Osaamista
ja oivallusta
tulevaisuuden
tekemiseen

Valtteri Pesu

Usean verkkopalvelun yhteinen koodi-pohja

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tieto- ja viestintätekniikka

Insinöörityö

25.11.2019

Tekijä Otsikko Sivumäärä Aika	Valtteri Pesu Useamman verkkopalvelun yhteinen koodipohja 32 sivua 25.11.2019
Tutkinto	insinööri (AMK)
Tutkinto-ohjelma	tieto- ja viestintätekniikka
Ammatillinen pääaine	Mobile Solutions
Ohjaajat	projektinhallintakonsultti Juha Rajamäki lehtori Ilkka Kylmäniemi

Insinöörityössä oli tarkoituksena kehittää yritykselle koodipohja, jolla uudistetaan yrityksen olemassa olevia palveluita. Työn tarkoituksena oli toteuttaa usealle verkkopalvelulle yhtenäinen koodipohja, jota on helppo ylläpitää. Myös uusien palveluiden lisääminen haluttiin nopeasti toteutettavaksi. Työssä opittuja tekniikoita oli tarkoitus hyödyntää muissakin tulevissa projekteissa.

Koska uudistus koski koko palvelun toimintaa, insinöörityö rajattiin selainpuolen teknologioihin, joihin kuuluivat Vue-ohjelmistokehys ja Vue-suunnittelujärjestelmä. Näiden lisäksi keskityttiin siihen, mitkä asiat vaikuttavat koodipohjan ylläpidettävyyteen ja miten koodipohjaa näillä teknologioilla kehitettiin. Koodipohjaan kehitettiin modulaarinen rakenne, eli koodipohjassa oli palvelukohtaista koodia ja kaikkien palveluiden käytössä olevaa koodia. Vue-suunnittelujärjestelmää käytettiin yleisten Vue-komponenttien jakamiseen palveluun npm-pakettienhallinnan kautta, ja logiikka pidettiin selainpuolen koodipohjassa.

Uusien kokonaisuuksien toteutuksen aikana huomattiin monia asioita, joita pystyttiin parantamaan etenkin asioiden nimeämisissä sekä suunnittelujärjestelmän ja selainpuolen koodipohjan välisen kehittämisen välillä. Näihin ongelmiin saatiin paljon parannuksia aikaan erilaisilla skripteillä ja sääntöjen sopimisilla. Kun ensimmäinen palvelu oli melkein valmis, toteutettiin samaan koodipohjaan toinen palvelu, jonka aikana huomattiin lisää puutteita koodipohjan rakenteessa. Kuitenkin lopputuloksena ongelmista huolimatta saavutettiin järjestelmälle asetetut tavoitteet ja koodipohjaa voidaan hyödyntää muidenkin palveluiden uudistamiseen. Teknologioista opittiin paljon hyviä asioita, ja koodipohja saatiin helpommin ylläpidettäväksi selainpuolen koodipohjan edetessä.

Avainsanat	Vue.js, Vue.ds, ylläpidettävyys

Author	Valtteri Pesu
Title	Multiple services in same code base
Number of Pages	32 pages
Date	25 November 2019
Degree	Bachelor of Engineering
Degree Programme	Information and Communications Technology
Professional Major	Mobile Solutions
Instructors	Juha Rajamäki, Project Management Consultant Ilkka Kylmäniemi, Senior Lecturer

The main purpose of this final year project was to create a codebase for multiple web-services. These webservices are hosted by a company and services contain a wide range of Finnish laws and statutes. The key features when developing the codebase was to have a codebase that is easy to maintain, and which scales well. Also learning new technologies for future purposes was important.

This final year project focuses mainly on front-end technologies and how they are used in this codebase. Technologies that were used in this project are Vue framework and Vue Design System. Vue framework handled all the logic how data flows in this codebase and the connections to API. Vue Design System contained components that showed data in user interface and Design System was connected to Vue framework via npm.

In conclusion, the codebase had two working webservices with unique user interfaces although they used the same codebase. Working with multiple services at the same time and connection the connection between Vue framework and Vue Design System caused plenty of issues. These problems were resolved with scripts and other methods depending on the issue. In the end improvements in the codebase made maintainability better and the new technologies embraced could be used in future projects.

Keywords	Vue.js, Vue.ds, Maintainability

Sisällys

Lyhenteet ja käsitteet

Angular	Javascript-ohjelmointikieleen perustuva ohjelmistokehys, jolla voidaan kehittää myös selainpuolen käyttöliittymiä.
API	Rajapinta, jonka kautta data kuljetetaan tietolähteen ja palvelun välillä.
DevTools	Selaimeen integroitava kehittäjätyökalu, jonka avulla pystytään tarkastelemaan Vue-komponenttien tapahtumia ja tilanhallintaa.
ESLint	Javascript-koodin muotoilua varten rakennettu työkalu, jolle voidaan asettaa sääntöjä siitä, miten koodi tulee muodostaa.
JSX	HTML tyyppinen lisäosa Javascript-ohjelmointikieleen. Sen avulla voidaan kirjoittaa HTML-koodia, joka käännetään Javascript-ohjelmointikieleksi.
npm	Sovellusrekisteri, pakettienhallintajärjestelmä ja asennustyökalu.
React	Javascript-ohjelmointikieleen perustuva kirjasto, jolla voidaan kehittää selainpuolen käyttöliittymää.
SCSS	Tiedostolyhenne Sass-tyylitiedostoista. Esiprosessoitua CSS-koodia, jonka tarkoitus on helpottaa CSS-koodin kirjoittamista.
TestCafe	Testiautomaatiokirjasto, jonka avulla voidaan kirjoittaa ja ajaa verkkopalveluun kytkettyjä testejä.
Vue.ds	Vue-ohjelmistokehyksen päälle rakennettu suunnittelujärjestelmä.
Vue.js	Vue-ohjelmistokehys. Tarkoitettu verkkosivujen käyttöliittymien kehittämiseen.
Vuex	Yhteinen tilanhallintajärjestelmä, jolla voidaan jakaa kaikille komponenteille sama muuttujan arvo.

1 Johdanto

Insinöörityön tarkoituksena oli kehittää usean palvelun koodipohja uusilla selainpuolen teknologioilla. Insinöörityö tehtiin osana isompaa palveluiden uudistusta suomalaiselle yritykselle, sillä Edita Publishing Oy:n ylläpitämät lakiverkkopalvelut haluttiin nykyaikaistaa.

Edita on sisällöntuottajayritys, jolla on paljon lakiin liittyviä verkkopalveluita. Näihin palveluihin kuuluvat Edilex ja siitä johdetut Plus-palvelut. Jokainen näistä palveluista sisältää paljon Suomen lakiin liittyviä säädöksiä ja muita oikeudellisia aineistoja. Plus-palvelulla tarkoitetaan sellaista palvelua, jossa on asiakkaan haluamat lait ja säädökset mukana. Asiakkaalle annetaan mahdollisuuksia päättää, miten palvelua räätälöidään asiakkaan käyttöönsä sopivaksi. (1.)

Insinöörityönä toteutettiin selainpuolen koodipohja usealle palvelulle, ja siihen on tarkoitus lisätä muita Plus-palveluita. Mukana olevat Plus-palvelut ovat Headpower ja Rajalex. Molemmissa on yrityksen työtehtäviin sopeutuvia ajantasaisia säädöksiä, uutisia ja muita kokonaisuuksia. (1.)

Toteutukseen valittiin uusia teknologioita, joista selainpuolella olevia teknologioita käsitellään tässä insinöörityössä. Näitä teknologioita ei ollut aiemmin käytetty Editan projekteissa, minkä takia niitä ehdotettiin insinöörityön aiheeksi. Päätin valita järjestelmän selainpuolen opinnäytetyön pohjaksi, sillä teknologiat ovat kiinnostavia enkä ollut niitä aiemmin käyttänyt.

Järjestelmän tarkoituksena oli saada kaikki Plus-palvelut uusittua mahdollisimman vähällä vaivalla ja tarvittaessa kehittää uusia palveluita samoilla tekniikoilla ja uutta osaamista hyödyntämällä. Tulevaisuudessa uusien teknologien oppimista hyödynnetään Edilex-palvelun uudistamiseen osa kerrallaan. Järjestelmässä kaikki pienemmät palvelut haluttiin yhteiseen koodipohjaan.

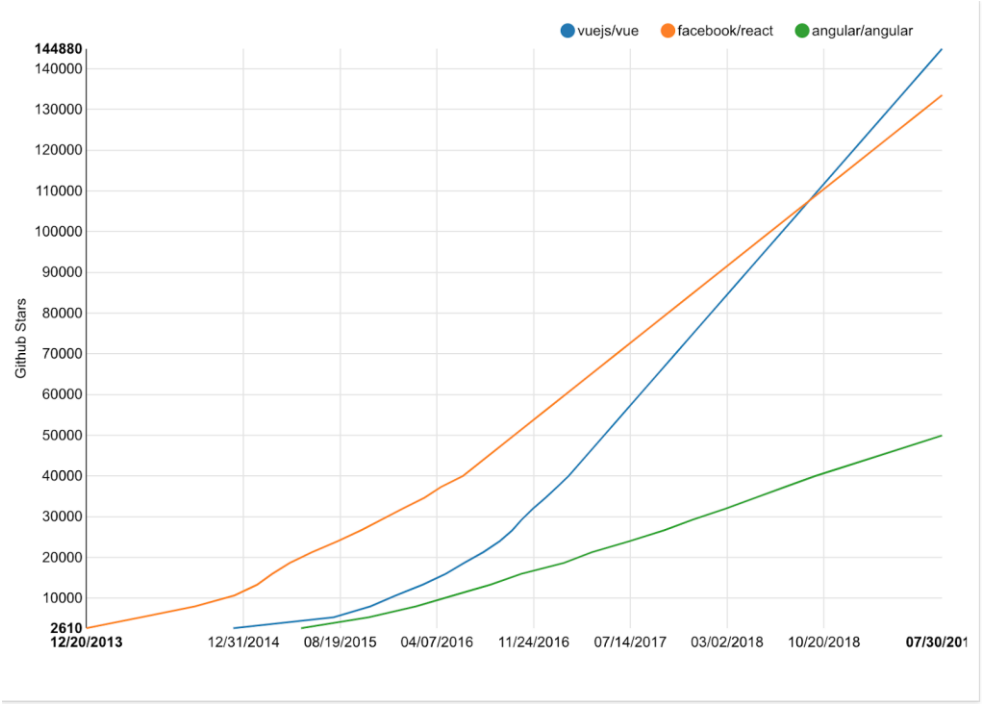
2 Selainpuolen kehitystavat

2.1 Selainpuolen teknologiat

Plus-palveluiden uudessa järjestelmässä käytettävien teknologioiden määrä oli suhteellisen suuri. Kuitenkin tässä insinöörityössä keskitytään selainpuolen teknologioihin, jotka ovat nimeltään Vue.js ja Vue.ds. Ne molemmat pohjautuvat Javascript-, CSS- ja HTML-ohjelmointikieliin. Molemmat ovat varsin tuoreita teknologioita, mutta kasvavassa suosiossa kehittäjien keskuudessa.

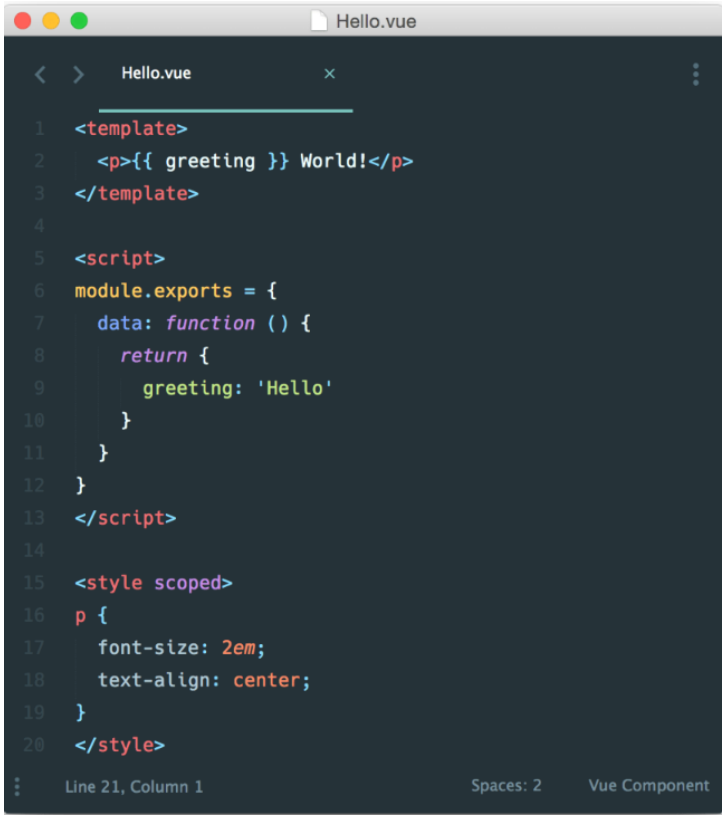
Vue-ohjelmistokehys

Vue on Javascript-ohjelmointikieleen pohjautuva ohjelmistokehys, jonka ensimmäinen versio julkaistiin vuonna 2014. Vue saa päivityksiä säännöllisesti, ja sen suosio on ollut nopeassa kasvussa viime vuosina. Vastaavanlaisia kirjastoja tai ohjelmistokehyksiä ovat React ja Angular. Näistä vanhin on Angular, joka on julkaistu vuonna 2010, ja siihen tulleet päivitykset ovat muokanneet sitä enemmän tai vähemmän samankaltaiseksi kuin Vue tai React. React julkaistiin vuotta aiemmin kuin Vue, ja Reactin suosio on ollut suurin kaikista kolmesta eri vaihtoehdosta. (2.) Insinöörityössä päädyttiin vaihtoehdoista huolimatta käyttämään Vue-ohjelmistokehystä. Sitä pidettiin yksinkertaisimpana ja ryhmän tekijöiden taustojen perusteella nopeimpana teknologiana opetella. Kuvassa 1 esitetään Vue-kehyksen suosion kasvu viime vuosina muihin vastaavanlaisiin tekniikoihin verrattuna.



Kuva 1. Vue-ohjelmistokehyksen suosion vertailu muihin teknologioihin Github-tähtien avulla (3).

Vue-komponentin voi rakentaa monella eri tavalla, mutta kaikista vaihtoehdoista päädyttiin käyttämään Single File Component -rakennetta. Single File Component eli yhden tiedoston komponentti on Vue-kehyksen käsite Vue-komponentista, jossa ovat HTML, CSS ja Javascript samassa tiedostossa. Rakenteessa HTML kirjoitetaan template-tagien sisään, Javascript tulee tuttujen script-tagien sisään ja CSS laitetaan puolestaan style-tagien sisään. (4.) Kuvassa 2 on yksinkertainen yhden tiedoston komponentti.



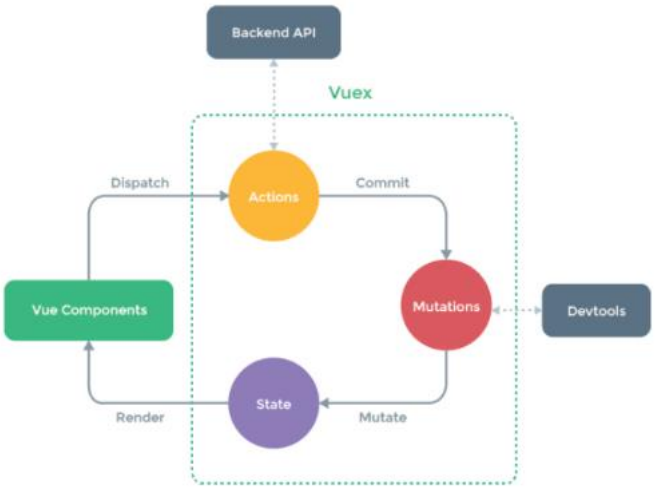
Kuva 2. Esimerkki Vue-kehyksen yhden tiedoston komponentti -rakenteesta (4).

HTML-elementteihin voi kytkeä toimintalogiikkaa v-direktiivien avulla. V-direktiivit sisältävät Angular-kirjastosta tuttuja ominaisuuksia, esimerkiksi if-lauseiden ja for-silmukoiden tekemiseen. Näillä saadaan tehtyä esimerkiksi dynaamisesti muuttuvia listoja ja määritettyä ehtoja mitä elementtejä näytetään tietyissä tilanteissa. (5.)

Komponenttien välinen kommunikointi tapahtuu ominaisuusarvojen avulla, ja niiden suunta on aina vanhemmilta lapsille. Komponenttia, joka sisältää alikomponentteja, kutsutaan vanhemmaksi, ja alikomponentteja kutsutaan lapsiksi. Jos halutaan lapselta lähettää takaisin informaatiota, käytetään emit-tapahtumaa. Näiden kahden erilaisen kommunikaatiotavan avulla voidaan suurin osa toiminnoista toteuttaa. (6.) Yhteisen tiedon jakaminen kuitenkin on helpompi toteuttaa käyttämällä Vuex-kirjastoa, sillä vuex on suunniteltu juuri siihen tarkoitukseen.

Vuex

Vuex on Vue-komponentteja varten lisätty ylimääräinen kirjasto ja sillä pystyy jakamaan kaikille komponenteille yhteisiä tiloja eli yhteisiä data-arvoja. Komponentit pystyvät lukemaan yhteisestä datatilasta asioita. Tämä on hyödyllinen toiminto esimerkiksi käyttöliittymän kielen kirjaamiseen ja rajapintakutsujen käyttämiseen, sillä jokaisella komponentilla on mahdollisuus lukea vuex-tilasta arvoja. (7.) Kuvassa 3 esitellään tarkemmin, miten Vue-komponentit keskustelevat vuex-kirjaston tilan kanssa.

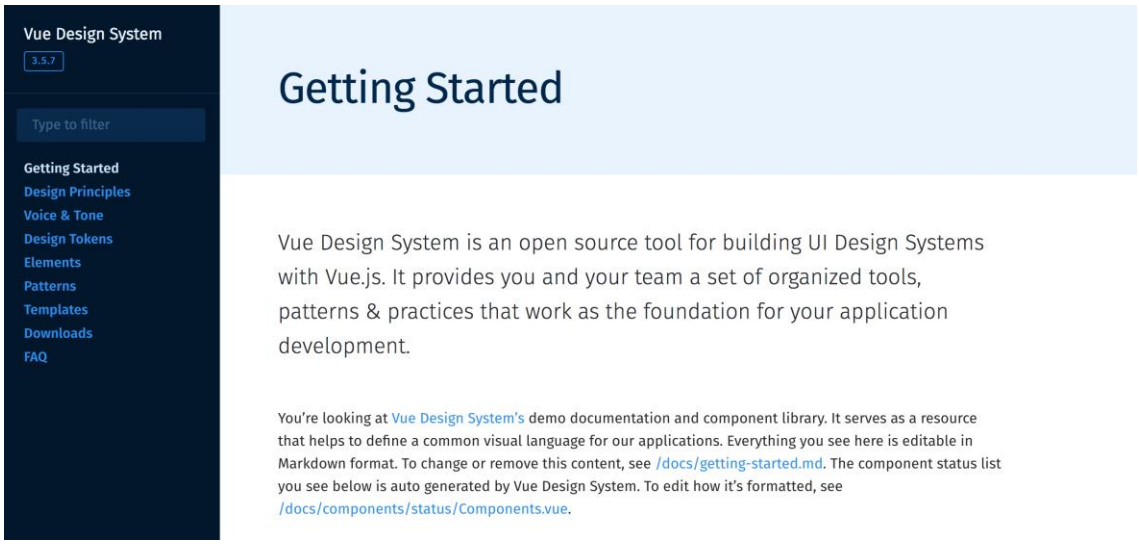


Kuva 3. Vuex-kirjaston jaettu tilanhallinta toimintalogiikka (7).

Vue-suunnittelujärjestelmä

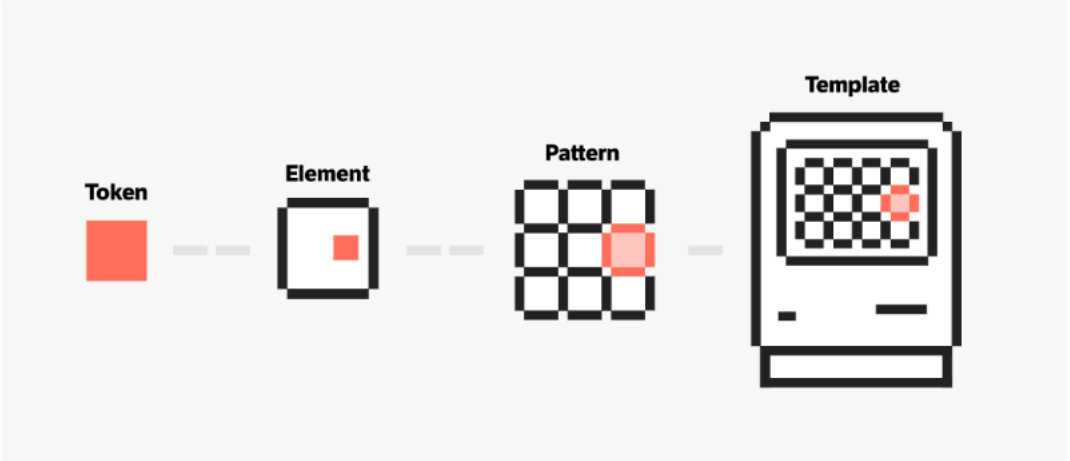
Vue-suunnittelujärjestelmä on Vue-ohjelmistokehystä varten luotu työkalu (8). Suunnittelujärjestelmällä tarkoitetaan kokonaisuutta, jolla voidaan suunnitella, toteuttaa ja kehittää tuote. Tässä tapauksessa tuotteena ovat usean palvelun käyttöliittymät. Suunnittelujärjestelmän tarkoitus on keskittää käyttöjärjestelmän työvaiheet yhteen kokonaisuuteen. (9.) Käyttöliittymän toteutusta varten Vue-suunnittelujärjestelmä antaa kattavat työkalut elementtien, isojen mallikokonaisuuksien rakentamiseen. Myös dokumentaatio saadaan lähelle lopputoteutusta, sillä suurin osa dokumentaatiosta luodaan koodin sekaan

lisättävillä kommenteilla. (10.) Kuvassa 4 on oletusnäkymä, johon jokainen Vue-suunnittelujärjestelmä avautuu ensimmäistä kertaa asennettuna.



Kuva 4. Oletusnäkymä kehitysympäristöön asennetusta Vue-suunnittelujärjestelmästä.

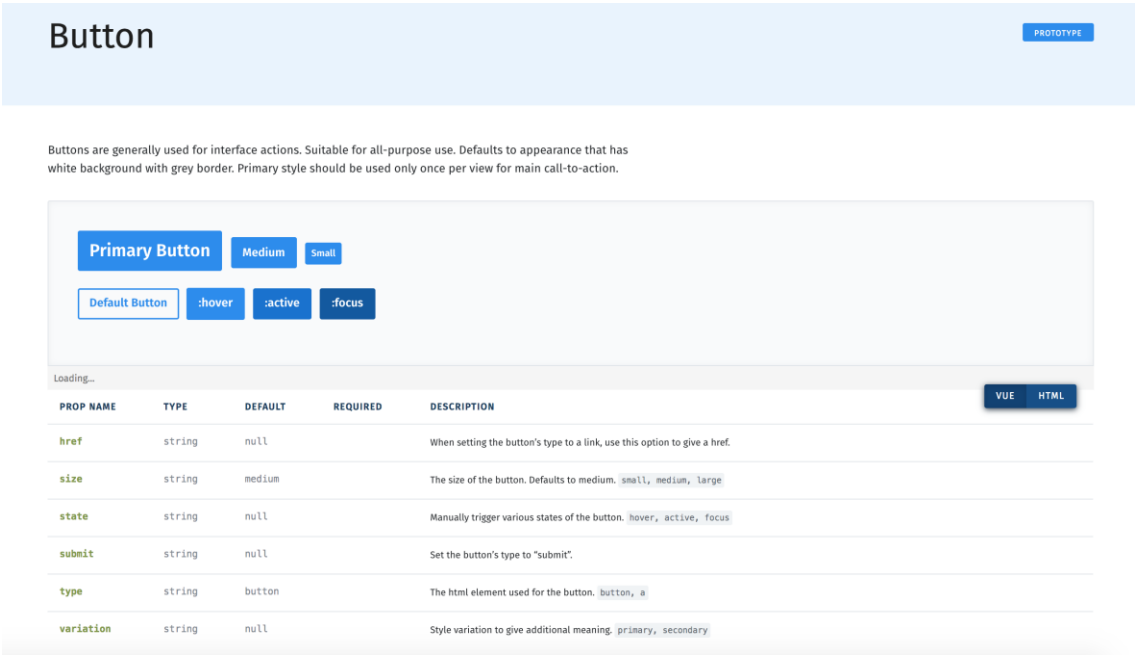
Suunnittelujärjestelmä hyödyntää Vue-komponenttirakennetta, ja yksinkertaisimpia komponentteja kutsutaan elementeiksi. Elementit vastaavat rakenteeltaan HTML-elementtejä hyvin pitkälti, mutta sisältävät räätälöidyt tyylit ja toiminnallisuudet. Räätälöidyt tyylit ja teksti saadaan aikaiseksi käyttämällä suunnittelumuuttujia. Tässä insinöörityössä keskitytään suunnittelujärjestelmän toiminnallisuuksista suunnittelumuuttujiin, elementteihin ja malleihin. (11.) Kuvalla 5 voidaan havainnollistaa rakennetta, jota suunnittelujärjestelmässä käytetään.



Kuva 5. Vue-suunnittelujärjestelmän rakenneosat (11).

Suunnittelumuuttujat ovat suunnittelujärjestelmän pienempiä rakenneosasia, ja ne toimivat samalla tavalla kuin muissakin kielissä muuttujat toimivat. Esimerkiksi niiden avulla voidaan nimetä väriyhdistelmiä tai asioiden kokoluokkia. Niiden tarkoitus on korvata kovakoodatut arvot ja jakaa elementeille arvoja tyylien määrittämiseen. (12.) Elementti voi olla esimerkiksi tyylitelty painike. Siinä on jo jonkin verran toiminnallisuuksia esimerkiksi painallustapahtumia, ja se voi lähettää tai vastaanottaa tietoa vanhemmiltaan. Vanhemmat yleensä koostuvat useasta elementistä ja siten luovat mallikokonaisuuden. Mallikokonaisuudeksi voidaan kutsua esimerkiksi elementeistä rakennettua listaa, jota voidaan käyttää palvelussa sellaisenaan. (13.)

Suunnittelujärjestelmässä Vue-komponentin rakenteeseen voi lisätä docs-tagit, joiden sisään kirjoitetaan JSX-koodia. JSX-koodi saa kuvan 6 dokumentaation painikkeet näkymään samalla tavalla kuin ne näkyisivät palvelun rakenteen sisällä.



Kuva 6. JSX-koodista generoitu komponentin dokumentaatio kehitysympäristön Vue-suunnittelujärjestelmässä.

JSX on HTML-tyyppinen kirjoitustyyli Javascript-ohjelmointikielen sisällä, ja se käännetään myöhemmin puhtaaksi Javascript-ohjelmointikieleksi. JSX mahdollistaa HTML- ja Javascript-kielien kirjoittamista samaan aikaan ja siten nopeuttaa kehittämistä. (14.) Ominaisuudella saadaan samaan aikaan hyvä dokumentaatio suunnittelujärjestelmän sisälle, ja käyttöliittymän suunnittelija voi kehittää omia komponentteja suunnittelujärjestelmässä ja jakaa valmiita komponentteja eteenpäin.

2.2 Skaalautuvuus ja ylläpidettävyys

Pitkäkestoisissa projekteissa tasapainoillaan ylläpidettävyyden ja skaalautuvuuden välillä. Skaalautuvuus verkkopalvelua kehittäessä on sitä, että koodipohjassa on arkkitehtuuri, johon voidaan laajentaa ominaisuuksia samanlaisella koodirakenteella. Ylläpidettävyydellä tarkoitetaan, että koodiin on helppo tehdä muutoksia ja sitä on helppoa ymmärtää. Skaalautuvuus tuo paljon omia asioita, jotka monimutkaistavat rakennetta, ja siten ylläpidettävyys yleensä heikkenee. (15.)

Skaalautuva koodipohja yleensä sisältää modulaarisen rakenteen, eli siinä on kokonaisuuksia koodille, joita voidaan lisätä ja poistaa. Koska järjestelmään on tarkoitus lisätä tulevaisuudessa useita eri palveluita, koodista täytyy tehdä modulaarista, jotta voidaan hyödyntää mahdollisimman paljon samaa koodia.

Ylläpidettävyyteen vaikuttaa moni asia, jotka helpottavat koodin seuraamista ja kehittämistä. Siihen vaikuttavat muun muuassa muuttujien ja funktioiden nimeämiset ja yksinkertaiset ja lyhyet funktiorakenteet. (16.) Koodin lisäksi ylläpidettävyyteen vaikuttaa se, kuinka hyvin päätökset ja toteutukset on dokumentoitu. Dokumentaatio pitää kuitenkin olla myös helposti ylläpidettävää, eli vain tärkeimmät ominaisuudet ja päätökset kuuluvat hyvään dokumentaatioon. (15.)

Koodia tehdessä on aina parantamisen varaa, mutta koodista ei kannata tehdä liian monimutkaista, jos monimutkaisuuteen ole erityistä tarvetta. Yksinkertaista koodia on helpompi ymmärtää ja siihen voi tehdä lisäyksiä jälkikäteen paremmin kuin monimutkaiseen koodiin. Koska järjestelmään halutaan useita palveluita samaan koodipohjaan, on tärkeää, että saadaan koodista mahdollisimman helppoa kehittää eteenpäin. (16.) Kuvassa 7 on tiivistettynä asiat, jotka vaikuttavat ylläpidettävyyteen ja skaalautuvuuteen.

Kuva 7. Yhteenveto ylläpidettävyydestä ja skaalautuvuudesta (15).

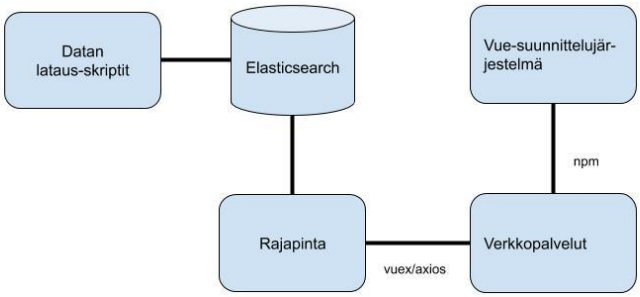
3 Kehittämistapojen suunnittelu

Plus-palveluiden uudesta järjestelmästä oli tarkoituksena kehittää mahdollisimman pitkäikäinen pohja monelle olemassa olevalle verkkopalvelulle ja myös potentiaalisille uusille palveluille. Tekniikoiden valinta oli tehty jo etukäteen ja oli valittu, minkälainen rakenne halutaan koodipohjalle. Järjestelmässä oli tarkoitus pitää koodi ylläpidettävänä ja skaalautuvana. Ensimmäinen palvelu, joka toteutetaan alustalle, sisältää vain muutamaa lakiaineistoa, ja sen takia siitä oli helpoin aloittaa. Myös järjestelmän laajuus oli tässä vaiheessa hieman epäselvä, samoin tarvittavien ominaisuuksien määrä.

3.1 Järjestelmän rakenne

Järjestelmään kuului insinöörityön osuuden lisäksi palvelinpuoli, josta tuotiin jokaiselle palvelulle omat dokumentit. Dokumentit ovat näissä palveluissa rakenteeltaan tarkasti määriteltyjä lakitekstejä. Dokumentit haettiin latausskriptien avulla ja tallennetaan Elasticsearch-hakumoottorin sisälle. Elasticsearch on hakumoottori, johon voidaan tallentaa kaikentyyppistä dataa ja hakea nopeasti sen sisältä (17). Sieltä data kuljetetaan rajapinnalle ja sen kautta selainpuolen käyttöön. Insinöörityössä keskityttiin kuitenkin selainpuolen rakenteeseen ja siihen, miten se on rakennettu.

Selainpuoli oli jaettu kahteen osaan. Vue-suunnittelujärjestelmä oli omana kokonaisuutena ja se jaettiin palvelimen selainpuolen koodille npm-pakettienhallinnan kautta. Npm-pakettienhallinta on suuri sovellusrekisteri, jolla voidaan jakaa kirjastoja ja muita ohjelmointiin liittyviä asioita (18). Selainpuolen koodi käytti suunnittelujärjestelmän komponentteja samalla tavalla kuin tavallisia Vue-komponentteja. Selainpuolen koodissa tuli huomioida, miten usean koodipohjan koodi jaetaan ja mitkä asiat kuuluivat yleiselle puolelle ja mitkä asiat olivat palvelukohtaisia asioita. Kuvassa 8 esitellään koko järjestelmän yleisrakenteen tärkeimmät osa-alueet.



Kuva 8. Plus-palveluiden uuden järjestelmän rakenteen hahmotelma.

Vue-suunnittelujärjestelmän koodi oli suunniteltu sillä tavalla, että komponentteja voidaan hyödyntää muihinkin tarkoituksiin. Tämä tarkoittaa, että komponenttien piti olla mahdollisimman vähän riippuvaisia niihin annettavasta datasta. Silloin logiikka täytyy sijoittaa selainpuolelle sijoitettuun koodiin. Datan haku ja sen lähettäminen eteenpäin komponenteille päätettiin toteuttaa ominaisuusarvojen avulla.

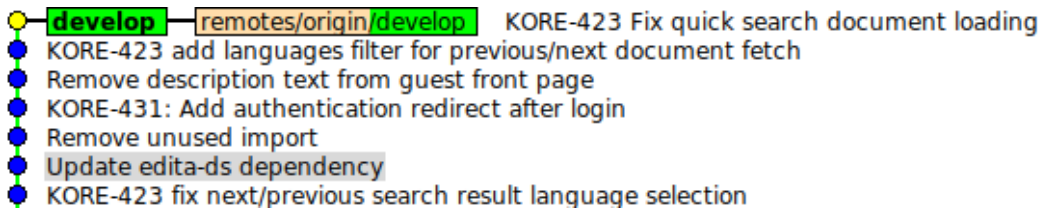
3.2 Dokumentointi

Järjestelmän eri vaiheiden ja päätöksien dokumentaatio oli hyvin tärkeää, sillä tekijät saattavat vaihtua nopeasti ja tieto unohtuu, jos sitä ei kirjata mihinkään. Dokumentaatiota tehdessä kuitenkin pitää muistaa, että järjestelmään tulee vielä paljon muutoksia, joten liian tarkasti ei kannata lähteä dokumentoimaan toimintaa tai koodia. Dokumentaatio alkoi jo suunnitteluvaiheessa ja silloin kerättiin määrittelyjä, joiden mukaan haluttiin toteuttaa järjestelmä. Palvelu, jota lähdettiin kehittämään, on olemassa oleva palvelu. Tämä vaihe jää melkoisen lyhyeksi, sillä suurimmalla osalla oli selkeä suunnitelma siitä, miten järjestelmää voitiin alkaa kehittää. Dokumentaatiolla saadaan parhaiten tieto leviämään kehittäjien kesken ja myös uusille tekijöille.

Järjestelmän koodipohjan edistymistä ja koodia on tarkoitus dokumentoida monin erilaisin tavoin. Jira-palvelulla voidaan hallita koko järjestelmän toteutusputkea, ja tässä insinöörityössä sitä käytettiin pääasiassa kokonaisuuksien jakamiseen pienempiin osiin. Myös järjestelmän kokonaisvaltaista edistymistä voitiin seurata Jira-järjestelmällä. (19.)

Selainpuolen koodin dokumentaatiota varten Vue-suunnittelujärjestelmässä komponenttien JSX-koodilla tuotettu dokumentaatio katsottiin riittäväksi. Git-versionhallinta on työkalu, joka sisältää kaikki tarpeelliset työkalut koodin jakamiseen ja versioiden ylläpitämiseen, ja seuraamiseen. (20.) Järjestelmässä sillä jaetaan koodi kaikkien yhteiseen kehittämiseen sekä se toimii myös eräänlaisena dokumentaationa koodista. Koodin sekaan lisätään tarvittaessa myös kommentteja, jotta koodin ymmärtäminen helpottuisi.

Jira-kortit haluttiin kytkeä samalla tavalla koodiin kuin aiemmissakin palveluissa. Kortin tiedot kirjoitetaan Git-lähetysviestiin muodossa järjestelmän nimen lyhenne isoilla kirjaimilla, minkä jälkeen tulee yhdysmerkki ja tehtävän numero. Lähetysviestin pitää myös kuvata mahdollisimman selkeästi, mitä koodiin toteutettiin tehtävästä, ja sen pitää olla myös sopivan lyhyesti kirjoitettuna. Tällä tavalla pystytään seuraamaan paremmin versionhallinnassa olevia muutoksia ja tarkastelemaan Jira-kortista, mitä kaikkea oli suunniteltu toteutettavaksi ja vastaako koodi suunnitelmaa. Komennolla gitk saadaan kuvan 9 mukainen listaus kaikista Git-versionhallinnan mukaisista muutoksista lähetysviesteineen.



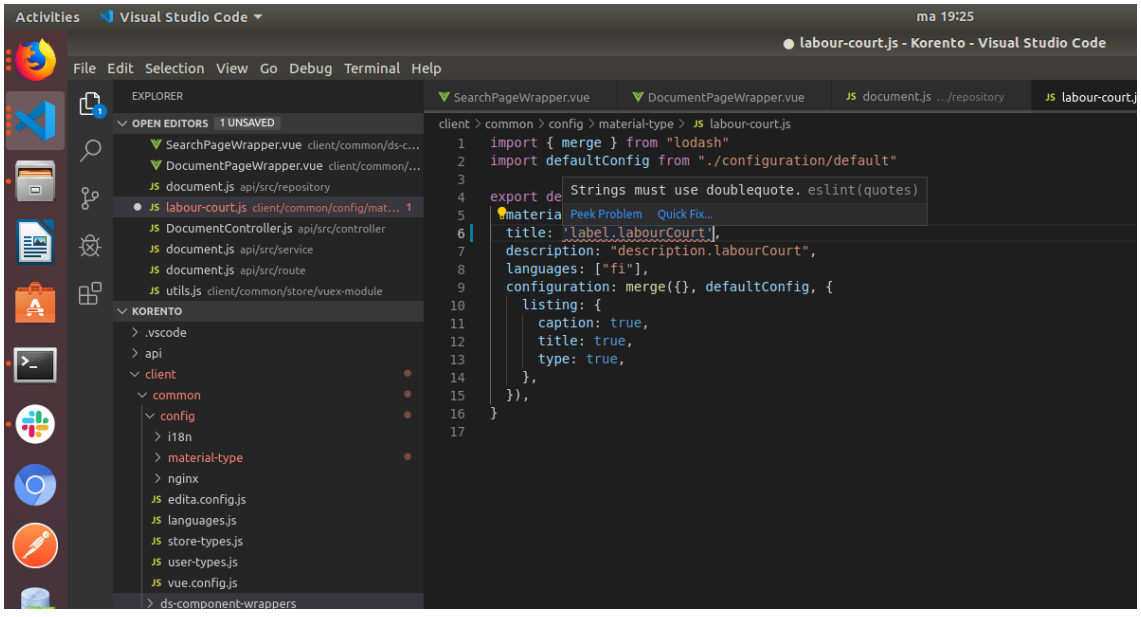
Kuva 9. Havainnollistus Git-lähetysviesteistä.

Kaikkia pieniä korjauksia varten ei ole järkevää tehdä Jira-korttia, joten sellaiset korjaukset voidaan kirjata versionhallintaan ilman Jira-tietoja kuten kuvasta näkyy. Jira-korteille määritettyjen tehtävien kokoluokka vaihteli huomattavasti, sillä tehtävien laajuutta ei voitu etukäteen arvioida. Tärkeintä on, että lähetysviesti on selkeä ja siitä käy ilmi, mitä on tehty.

3.3 Koodin yhteiset säännöt

Järjestelmää kehitettäessä on tärkeää, että käytetään samoja nimityksiä ja merkintätapoja, jotta koodi on yhtenäistä ja siistiä. Nimeämisessä pitää olla erityisen huolellinen, ettei käytetä liian yleisiä termejä. Esimerkiksi pelkästään sana ”lista” ei välttämättä ole tarpeeksi kuvaava termi, jos sillä tarkoitetaan jotain tiettyä listaa. Kuvaavampi nimi on esimerkiksi dokumenttilista, jonka nimestä selviää, että listassa käsitellään dokumentteja.

Kehittämistapojen yhdistämiseen on olemassa työkaluja, joista päädyttiin käyttämään ESLint-kirjastoa. ESLint on Javascript-ohjelmointikieltä varten rakennettu sääntökirjasto, jonka avulla voidaan kertoa millä tavalla koodin pitää olla rakennettu (21). Näitä sääntöjä voi myös itse muuttaa omien tottumusten mukaisiksi. Kehittämistapojen yhdistäminen haluttiin erityisesti mukaan, sillä aiemmissa projekteissa oli toteutustavoissa monenlaisia ratkaisuja ja siten koodin rakenteesta tuli epäselvää. ESLint-sääntöjen avulla ohjelmointieditori pystyy huomauttamaan virheistä ja ehdottamaan oikeaa toteutustapaa. Ehdotukset auttavat yleensä korjaamaan koodin oikeanlaiseksi jo tekovaiheessa, ja niiden aktiivisella korjaamisella vältytään tilanteen pahentumiselta. ESLint-virheilmoitus näkyy esimerkiksi Visual Studio -editorissa kuvan 10 mukaisella tavalla.



Kuva 10. ESlint-virheilmoitus Visual Studio Code -käyttöliittymässä.

3.4 Testaus ja koodikatselmointi

Toimintojen testaukseen oli tarkoitus tehdä automaattisesti generoituja testejä TestCafe-kirjastolla. TestCafe on työkalu, jolla voidaan kehittää testejä kaikkiin insinöörityön toiminnallisuuksiin (22). Tärkeimpiä ominaisuuksia varten kuitenkin tehdään samalla kirjastolla räätälöityjä testejä, jotta ne varmasti toimivat oikein. Automatisoidut testit on helppo rakentaa uudelleen, jos koodipohjaan tulee muutoksia. Tarkemmat testit kannattaa kuitenkin rakentaa vasta, kun järjestelmän runko alkaa olla valmis, jotta vältytään turhalta testien uudelleen kirjoittamiselta.

Testiautomaation lisäksi haluttiin jokainen Jira-kortin tehtävä testata myös manuaalisesti. Manuaalisen testauksen alustavasti hoitaa se, joka on itse kehittänyt kyseisen toiminnallisuuden, minkä jälkeen sen voi siirtää Git-haaraan. Git-haarasta viedään uusin versio palvelimella olevaan kehitysympäristöön, ja sen kautta testaaja pääsee testaamaan oikeassa ympäristössä. Jos ominaisuus on kunnossa, voidaan siirtää Jira-kortti valmiiseen tilaan. Toiminnallisuudesta löytyvän virheen takia tehtävä voidaan kuitenkin palauttaa takaisin kehittäjälle korjattavaksi. Monivaiheisen testauksen tarkoituksena on vähentää vikojen päätymistä loppupalveluun ja nopeuttaa vikojen löytymistä. Nopea vian tiedostaminen helpottaa myös korjaamista, sillä kehittäjällä on tuoreessa muistissa se, mitä asioita koodipohjassa on muutettu.

Suurimmissa muutoksissa käytetään jonkintasoista koodikatselmointimenetelmää. Koodikatselmoinnilla tarkoitetaan, että joku muu kuin koodin kirjoittanut lukee koodin läpi. Tässä vaiheessa voidaan esittää kysymyksiä toteutustavoista, mutta tärkeintä on saada tieto muutoksista leviämään. Koodin läpikäymisessä huomataan varhaisessa vaiheessa, jos jokin asia on epäselvästi tehty tai jotain tilannetta ei ole otettu huomioon. Koodikatselmointia voisi hyödyntää myös useammin, mutta sitä ei koettu tarpeelliseksi tämän järjestelmän kehityksen aikana.

4 Selainpuolen toteutus

Järjestelmän insinöörityöosuutta lähdettiin toteuttamaan ottamalla mukaan yksinkertainen uusittava verkkopalvelu, jonka ominaisuuksien mukaan suunnitelmat tehtiin. Ryhmän käyttöliittymän suunnittelija ja yksi ohjelmoija kehittivät järjestelmän koodipohjan rungon. Muiden tekijöiden mukaan tullessa huomattiin nopeasti, että olisi ollut parempi, että useampi ryhmän jäsen olisi ollut alusta asti mukana päätöksien teossa. Pääosa tarvittavista muutoksista täytyi kuitenkin tehdä funktioiden ja käsitteiden nimeämisiin. Esimerkiksi sanan ”section” käyttö koodissa monen erilaisen osion nimeämiseen aiheutti huomattavaa sekavuutta uusien tekijöiden näkökulmasta. Se saattoi tarkoittaa lakitekstin osaa tai sivun eri osia, ja sillä oli myös nimetty reittimäärittelyiden konfiguraatioita.

4.1 Toteutettavat kokonaisuudet

Muiden kehittäjien tullessa mukaan kehittämään täytyi keksiä keino, miten saadaan mahdollisimman nopeasti edistettyä selainpuolen eri osa-alueita. Selainpuolen kehittämistä pilkottiin pienempiin kokonaisuuksiin, jotta saatiin jokaiselle tekijälle jokin alue, jota lähteä kehittämään. Selkeitä isoja kokonaisuuksia olivat navigaatiot, dokumenttinäkymä, hakutoiminnot ja listaukset. Jokainen ominaisuus vaati kuitenkin, että rajapinnasta saadaan jotain oikeaa dataa, jonka avulla voidaan rakentaa kokonaisuudet oikein.

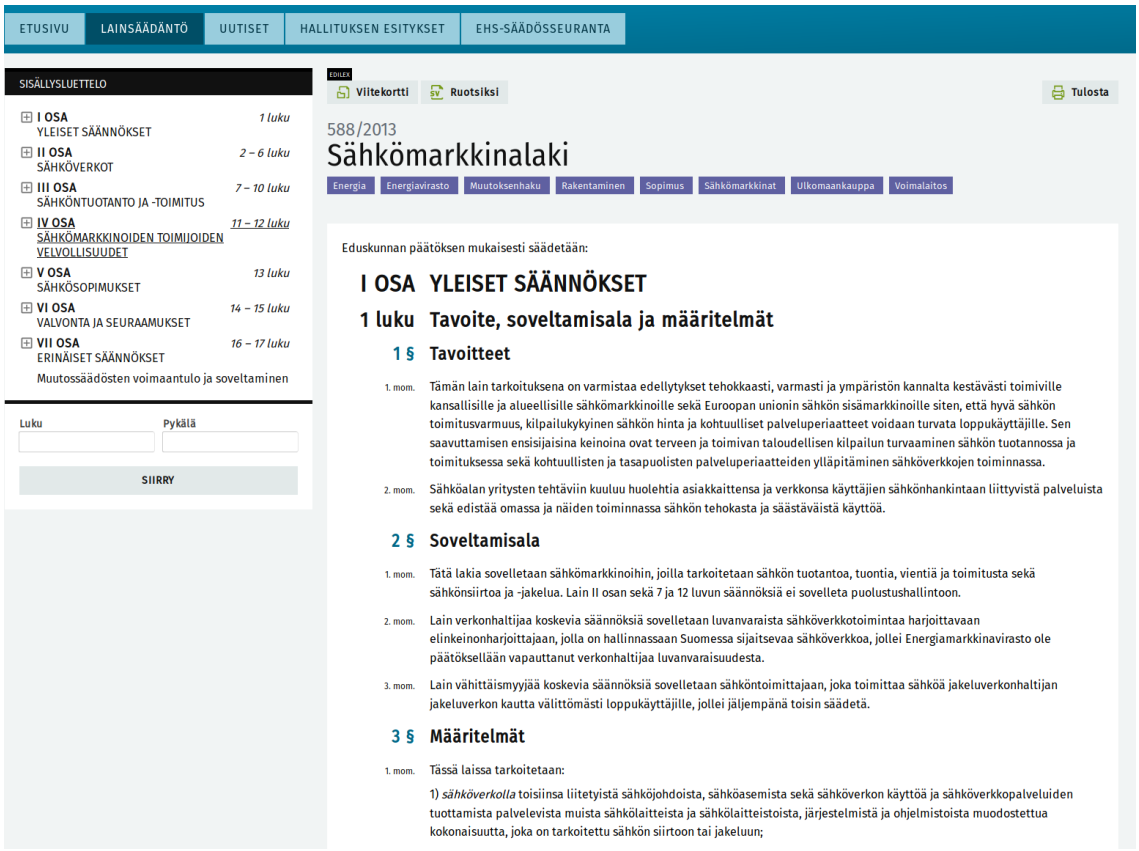
Navigaation tärkeimmät osat ovat palvelussa ylänavigaatio ja sivupalkki, joiden kautta voi selailla erilaisia lakialueita. Navigaatioissa käytettiin Vue-kehyksen reititystä. Reitityksen avulla toteutettiin myös sivujen asetukset, jotta tiedetään mitä asioita voidaan kullakin sivulla näyttää. Tämä kuitenkin toi huomattavia haasteita uusille tekijöille ymmärtää, mistä ja miten tiedot tulevat komponenteille käytettäviksi. Kuvassa 11 on alustalla toteutetun palvelun päänavigaatio.

Kuva 11. Plus-palvelun navigaatiokomponentti.

Seuraava selkeä kokonaisuus on listaukset, ja niitäkin pystyttiin jakamaan erityyppisiin listauksiin. Listauksissa otettiin huomioon eri dokumenttityypit ja se mitä tietoja niistä halutaan näyttää. Jokaista listaa varten täytyi suunnitella, voidaanko jotain olemassa olevaa listaa hyödyntää vai ovatko listat niin erilaisia, ettei niitä kannata tehdä samaan komponenttiin. Listauksia palvelussa on monenlaisia eri tarkoituksiin, vaikka ne käyttävät samoja komponentteja. Kuvassa 12 on yhden dokumenttityypin listausrakenne.

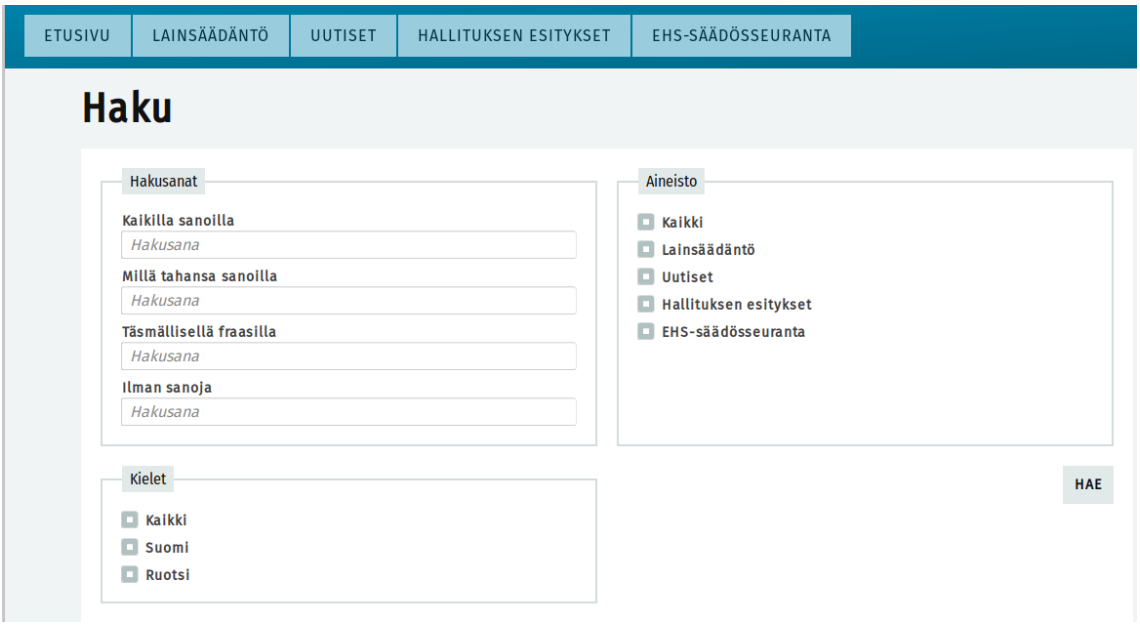
Kuva 12. Kaikissa insinöörityön palveluissa käytettävä listaus suunnittelujärjestelmän puolelta.

Tämän palvelun dokumenteilla on monenlaisia rakenteita, ja ne haluttiin yhtenäistää käyttämällä mahdollisimman paljon samoja komponentteja. Kenttien yhtenäistäminen toteutetaan jo datan latausvaiheessa palvelinpuolella. Tämän on tarkoituksena helpottaa samojen komponenttien käyttämistä dokumentin rakentamiseen. Myös dokumenttien pituudet vaihtelevat hyvin paljon, joten joitakin dokumenttityyppejä varten tuotiin myös sisällysluettelot, joiden avulla navigointi dokumentin sisällä helpottuu. Dokumenttien rakenne vaatii paljon suunnittelua, jotta komponentteja voidaan hyödyntää mahdollisimman hyvin, vaikka rakenteet olisivat erilaisia. Kuvassa 13 on valittuna iso dokumentti, johon on haluttu mukaan sisällysluettelo.



Kuva 13. Plus-palvelun yleinen dokumenttirakenne ja sisällysluettelo.

Lakitekstien selaaminen on varsin työlästä, ja sen vuoksi haluttiin tehokas hakumoottori, eli Elasticsearch, jonka avulla voidaan hakea tiettyjä dokumentteja nopeasti ja helposti. Hakulomakkeita oli kahdentyyppisiä, pikahaku ja tarkennettu haku. Pikahakuun haluttiin hakuehdotukset mukaan ja tarkennettuun hakuun yleiset hakutoiminnot kuten haku kaikilla sanoilla ja haku ilman sanoja. Elasticsearch-hakumoottorista ei vielä tässä vaiheessa ollut tarpeeksi kokemusta, joten näiden toiminnallisuuksien kanssa haluttiin myös testata miten hyvin hakumoottori löytää dokumentteja suomenkielisillä hakusanoilla. Yleisten hakuehtojen lisäksi haluttiin omia rajauksia, joita lähdettiin toteuttamaan dokumenttityyppirajauksella. Kuvassa 14 on tarkennetun haun lomakerakenne.



Kuva 14. Tarkennettu hakulomake kokonaisuudessaan.

4.2 Palveluiden yhteinen koodi

Selainpuolen yhteiseen koodiin kuuluivat kaikki rakenteisiin liittyvät osat ja datan hakuun liittyvät osat. Alkuun yhteisen koodin puolella oli asioita, jotka kuuluivat enemmän palvelukohtaiseen koodiin, mutta niitä siirreltiin sopivissa väleissä pikkuhiljaa oikeille paikoille. Mahdollisimman paljon logiikkaa haluttiin yleiselle puolelle, jotta koodia pystytään paremmin jakamaan palvelukohtaisiin koodeihin.

Koodipohjan monimutkaisimmaksi kokonaisuudeksi osoittautui sivujen reititys, jota pitkin päätettiin tuoda suurin osa sivujen asetuksista. Tähän ongelmaan ei kuitenkaan vielä ole keksitty parempaa ratkaisua, ja se on liian suuri kokonaisuus tehdä uudelleen, joten sitä täytyy pikkuhiljaa parantaa. Vaihtoehtoja parantamiseen ovat tiedon levittäminen siitä, miten tämä kokonaisuus toimii, ja samalla koodin selkeyttäminen kohdista, joissa se on hieman epäselvää. Ylläpidettävyyden kannalta tämän alueen parantaminen on tärkeää, mutta palvelun toimintaan vaikuttavat asiat täytyy toteuttaa ensisijaisesti. Jokainen reitti sisältää oletusmäärittelyt ja samankaltaisen rakenteen, ja oletusmäärittelyistä saadaan kaikki materiaalikohtaiset räätälöidyt määrittelyt. Kuvassa 15 on erään dokumenttityypin reittimääritys.

client > common > router > routes > JS labour-court.js > ...

```
import { merge } from "lodash"
import { makeDefaultMaterialRoute } from "@common/router/utils"

import DocumentListWrapper from "@wrappers/DocumentListWrapper.vue"
import DocumentPageWrapper from "@wrappers/DocumentPageWrapper.vue"

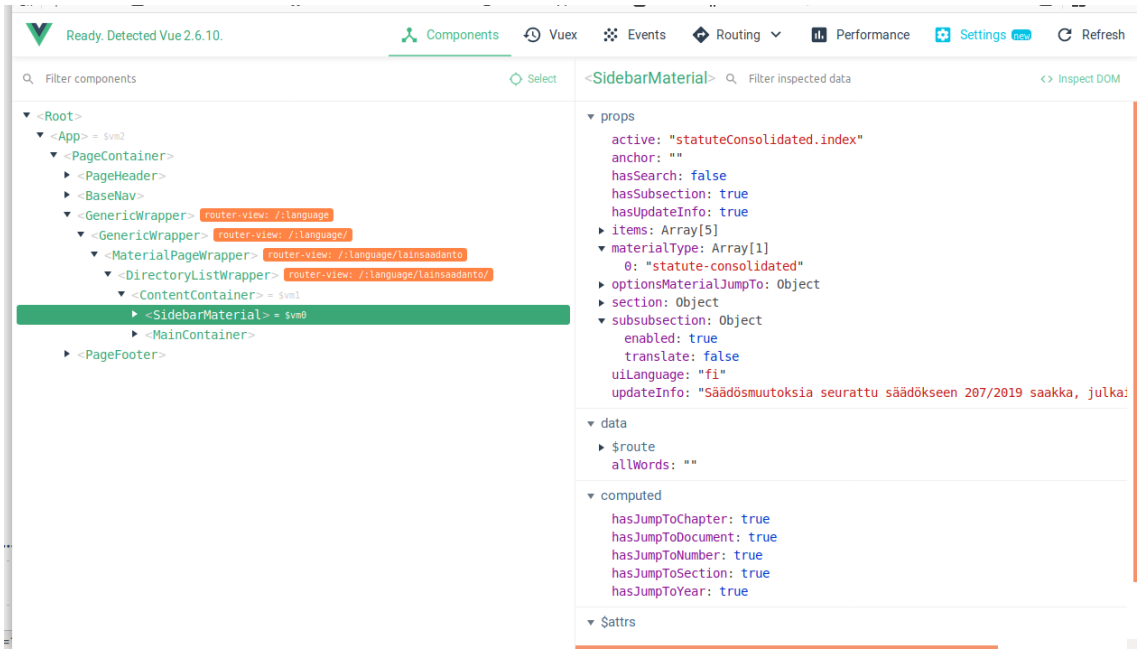
export default function(material, { highlight = false, isPublic = true } = {}) {
  return merge({}, makeDefaultMaterialRoute(material), {
    path: "tt",
    children: [
      {
        path: "",
        name: "labourCourt.index",
        component: DocumentListWrapper,
        meta: {
          title: "heading.labourCourt.index",
          isPublic: !!isPublic,
        },
      },
      {
        path: ":documentKey(\\d+)",
        name: "labourCourt.documentView",
        component: DocumentPageWrapper,
        meta: {
          validateDocumentLanguage: true,
          isPublic: !!isPublic,
        },
        hidden: true, // Do not show in Nav
      },
    ],
    highlightItem: highlight,
  })
}

```

Kuva 15. Vue-router-reititysmäärittely yhdelle dokumenttityypille.

Reititykset jakavat tietoa selainpuolen sivuille, ja sivut rakennettiin isoina komponenttikokonaisuuksina. Kokonaisuudet koostuivat sivujen vaihtuvista osista, joita kutsuttiin käärekomponenteiksi. Nimeämisessä haluttiin käyttää jotain muuta sanaa kuin sivu, koska kaikki komponentit eivät olleet sivuja vaan pienempiä osia sivuista, ja siksi päädyttiin kääre-sanaan. Nämä komponentit sisältävät kaiken sivulla vaadittavan logiikan, ja ne käyttävät paljon erilaisia Vue-suunnittelujärjestelmän komponentteja. Näistä käärekomponenteista jaettiin ominaisuusarvojen avulla dataa suunnittelujärjestelmän puolelle hyödyntäen aikuiselta lapselle-tekniikkaa. Lapsikomponenteilta tuotiin dataa takaisin päin käyttäjän syötteiden mukaisesti emit-tapahtumilla. Komponenttien välistä datan siirtoa selainpuolella tehtiin hyvin paljon, sillä komponentteja ketjutettiin toisiinsa paljon.

Dataa vietiin yleensä aina Vue-suunnittelujärjestelmän elementti komponenteille asti. Kuvalla 16 pyritään havainnollistamaan käärekomponenttien kytkentä isommasta kokonaisuudesta pienempään.

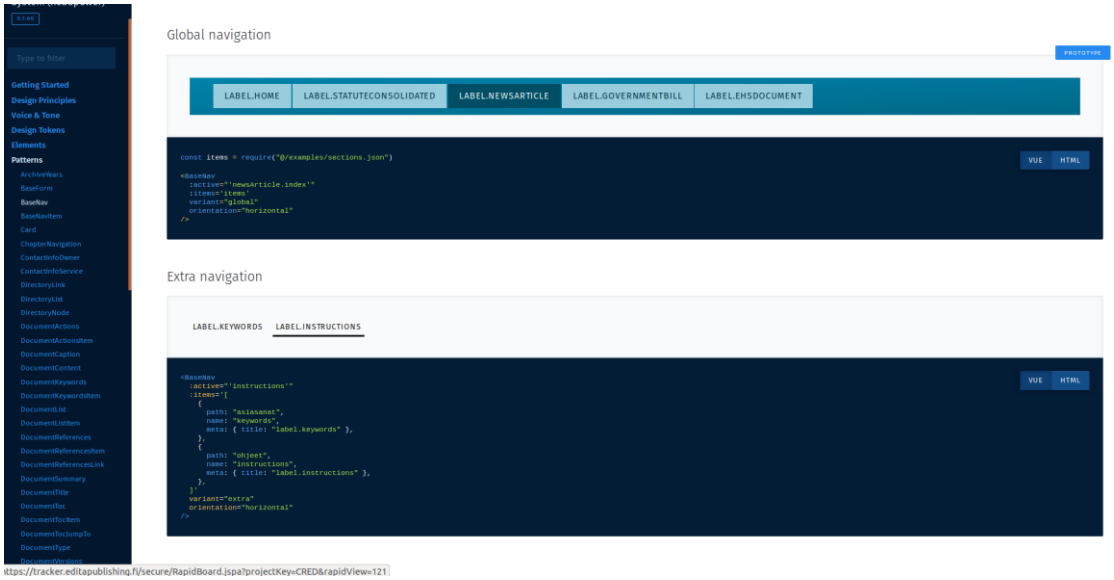


Kuva 16. Selainpuolen yleinen komponenttirakenne.

Vue-suunnittelujärjestelmän kaikki komponentit ovat yhteisessä koodissa käytettävissä, mutta niiden tyylit vaihtelevat palveluittain suunnittelumuuttujien ansiosta. Suunnittelumuuttujissa on myös yleiset ja palvelukohtaiset rakenteet, jotta saadaan palveluille käyttöön omia tyylimuutoksia. Kaikki muu on pystytty toteuttamaan käyttämällä samoja yleisiä komponentteja.

Nimeämiseen Vue-suunnittelujärjestelmän komponenteissa käytettiin logiikkaa, jossa alkuosa kertoo, mihin ominaisuuteen se liittyy, ja loppuosa millainen komponentti on kyseessä. SearchSuggest- ja SearchSuggestList-komponenttien nimistä näkyy, että alku kertoo, mihin ominaisuuteen komponentit liittyvät, ja jälkimmäisessä nimessä kerrotaan mikä osuus ominaisuudesta on kyseessä. Tämä nimeämistapa katsottiin hyväksi, sillä listoja ja muita komponentteja tehtiin suhteellisen paljon ja haluttiin jo nimessä kertoa mahdollisimman yleisesti ja tarkasti, mitä komponentti tekee. Nimistä myös näkee, mitkä komponentit kuuluvat yhteen.

Osa suunnittelujärjestelmän komponenteista on rakenteeltaan yleisiä, jotta niitä voidaan hyödyntää monissa eri paikoissa. Esimerkiksi navigaatioita varten suunniteltu BaseNav-komponentti sisältää monia erilaisia tyylejä, joilla saadaan komponentti toimimaan erilaisiin navigaatiotoimintoihin. Yleisissä komponenteissa kuitenkin on huomattavasti enemmän koodia, jotta ne toimivat oikein erilaisissa tapauksissa. Yleisien komponenttien rakentamisessa kannattaa kiinnittää huomiota siihen, milloin eriyttää omaksi komponentiksi samankaltainen toiminto. Esimerkiksi navigaatiokomponentissa voitiin hyödyntää samaa navigaatiota monessa eri paikassa ja kuvassa 17 on pari esimerkkiä siitä, miten komponentille on määritelty eri tyylit.



Kuva 17. BaseNav-komponentin kaksi erilaista tyylirakennetta.

Jotta komponenteille saadaan dataa, sitä on haettava rajapinnasta. Tätä toteutusta varten otettiin käyttöön vuex. Vuex-kirjastolla saatiin kätevästi kaikki rajapintakyselyt yhtenäistettyä ja samalla helposti kaikkien käärekomponenttien käyttöön tarvittaessa. Koska samaa dataa haetaan monesta komponentista samalla funktiolla, on hyvä käyttää jaettua tiedostoa, jossa kaikki nimet on määritelty. Monessa kohtaa nimen määritteleminen heikentää ylläpidettävyyttä ja yhteisellä määrittelytiedostolla saadaan parannettua ylläpidettävyyttä.

Määrittelytiedostoja on yleisessä koodissa paljon, sillä jokainen dokumenttityyppi tarvitsee oman määrittelyn. Määrittelyissä kerrotaan mm. dokumenttien tyyppi, kieli ja yleiset

asetukset, joita listauksiin ja sivunavigaatioon tulee mukaan. Nämä määrittelyt ovat niitä, jotka päätettiin viedä reitityksen mukana komponenteille. Niiden takia reitityksestä tuli tavallista monimutkaisempaa hahmottaa, ja niistä on myös kytköksiä palvelukohtaiseen koodiin. Kuvalla 18 havainnollistetaan yksinkertaisen määrittelytiedoston rakennetta.

client > common > config > material-type > JS labour-court.js

```
1  import { merge } from "lodash"
2  import defaultConfig from "./configuration/default"
3
4  export default {
5    materialType: "labour-court",
6    title: "label.labourCourt",
7    description: "description.labourCourt",
8    languages: ["fi"],
9    configuration: merge({}, defaultConfig, {
10     listing: {
11       caption: true,
12       title: true,
13       type: true,
14     },
15   }),
16 }
17
```

Kuva 18. Yksinkertainen määrittelytiedosto yhdelle materiaalille.

4.3 Palvelukohtainen koodi

Jokaisella palvelulla on ominaisuuksia ja asioita, jotka näkyvät vain kyseisen palvelun alla. Tämän takia päätettiin tehdä modulaarinen rakenne, jotta voidaan hyödyntää mahdollisimman paljon koodia yhteisellä puolella. Palvelukohtaiseen koodiin tullaan sijoittamaan kaikki ominaisuudet, joita halutaan räätälöidä. Palvelun koodi sisältää valmiin pohjan mukaisen rakenteen, johon tuodaan erilaisia räätälöitäviä ominaisuuksia määrittelytiedostoista. Tämä osuus haluttiin pitää mahdollisimman suppeana ja tuoda palveluille kaikki ominaisuudet yleiseltä puolelta.

Palvelukohtaista koodipohjaa varten rakennettiin skriptejä, joiden avulla saadaan uusi palvelukohtainen koodipohja oletusasetuksilla. Skriptien on tarkoitus pitää jokaisen

palvelun koodipohja samanlaisena, mutta tätä ei aina ylläpidetty ja sieltä saattoi puuttua uusimpia ominaisuuksia. Kuitenkin puutteita on helpompi korjailla kuin rakentaa liian monimutkainen valmis koodipohja.

Palvelukohtainen koodi sisältää pääasiassa määrittelytiedostoja, sillä suurin osa logiikasta on yleisellä puolella. Määrittelytiedoissa otetaan yleiseltä koodipuolelta tarvittavat reittimäärittelyt, jotka halutaan mukaan palveluun, ja määritellään, mitä asioita näytetään eri tavalla. Määrittelyissä voidaan myös muuttaa yleisiä määrittelyjä, jos jokin asia halutaan toteuttaa eri tavalla kuin mahdollisissa muissa palveluissa.

Kuitenkin logiikan toimimiseen tarvitaan yleiset Vue-kehyksen omat määrittelyt, jotka mahdollistavat kaikille komponenteille Vue-kehyksen toiminnallisuuksia. Muun muassa Vue-reitityksen ja Vue-suunnittelujärjestelmän käyttöönotto tehdään täällä. Kaikki yleiset asiat tuodaan npm-pakettienhallinnan kautta, kuten myös Vue-suunnittelujärjestelmä.

Suunnittelujärjestelmä käyttää semanttista versionhallintaa, jonka mukaan npm osaa ladata viimeisimmät muutokset. Semanttinen versionhallinta tarkoittaa sitä, että versionhallintaan annetaan versionumeromerkintä, joka on muotoa x.y.z. X tarkoittaa aiemman toiminnallisuuden rikkovaa ominaisuutta, y tarkoittaa isoa muutosta, ja z tarkoittaa pientä muutosta. Semanttista järjestelmää varten suunnittelujärjestelmän koodi pitää paketoida dist-kansioon build-komennolla ja viedä versionhallintaan versiomerkinnän kanssa.

Tämän jälkeen pitää antaa palvelukohtaisen koodin puolella npm install edita-ds -komento, minkä jälkeen uudet muutokset ovat palvelukohtaisessa koodissa mukana. Palvelukohtaisessa koodissa suunnittelujärjestelmän komponentit on otettu käyttöön kahdella rivillä koodia main.js-tiedostossa esimerkkikoodin 1 mukaisesti, minkä jälkeen niitä ei tarvitse enää uudelleen tuoda jokaiseen komponenttiin erikseen.

```
import DesignSystem from "edita-ds/dist/rajalex/rajalex.js"
Vue.use(DesignSystem)
```

Esimerkkikoodi 1. Vue-suunnittelujärjestelmä käyttöön main.js-tiedoston kautta.

4.4 Kehittämisen esimerkkivaiheet

Jokaista ominaisuutta varten täytyi suunnitella ensimmäiseksi, kuuluuko ominaisuus yleisen vai palvelukohtaisen koodin puolelle. Jos asiasta ei ollut vielä varmuutta, tehtiin toteutus yleiselle puolelle. Jokainen ominaisuus sisältää logiikan ja tyylirakenteen, mutta logiikka oli palvelun koodipuolella ja samaan aikaan saatettiin toteuttaa suunnittelujärjestelmän puolelle valmista komponenttia. Tässä kuitenkin kommunikaatio ja työskentelytapojen hiominen oli tärkeää, sillä työvaiheissa oli paljon asioita, jotka eivät aluksi toimineet tarpeeksi ketterästi.

Hakua toteuttaessa tarvittiin monenlaisia komponentteja, joiden koko vaihteli käärekomponentista aina yksinkertaisimpiin painikekomponentteihin. Toteutuksen tekijällä on hyvä olla kattava pohjatieto, millä tavalla koodipohja on toteutettu. Uusien tekijöiden tullessa kehittämään uusia ominaisuuksia huomattiin monia puutteita dokumentaatiossa ja tiedon leviämisessä. Tietoa kuitenkin saatiin kehittämisen yhteydessä tuotua uusille tekijöille ja dokumentaatiota parannettua huomattavasti. Kun koodipohjasta oli selvitetty mitä komponentteja voidaan hyödyntää esimerkiksi hakutoimintoja varten, voitiin alkaa kehittää jotain tiettyä haun ominaisuutta.

Komponentin logiikan kehitys

Ominaisuuden rakentaminen aloitettiin luomalla palvelun koodipohjaan komponentti, joka oli tarkoitus jälkeenpäin siirtää suunnittelujärjestelmän puolelle. Tämäntyyppinen kehittäminen otettiin käyttöön sen takia, ettei tarvinnut jokaisen muutoksen jälkeen siirtää koodia suunnittelujärjestelmän versionhallinnan kautta npm-pakettina palveluun käytettäväksi. Palvelun koodipohjassa komponenttiin tehtävät muutokset näkyivät suoraan, ja sitä on helpompi sen takia kehittää.

Logiikan tuomista komponentin käyttöön joutuu paljon miettimään, sillä lähes jokainen toiminto on määriteltävissä ja kaikissa palveluissa on eroja toteutuksessa. Määrittelytiedostoihin tuli paljon asioita, joissa kerrotaan, näytetäänkö kehitettyä ominaisuutta vai ei tämän dokumenttityypin yhteydessä. Esimerkiksi jokaisella dokumenttityypillä ei välttämättä ole sisällysluetteloa, joten sisällysluettelokomponentin täytyi olla määriteltävissä, näytetäänkö sitä vai ei, kun ollaan tietyn dokumenttityypin näkymässä.

Komponentin rakentamiseen käytettiin joko HTML-elementtejä suoraan tai valmiita komponentteja suunnittelujärjestelmän puolelta. Niiden avulla rakennettiin perusrakenne komponenttiin ja voitiin määritellä paremmin, mihin ominaisuusarvoista tuleva data viedään. Ominaisuusarvojen pitää olla mahdollisimman yleisiä rakenteeltaan, jotta komponentti voidaan siirtää lähes sellaisenaan suunnittelujärjestelmään. Komponentin tarvittavien tietojen haku suoritettiin joko vuex-tilahallinnan kautta tai määrittelytiedostoista käärekomponentin sisällä. Nämä tiedot annettiin komponentin omiin ominaisuusarvoihin.

Komponentti suunnittelujärjestelmään

Kun logiikka saatiin toimimaan palvelun sisäisessä komponentissa, se voitiin siirtää suunnittelujärjestelmän puolelle. Suunnittelujärjestelmän puolella saatettiin haluta pilkkoa komponentti pienempiin osiin, jotta toteutus on tarpeeksi yleisesti toteutettu. Myös toiminnallisuus saattoi mennä rikki, kun rakennetta muutettiin oikeantyyppiseksi, minkä jälkeen jouduttiin tekemään korjauksia logiikkaan asti. Kuitenkin jos saatiin komponentin rakenne aiemmin oikeaksi, komponenttiin voitiin samanaikaisesti lisätä tyylejä suunnittelujärjestelmän puolella. Tyylejä varten käytettiin SCSS-tyylitiedostoja, joiden avulla voidaan käyttää funktioita ja muuttujia tyylien mukana (23). SCSS on hyödyllinen suunnittelujärjestelmän suunnittelumuuttujien tuomiseen tyylitiedostojen käyttöön.

Koska suunnittelujärjestelmään ei tuotu logiikalla dataa, tarvittiin testidataa jostakin. Tämä data tuotiin kopioimalla selainpuolelle tuleva data json-tiedostoon, josta se voidaan tuoda suunnittelujärjestelmän komponentin käyttöön. Nämä json-tiedostot tallennettiin suunnittelujärjestelmään, jotta niitä voidaan käyttää hyödyksi myös myöhemmin. JSX-koodilla voidaan lukea tiedosto ja antaa siitä oikeat osat komponentin luettavaksi koodiesimerkin 2 mukaisesti.

```
<docs>
  ```jsx
  const searchSuggestions = require("@/examples/searchSuggestList.json")

  <SearchSuggest
    :suggestion-groups="searchSuggestions"
    @clearSuggest="clearSuggest"
  />
  ```
</docs>
```

Esimerkkikoodi 2. JSX-koodi suunnittelujärjestelmän dokumentaatiota varten.

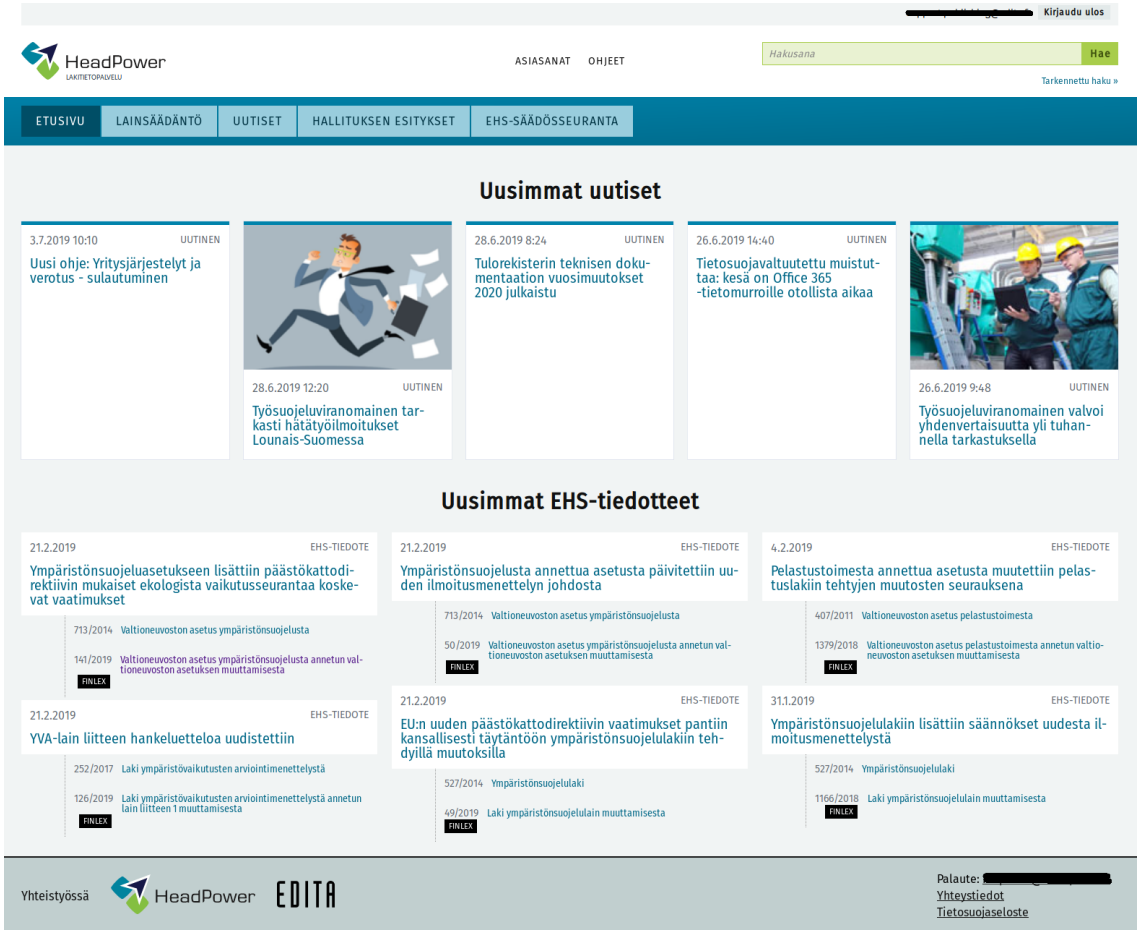
Jotta suunnittelujärjestelmään viety komponentti näkyisi palvelussa oikeassa paikassa, täytyy suunnittelujärjestelmä viedä npm-pakettienhallinnan läpi. Läpivientiin ensimmäiseksi vietiin muutokset Git-versionhallintaan, ja sinne luotiin merkki, jonka mukaan npm osaa hakea ja asentaa oikean version.

Kun kokonaisuus on kehittäjän puolesta valmis ja toimii omassa kehitysympäristössä, voidaan viedä koodi versionhallintaan ja sieltä testipalvelimelle, jossa testaaja pääsee testaamaan. Tarvittavat tiedot ominaisuudesta ja siitä miten sen pitäisi toimia, laitetaan Jira-korttiin ja kortti viedään eteenpäin testaajalle. Tästä testaaja tietää, että ominaisuus on valmis ja se voidaan manuaalisesti testata. Jos tehtävässä on vielä puutteita, laitetaan tehtävä takaisin kehittäjän ratkaistavaksi. Valmiiseen ominaisuuteen tehdään myös testiautomaatio, jotta tiedetään, jos ominaisuus hajoaa tai muuttuu.

5 Selainpuolen tulokset

5.1 Selainpuolen kehityksen havainnot

Koodialustan valmistuessa löydettiin paljon asioita, joita voidaan parantaa, etenkin siinä vaiheessa, kun ryhmään lisättiin tekijöitä. Asioita saatiin hyvin ratkottua, kehittäminen saatiin helpommaksi ja koodin ylläpidettävyyttä saatiin parannettua. Järjestelmän kokonaisuus oli varsin laaja, ja siinä uusittiin käytännössä koko dataputki uusilla tekniikoilla. Uusista tekniikoista Vue-kehys taipui tarpeisiin varsin hyvin ja sillä saatiin toteutettua kaikki ominaisuudet mitä haluttiinkin toteuttaa. Palvelun rakenne saatiin pidettyä koossa, vaikka komponentteihin jakaminen ei aina ollut selkeää. Kuvassa 19 on alustan ensimmäisen palvelun etusivu, jossa on monenlaisia komponentteja.



Kuva 19. Järjestelmällä toteutetun Plus-palvelun etusivu.

Kehityksen kulku oli hieman vaihtelevaa, sillä datamallien suunnittelu palvelun hakumoottorille jäi vähäiselle. Data haluttiin nopeasti selainpuolen koodin käsiteltäväksi, jotta saatiin jotain konkreettista näkyviin ja testattua Vue-kehyksen ominaisuuksia. Datamallit rakennettiin sitä mukaa, kuin saatiin asioita oikeille paikoille käyttöliittymässä.

Palvelu kuitenkin vaatii vielä parannuksia, jotta sillä voidaan korvata olemassa oleva vanha palvelun versio. Palvelu annettiin testattavaksi asiakkaalle, jotta asioita voidaan hienosäätää. Vanhaan versioon nähden muutokset ovat huomattavia, ja koodipohjaa saatiin nykyaikaistettua hyvin. Kuitenkin uusi koodipohja tarvitsee vielä paljon muutoksia, jotta siitä saadaan todellinen hyöty. Koska Vue-suunnittelujärjestelmä on erillinen osa eikä osa koodipohjaa, sitä voidaan hyödyntää myös vastaavanlaisissa palveluissa ja ottaa käyttöön myös muihin palveluihin esimerkiksi Edilex-palveluun.

Kehityksen aikana dokumentointitapoja kehitettiin eteenpäin, ja tärkeimmäksi dokumentaatioksi osoittautui readme-tiedostot järjestelmän osien juurikansioissa. Siellä oli kattavat ohjeet järjestelmän osien kehittämiseen sekä kehitysympäristön ylläpitoon liittyviä asioita. Myös suunnittelujärjestelmällä kehittämisen hitauden ratkaisuun kehitetyt skriptit vähensivät koodin käännösaikoja useita minuutteja, joten siitäkin oli suuresti hyötyä ylläpidettävyyden parantamisessa.

5.2 Jatkokehitys

Järjestelmän kehityksessä opittuja taitoja on tarkoitus hyödyntää muissakin palveluissa, joita Edita Publishing Oy ylläpitää esimerkiksi Edilex-palvelun uusimiseen. Edilex-palvelua ei kuitenkaan rakenneta samaan koodipohjaan Plus-palveluiden kanssa. Myös koodipohjaan lisätään monia samanlaisia palveluita ja tarkoituksena olisi saada ne mahdollisimman vähällä työllä valmiiksi asti.

Uudet palvelut samaan koodipohjaan

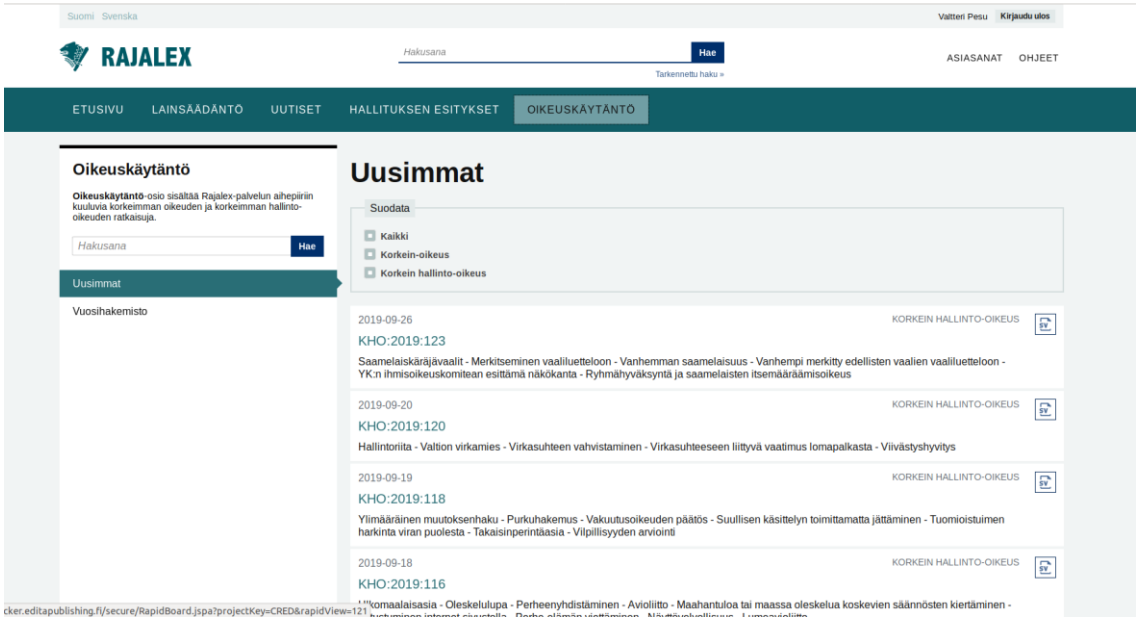
Kun ensimmäinen palvelu oli loppusuoralla, alettiin rakentaa uutta palvelua mukaan, jotta voidaan paremmin reagoida koodipohjassa oleviin puutteisiin ja ongelmiin. Alussa ei tiedetty minkälainen rakenne tulee olemaan, eikä voitu kehittää samanaikaisesti useaan palvelua. Palvelukohtaisen koodipohjan rakentaminen skriptien avulla osoittautui

hankalammaksi, sillä oletus koodipohjasta puuttui huomattavasti määrittelytiedostoja eikä niiden käyttöä ollut vähään aikaan testattu. Ongelmakohtiin lisättiin virheenhallintaa, jotta uuden palvelun kehittämisessä ei jouduta tutkimaan samoja ongelmia uudelleen.

Uuden palvelun oletuskoodipohjaan lisättiin ensimmäisenä Vue-suunnittelujärjestelmän riippuvuuden vaihtaminen, koska suunnittelujärjestelmään kehitettiin myös modulaarinen rakenne. Suunnittelujärjestelmän rakenteessa tuodaan oikeat suunnittelumuuttujat palveluun, kun se ladataan palvelun koodipohjaan npm-pakettina. Oletuksena uuteen palveluun suunnittelujärjestelmässä otetaan samat tyylit, minkä jälkeen käyttöliittymän suunnittelija suunnittelee ja muokkaa niitä palveluun sopivaksi.

Määrittelytiedostojen logiikkaan jouduttiin tekemään paljon muutoksia, sillä jokainen sivu oli toteutettu sisältämään vain yhden dokumenttityypin kerrallaan. Tätä mahdollisuutta ei ollut otettu huomioon ensimmäistä palvelua tehdessä, ja tämä mahdollisuus pitää olla monessa muussa palvelussa mukana. Määrittelytiedostojen logiikan muutokset vaikuttavat aina kaikkien alustan palveluiden toimintaan, joten myös ensimmäisen palvelun toiminta on testattava ja varmistettava, että asiat toimivat oikealla tavalla.

Kuvassa 20 on toinen samalla koodipohjalla toteutettu palvelu. Kuvan sivulla on useampia dokumenttityyppejä viety samaan listausrakenteeseen ja tuotu suodatinominaisuus, jotta voidaan eritellä dokumenttityyppejä halutessa. Ulkoisesti muutos ei näytä kovin isolta, mutta logiikan muutos oli ylläpidettävyyden kannalta varsin hankala hahmottaa.



Kuva 20. Toinen Plus-palvelu samassa koodipohjassa.

Vaikka selainpuolen perusrakenne saadaan nopeasti kokoon ja siten kätevästi ylläpidettäväksi, tällaiset pienet asiat, joita ei ole suunniteltu järjestelmää tehdessä, voivat aiheuttaa paljon lisää työmäärää. Kuitenkin alustaan on mahdollista lisätä palveluita, ja kun sitä parannetaan säännöllisesti toimimaan paremmin, voidaan saada nopeammin tehtyä uusia, koska kaikki tarvittavat ominaisuudet on toteutettu valmiiksi.

Edilex-palvelun uudistus

Edilex on mittakaavaltaan laajempi kokonaisuus kuin räätälöitävät Plus-palvelut. Edilex-palvelun koodipohjassa on paljon asioita, joita halutaan tehdä uudelleen, ja uudet onnistuneet palvelut osoittavat, että Edilex olisi mahdollista myös uudistaa samoilla teknologioilla. Koska järjestelmässä toteutetussa koodipohjassa huomattiin datamallien kanssa ongelmia, halutaan Edilex-uudistus aloittaa kehittämällä datamallit ja Elasticsearch-hakumoottori riittävän hyvään kuntoon. Tämän jälkeen voidaan pohtia, miten selainpuolen käyttöliittymän osia voidaan hyödyntää Edilex-palvelun uudistuksessa.

Edilex sisältää samoja dokumenttityyppejä, joten siinä voidaan hyvin hyödyntää Vue-suunnittelujärjestelmän komponentteja ja käyttää niitä omassa koodipohjassa. Tällöin saataisiin kehitystä nopeammaksi, sillä komponentteja olisi jo valmiiksi rakennettu

oikealla rakenteella. Myös kaikkia elementtitason komponentteja voidaan vapaasti hyödyntää Edilex-palvelussa.

Komponenttien lisäksi halutaan käyttää uusia tekniikoita, joita kehityksen aikana on opittu käyttämään paremmin. Uusissa tekniikoissa on käytetty monenlaisia toimintatapoja, ja niistä halutaan ottaa käyttöön eniten hyötyä tuovat toiminnot. Tekniikoiden ja toimintatapojen yhdistäminen helpottaa pidempiaikaista ylläpitoa. Kaikkia palveluita kehittää sama kehitystiimi, joten samalla tavalla toteutetut asiat on helpompi muistaa ja ongelmien etsiminen helpottuu, kun rakenteet eivät paljoa eroa toisistaan.

6 Yhteenveto

Plus-palveluiden uutta järjestelmää aloitettaessa ei vielä tiedetty, millaisella se aikataululla saadaan valmiiksi ja mitä esteitä tulee vastaan. Työn aikana uudistettiin kokonaisuudessaan vanha järjestelmä tietolähteestä palvelun käyttöliittymään asti, kaikki rakenneosaset. Insinöörityön osuudessa tehtiin pääasiassa Vue-kehystä ja Vue-suunnittelujärjestelmää. Palveluiden uudistuksessa haluttiin saada sama koodipohja kaikille Plus-palveluille ja nopeuttaa uusien palveluiden toteutusta.

Järjestelmän suunnitteluun olisi voinut käyttää enemmän aikaa, etenkin siihen, missä rakenteessa data tuodaan palveluun. Kuitenkin kehittämisessä saatiin korjattua ongelmia nopeasti tehdyillä korjauksilla ja kehittäminen helpottui huomattavasti. Näihin ongelmiin ei olisi pidempi suunnittelu auttanut. Koodipohjan kehittämisessä huomattiin myös, että dokumentaatio ja tiedon levittäminen muiden kehittäjien tietoon on hyvin tärkeää, jotta kehittäminen olisi nopeaa. Yhteisien sääntöjen sopimisessa huomattiin, että sopiminen kannattaa tehdä hyvissä ajoin, jotta koodiin ei tule epäselvästi nimettyjä funktioita ja tiedostonimiä.

Koodipohjan ensimmäisen palvelun kokonaisuus saatiin suurimmalta osalta valmiiksi, ja toinen palvelu saatiin myös hyvälle alulle. Niitä on tarkoitus vielä hienosäätää ja testata, jotta ne voidaan julkaista asiakkaiden käyttöön. Plus-palveluiden uudistaminen on hyvässä vauhdissa, ja lisää uudistettavia palveluita on tulossa. Vielä ei tiedetä, auttaako yhteinen koodipohja usean palvelun kehityksen nopeuttamiseen, sillä toisen palvelun aikana oli vielä paljon asioita, joita ei ollut otettu huomioon. Yhteisen koodipohjan kanssa myös määrittelytiedostojen määrä on suuri ja niiden ylläpitäminen vaatii paljon pohjatietoa koko järjestelmän rakenteesta.

Uusien teknologioiden käyttämisen opiskelu onnistui varsin hyvin ja opittiin paljon uutta siitä, miten niiden avulla voidaan kehittää tulevia projekteja paremmin. Mutta Vue-suunnittelujärjestelmän tuomia hyötyjä ei vielä suuremmin pystytty käyttämään, sillä kaikki palvelut ovat samassa koodipohjassa ja npm-pakettienhallinnan kautta tuominen hidastaa kehitystä. Jatkossa voidaan suunnittelujärjestelmän komponentteja hyödyntää Edilex-palvelun uudistuksessa, joten silloin vasta tiedetään, onko suunnittelujärjestelmän käyttö hyödyllistä.

Lähteet

1 Edilex Plus. Verkkoaineisto. Edilex. <https://www.edilex.fi/tietoa_palvelusta/plus>. Luettu 1.11.2019.

2 Daityari, Shaumik. 2019 Angular vs React vs Vue: Which Framework to Choose in 2019. Verkkoaineisto. Codeinwp. <https://www.codeinwp.com/blog/angular-vs-vue-vs-react/>. 13.6.2019. Luettu 15.10.2019.

3 Qian, Tim. Star history. Verkkoaineisto. <https://star-history.t9t.io>. Luettu 20.10.2019.

4 Single File Components. Verkkoaineisto. Vue.js. <https://vuejs.org/v2/guide/single-file-components.html>. Luettu 21.10.2019.

5 Directives. Verkkoaineisto. Vue.js. <https://012.vuejs.org/guide/directives.html>. Luettu 21.10.2019.

6 Sumeet, Roy. 2019. Component Communication in Vue.js. Verkkoaineisto. Medium. <https://medium.com/js-dojo/component-communication-in-vue-js-ca8b591d7efa>. 8.4.2019. Luettu 22.10.2019.

7 What is Vuex? Verkkoaineisto. Vuex. <https://vuex.vuejs.org>. Luettu 22.10.2019.

8 Salminen, Viljami. 2018. Vue Design System Docs. Verkkoaineisto. Github. <https://github.com/viljamis/vue-design-system/wiki>. 29.3.2018. Luettu 23.10.2019.

9 Hacq, Audrey. 2018. Everything you need to know about Design Systems. Verkkoaineisto. UX Collective. <https://uxdesign.cc/everything-you-need-to-know-about-design-systems-54b109851969>. 22.5.2018. Luettu 23.10.2019.

10 An open source tool for building Design Systems with Vue.js. Verkkoaineisto. Vueds. <https://vueds.com>. Luettu 23.10.2019.

11 Salminen, Viljami. 2018. Terminology. Verkkoaineisto. Github. <https://github.com/viljamis/vue-design-system/wiki/terminology>. 29.3.2018. Luettu 23.10.2019.

12 Design Tokens. Verkkoaineisto. Vueds. <https://vueds.com/example/#!/Design%20Tokens>. Luettu 24.10.2019.

13 Patterns. Verkkoaineisto. Vue Design System. <https://vueds.com/example/#/Patterns>. Luettu 24.10.2019.

14 What is JSX? Verkkoaineisto. React Enlightenment. <https://www.reactenlightenment.com/react-jsx/5.1.html>. Luettu 24.10.2019.

15 Choose the right Technology and Framework to your Projects. Verkkoaineisto. Morioh. <https://morioh.com/p/6b4b689a3d6b>. Luettu 25.10.2019.

16 Williams, Anthony. Writing Maintainable Code. Verkkoaineisto. Just Software Solutions. <https://www.justsoftwaresolutions.co.uk/articles/maintainable_code.html>. Luettu 25.10.2019.

17 What is Elasticsearch? Verkkoaineisto. Elastic. <https://www.elastic.co/what-is/elasticsearch> Luettu 3.11.2019.

18 What is npm? Verkkoaineisto. W3schools. <https://www.w3schools.com/whatis/whatis_npm.asp>. Luettu 1.11.2019.

19 Ketterien kehitystiimien ykköstyökalu. Verkkoaineisto. Atlassian. <https://www.atlassian.com/fi/software/jira>. Luettu 2.11.2019.

20 Git –everything-is-local. Verkkoaineisto. Git-scm. <https://git-scm.com>. Luettu 1.11.2019.

21 About. Verkkoaineisto. ESLint. <https://eslint.org/docs/about/>. Luettu 1.11.2019.

22 A node.js tool to automate end-to-end web testing. Verkkoaineisto. TestCafe. <https://devexpress.github.io/testcafe/>. Luettu 3.11.2019.

23 Documentation. Verkkoaineisto. Sass. <https://sass-lang.com/documentation>. Luettu 1.11.2019.