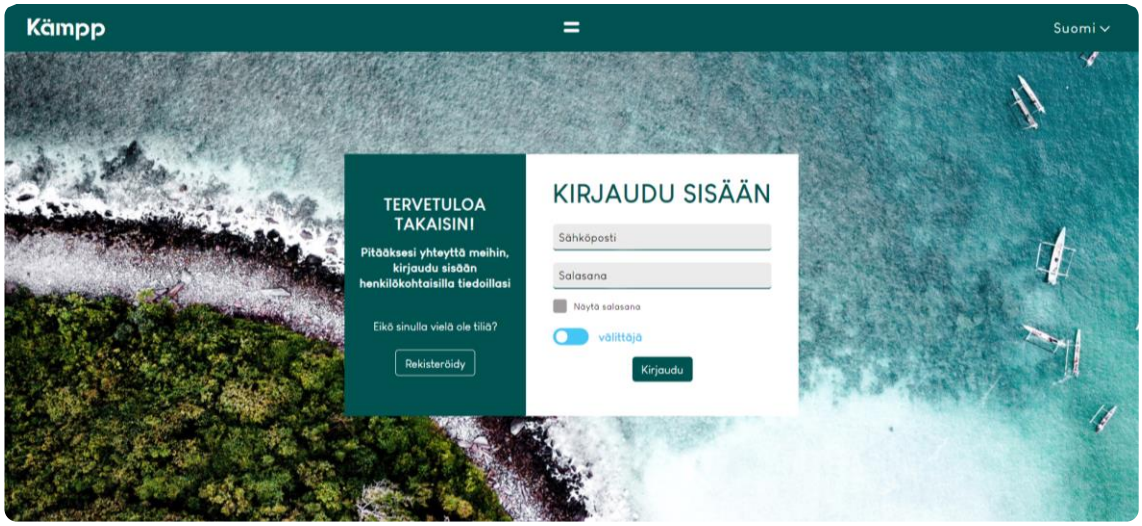


Kuva 14. Kämpp-palvelun sivuarkkitehtuuri.

Etusivulla on heti ensimmäiseksi asuntonosto sekä hae-kenttä asunnoille, joiden avulla käyttäjää johdatetaan käyttämään sivustoa. Sivuston yläpalkissa on sivuston nimi, navigointivalikko sekä sivuston kielivalinta. Poikkeuksellisesti päätimme piilottaa sivuston navigointivalikon kuvakkeen taakse myös laajemmassa näkymässä mobiilinäkymän lisäksi. Ratkaisulla halutaan pitää näkymä mahdollisimman rauhallisena ilman ylimääräistä informaatiota. Piilotettu valikko aukeaa viemällä kursori kuvakkeen päälle eli käyttäjän ei tarvitse tehdä ylimääräistä hiiren klikkausta. Yläpalkki häviää sivua vierittäessä alaspäin ja tulee näkyville ylöspäin vieritettäessä. Tällä tavalla annamme mahdollisimman paljon lisää tilaa sivuston näkymälle silloin, kun käyttäjä ei yläpalkkia tarvitse.

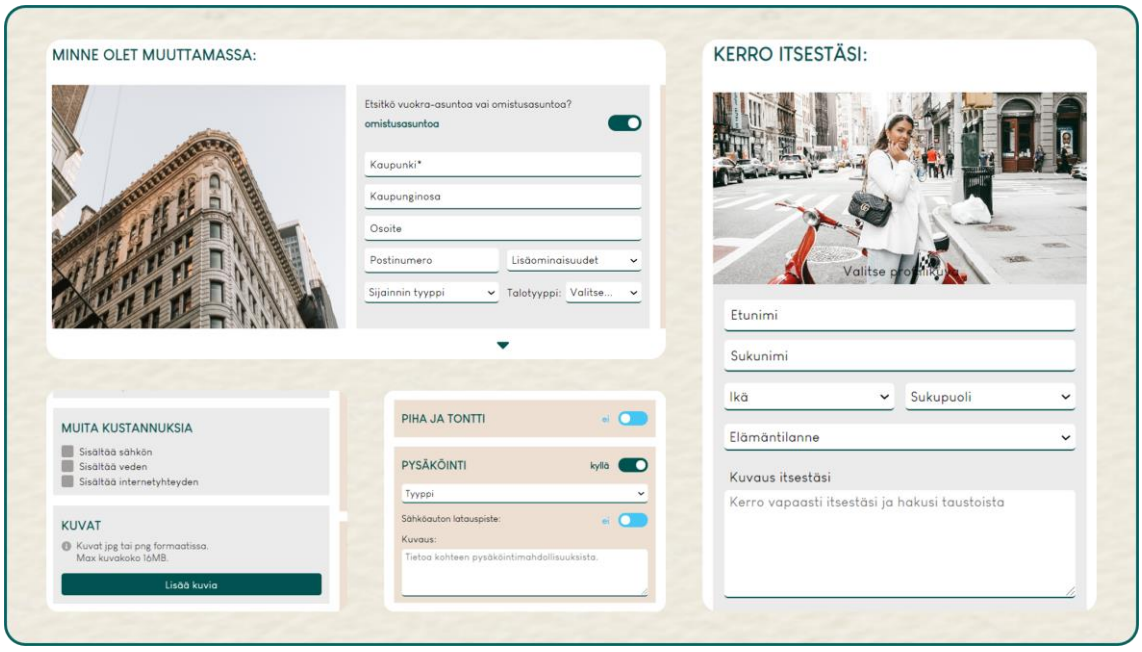
Kirjautumissivulla tapahtuu sivustoon kirjautuminen sekä sivustolle rekisteröityminen (kuva 15). Halusimme sijoittaa nämä molemmat samalle sivulle, jotta sivuston sivurakenne pysyisi mahdollisimman pienenä ja selkeänä. Näkymän vaihto kirjautumisen ja

rekisteröitymisen välillä tapahtuu napin painalluksesta, joka vaihtaa kolumnien järjestyksen ja sisällöt sekä oikean lähetyslomakkeen. Kahden eri käyttäjätyypin määrittely tapahtuu rekisteröitymisen yhteydessä, ja molemmat käyttäjätyypit kirjautuvat palveluun sisään samasta kirjautumisikkunasta. Kirjautumissivun taustakuva tehtiin sivun päivityksen mukaan vaihtuvaksi, jotta sivustolle saatiin elävyyttä.



Kuva 15. Kirjautumissivun näkymä kirjauduttaessa sivustolle.

Profiilinluomissivulla puolestaan luodaan profiililomakkeen sisältö (kuva 16). Profiilinluomissivulla valittu kämppis-, asunto- tai kämppis ja asunto -vaihtoehdon valinta määrää, mitä Vue-komponentteja profiililomakkeessa näytetään. Profiililomakkeen kaikki samaa kokonaisuutta olevat syötekentät on tehty omiin komponentteihinsa. Syötekenttäkomponenteista emitoidaan syötetyt tiedot profiililomakkeeseen, josta ne kootaan oikeanmuotoiseen JSON-olioon tietokantaan lähettämistä varten.



Kuva 16. Otteita profiililomakkeen eri osioista. Ylimpänä vasemmalla on ote kämppishakuun vaadittavasta asunto-osiosta kuvituskuvineen. Sen alla on otteita asunnon myynti-ilmoituslomakkeesta, ja oikealla ote oman henkilökohtaisen profiilin lomakeosiosta, johon voi halutessaan liittää itsestään profiilikuvan. Yksinkertaisemman ilmeen luomiseksi osassa syötekentissä on käytetty *label*-tekstiä, joka on muotoiltu syötekentän jatkeeksi erillisen syötekentän yläpuolella sijaitsevan *label*-tekstin sijaan. Samannäköinen ilme saataisiin toteutettua myös siten, että *label*-teksti jätettäisiin kokonaan pois, ja sen tilalla käytettäisiin *placeholder*-tekstiä. Tällainen ratkaisu kuitenkin heikentäisi merkittävästi saavutettavuutta, joka on nykypäivänä erityisen tärkeää huomioida.

Navigointivalikossa sijaitsevasta kämppikset-vaihtoehdosta pääsee suoraan selailemaan kämppisehdokkaita, mutta asunnot-vaihtoehdosta tulee ensin valita haettavien asuntojen tyyppi eli haluaako käyttäjä selailla myytäviä vai vuokrattavia asuntoja. Asuntojen ja kämppisehdokkaiden listaukset käyttävät samaa visuaalista tyyliä. Vaikka tyyli on täysin sama, sijoitimme listaukset omiin komponentteihin, jotta komponenttien sisällöt pysyisivät yksinkertaisia. Liiallinen *if-else*-rakenne kolmen eri sisällön näyttämiseen ei olisi palvellut meitä mitenkään järkevästi. Listausnäkymän kämppisehdokasta tai asuntoa klikkaamalla pääsee kyseisen kämppiksen tai asunnon bioon, jossa on yksityiskohtaisempi kuvaus kohteesta.

6.2 Palvelintoteutus

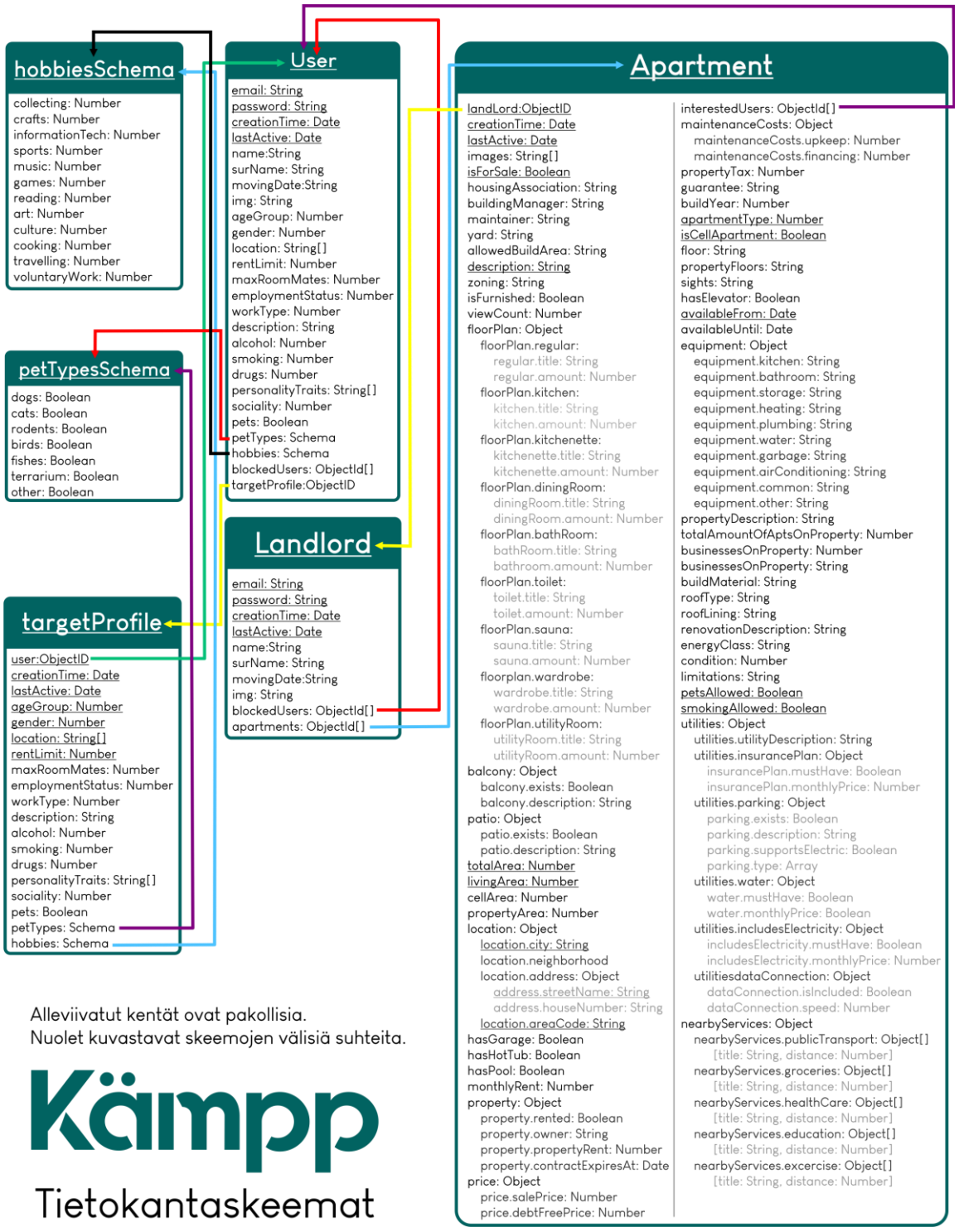
Oleellinen osa web-palvelumme rakennetta on palvelinpuolen toteutus. Se käsittelee suurimmaksi osaksi tietokantaa, johon sovelluksemme tallentaa käyttäjiensä profiilitietoja. Koska projektimme on luonteeltaan vielä prototyyppi, emme ole hankkineet erillistä serveri-infrastruktuuria palvelinkoodin alustaksi. Projektia kehittäessämme pyöritimme palvelintoteutusta selaintoteutuksen rinnalla omilla tietokoneillamme. Työryhmämme kesken tilaltaan yhtenäisenä pysyvän tietokannan olemme pystyttäneet ilmaiseen pilvipalveluun, ja sovelluksen käyttökokemuksen kannalta olennaiset kuvat varastoidaan Google Driveen Googlen rajapintaa hyödyntämällä.

Palvelussamme palvelimen tarkkailemat reitit määritetään expressillä. Luotuihin express-reitteihin lisäsimme erilliset ohjainfunktiot, jotka käsittelevät reitteihin saapuvia pyyntöjä ja niiden sisältämää dataa asianmukaisesti. Rajapintaan lisäsimme myös muita pakkauksia npm-paketinhallintapalvelusta.

6.2.1 Tietorakenteet

Päädyimme toteuttamaan palvelua varten neljä skeematyyppistä päätietorakennetta ja kaksi tukiskeemaa (kuva 17). Päätietorakenteita noudattavat oliot mallinnetaan kokoelmina sovelluksen MongoDB-tietokannassa.

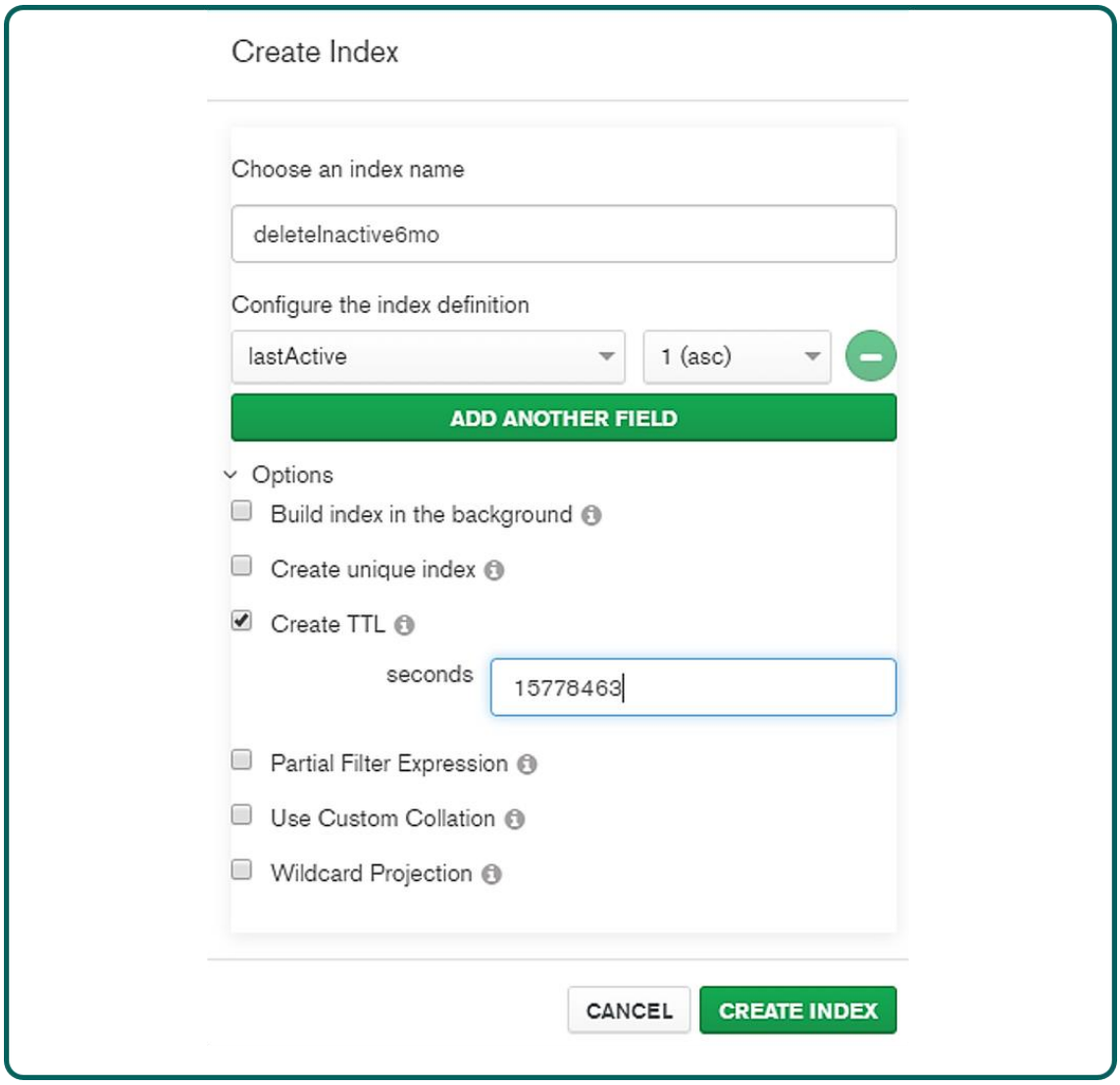
Tietorakenne Apartment kuvaa asuntoa ja sen myynti-ilmoitusta. Käyttäjämalli User on tehty kämppistä tai asuntoa etsiville käyttäjille, ja se sisältää erilaisia tietueita, joiden avulla käyttäjä voi kertoa itsestään. Landlord-käyttäjämalli on tehty asunnonomistajille, jotka haluavat luoda asunnostaan vuokra- tai myynti-ilmoituksia palveluun. Viimeinen malli on targetProfile, johon User-käyttäjä määrittelee tulevalta kämppikseltään toivomia piirteitä. TargetProfile-mallia käytetään myös palvelun suosittelualgoritmissa. Algoritmi vertaa muiden palvelun käyttäjien profiileja kämppistä hakevan käyttäjän targetProfileen, ja listaa profiilit yhtenevyyden mukaan. Lisäksi käyttäjien lemmikkejä ja harrastuksia käsittelevät tietorakenteet on eritelty omiksi sivuskeemoikseen, sillä rakenteet toistuvat sekä User- että targetProfile-tietorakenteissa.



Kuva 17. Sovelluksen Mongoose-tietokantaskeemat ja niiden suhteet.

Tietueiden eheyden varmistamiseksi laadimme skeemoihin validaatiosääntöjä. Säännöt voivat koskea esimerkiksi tietueen jonkin kentän pakollisuutta tai arvon, kuten id-arvojen, uniikkiusvaatimusta. Validaation rikkova tietue ei pääse lainkaan tietokantaan, ja Mongoose palauttaa virheen kyseisissä tapauksissa. Validaatiosääntöjen lisäksi

hyödynnämme TTL-indeksiä. Puoli vuotta muokkaamattomana tai inaktiivisina säilyneet dokumentit poistetaan automaattisesti (kuva 18).



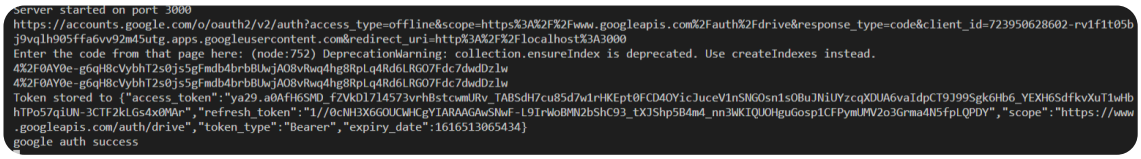
Kuva 18. TTL-indeksin määrittäminen MongoDB Compass -tietokantatulkin avulla.

TTL-indeksin tarkoituksena on poistaa palvelumme tietokannasta ylimääräistä dataa, jota käyttäjät eivät enää käytä. Ei ole järkevää säilöä yli puoli vuotta inaktiivisena olevia tietoja, koska mitä enemmän tietokanta sisältää tietoa, sitä enemmän se vie muistia palvelimelta. Tietojen poistamisen määräajaksi määrittelimme puoli vuotta, koska se on mielestämme riittävän pitkä aika odottaa käyttäjän kirjautuvan uudelleen palveluumme. Jos käyttäjä kirjautuu uudelleen palveluumme vasta yli puolen vuoden kuluttua, on käyttäjän elämäntilanne ja mieltymykset mitä luultavammin muuttuneet eivätkä aiemmat tiedot enää vastaisi uutta tilannetta.

6.2.2 Google Drive API ja Google Driven käyttö web-palvelun kuvahakemistona

Normaalisti verkkopalveluiden käyttäjien tuottamaa dataa säilötään joko palvelimen tallennustilassa tai jossakin palvelinkäyttöön optimoidussa pilvitallennustilassa. Koska kehitystietokantamme sijaitsee jo ilmaispalvelussa, oli luontevaa lähestyä kuvien säilömistä samalla tavalla. Google Driven käyttöä varten luotiin projektille oma yritystili, ja sille oma *service account* -tili, joka on valtuutettu käyttämään Drivea yritystilin onnistuneen OAuth 2.0 -verifikaation jälkeen.

Palvelinsovelluksen autentikointi tapahtuu sovelluksen käynnistyessä. Ohjelma tulostaa sovelluksen konsoliin URL-osoitteen, jota seuraamalla pääsee autentikoitumaan Googlen palveluihin (kuva 19). Autentikoitumisen onnistuttua selaimessa esitetään koodi, joka on kopioitava palvelinsovelluksen konsoliin. Jos palvelinkoneella ei ole asennettuna web-selainta, voi URL-osoitteeseen mennä suorittamaan autentikoitumisen toisella laitteella ja kopioida koodi käsin palvelimen konsoliin. Koodin liittämisen jälkeen saadaan Driven käyttöä varten tarvittavat access ja refresh tokenit. Nämä suositellaan tallentamaan palvelinkoneen muistiin, mutta koekäytössä olemme tallentaneet ne suorituksenaikaisiin ympäristömuuttujiin.



Kuva 19. Google-autentikaatio palvelimen käynnistyksen yhteydessä. Autentikaatiopalvelu huolehtii itsenäisesti tokeneiden käyttämisestä ja vanhentuneiden tokeneiden uusimisesta, kunhan ne ovat tallennettuna ohjelmalle määriteltyyn paikkaan.

Onnistuneen autentikaation jälkeen muunnamme Google-autentikoinnin globaaliksi, jolloin pyyntökohtainen uudelleen autentikoituminen on tarpeetonta. Sekä autentikaation että itse Driven käytön mahdollistaa npm:n googleapis-paketti [35]. Driveen loimme kansion sovelluksellemme, johon kuvakansiot tulevat. Luodun kansion asetimme ohjelmakoodissa kuvasarjakohtaisten kansioiden emokansioksi (esimerkkikoodi 2). Projektin edetessä olemme suojanneet eri kuvakansioiden tunnisteet siirtämällä ne ympäristömuuttujiin.

```
exports.createFolder = async (name) => {
     var fileMetadata = {
          name: name,
          mimeType: "application/vnd.google-apps.folder",
          parents: ["11YHNVJmAv7FRX2huo_9Q5JJy8u3BHJDM"],
     }
     const drive = google.drive({ version: "v3" })
     try {
          const res = await drive.files.create({
               resource: fileMetadata,
               fields: "id",
          });
          return res.data.id
     } catch (err) {
          return err
     }
}
```

Esimerkkikoodi 2. Kansion luominen Google Driveen ohjelmallisesti. Parents-kenttä määrittää mahdollisen emokansion. Onnistuneen suorituksen jälkeen funktio palauttaa luodun kansion id:n, jota voidaan käyttää lisättävien kuvien parent-kansiona.

Kuvan lataus kansioon toteutetaan samantyylisesti kuin kansioiden luominen (esimerkkikoodi 3). Kuvadatasta kerätään kuvan metatiedot, mediatyyppi sekä itse kuvan raakatieto:

```
exports.uploadToFolder = async (folderId, photoName, photo) => {
     try {
          var fileMetadata = {
               name: photoName,
               parents: [folderId],
          }
          var media = {
               mimeType: photo.mimetype,
               body: bufferToStream(photo.buffer),
          }
          const drive = google.drive({ version: "v3" })

          const res = await drive.files.create({
               resource: fileMetadata,
               media: media,
               fields: "id",
          })
          return res
     } catch (err) {
          return err
     }
}

---

function bufferToStream(buffer) {
     var stream = new readable()
     stream.push(buffer)
     stream.push(null)

     return stream
}
```

Esimerkkikoodi 3. Kuvan lisääminen sekä *bufferToStream*-funktio.

Lisättävän kuvan *buffer*-ilmaisumuoto on muutettava *stream*-tyyppiseksi, jotta Google Drive API osaa käsitellä sitä. Streamiin eli virtaan syötetään *readable.push*-metodilla ensiksi kuvan *buffer*-muoto, ja sen jälkeen *null*, joka on merkki virran sulkemiselle. [36.]

Kuvien lähettäminen asuntoa luodessa tuotti aluksi hankaluuksia. Kuvien tietomuodot eivät näkyneet palvelimelle saapuvassa JSON-rakenteessa ollenkaan. Asian selvittelyn jälkeen JSON-tietoja käsittelevän body-parser-väliohjelmiston lisäksi piti ottaa käyttöön *form-data/multipart*-enkoodaustyyppiä käsittelevä multer-väliohjelmisto. Multer käsittelee vain edellä mainittua *multipart*-enkoodaustyyppiä olevat kutsut, joten body-parser voi jatkaa muiden, pelkkää JSON:a sisältävien pyyntöjen käsittelyä. Multer toimittaa palvelimelle normaalin JSON-muotoisen sisällön tuttuun tapaan request.body-muuttujaan tallennettuna (kuva 20). Lisätyt kuvatiedostot toimitetaan pääoliosta erillään muuttujassa request.files.

Kuva 20. Palvelinsovelluksessa konsoliin tulostettuina asunnon lisäävän pyynnön asuntotiedot omaava muuttuja, sekä lisänä tulevat kaksi kuvatiedostoa.

Kuvatieto käsitellään sen jälkeen siten, että palvelin lisää JSON-kenttään tietoja asuntoilmoituksen luomisajankohdasta ja sen luoneesta käyttäjästä. Kuvat tallennetaan Google Driveen (kuva 21), ja niiden tallennuksen yhteydessä syntyvät kuvakohtaiset id:t palautetaan asunto-JSON:n images-kenttään. Tämän jälkeen itse asunto tallennetaan tietokantaan.

Kuva 21. Google Driveen asunnon luonnin yhteydessä tallennetut testikuvat. Kuvat tallennetaan kansioon, joka nimetään asuntoilmoituksen osoitteen ja ilmoituksen luoneen käyttäjän id:n mukaan.

Kuvien näyttäminen Drivesta on yksinkertaisempaa kuin kuvien lataaminen sinne. Kuvan emokansion jakoasetukseksi on asetettava ”Kaikki tämän linkin saaneet internetissä voivat katsoa”. Tämän jälkeen kuvaan voi viitata käyttämällä tiedoston *webContentLink*:ä. URL-osoite on mallia *https://drive.google.com/uc?id={tiedoston_id}&export=download*. Tällainen URL-osoite toimii suoraan HTML:n kuvaelementin lähteenä.

6.3 Suosittelualgoritmi

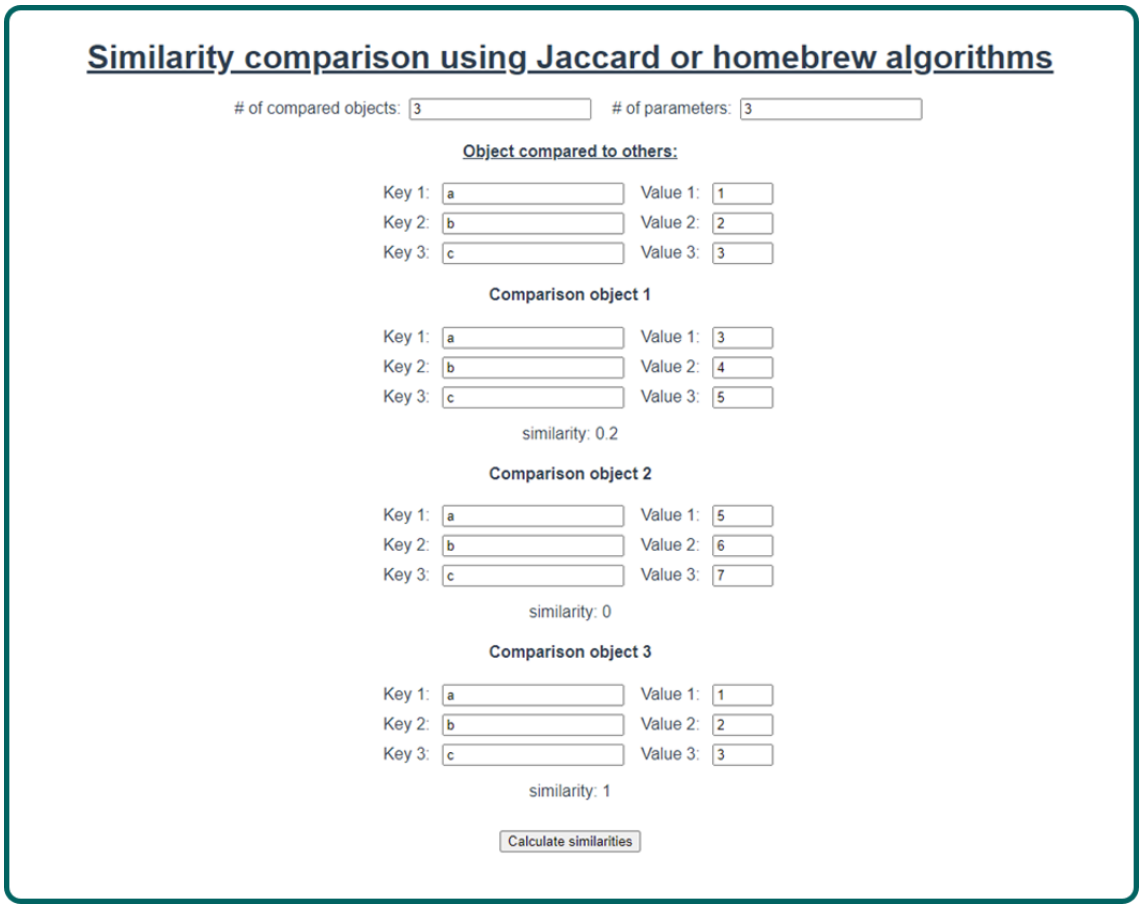
Yhtenä työmme tavoitteista oli pystyä vertaamaan käyttäjätietueiden numeroarvollisten kenttien vastaavuutta kämppäkaveria tai asuntoa etsivän käyttäjän määrittelemän toivekämppiksen tai toiveasunnon arvoihin. Jotta suositteluja voidaan tehdä, tarvitaan niiden arvojen samankaltaisuutta vertaileva algoritmi. Tällaisen toteuttamisesta meillä ei ollut aikaisempaa kokemusta, joten sopivan algoritmin määrittämiseksi teimme pienimuotoisen testisovelluksen. Tiedonhaun tuloksena ensimmäisenä kokeiluun valikoitui Jaccard-indeksi.

6.3.1 Jaccard-indeksin testisovelluksen toteutus

Algoritmista löytyi valmis npm-pakkaus [37], joka asennettiin testisovellukseen. Testisovelluksessa vertaillaan avain-arvoparikokoelmien samankaltaisuuksia. Sekä avain-arvoparien lukumäärää että verrattavien kokoelmien lukumäärää voi muuttaa. Lisäksi avaimien nimet ja arvot ovat vapaasti määriteltävissä.

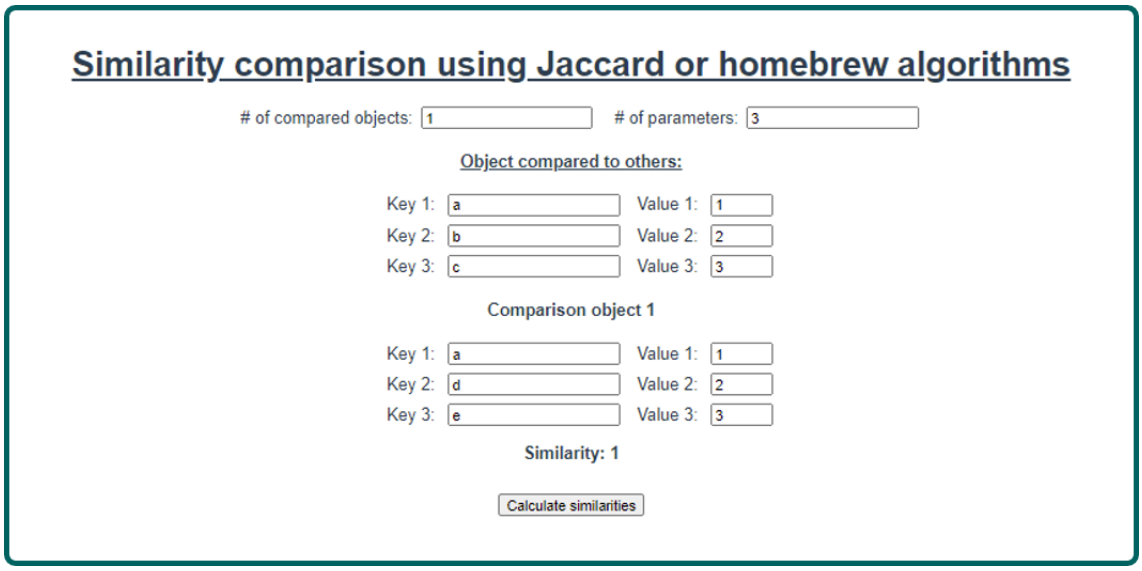
Testisovellus laskee samankaltaisuuden vain kohdeolion ja vertailuolion väliltä löydettyjen samannimisten avainten suhteen. Tällä tavoin voidaan pääsovelluksessa välttää tilanteet, joissa olioilla on eri lukumäärä kenttiä käyttäjän jätettyä täyttämättä omasta mielestään merkityksettömiä tietoja.

Testisovellusta ajettaessa, havaitsimme Jaccard-indeksin käyttämisessä ongelman (kuva 22). Kaikkien avainten täsmätessä, mutta arvojen ollessa erilaiset, indeksi määrittyy vain yhteisten arvojen mukaan. Kohdeolion ja vertailuolio 1:n arvojoukon {1, 2, 3, 3, 4, 5} viidestä eri numeroarvosta yksi esiintyy kahdesti, jolloin samankaltaisuuden tulokseksi saadaan *toistuvien arvojen lukumäärä / eri numeroarvojen lukumäärä = 0.2*. Vertailuolio 2:ssa avaimet täsmäävät, mutta arvojen ollessa erilaiset samankaltaisuutta ei Jaccard-indeksin kannalta ole. Vertailuolio 3 on täysin identtinen kohdeolion kanssa, ja samankaltaisuustulokseksi tuleekin yksi. Tällä tavoin havaitsimme Jaccard-indeksin puutteellisuuden käyttötarkoitukseemme nähden.



Kuva 22. Jaccard-indeksi testisovelluksessa, jossa kolme vertailtavaa oliota, ja niille laskettu samankaltaisuus.

Halusimme, että algoritmimme ottaa huomioon kaikki vertailuolioiden yhteiset avain-arvoparit arvojen ollessa erilaiset. Lisäksi Jaccard-indeksillä tulisi saada toteutuksessamme samankaltaisuuden arvoksi yksi, jos olioilla olisi vain yksi yhteinen kenttä, jonka arvo sattuisi olemaan sama (kuva 23). Käyttämämme algoritmin pitäisi siis laskea arvojen eroavaisuuksia sekä painottaa tuloksessa yhteisten avainten lukumäärää.



Kuva 23. Jaccard-indeksi testisovelluksessa tilanteessa, jossa olioilla on vain yksi yhteinen avain-arvopari ja jossa niillä on sama arvo.

6.3.2 Oman suosittelualgoritmin suunnittelu ja toteutus

Suosittelualgoritmin suunnittelu alkoi pohdinnalla siitä, minkälaisia eri matemaattisia laskutoimituksia kannattaa käyttäjien ja asuntojen tietoihin kohdistaa. Keräsimme tietokannasta kaikki tiedot taulukko-ohjelmaan, jossa karsimme ne tiedot pois, joita suosittelualgoritmiin ei tarvita. Näitä tietoja olivat esimerkiksi käyttäjän nimi, vapaat kuvaukset ja kaikki id-tunnisteet. Jäljelle jääneille tiedoille kokeilimme erilaisia laskentavaihtoehtoja ja katsoimme, minkälaisia tuloksia tietueille annetut eri arvot antavat. Tuloksissa oli paljon käsiteltävää ja yksi tärkeimmistä huomioista oli, ettei tietenkään kaikkia tietoja voida käsitellä samalla tavalla. Tiedot tulisi jakaa kahteen eri ryhmään, joihin kohdistettaisiin erilaiset matemaattiset operaatiot.

Toinen huomio kohdistui siihen, miten käsittelemme käyttäjiä, jotka ovat täyttäneet eri määrän tietoja. Koska aikaa oli rajallisesti, emme lähteneet syvällisemmin miettimään kaikkia mahdollisia tapauksia, mitä suosittelualgoritmin tulisi osata käsitellä. Ratkaisun yksinkertaistamiseksi päätimme kerätä ne kentät, joita toinen käyttäjä ei ole täyttänyt, mutta toinen on. Kerätyt kenttien nimet listataan tuloksen yhteyteen, jotta kämppistä tai asuntoa etsivä käyttäjä näkee, mihin suosittelualgoritmin tulos perustuu. Näin ei algoritmissa tarvitse huomioida täytettyjen kenttien lukumäärää ja puuttuvan datan imputoimista. Emme voi tietää, miksi käyttäjä on jättänyt tietoja tyhjiksi. Vastaamatta jättäminen ei kuitenkaan tarkoita sitä, etteikö käyttäjällä olisi mielipidettä kyseiseen kysymykseen.

Jos puuttuvia tietoja lähdetään täyttämään esimerkiksi keskiarvolla tai moodilla, voi korvaava arvo olla käyttäjälle virheellinen, jolloin sopivuus toiseen käyttäjään vääristyy.

Kun käsitys eri tietueista ja niiden arvoihin kohdistettavista operaatioista oli selvillä, lähdimme etsimään sopivia suosittelualgoritmeja. Tehdessämme tiedonhakua löysimme ajatuksia herättävän kommentin, josta inspiroituneena toteutimme ensimmäisen version suosittelualgoritmista [38].

Similarity is defined as the average difference in common metrics.

Suosittelualgoritmissa tullaan vertailemaan sisään kirjautuneen käyttäjän antamia tietoja muiden käyttäjien tietoihin. Ensimmäisessä versiossa vertailimme yksinkertaisia olioita, jotka sisälsivät vain numeroarvoja. Vertailtavista olioista eritellään ensin yhteisen avaimen omaavat avain-arvoparit. Taulukko, jossa on vain yhteiset avain-arvoparit, käydään läpi, jonka perusteella vertailtavat arvot siirretään avainten mukaisessa järjestyksessä omiin taulukoihinsa (esimerkkikoodi 4). Tällä tavalla varmistetaan, että algoritmi vertailee oikeita arvoja toisiinsa.

```
exports.calculateCustomSimilarity = (comparisonObjects,
     comparedObject) => {
     comparisonObjects.forEach((object) => {
          let commonKeys = this.getCommonKeys(comparedObject, object)
          let comparedValues = []
          let comparisonValues = []
          commonKeys.forEach((key) => {
               comparedValues.push(comparedObject[key])
               comparisonValues.push(object[key])
          })
          let similarity = this.customSimilarityIndex(
               comparedValues, comparisonValues)
          similarity ? object.similarity = similarity :
               object.similarity = 0
  })
}
```

Esimerkkikoodi 4. Funktio, joka erittelee vertailukohteista verrattavaan kohteeseen nähden yhteiset avain-arvoparit, lisää niiden arvot taulukoihin ja ajaa ne itse vertailun läpi.

Kun yhteiset avaimet on löydetty ja arvot siirretty omiin taulukoihin, tapahtuu varsinainen vertailu (esimerkkikoodi 5). Vertailussa lasketaan sitä, kuinka paljon arvot eroavat toisistaan. Ensin funktiossa lasketaan arvoparin erotus, jonka jälkeen erotuksen itseisarvo lisätään muuttujaan, johon summataan eroavaisuusarvo jokaisen arvoparin osalta. Kun

kaikki arvoparit on käyty läpi, jaetaan kyseisen muuttujan arvo vertailtujen arvoparien lukumäärällä (kaava 2). Näin saavutetaan ”*average difference in common metrics*”.

```
exports.customSimilarityIndex = (comparedValues,
     comparisonValues) => {
     if (comparisonValues.length) {
          let differenceSum = 0
          for (let i = 0; i < comparisonValues.length; i++) {
               differenceSum += Math.abs(comparisonValues[i] –
                    comparedValues[i])
          }
          let avgDifference = (differenceSum / comparisonValues.length)
          return avgDifference
     }
     return "No similar fields"
}
```

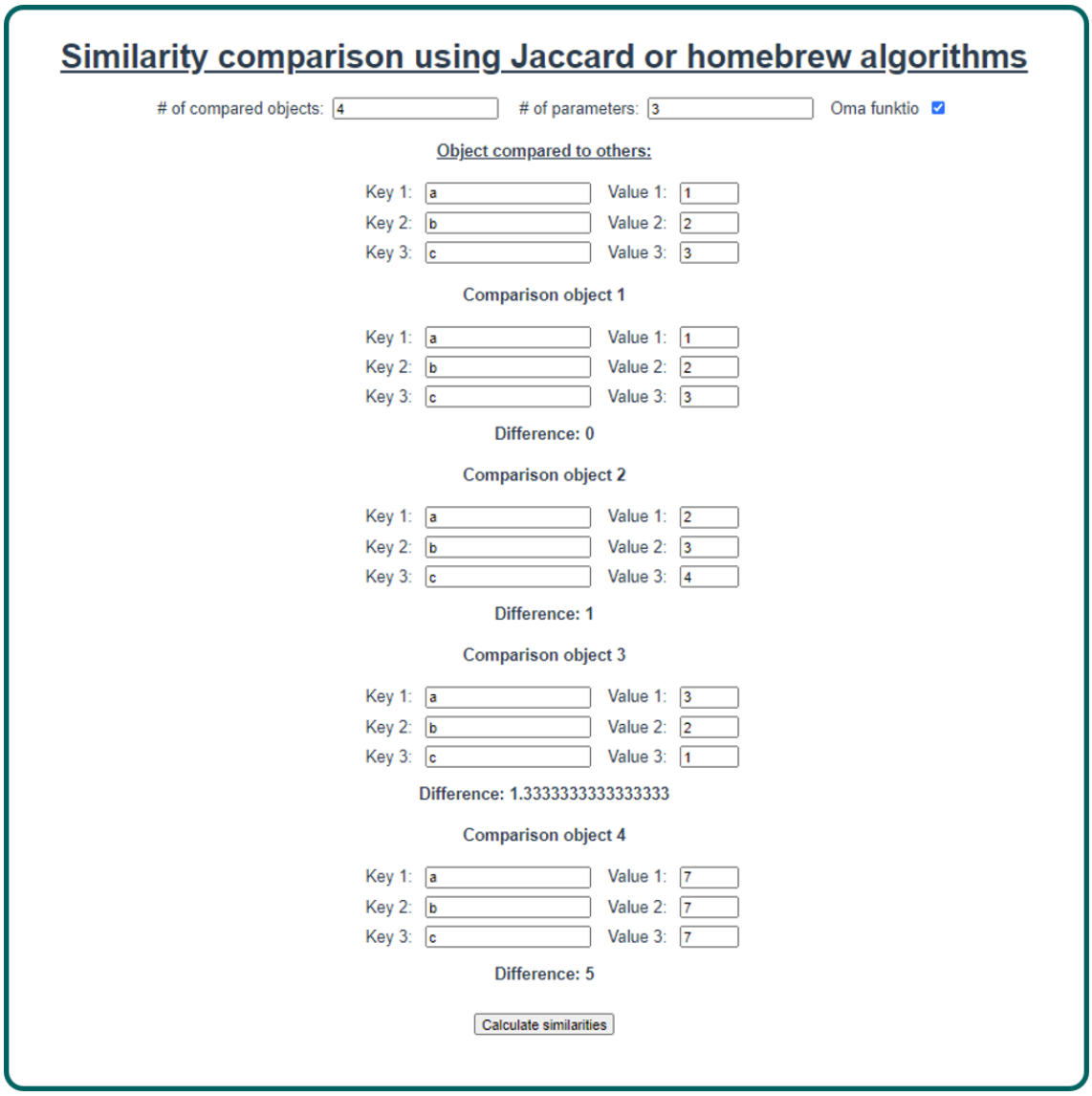
Esimerkkikoodi 5. Vertailufunktio.

Kuten aiemmin tuli ilmi, emme lähteneet toteuttamaan algoritmia, jossa huomioimme tuloksiin vaikuttavien kenttien lukumääriä. Täten päätimme jättää jakolaskun pois ratkaisustamme, sillä sen käyttäminen ei aivan sopinut siihen, mitä haimme tämänhetkiseltä lopputulokselta.

$$Ero(A,B) = \frac{\sum_{k=a}^{\infty} |A.k - B.k|}{A{:}\,n\ ja\ B{:}\,n\ yht.avainten\ lkm}$$

Kaava 2. Ensimmäinen hahmotelma omasta algoritmista.

Suosittelualgoritmimme ensimmäinen versio toimi aivan niin kuin oletimmekin (kuva 24). Ainoa ongelma, jonka huomasimme ja jota osasimme odottaa, olivat tilanteet, joissa verrattavilla olioilla on vain vähän yhteisiä avaimia. Silloin funktio palauttaa korkean samankaltaisuustuloksen, joka ei ole vertailukelpoinen niihin tuloksiin, joissa vertailtavia arvoja on paljon.



Kuva 24. Omatekoisen funktion kokeilua.

Todettuamme ensimmäisen suosittelualgoritmimme toimivan oli aika siirtää algoritmi varsinaiseen sovellukseemme käyttöä ja jatkotestausta varten. Testausvaiheessa suosittelualgoritmi sijoitettiin selainpuolelle, jotta algoritmin kehittäminen ja testaaminen olisi vaivattomampaa. Järkevintä lopullisessa versiossa olisi siirtää algoritmi palvelinpuolelle, jolloin pystytään vähentämään tietoliikenteen määrää selainpuolelle.

Algoritmi ajetaan, kun sisään kirjautuneen käyttäjän profiilia verrataan samalle alueelle muuttamisesta kiinnostuneiden käyttäjien profiiliin tai samalla alueella sijaitseviin asuntoihin. Se, mihin algoritmia käytetään, perustuu käyttäjän luomaan profiiliin. Profiili kertoo, etsiikö käyttäjä kämppäkaveria ja asuntoa vai ainoastaan jompaakumpaa.

Jotta käyttäjien antamia tietoja voidaan verrata oikealla tavalla, tulee tiedot ensin suodattaa ja esiprosessoida. Kun käyttäjien ja asuntojen tiedot tulevat tietokannasta, niiden sijaintitiedot tarkistetaan. Ne oliot, joilla on samat sijaintitiedot kuin sisään kirjautuneen käyttäjän antamat sijaintitiedot, poimitaan seuraavaan suodatukseen. Toisessa suodatuksessa oliot suodatetaan kämppishaussa sukupuolen mukaan ja asuntohaussa rakennuksen tyypin mukaan.

Tämän jälkeen suosittelualgoritmille tarpeettomat tietueet poistetaan. Tietokannan antamat oliot sisältävät sisäkkäisiä olioita, joten oliot muutetaan yhdentasoisiksi, jotta tietojen käsitteleminen ja vertaileminen olisi helpompaa. Tietueista ei tässä kohdin tarkisteta tietojen oikeellisuutta, sillä ne on jo tarkistettu. Puuttuva data eli *null*- ja *undefined*-arvot käsitellään vasta suosittelualgoritmia ajettaessa. Suosittelualgoritmissa puuttuvan datan sisältävät avaimet kerätään listaan, joka myöhemmin esitetään käyttäjien tai asuntojen samankaltaisuustuloksissa.

Seuraavaksi on määriteltävä tietojen tyyppi, jotta tiedetään, miten muuttujia voidaan käsitellä. Tiedonlouhinnassa muuttujien tyyppejä ovat nominaaliasteikon, järjestysasteikon, intervalliasteikon ja suhdeasteikon muuttujat. Palvelumme olioissa on kahdentyyppisiä muuttujia, joissa toisissa voi laskea arvojen välisiä etäisyyksiä, ja toisissa ei. Tästä eteenpäin käymme tarkemmin läpi kämppishaussa käytettävän suosittelualgoritmin toimintaa.

Tiettyjen kämppishaussa olevien muuttujien, kuten esimerkiksi harrastukset-muuttujan arvojen välisiä etäisyyksiä voidaan laskea, sillä harrastuksissa käytämme asteikkoa mittaamaan sitä, kuinka aktiivisesti käyttäjä harrastaa mitäkin harrastusvaihtoehtoa. Harrastukset-muuttuja kuuluu näin ollen intervalliasteikon muuttujiin, joissa arvojen välinen etäisyys voidaan laskea, mutta arvojen välistä suhdetta ei voi. Luonteenpiirteissä, sukupuolen määrittelyssä tai sellaisissa tiedoissa, joiden arvo on tosi tai epätosi, ei arvojen välisiä etäisyyksiä voida laskea, sillä ne antaisivat vääränlaisia tuloksia tiedon luonteeseen nähden. Tiedot kuuluvat näin ollen nominaaliasteikon muuttujiin, joka kertoo mihin luokkaan havainto kuuluu. Niiden välille ei voi muodostaa mitään järjestystä. Koska kämppisehdokkaat-oliossa ja sisään kirjautuneen käyttäjän toivekämppisprofiilissa eli targetProfile-oliossa on kahdentyyppisiä tietoja, tiedot jaetaan kummankin olion osalta kahteen uuteen olioon muuttujien tyypin mukaan (esimerkkikoodi 6).

```
//targetProfile
export function createComparedTargetObject(targetProfile) {

     let targetProfileObject = removeKeyValuePairs([targetProfile])
     let comparedTargetObject = createOneLevelObject(
          targetProfileObject[0])

     let algoObjects = whichAlgorithm([comparedTargetObject])
     return algoObjects
}

//user objects
export function createComparisonUserObjects(users) {

     let userObjects = removeKeyValuePairs(users)

     let comparisonUserObjects = []
     userObjects.forEach(user => {
          comparisonUserObjects.push(createOneLevelObject(user))
     })

     let algoObjects = whichAlgorithm(comparisonUserObjects)
     return algoObjects
}
```

Esimerkkikoodi 6. Käyttäjäolioiden esiprosessointia, jossa tuloksina palautetaan targetProfile-olio ja kämppisehdokkaat-olio. Molemmat oliot sisältävät kaksi sisäkkäistä oliota. Toinen sisäkkäisistä olioista sisältää nominaaliasteikon muuttujia, ja toinen sisältää intervalliasteikon muuttujia.

Muodostetut oliot viedään suosittelualgoritmille (esimerkkikoodi 7). Suosittelualgoritmin ensimmäisessä vaiheessa vertailukohteet viedään funktioihin, joista toinen laskee samankaltaisuutta ja toinen erilaisuutta.

```
let similarities = getSimilarities(
     //nominal
     this.comparisonUserObjects[0], this.comparedTargetObject[0][0],

     //interval
     this.comparisonUserObjects[1], this.comparedTargetObject[1][0]
)
```

Esimerkkikoodi 7. Kämppisehdokkaat ja sisään kirjautuneen käyttäjän toivekämppistiedot viedään suosittelualgoritmille. ComparisonUserObjects sisältää kämppisehdokkaat, joihin comparedTargetObject eli sisään kirjautuneen käyttäjän targetProfilea verrataan.

Samankaltaisuutta laskevassa funktiossa etsitään arvoja, jotka ovat täysin samoja. Jos vertailukohteiden arvot ovat täysin samoja, kasvatetaan samankaltaisuutta laskevan muuttujan arvoa yhdellä. Lisäksi se laskee maksimiarvon samankaltaisuudelle. Erilaisuutta laskevassa funktiossa puolestaan lasketaan sitä, kuinka paljon arvot eroavat toisistaan. Jos arvot ovat täysin samoja, tuloksena on nolla. Mitä suurempi tulos on, sitä suurempi eroavaisuus. Vertailtavien arvojen erilaisuudet lasketaan yhteen ja lopputulos vähennetään maksimierilaisuudesta (kaava 3).

$$Ero(A,B) = \sum_{k=a}^{\infty} A.k, B.k \text{ maksimieroavaisuus} - \sum_{k=a}^{\infty} |A.k - B.k|$$

Kaava 3. Suosittelualgoritmin eroavaisuutta laskevan funktion tämänhetkinen kaava.

Näin ollen tuloksena saadaan arvojen samankaltaisuus ja myöhemmin kahden eri vertailufunktion tuloslistojen arvoja voidaan laskea yhteen, jotta yhden käyttäjän lopullinen samankaltaisuus verrattuna sisään kirjautuneen käyttäjän toivekämppiksen profiiliin saadaan selville (esimerkkikoodi 8).

```
exports.getSimilarities = (ifExistsComparisonObjects,
     ifExistsComparedObject, differenceComparisonObjects,
     differenceComparedObject) => {

     let ifExistsArr = this.calculateIfExists
          (ifExistsComparisonObjects, ifExistsComparedObject)

     let distancesArr = this.calculateDistance
          (differenceComparisonObjects, differenceComparedObject)

     ---

     let similarityArr = ifExistsArr.map(function (num, idx) {
          return num + distancesArr[idx]
     })

     let similarities = []
     similarityArr.forEach(item => {
          similarities.push(this.percentageSimilarity(item, max))
     })

     return similarities
}
```

Esimerkkikoodi 8. Funktio, joka palauttaa kämppisehdokkaiden samankaltaisuuden verrattuna sisään kirjautuneen käyttäjän toivekämppikseen. Tulos on muutettu prosenteiksi.

Erilaisuutta laskevan funktion maksimiarvo erilaisuudelle on yhtä suuri kuin maksimiarvo samankaltaisuudelle. Täten samankaltaisuutta laskevan funktion maksimiarvo samankaltaisuudelle voidaan laskea yhteen erilaisuutta laskevan funktion maksimiarvon kanssa. Lopullinen yhteenlaskettu maksimiarvo sekä vertailtavien käyttäjien samankaltaisuustulokset viedään funktiolle, joka muuttaa jokaisen käyttäjän samankaltaisuustuloksen prosenteiksi jakamalla samankaltaisuustuloksen maksimiarvolla ja kertomalla tuloksen sadalla.

Lopulta suosittelualgoritmin tuloksena saadaan lista, josta näkyy prosentteina, kuinka sopiva kukin käyttäjä tai asunto on sisään kirjautuneelle käyttäjälle. Listan tulokset viedään vertailtavien käyttäjien tai asuntojen olioihin, jotta käyttäjät ja asunnot voidaan lajitella tuloksena saadun samankaltaisuuden mukaan. Tämän jälkeen oliot lähetetään selaimeen, josta kämppäkaveria tai asuntoa etsivä käyttäjä voi tarkastella tietoja kämppis- ja asuntolistauksista.

Myöhemmin lopullisessa versiossa algoritmia pitää suunnitella vielä tarkemmin erilaisille lähtökohdille, joita sen tulee osata käsitellä. Lisäksi pitää miettiä tulosten suhteuttamista toisiinsa nähden huomioiden tulokseen vaikuttavien avainten lukumäärä sekä varmistaa se, että kaikilla muuttujien arvoilla on sama painoarvo.

6.4 Autentikaatio

Sovelluksen rajapinnan vapaa käyttö on osin estetty autentikaatiosäännöillä. Autentikaatiosääntöjä tarvitaan, ettei kuka tahansa pääse vapaasti muuttelemaan tietokannan sisältöä lähettämällä HTTP-pyyntöjä palvelimelle. Palveluumme autentikaatio on toteutettu käyttämällä jsonWebToken-kirjastoa. Käyttäjien kirjautumiseen vaadittavat salasanat säilytetään tietokannassa salattuina. Salaukseen käytetään bcrypt-pakkausta. Kun salasana lähetetään kirjautumisen yhteydessä palvelimelle, sitä verrataan tietokantaan tallennettuun, salattuun versioon bcryptin avulla.

Asuntoilmoituksia palvelun tietokannasta hakevat palvelinreitit on jätetty autentikaation ulkopuolelle, jotta palveluun kirjautumattomat käyttäjät voivat myös tarkastella asuntotarjontaa. Lisäksi kaikille avoinna ovat luonnollisesti reititykset uuden käyttäjän luontia ja sisäänkirjautumista varten. Esimerkkikoodi 9 käsittelee reitityksiä.

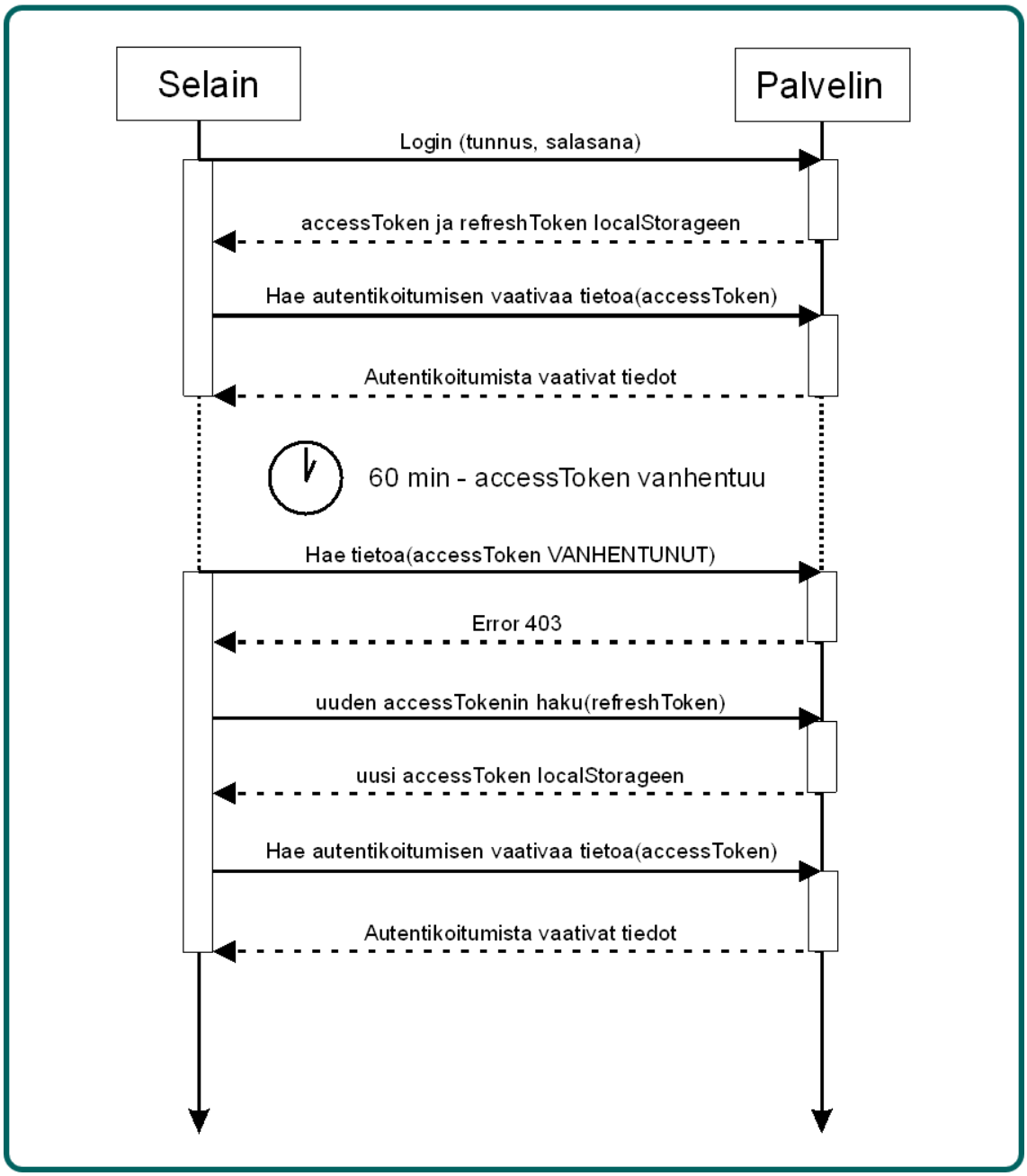
```
module.exports = (app) => {
     app
          //Read all, CREATE
          .route("/users")
          .get(helper.authenticateJWT, userController.getAllUsers)
          .post(userController.createUser)

     app
          //Get users by their location values
          .route("/users/location/?")
          .get(helper.authenticateJWT,
               userController.findUsersByLocation)

     app
          //RUD
          .route("/users/:userId")
          .all(helper.authenticateJWT)
          .get(userController.getUser)
          .put(userController.updateUser)
          .delete(userController.deleteUser)
}
```

Esimerkkikoodi 9. Käyttäjätietojen express-reititykset. .route-rivillä määritellään polku palvelimelle, johon pyyntö tehdään. Polun jälkeen määritetään ajettava funktio pyyntötyypin mukaan ja mahdollinen autentikaation tarve helper.authenticateJWT:llä.

Käyttäjän kirjautuessa sisään palveluun onnistuneen kirjautumispyynnön vastauksen mukana palautetaan kaksi tokenia: access token ja refresh token. Ne säilötään istunnon ajaksi selaimen tallennustilaan. Access tokenia käytetään lähetettäessä kutsuja kaikkiin autentikoitumista vaativiin palvelimen reitteihin. Se lähetetään pyynnön mukana authorization-otsakkeessa. Access tokenilla on tietty, palvelimella ennalta määritetty elinaika, jonka jälkeen se ei enää kelpaa palvelimelle. Tätä varten on refresh token. Kun access tokenin vanhentuminen havaitaan palvelimelta vastauksena tulevan virheviestin yhteydessä, lähetetään palvelimelle uusi pyyntö, jossa on refresh token (kuva 25). Jos refresh token on oikeellinen, palvelin palauttaa käyttäjälle uuden access tokenin, jolla käyttäjä voi taas käyttää autentikoitunutta istuntoa vaativia toimintoja. Sovelluksessamme tämä uusintaprosessi sekä access tokenien liittäminen kutsuihin on automatisoitu hyödyntämällä axios-tietoliikennekirjaston request- ja response-interceptoreita.



Kuva 25. Autentikoituminen ja autentikaatiotokenin automatisoitu uusinta.

Token-järjestelmä suojaa myös käyttäjän istunnon joutumiselta ei-toivottuihin käsiin. Jos käyttäjän istuntoa varten luotu access token joutuu vääriin käsiin, se vanhenee melko nopeasti, jolloin hyökkääjän pitäisi tarjota palvelimelle refresh token jatkaakseen palvelimen kanssa viestintää. Refresh tokenin puuttuessa istunto päättyy, ja käyttäjä kirjataan ulos palvelusta.

7 Lopputulos

7.1 Etätyöskentely poikkeuksellisena aikana

Insinöörityön toteuttaminen itsenäisenä parityönä on melko epätyypillistä. Vielä epätyypillisempiä olivat olot, joissa teimme työtämme. Koronaviruspandemiasta johtuen työskentelimme suurelta osin etänä, mutta työryhmän ollessa vain kahden ihmisen kokoinen, tapasimme kampuksella noin kerran viikossa. Ohjaavan opettajamme varaama työskentelytila oli meidän tarpeisiimme erinomainen, sillä esimerkiksi tilassa olleiden suurien näyttöjen avulla pystyimme vaivattomasti käymään läpi insinöörityömme kulkua. Koululla tapahtuvien tapaamisten lisäksi kommunikoimme pikaviestipalvelujen välityksellä. Lisäksi pidimme noin kahden viikon välein etäpalavereja ohjaavan opettajamme kanssa.

Projektin puolivälin paikkeilla harkitsimme ottavamme käyttöön projektinhallintatyökalun työskentelymme avuksi. Vaihtoehtoina olivat Jira, Monday ja Trello. Aiempaa kokemusta meillä oli jo Mondaysta, jota olimme käyttäneet aiemmissa projekteissamme. Trelloa olimme käyttäneet hyvin vähän ja siihen halusimme tutustua hieman paremmin. Perehdyttyämme siihen syvällisemmin, emme kuitenkaan kokeneet sen käyttöä työhömme sopivaksi. Lopulta päädyimme kokeilemaan Jiraa. Tehtyämme tarvittavat käyttäjätunnukset ja tutustuttuamme sovelluksen käyttöliittymään tulimme kuitenkin siihen tulokseen, että järeämmän projektinhallintatyökalun kunnollinen käyttöönottaminen tässä vaiheessa projektia tulisi kuluttamaan enemmän meille arvokasta aikaa kuin säästämään sitä. Siirsimme kuitenkin projektin versionhallinnan henkilökohtaiselta GitHub-tililtä erilliselle organisaatiotilille. Tämän jälkeen jatkoimme kehitystyötä tuttuun tapaan lähinnä ilmoittamalla, mitä kumpikin on tehnyt ja mitä seuraavaksi aiomme tehdä.

Työskentely etänä toimi mielestämme hyvin olosuhteisiin nähden. Hyvät työympäristöt, niin kodin kuin tekniikankin osalta, mahdollistivat mielekkään ja tehokkaan työskentelyn. Erinomainen työskentelyryhmähenki ja taustatuki edesauttoivat hyvän lopputuloksen saavuttamisessa. Haluammekin painottaa sitä, kuinka tärkeää on työskennellä juuri itselle sopivassa ja kannustavassa ilmapiirissä, jotta työskentely olisi mielekästä, opettavaista ja antoisaa.

7.2 Tavoite tuotantovalmiista sovelluksesta

Insinöörityöllämme oli alusta alkaen tiukka aikataulu. Painetta loivat kevään aikana alkavat palkkatyöt sekä toukokuun valmistumispäivämäärä. Koulusta on valmistuttava viimeistään toukokuun aikana, jos aikoo hakea jatkokoulutukseen hyödyntämällä insinöörin tutkintotodistusta kevään yhteishaussa. Alkuperäinen, joskin kunnianhimoinen tavoite sovelluksen julkaisemisesta kevään tai kesän aikana jää valitettavasti saavuttamatta. Projektissa olisi ollut meille vielä paljonkin tekemistä täysin tuotantovalmiin ja sulavasti toimivan sovelluksen saavuttamiseksi. Vielä ei ole tietoa, milloin ja millä voimin projekti jatkuu, mutta hyvä pohjatyö on nyt tehty. Etenkin rajapintatoteutus on pyritty dokumentoimaan mahdollisimman kattavasti, jotta sen voisi pitää käytössä mahdollisimman samankaltaisena, vaikka selainsovellus tehtäisiin joskus uusiksi kokonaan eri tekniikoin.

Projektijakson loppupuolella kirjoitimme erillisen dokumentin sovelluksen senhetkisistä puutteista, bugeista ja muista tulevaisuuden kannalta huomionarvoisista seikoista. Dokumentin on tarkoitus olla muistilistana, jos projektin pariin vielä tulevaisuudessa palataan. Saatoimme myös versionhallinnan mahdollisimman siistiin kuntoon projektijakson päätteeksi.

Lähteet

1 How many adults under 30 are still living at home with their parents? Verkkoaineisto. <https://www.thejournal.ie/factfind-under-living-with-parents-4981426-Feb2020/> Luettu 1.2.2021.

2 More Dutch and European young adults living at home. Verkkoaineisto. <https://www.cbs.nl/en-gb/news/2018/46/more-dutch-and-european-young-adults-living-at-home> Luettu 1.2.2021.

3 Percentage of Young Adults in Europe, aged 25-34, Who Still Live With Their Parents. Verkkoaineisto. <https://brilliantmaps.com/europe-live-parents/> Luettu 1.2.2021.

4 Impact of Lehman Collapse on Housing Prices in Europe. Verkkoaineisto. <https://www.millersamuel.com/tag/europe/> Luettu 29.3.2021.

5 Näin Euroopassa sinnitellään, kun vuokrat nousevat pilviin. Verkkoaineisto. <https://yle.fi/uutiset/3-11662751> Luettu 1.12.2020.

6 All over Europe, housing has become all too expensive. Verkkoaineisto. <https://www.europeandatajournalism.eu/eng/News/Data-news/All-over-Europe-housing-has-become-all-too-expensive> Luettu 1.2.2021.

7 Pk-seudun asuntokaupassa pelätään nyt 0-asunnon loukkua. Verkkoaineisto. <https://www.helsinginuutiset.fi/paikalliset/3987387> Luettu 16.3.2021.

8 Pienten kaupunkien keskustat ovat vaarassa autioitua, vaikka voisivat pärjätä. Verkkoaineisto. <https://yle.fi/uutiset/3-10998597> Luettu 16.3.2021.

9 Kehä I:n tunneli saa Keilaniemen vihertämään. Verkkoaineisto. <https://www.ncc.fi/media/ajankohtaista/keha-in-tunneli-saa-keilaniemen-vihertamaan/> Luettu 2.4.2021.

10 Keskustan kehityksessä tavoitteena viihtyisä kaupunkitila. Verkkoaineisto. <https://www.uuttahelsinkia.fi/fi/keskusta/uudistukset#keskustan-kehityksessa-tavoitteena-viihtyisa-kaupunkitila> Luettu 2.4.2021.

11 YIT nappasi jättihankkeidensa jatkoksi Helsinki Gardenin. Verkkoaineisto. <https://yle.fi/uutiset/3-10533075> Luettu 2.4.2021.

12 The World’s 10 Greenest Cities of 2020. Verkkoaineisto. <https://www.afar.com/magazine/greenest-cities-in-the-world-in-2020> Luettu 29.3.2021.

13 Asuntosijoittaminen. Verkkoaineisto. <https://sijoitusasunnot.com/asuntosijoittaminen/> Luettu 28.3.2021.

14 Vuokrat nousevat vuodesta toiseen – tässä syyt. Verkkoaineisto. <https://www.salkunrakentaja.fi/2016/05/vuokrat-nousevat-vuodesta-toiseen-tassa-syyt/> Luettu 28.3.2021.

15 Vapaarahoitteiset vuokrat nousivat eniten pääkaupunkiseudulla. Verkkoaineisto. <http://www.stat.fi/til/asvu/2019/04/asvu_2019_04_2020-02-06_tie_001_fi.html> Luettu 5.4.2021.

16 The Top 5 Most Profitable Real Estate Markets in Europe. Verkkoaineisto. <https://www.rubinarealestate.com/en/berlin/the-top-5-most-profitable-real-estate-markets-in-europe/> Luettu 29.3.2021.

17 Tietosuojaseloste. Verkkoaineisto. <https://sopimustieto.fi/sopimukset/7ajgyO-tietosuojaseloste?gclid=Cj0KCQjwo-aCBhC-ARIsAAkNQiv-CWqd9KPJ1oZC0IWpaePD27cMdaOinf-seBDgBMhNSpY9bZGy3On-MaAh2nEALw_wcB> Luettu 31.03.2021.

18 Näin teet helposti tietosuojaselosteen nettisivuillesi. Verkkoaineisto. <https://www.helpotkotisivut.fi/blogi/nain-teet-helposti-tietosuojaselosteen-nettisivuillesi/> Luettu 25.03.2021.

19 Usein kysyttyä EU:n tietosuoja-asetuksesta. Verkkoaineisto. <https://tietosuoja.fi/gdpr>. Luettu 25.03.2021.

20 Tätä EU:n tietosuoja-asetus GDPR vaatii yrityksiltä. Verkkoaineisto. <https://www.helpotkotisivut.fi/blogi/tata-eun-tietosuoja-asetus-gdpr-vaatii-yrityksilta/> Luettu 31.03.2021.

21 Vue - tehokasta UI-kehitystä. Verkkoaineisto. <https://lamia.fi/blogi/vue-tehokasta-ui-kehitysta> Luettu 26.3.2021.

22 WTF is Vuex? Verkkoaineisto. <https://vuejsdevelopers.com/2017/05/15/vue-js-what-is-vuex/> Luettu 26.3.2021.

23 Redux vs. Vuex. Verkkoaineisto. <https://medium.com/@Musclenun/redux-vs-vuex-9b682529c36> Luettu 26.3.2021.

24 7 benefits of using SASS over conventional CSS. Verkkoaineisto. <https://www.mugo.ca/Blog/7-benefits-of-using-SASS-over-conventional-CSS> Luettu 12.3.2021.

25 Why The Hell Would I Use Node.js? Verkkoaineisto. <https://www.toptal.com/nodejs/why-the-hell-would-i-use-node-js> Luettu 27.3.2021.

26 Node.js – Introduction. Verkkoaineisto. <https://www.tutorialspoint.com/nodejs/nodejs_introduction.htm> Luettu 27.3.2021.

27 Express.js– Express for Everyone. Verkkoaineisto. <https://www.simform.com/best-nodejs-frameworks/#Express> Luettu 27.3.2021.

28 Building a REST API with Node and Express. Verkkoaineisto. <https://stackabuse.com/building-a-rest-api-with-node-and-express/> Luettu 27.3.2021.

29 Data-analytiikka, tekoäly ja koneoppiminen. Verkkoaineisto. <https://www.vincit.fi/fi/data-analytiikka-tekoaly-ja-koneoppiminen-trendi-termit-suomeksi/> Luettu 26.3.2021.

30 8 Key Differences Between Data Science and Data Mining. Verkkoaineisto. <https://becominghuman.ai/8-key-differences-between-data-science-and-data-mining-674f09599df2 > Luettu 26.3.2021.

31 A simple way to explain the Recommendation Engine in AI. Verkkoaineisto. <https://medium.com/voice-tech-podcast/a-simple-way-to-explain-the-recommendation-engine-in-ai-d1a609f59d97> Luettu 26.3.2021.

32 Näin sinua ohjataan Facebookissa ja internetissä. Verkkoaineisto. <https://yle.fi/aihe/artikkeli/2016/12/19/nain-sinua-ohjataan-facebookissa-ja-internetissa> Luettu 26.3.2021.

33 The Ethical and Privacy Issues of Recommendation Engines on Media Platforms. Verkkoaineisto. <https://towardsdatascience.com/the-ethical-and-privacy-issues-of-recommendation-engines-on-media-platforms-9bea7bcb0abc> Luettu 26.3.2021.

34 Jaccard index. Verkkoaineisto. <https://en.wikipedia.org/wiki/Jaccard_index> Luettu 31.03.2021.

35 Google APIs Node.js Client. Verkkoaineisto <https://github.com/googleapis/google-api-nodejs-client#google-apis-nodejs-client> Luettu 31.03.2021.

36 Node.js v15.13.0 documentation. Verkkoaineisto. <https://nodejs.org/api/stream.html> Luettu 31.03.2021.

37 Jaccard. Verkkoaineisto. <https://www.npmjs.com/package/jaccard> Luettu 31.03.2021.

38 MySQL how to rank objects by similarity of multiple property rows? Verkkoaineisto. <https://stackoverflow.com/a/14111439> Luettu 31.03.2021.