Haaga-Helia
ammattikorkeakoulu Oy

CMS-ratkaisu Maanmittauslaitoksen kirjaamissovelluksissa

Sulo Heija

Opinnäytetyö
Tietojenkäsittelyn
koulutusohjelma
2020

Tiivistelmä

Haaga-Helia ammattikorkeakoulu Oy

Tekijä(t)
Sulo Heija

Koulutusohjelma
Tietojenkäsittelyn koulutusohjelma

Raportin/Opinnäytetyön nimi
CMS-ratkaisu Maanmittauslaitoksen kirjaamissovelluksissa

Sivu- ja liitesivumäärä
46 + 1

Sisällönhallintajärjestelmä eli CMS on nykypäivänä tärkeä osa sovelluskehitystä. Kun sisältö on irrotettu sovelluksen rakenteesta, voivat kehittäjät keskittyä kehittämiseen, ja sisällöntuottajat sisällön tuottamiseen – sekä kaikkien osapuolten aikaa, rahaa ja hermoja säästyy. Toisaalta CMS-ohjelmiston valinta on haastavaa, koska ”hyvän” ominaisuuden määritelmä riippuu täysin siitä, mihin ohjelmisto tulee käyttöön.

Opinnäytetyö tehtiin toimeksiantona Maanmittauslaitokselle. Vaikka Maanmittauslaitoksen päätehtävä on mitata maata, on sillä myös kohtuullisen suuri sovelluskehitysosasto, jonka eräs osa-alue on nimeltään ”kirjaamissovellukset”. Kirjaamissovelluksiin vastaavasti kuuluu useampi sovelluskehitysprojekti. Opinnäytetyön päätehtävänä oli tutkia, millaisia CMS-tarpeita kirjaamissovelluksien eri projekteissa on, mitä CMS-sovelluksia kirjaamissovelluksissa on jo käytössä, ja pystytäänkö näillä sovelluksilla vastaamaan täyttämättömiin CMS-tarpeisiin. Mikäli ei pystyttäisi, kartoitettaisiin kaupallisia sovelluksia.

Käyttäjien tarpeet ja käytössä olevat CMS-ohjelmistot kartoitettiin haastattelemalla 15:tä kirjaamissovelluksissa työskentelevää henkilöä. Haastatteluiden perusteella todettiin, että kirjaamissovelluksissa ei ole sovelluskehitykseen soveltuvia CMS-ohjelmistoja käytössä, ja erityisesti kehittäjät joutuvat tekemään tehtäviä, jotka CMS-ohjelmistolla olisivat helposti jonkun ei-teknisen henkilön tehtävissä.

Koska jo käytössä olevia ohjelmistoja ei ollut, tehtiin vertailu kaupallisten ohjelmistojen välillä. Vertailussa painotettiin MML:n erityistarpeita valtiollisena toimijana, sekä haastatteluissa ilmenneitä konkreettisia tarpeita. Voittajaksi valikoitui Directus Suite, joka oli ainoa, joka vastasi MML:n teknisiä tarpeita täydellisesti. MML oli tyytyväinen opinnäytetyön tulokseen, ja tutkii, voidaanko ratkaisu ottaa käyttöön.

Asiasanat
CMS, sisällönhallinta, tarvekartoitus, maanmittauslaitos

Sisällys

1 Johdanto

Opinnäytetyö tehtiin toimeksiantona Maanmittauslaitokselle (MML). Opinnäytetyön tarkoituksena oli kartoittaa MML:n sisällönhallintaohjelmistotilannetta, ja etsiä ratkaisuja mahdollisesti löytyviin sisällönhallinnan ongelmiin.

MML:n päätehtävänä on nimensä mukaisesti mitata maata, eli muun muassa tuottaa maastodataa, tehdä maanmittaustoimituksia, sekä huolehtia kiinteistöjen omistukseen ja kauppaan liittyvästä rekisterinpidosta (Maanmittauslaitos 2020). MML on kuitenkin myös keskisuuri ohjelmistoalan yritys, koska se kehittää suuren osan tarjoamistaan digitaalisista palveluista itse (Maanmittauslaitos 2020). MML:n tarjoamiin julkisiin palveluihin kuuluvat esimerkiksi maastotietopalvelu Karttapaikka, useita rajapintapalveluita, sekä erilaisia kiinteistökauppaan liittyviä asiointipalveluja. MML vastaa myös useiden perusrekisterien ylläpidosta, ja tuottaa näiden rekistereiden ylläpitoon tarvittavia sovelluksia.

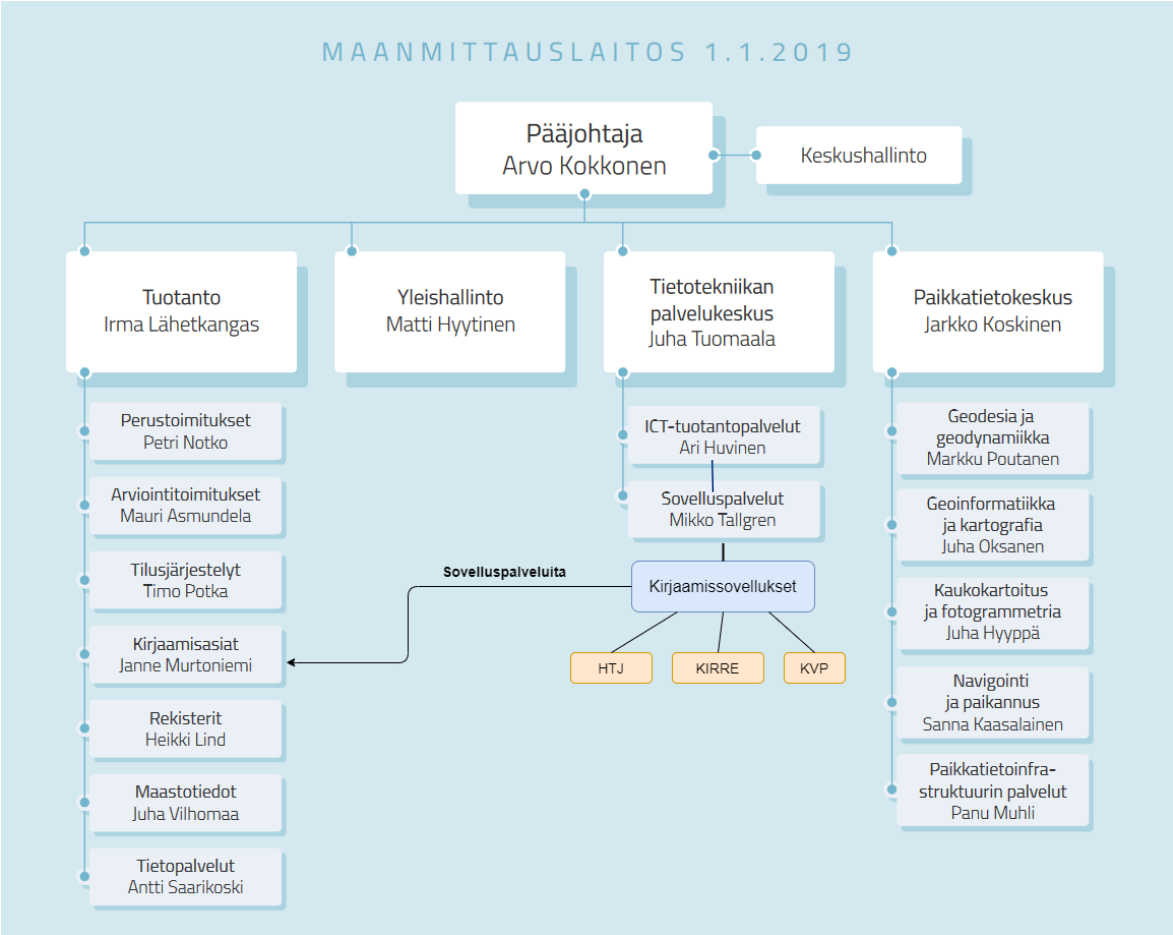
Nykypäivänä sovelluskehitys on usein monimutkaista, sekä teknisesti että organisaatiollisesti. Laadukas sovellus vaatii varsinaisten ohjelmistokehittäjien lisäksi käyttöalan tuntevia määrittelijöitä, käyttöliittymäsuunnittelijoita, sisällöntuottajia ja kääntäjiä - sekä paljon muuta osaamista, jota on mahdoton tyhjentävästi luetella. Jotta kaiken tämän osaamisen tuominen yhteen olisi sujuvaa, kuuluvat moderniin sovelluskehitykseen erilaiset ”apuohjelmistot”, kuten esimerkiksi projektinhallinta- tai sisällönhallintasovellukset.

Koska MML:n on laaja organisaatio, on uusien ohjelmistojen käyttöönotto raskas toimenpide. Tämän takia sovellushankintojen tulee olla tarkkaan pohdittuja, ja olemassa olevien ohjelmistojen hyödyntämismahdollisuudet tulee kartoittaa huolella. Koska MML:n on osa julkishallintoa, kohdistuu ohjelmistovalintoihin myös enemmän sellaisia turvallisuus- ja luotettavuusvaatimuksia, joita ei huomioida samalla tavalla yksityissektorin hankinnoissa. Näytä vaatimuksia käsitellään tarkemmin opinnäytetyön produktiosuudessa, luvussa 6.1 MML:n yleiset vaatimukset.

1.1 Aiheen tausta

Opinnäytetyö syntyi tarpeesta saada MML:n Huoneistotietojärjestelmää (HTJ) kehittävälle HTJ-projektille käyttöön sisällönhallintajärjestelmä (Content Management System, CMS), jotta kehittäjien ei tarvitsisi päivittää tekstejä ja tehdä muita ei-teknisiä päivityksiä. HTJ kuuluu kirjaamissovelluksiin, joka on MML:n sovelluspalveluyksikköön kuuluva osa-alue, joka vastaa MML:n kirjaamisasioiden sovelluspalveluista (kuva 1). Kirjaamissovelluksiin kuuluvat HTJ:n lisäksi lainhuuto- ja kiinnitysrekisterin ylläpitojärjestelmä (KIRRE) ja

kiinteistövaihdannan palvelu (KVP) – näiden sovellusten kehittämisprojekteja vastaavasti kutsutaan KIRRE- ja KVP-projekteiksi.



Kuva 1. MML:n organisaatiorakenne. Vaaleansinisellä on kuvattu organisaatioyksiköt, ja oranssilla projektit. (muokattu, alkuperäinen Maanmittauslaitos 2019)

Kirjaamisasiat sisältävät käytännön virkamiespalvelut, eli HTJ:n tapauksessa kiinteistöjen vaihdantaan ja panttauksiin liittyvien hakemusten käsittelyn, rekisteröinnin ja ratkaisun.

Opinnäytetyön teon mahdollisuutta kartoittaessa paljastui, että MML:n sovelluspalveluyksikössä on tällä hetkellä käytössä useita ominaisuuksiltaan päällekkäisiä CMS-ohjelmia, jotka vaihtelevat kaupallisista itsekoodattuihin. Työn aiheeksi valikoitui näin tarve selvittää, mitä ohjelmistoja kirjaamissovelluksilla on jo käytössä, mitä täyttämättömiä tarpeita kirjaamissovelluksilla on CMS-ohjelmiston osalta, ja pystyisivätkö jo käytössä olevat ohjelmistot vastaamaan näihin tarpeisiin. Projektissa tutustutaan myös markkinoilla oleviin CMS-sovelluksiin siltä varalta, että käytössä olevista ohjelmistoista mikään ei sovellu kirjaamissovellusten tarpeisiin. Käytössä olevat sovellukset ja täyttämättömät tarpeet selvitetään vapaamuotoisten haastatteluiden avulla, ja kaupallisiin sovelluksiin tutustuminen tehdään kirjallisena vertailuna.

MML:llä on tehty aikaisemmin sisällöltään tämän opinnäytetyön tyyppisiä selvityksiä käytössä olevista CMS- ja muista ohjelmistoista, mutta ne ovat kohdistuneet eri osastoihin. Tuloksia voidaan hyödyntää tässä projektissa.

Projekti hyödyttää sovelluspalveluita tuomalla selkeyttä siihen, millaisia ohjelmistoja osastolla on käytössä, ja kartoittamalla kenties tuntemattomia kipukohtia kirjaamissovelluksissa, joita CMS voisi helpottaa. Mikäli tämän lisäksi jokin CMS-ohjelmisto otetaan kirjaamissovelluksissa käyttöön, järjestelmien tuki ja ylläpito virtaviivaistuu.

1.2 Rajaukset

MML:n sovelluspalvelut ovat erittäin laaja kokonaisuus, joten projektissa ei kartoitettu kirjaamissovellusten ulkopuolella käytössä olevia CMS-sovelluksia. Kaupallisten CMS-ohjelmistojen kartoitus ja vertailu tehdään puhtaasti kirjallisena, koska MML:llä ei ole mahdollista tehdä tämän opinnäytetyön puitteissa pilottihankkeita tai muita vastaavia käytännön kokeiluja.

1.3 Opinnäytetyön rakenne ja tutkimusmetodit

Koska opinnäytetyö toteutettiin toiminnallisena opinnäytetyönä, koostuu se kahdesta osasta: teoriaosuudesta ja ns. produktista, joka on tässä tapauksessa käytännön selvitystyö.

Teoriaosuudessa käsitellään sisällönhallintaa konseptina ja eritellään CMS-ohjelmistojen kaksi päätyyppiä. Tämän lisäksi käsitellään tiettyjä CMS-ohjelmistoilta usein löytyviä ominaisuuksia – käsiteltävät ominaisuudet ovat sellaisia, jotka ovat merkityksellisiä tämän opinnäytetyön selvitysosuuden kannalta.

Selvitystyö toteutettiin kesällä 2020 ja se koostui karkeasti kolmesta osasta: asiasta aikaisemmin tehtyjen selvitysten analysoinnista, kartoitusvaiheesta ja CMS-vaihtoehtojen vertailusta. Aikaisemmat selvitykset koostuivat MML:n sisäisistä asiakirjoista, ja niistä tehtiin yhteenveto tämän opinnäytetyön näkökulmasta. Kartoitusvaiheessa selvitettiin täyttämättömät CMS-tarpeet, sekä mitä CMS-ohjelmistoja kirjaamissovelluksissa on jo käytössä. Sekä tarpeiden selvittämiseen että CMS-ohjelmistojen kartoitukseen käytettiin haastatteluja ja vapaamuotoisia keskusteluja MML:n henkilöstön kanssa – metodi valittiin, koska asiasta ei yksinkertaisesti ollut kirjoitettua tietoa. Haastatteluita analysoitiin

syvällisesti, ja ongelmakohdat sekä niistä syntyvät tarpeet eriteltiin. CMS-ohjelmistojen vertailu suoritettiin laadullisena vertailuna kirjaamissovellusten tarpeita vasten.

Laadullisessa tutkimuksessa perehdytään tutkittavan kohteen tarkoitukseen ja ominaisuuksien merkitykseen esiintymisympäristössään (Jyväskylän yliopisto 2015a), ja sillä vastataan kysymykseen ”millainen?”. Laadulliseen tutkimukseen päädyttiin, koska CMS-ohjelmistojen eroja on vaikea esittää numeerisesti kadottamatta valinnan kannalta merkityksellistä informaatiota. MML ei myöskään tee sitovaa päätöstä pelkästään opinnäytetyön pohjalta, vaan asiaa käsitellään lisää – tämän vuoksi analyysi ohjelmiston ominaisuuksista koettiin jatkon kannalta hyödyllisemmäksi, kuin pelkkä toteamus siitä, kuinka monta kohtaa asetetuista vaatimuksista sovellus täyttää.

Vertailevassa tutkimuksessa verrataan mitä tahansa vertailukelposeksi todettuja tapauksia tai ilmiöitä (Jyväskylän yliopisto 2015b). Tässä tapauksessa vertailtiin eri CMS-ohjelmistoja keskenään. Vertailuun otetut kaupalliset sovellukset etsittiin google-hakukoneesta hakemalla ja AlternativeTo-palvelua käyttäen. AlternativeTo-palvelussa käyttäjä voi hakea jotain tuntemaansa ohjelmistoa, ja saada listan muista samankaltaisista ohjelmistoista (AlternativeTo 2020).

Koska tarpeiden kartoituksessa ilmenneistä asioista tehtiin kohtuullisesti analyysia, voidaan opinnäytetyön haastatteluosuuden katsoa sisältävän myös tapaustutkimuksellisia elementtejä. Tapaustutkimuksessa tutkitaan jotain tiettyä tapausta syvällisesti, kuitenkin tavoitteena tuottaa yleistettävissä olevaa tietoa (Jyväskylän yliopisto 2015c). Tässä tapauksessa se, millainen sisällönhallinnan prosessi on kirjaamissovelluksissa, voidaan katsoa tutkimuksen kohteena olevaksi tapaukseksi.

1.4 Käsitteet

Scrum: Yksi malli sovelluskehityksen organisoimiseksi. Scrumin pääpointteina ovat kehityksen jakaminen lyhyisiin ”sprintteihin”, ja kehitystiimien suuri itseohjautuvuus verrattuna muihin malleihin. (Sininen meteoriitti 2020.)
Sprint: Tyypillisesti muutaman viikon mittainen kehitysjakso, jonka aikana sovelluksesta tehdään uusi, käyttökelpoinen versio. Vääntyy suomeksi yleensä muotoon ”sprintti”. (Sininen meteoriitti 2020.)
CMS: Content Management System, eli sisällönhallintajärjestelmä. Käytetään sisällöntuoton erottamiseen sovelluslogiikasta. (Burgy 2020.)
CDA: Content Management Application - sisällönhallintajärjestelmän osa, joka tarjoaa käyttöliittymän sisällön tuottamiseen ja hallitsemiseen. (Burgy 2020.)
CMA: Content Delivery Application – sisällönhallintajärjestelmän osa, joka yhdistää sisällön ja rakenteen esittäväksi kerrokseksi, jota voidaan katsella selaimella. (Burgy 2020.)
Front-end, FE: Esittävä kerros, eli selaimessa näkyvä osuus verkkosivusta tai muusta web-sovelluksesta (Pastorino s.a.).
Back-end, BE: Logiikkakerros sovelluksesta. Sovelluksen varsinainen toiminta tapahtuu tyypillisesti back-endissä, josta tiedot lähetetään front-endille näytettäväksi (Pastorino s.a.).
REST: Representational Data Transfer. Hyvin yleinen arkkitehtuurimalli sovellusrajapintojen rakentamiseen. Data tarjotaan asiakassovellukselle palvelimen määrittämässä muodossa. (Red Hat 2020.)
GraphQL: Kyselykieli rajapintojen rakentamiseen. Palvelin tarjoaa dataa asiakassovelluksen tekemän kyselyn määrittämässä muodossa. (Sturgeon 2017.)
Webhook: ”käänteinen” rajapinta reaaliaikaiseen reagointiin – asiakassovellus ei tee kyselyä, vaan antaa rajapinnan, johon palvelin voi lähettää dataa, kun jotain tapahtuu. (Sengrid 2017.)
Perusrekisteri: Viranomaisen ylläpitämä hallinnollinen rekisteri, esimerkiksi kiinteistörekisteri tai väestötietorekisteri. (Tieteen termipankki 2020.)
Admin-sovellus: Tässä opinnäytetyössä tarkoittaa sovellusta, jolla voidaan tehdä erinäisiä ylläpidollisia tehtäviä, kuten muokata tiettyjä sovelluksen arvoja tai käynnistää eräajoja.
Eräajo: Ajastetusti tai manuaalisesti käynnistettävä ohjelma, joka voidaan ajaa ilman käyttäjän interaktiota (IBM 2010). Käytetään MML:llä esimerkiksi kuntatietojen ajantasaistamiseen vuosittain, kun uudet kuntaliitokset ovat astuneet voimaan.
HTJ: Huoneistotietojärjestelmä. Sovelluskokonaisuus, jota kehitetään HTJ-projektissa.
Osakeluettelon siirtosovellus: Julkisessa verkossa näkyvä osuus HTJ:stä; Asiointipalvelu osakeluettelon sähköistämiseen.

KIRRE: Lainhuuto- ja kiinnitysrekisterin ylläpitopalvelu. MML:n sisäisesti käyttämä sovellus, jota kehitetään KIRRE-projektissa.
KVP: Kiinteistövaihdannan palvelu. Julkinen palvelu, jota kehitetään KVP-projektissa.
Käyttöönottoprojekti: Täsmällisemmältä nimeltään ”HTJ-käyttöönottoprojekti”, joka on HTJ-projektin sisarprojekti. Projektin merkityksellisin tehtävä *tämän opinnäytetyön kannalta* on sisällöntuotto HTJ-projektille.

2 CMS-ohjelmistot

2.1 Sisällönhallinnasta yleisesti

Kun puhutaan sisällönhallintajärjestelmästä (Content Management system, CMS), voidaan asiayhteydestä riippuen tarkoittaa mitä tahansa yrityksen dokumenttienhallintaohjelmistosta selainpohjaiseen blogipalveluun. Sovelluskehityksen kontekstissa CMS-ohjelmistolla tarkoitetaan tyypillisesti web-sisällön hallinnointiin tarkoitettua järjestelmää, ja se on myös se CMS:n laji, josta tässä opinnäytetyössä puhutaan. (Heslop 2018.)

Web-sisältö, eli verkkosivut, voidaan karkeasti jakaa kahteen osaan: sisältöön ja rakenteeseen. Sisällöllä viitataan mihin tahansa palaan informaatiota, kuten esimerkiksi kuvaan, artikkeliin, tai infotekstiin. Rakenteella taas tarkoitetaan sitä, miten sisältö sijoitetaan verkkosivulle. Sivusto ilman rakennetta on muodoton kasa tekstiä, ja sivusto ilman sisältöä on ihmissilmälle tyhjä. (Vaxter & Vogt 2002.)

Perinteisesti verkkosivut ovat koostuneet palvelimella sijaitsevista staattisista HTML-tiedostoista, joissa rakenne ja sisältö sijaitsevat rinnakkain koodissa, mahdollisesti samassa tiedostossa. Mikäli sivuston sisältöä halutaan muuttaa, on muutoksen vienyt paikalleen asiaan vihkiytynyt henkilö, jolla on sivun muokkaamiseen vaadittava tekninen osaaminen. (Vaxter & Vogt 2002.)

Nykyaikana web-kehitys on siirtynyt voimakkaasti teknisesti monimutkaisempien, dynaamisten verkkosivujen suuntaan. Siinä missä staattisen verkkosivun rakenne on lyöty lukkoon, voi dynaamisen verkkosivun sisältö ja rakenne muuttua käyttäjän toimien tai muun ulkoisen tekijän toimesta erilaiseksi ilman, että sivua tarvitsee ladata uudelleen - hyvänä esimerkkinä dynaamisista verkkosivuista ovat erilaiset sosiaalisen median palvelut (Heslop 2018). Vaikka dynaamiset verkkosivut ovat moderneja, on sisällönhallinnan ongelma kuitenkin säilynyt samana – sisällön muokkaamiseen tarvitaan sisällön tuottajan lisäksi koodausosaamista omaava henkilö.

Ongelman ratkaisemiseksi on kehitetty sisällönhallintajärjestelmän (CMS) konsepti. CMS koostuu kahdesta osasta: varsinaisesta sisällönhallintaohjelmistosta (content management application, CMA) jolla sisältöä voidaan luoda ja muokata, sekä sisällön tarjoavasta osasta (content delivery application, CDA) joka yhdistelee sisällön rakenteeseen ja luo selaimessa näytettävän verkkosivun (Heslop 2018). Koska sisältö on

täysin eriytetty koodista, voivat sisällöntuottajat muokata sitä omilla työkaluillaan, ilman teknistä osaamista (Vaxter & Vogt 2002).

2.2 Perinteinen CMS

Perinteisesti CMS-ohjelmistot ovat olleet kokonaisvaltaisia ohjelmistoja, jotka sisältävät sekä CMA-osuuden että CDA:n (kts. 2.1 Sisällönhallinnasta yleisesti). Sisältö voidaan tuottaa ohjelmiston käyttöliittymän tekstieditorissa, ja kun se julkaistaan, generoidaan staattinen verkkosivu automaattisesti sisällön pohjalta. Perinteinen CMS mahdollistaa näin web-sisällön luomisen lähes ilman koodausosaamista - vaikka CDA-osuuden asetukset vaativat yhä teknistä osaamista kyseisestä sovelluksesta, voidaan järjestelmän pystyttämisen jälkeen verkkosivua muokata puhtaasti käyttöliittymästä käsin. (Heslop 2018.)

Perinteisellä CMS:llä on tyypillisesti luotu staattista tekstisisältöä, esimerkiksi blogeja, ja ohjelmiston CDA-osuus yleensä tukee tätä käyttöä parhaiten (Burgy 2020). Tämä heijastuu myös ohjelmistojen sisällön rakenteeseen, joka on yleensä perusoletuksena artikkelimuotoista (Wordpress 2020). Näistä syistä perinteisellä CMS:llä on vaikeaa luoda dynaamisia verkkosivuja, jotka dynaamisen rakenteen lisäksi vaativat usein monimutkaisia datamalleja.

Useimmissa perinteisissä CMS-ohjelmistoissa on mahdollisuus CDA:n luoman verkkosivuston kustomointiin, mutta se vaatii sovelluskohtaista erikoistunutta teknistä osaamista. CMA ei yleensä tarjoa sisältöään rajapinnan kautta, jolloin oman CDA:n tai front-ending luominen ei mahdollista. Mikäli samaa sisältöä halutaan käyttää useammassa sovelluksessa ns. monikanavaisesti, on CMS-ohjelmiston spesifisesti tuettava tätä ominaisuutta. (Burgy 2020.)

Eräitä suosittuja perinteisiä CMS-ohjelmistoja ovat Wordpress, Joomla ja Drupal. Näistä MML:llä on käytössä Drupal. Perinteiset CMS:ssät ovat erittäin suosittuja verkkosivustojen alustoina, ja vuonna 2020 63,5% kaikista CMS-ohjelmistolla rakennetuista sivuista oli tehty nimenomaan Wordpressiä käyttäen. (W3Techs 2020.)

2.3 Headless CMS

Siinä missä perinteisessä CMS:ssä CMA ja CDA on nivottu tiukasti yhdeksi ohjelmistoksi, ei headless CMS:ssä ole CDA:ta ollenkaan. Sisältö luodaan perinteisen CMS:n tapaan ohjelman käyttöliittymästä käsin, jonka jälkeen se tarjoillaan ulos standardin rajapinnan

kautta. Koska CMS-ohjelmisto on näin täysin irrotettu esittävästä kerroksesta, ei valittu CMS-ohjelmisto rajoita front-endissä käytettäviä tekniikoita. (Heslop 2018.)

Sisältörajapintana käytetään tyypillisesti REST-rajapintoja, toisinaan tarjolla on myös GraphQL-rajapinta. Molemmat tyypit mahdollistavat usean front-end sovelluksen yhdistämisen samaan rajapintaan, jolloin samaa sisältöä voidaan käyttää monikanavaisesti webissä, mobiilissa ja vaikkapa intranetissä. Front-endissä käytettävät teknologiat eivät ole sidottu käytettävään CMS-ohjelmistoon, ja esimerkiksi vanhentunut front-end voidaan vaihtaa uuteen sisällön katoamatta. Tiedonsiirto rajapintojen yli mahdollistaa myös integroinnin esimerkiksi lokalisointipalveluiden tai data-analytiikkaohjelmistojen kanssa. (Heslop 2018.)

Koska headless CMS-ohjelmistot on suunniteltu toimimaan hyvin monenlaisen sisällön kanssa, tukevat ne hyvin monenlaisia ja monimutkaisia datamalleja. Osa ohjelmistoista tukee myös sisältöjen välisiä relaatioyhteyksiä, joiden avulla voidaan saavuttaa relaatiotietokantamaista logiikkaa sisältötyyppien välillä. (Contentful 2020.)

Headless CMS:ssän negatiivisena puolena on sen käytön vaatima tekninen osaaminen. Koska ohjelmistolla ei voi luoda front-endiä, tarvitaan sen tekemiseen kehittäjiä ja sisällön hyödyntämiseen rajapintaosaamista. Mikäli käytettävä datamalli on monimutkainen, voidaan tarvita myös tietokantaosaamista.

Eräitä suosittuja headless CMS ohjelmistoja ovat Strapi ja Contentful. MML:llä ei ole headless CMS:ssää käytössä. Isossa mittakaavassa headless CMS-ohjelmistot eivät ole erityisen suosittuja, mutta niiden käytön uskotaan yleistyvän voimakkaasti lähivuosina (Kentico Software 2018). Kentico Softwaren teettämässä kyselyssä 38 % vastanneista kertoi suunnittelevansa headless CMS:n käyttöönottoa seuraavan 3–12 kuukauden aikana (Kentico Software 2018).

3 CMS-ohjelmistolta vaadittavia teknisiä ominaisuuksia

CMS-ohjelmistot eroavat ominaisuuksiltaan huomattavasti. Myös käyttökohde vaikuttaa merkittävästi siihen, mikä CMS-ohjelmisto on ominaisuuksiltaan “paras”. Mikäli CMS-ohjelmisto halutaan ottaa käyttöön isossa yrityksessä tai virastossa, täytyy siinä tyypillisesti olla itse CMS-ominaisuuksien lisäksi liuta muita, nimenomaan yrityskäyttöön räätälöityjä ominaisuuksia. Koska kaikkien mahdollisten ominaisuuksien läpikäynti on mahdotonta, on tässä luvussa käsitelty nimenomaan tämän opinnäytetyön haastatteluosuudessa ja teknisessä vertailussa esiin tulleita ominaisuuksia.

3.1 Lähdekoodi

Lähdekoodi tarkoittaa kehittäjän luomaa koodia, joka “käännetään” konekielelle varsinaiseksi käytettäväksi sovellukseksi. Mikäli lähdekoodi on avointa, on se kenen tahansa saatavilla. Useat avoimen lähdekoodin projektit ovat myös sellaisia, että kehittämiseen voi osallista kuka vain. Suljettu lähdekoodi sen sijaan ei ole saatavilla, ja sovellusta kehittää yleensä jokin yritys. (Wikipedia 2020a.)

Koska kuka tahansa voi ottaa avoimen lähdekoodin sovelluksesta kopion itselleen lähdekoodin kautta, on sen kaupallistaminen hankalaa – tyypillisesti sovellus itse on ilmainen, ja kehittäjät myyvät erinäisiä tukipalveluita, tai sovelluksen kehitys voidaan rahoittaa säätiön tai lahjoitusten avulla. Suljetun lähdekoodin kehittäjä voi taas yksinkertaisesti myydä sovelluksestaan lisenssejä, joka on helposti lähestyttävää ja tuottoisaa. (Wikipedia 2020a.)

Mutta kumpi on parempi? Vastaus tähän riippuu täysin siitä, keneltä kysytään. Avoimen lähdekoodin sovelluksen käytettävyys ja ominaisuudet voivat olla rahanpuutteen takia huonommat, ja kehitystiimi ylläpitää sovelluksen tietoturvaa silloin kun päivätöiltään ehtivät. Vaihtoehtoisesti sovelluksen kehitykseen aktiivisesti osallistuva monipäinen yhteisö kiillottaa käyttöliittymää minkä ehtii, ja lisäpalveluiden myynti kattaa jämäkän tietoturvaosaamisen tuomat kulut. Suljetun lähdekoodin sovellus voi kaupallistaa sovelluksensa helposti, ja käyttää tuottonsa sovelluksen ominaisuuksien parantamiseen, hiomiseen ja tietoturvaan. Yritys myös voi jakaa rahat omistajien taskuun, ja lopettaa sovelluksen kehittämisen seinään. Valinta avoimen ja suljetun lähdekoodin välillä ei siis vaikuta suoraan sovelluksen laatuun, mutta se vaikuttaa siihen, mitkä asiat ovat tärkeitä kehittäjän luotettavuutta tarkasteltaessa.

3.2 Autentikointi

Kertakirjautuminen (single sign-on, SSO) on tekniikka, joka mahdollistaa yksien kirjautumistietojen käyttämisen useampaan palveluun, joihin jokaiseen käyttäjä joutuisi normaalisti kirjautumaan erikseen. Eri SSO-palvelut eroavat teknisesti toisistaan, mutta perusajatus on kaikissa sama: käyttäjä kirjautuu SSO-palveluun, jonka jälkeen muut palvelut tarkistavat käyttäjän kirjautumisen SSO-palvelusta kirjautumistietojen kyselyn sijaan. Mikäli käyttäjällä on SSO-palvelussa aktiivinen kirjautuminen, ja hänellä on oikeus käyttää tätäkin palvelua, hänet kirjataan automaattisesti sisään. (Rouse 2020.)

SSO:n käyttö helpottaa kirjautumistietojen hallintaa yrityksessä, koska eri tunnusten määrä pienenee. SSO-palvelu voidaan myös laittaa esimerkiksi kaksivaiheisen tunnistautumisen taakse, jolloin tietomurto vaikeutuu huomattavasti. (Rouse 2020.)

Kaksivaiheinen tunnistautuminen (two-factor authentication, 2FA) tarkoittaa periaatetasolla sitä, että palveluun kirjautuva käyttäjä tunnistetaan tunnus-avainparin lisäksi jollain muulla tavalla – tyypillisesti noudatetaan sääntöä “jotain mitä käyttäjä tietää, jotain mitä hänellä on, ja jotain mitä hän on”. Käytännössä tämä voi tarkoittaa esimerkiksi kirjautumista älykorttia käyttäen, jolloin käyttäjän on tiedettävä tunnuksensa, älykortin on oltava käyttäjän hallussa, ja käyttäjän pitää olla fyysisesti paikalla älykortin kanssa kirjautumistilanteessa. (Ciprlani & Rosenblatt 2015.)

2FA:n käyttö tekee kirjautumisesta hitaampaa, mutta parantaa tietoturvaa huomattavasti (Ciprlani & Rosenblatt 2015). Se ei kuitenkaan ole aukotonta – etenkin kohtaa “jotain mitä käyttäjä on” on teknisesti vaikea toteuttaa, koska esimerkiksi älykortti voidaan aina varastaa. Biometriikasta toivotaan helpotusta kaksivaiheisen tunnistuksen turvallisuuden parantamiseen (Ciprlani & Rosenblatt 2015).

3.3 Palvelutasosopimus, SLA

Palvelutasosopimus (service level agreement, SLA) on nimensä mukaisesti sopimus palvelun tasosta asiakkaan ja palvelun tuottajan välillä. Sopimuksessa sovitaan esimerkiksi, kuinka usein palvelu saa vikaantua, miten nopeasti tilanteeseen reagoidaan ja kuinka nopean ajan sisällä palvelun tulee olla korjattu. Mikäli palvelu ja palveluntarjoaja ei pysy sovittujen rajojen sisällä, seuraa siitä sovittu sanktio. SLA:n tekeminen on yleistä yritysmaailmassa, sillä se vähentää epäselvyyttä ongelmatilanteissa. (Wikipedia 2020b.) Koska SLA:n tekeminen on yleistä, otettiin sen tarjoaminen yhdeksi CMS-ohjelmistojen teknisen vertailun kriteeriksi.

3.4 On-premise

Mikäli sovellus asennetaan “on-premise”, tarkoittaa se asennusta käyttäjän itsensä ylläpitämille palvelimille. Hyötyinä ovat asennuksen kustomoitavuus, ja haittoina taas palvelimen ylläpidon ja sovelluksen käyttöönoton tuomat kustannukset. Tieturva ja datan turvaaminen ovat myös käyttäjän omissa käsissä - onko tämä positiivinen vai negatiivinen asia, riippuu tapauksesta. (Cequens 2018.)

On-premise on periaatteessa termin väärinkirjoitettu muoto, oikea muoto on “*on-premises”* (Merriam-Webster 2020). Väärä muoto on kuitenkin huomattavasti yleisempi ja isojen

palveluntarjoajien käytössä, joten tässä opinnäytetyössä asetutaan kielisodassa voittajien puolelle.

3.5 Pilvipalvelu

Pilvipalveluksi kutsutaan palvelua, jossa sovelluksen asennus ja kaikki käyttäjän data sijaitsevat palveluntuottajan palvelimilla. Pilvipalvelun hyötyinä ovat palvelinten ylläpitokustannusten ulkoistaminen ja käyttöönoton helppous, huonoina puolina rajatut kustomointivaihtoehdot. Tietoturva ja datan suojelu ovat palveluntarjoajan vastuulla, joka voi olla jokohyvä tai huono asia tilanteen mukaan. (Cequens 2018.)

3.6 WYSIWYG

WYSIWYG-lyhenne tulee englannin kielen lauseesta “what you see is what you get” eli vapaasti suomennettuna “saat sen mitä näet”. Sitä käytetään kuvaamaan käyttöliittymiä, joissa käyttäjän luoma dokumentti näyttää luomisvaiheessa samalta kuin lopputulos - esimerkiksi tekstieditoria, jossa tekstin kursivointi näytetään *kursivointina,* eikä kursivointimerkkeinä “ ” (Wikipedia 2020c). CMS-ohjelmistossa tämä helpottaa sisällöntuottajan työtä, koska se mahdollistaa myös rakennetietoa sisältävän sisällön luomisen ilman teknistä osaamista. Tästä syystä WYSIWYG otettiin yhdeksi CMS-ohjelmistojen teknisen vertailun kriteeriksi.

3.7 REST-rajapinta

REST eli Representational State Transfer on arkkitehtuurimalli sovellusrajapintojen rakentamiseen. Eräitä REST-mallin pääpiirteitä ovat: selkeä jako asiakkaan ja palvelimen välillä, palvelimen päättämä tiedonsiirtoformaatti asiakaskohtaisen formaatin sijasta, sekä tilattomuus. Tilattomuudella tarkoitetaan, että jokainen kutsu käsitellään riippumattomana muista kutsuista. REST-rajapinnat ovat suosittuja keveytensä, luettavuutensa ja monipuolisuutensa vuoksi. (Red Hat 2020.)

REST-rajapinnat käyttävät lähes aina HTTP-protokollaa, jolloin käytössä ovat yleisimmin komennot GET, POST, DELETE ja PUT (hae, lisää, poista ja muokkaa). Tietojen siirto tapahtuu yleensä JSON-dataformaatissa, joskin arkkitehtuurimalli tukee monia muitakin formaatteja, muun muassa XML:llää. (Red Hat 2020.) Varsinaista url-osoitetta, johon kutsu tehdään, kutsutaan ”endpointiksi”.

Käytännössä REST-rajapintaa voi hyödyntää CMS-kontekstissa esimerkiksi seuraavasti: Palvelimella pyörivä headless CMS-ohjelmisto (kts. 2.3 Headless CMS) tarjoaa REST-

rajapinnan, jossa on tietty määrän endpointteja eri asioita varten. Yhdestä endpointista tarjotaan lista kaikista ”blogiartikkeli”-tyyppisistä datariveistä. Mikä tahansa asiakassovellus voi hakea kyseistä listaa GET-pyyntönä - CMS-palvelin käsittelee pyynnön ja palauttaa taulukon JSON-objekteja, jokainen objekti ollen yksi artikkeli. Asiakassovelluksen front-end voi nyt esittää saamansa datan listana kaikista artikkeleista. Jos asiakassovellus haluaa jotain muuta, joutuu se tekemään uuden pyynnön eri endpointtiin.

3.8 GraphQL-rajapinta

GraphQL on verrattain uusi keksintö – se julkaistiin 2015 Facebookin toimesta. Siinä missä REST on arkkitehtuurikuvaus rajapinnalle, on GraphQL kyselykieli rajapintakutsujen tekemiseen. Molemmat käyttävät HTTP-protokollaa, ja palauttavat tietoa JSON-muodossa. (Sturgeon 2017.) GraphQL-rajapintoja voidaan käyttää rinnan REST-mallin kanssa (Sturgeon 2017), mutta pelkällä GraphQL-rajapinnalla varustettu palvelu ei ole REST-mallin mukainen GraphQL:n erityispiirteiden vuoksi.

GraphQL:n pääpiirteenä on asiakassovelluksen mahdollisuus vapaasti päättää, mitä tietoja se palvelimelta hakee. Siinä missä REST-mallin mukaan palvelimella on olemassa tietyt endpointit, joista saa tiettyjä ennalta määriteltyjä asioita, voidaan yhteen GraphQL-endpointtiin tehdä minkälainen kysely tahansa. Tämä tekee GraphQL-rajapinnasta erittäin joustavan, koska rajapintaa rakentaessa ei tarvitse erikseen miettiä, mitä tietoa rajapinnasta halutaan ulos. Kääntöpuolena on, että yksinkertaisten toimintojen kohdalla GraphQL-kutsu on vastaavaa REST-kutsua monimutkaisempi, koska rajapinta vaatii aina eksaktin tiedon siitä, mitä halutaan tehdä (kuva 2). (Sturgeon 2017.)

```
GraphQL:
POST /graphql HTTP/1.1
Host: localhost:3000
Content-Type: application/graphql

mutation addImage {
  addImageFromUrl(image_url: "https://example.org/pic.png") {
    id,
    image_url
  }
}

REST:
POST /images HTTP/1.1
Host: localhost:3000
Content-Type: application/json

{
  "image_url" : "https://example.org/pic.png"
}
```

Kuva 2. Esimerkki kuvan lisäyksestä GraphQL- ja REST-rajapintojen kautta (mukailtu, alkuperäinen Sturgeon 2017)

Mikäli edellisen luvun (3.8 Rest-rajapinta) headless CMS-esimerkki toteutettaisiin GraphQL-rajapinnalla, tarjoaisi CMS-palvelin vain yhden endpointin – asiakassovellus voisi tästä endpointista hakea mitä vain, mitä palvelimella on tarjolla.

3.9 Docker

Docker on sovellus, joka mahdollistaa sovellusten asentamisen ns. ”kontteihin” -englanniksi ”container”. Kontti sisältää kaikki riippuvuudet, mitä sovellus toimii toimiakseen. Mikäli samalla fyysisellä palvelimella halutaan pyörittää useampaa ohjelmaa, on ohjelmistojen eristäminen toisistaan tärkeää tietoturvan kannalta – perinteisesti fyysinen palvelin jaetaan useaan virtuaalipalvelimeen, joilla on jokaisella oma käyttöjärjestelmänsä. Konttiteknologia mahdollistaa saman lopputuloksen ilman virtuaalipalvelimia, joka on resurssitehokkaampaa. (Docker 2020.)

3.10 Webhook

Webhookit, joita toisinaan kutsutaan myös nimellä ”HTTP push API”, ovat sovellusten tapa lähettää tietoa toisiin sovelluksiin automaattisesti tiettyjen ehtojen täyttyessä. Perinteisessä rajapinnassa asiakassovellus kyselee palvelimelta tietoa, ja reaaliaikainen seuranta vaatii jatkuvia kyselyjä. Webhookkia käytettäessä tilanne on päinvastainen: asiakassovellus antaa palvelimelle rajapintaosoitteen, johon palvelin lähettää asiakassovellukselle tietoa silloin, kun jotain tapahtuu. (Sengrid 2014.)

4 MML:llä aikaisemmin tehdyt selvitykset

Maanmittauslaitoksella on aikaisemmin tehty useita selvityksiä sisällön- ja dokumentinhallinnan tilaan liittyen. Selvitykset ovat iteroivasti perustuneet edellisiin selvityksiin, ja niissä on kartoitettu sisällönhallinnan nykytilaa, sen mahdollisia ongelmia ja esitetty mahdollisia korjaavia toimenpiteitä perusteluiden kera. Selvitykset ovat kuitenkin kohdistuneet laitokseen yleisesti tai muihin osastoihin, eikä niissä ole kartoitettu käytännön sovelluskehitystyötä tekevien tarpeita tai mielipiteitä asian suhteen. Näin ollen niitä ei voitu suoraan soveltaa kirjaamissovelluksiin, jossa kipupiste oli nimenomaan kehittäjien ahdinko.

Selvityksissä on nostettu esiin mm. sisällönhallinnan terminologian epäyhtenäisyys, laitostasoisen ohjeistuksen ja yhtenäisen toimintavan puuttuminen, sisällönhallinnan strategian puuttuminen, sekä tiedon tallennuspaikkojen liian suuri määrä. Terminologian epäyhtenäisyys vaikeuttaa yhtenäisen sisällönhallinnan ohjeistuksen muodostamista, joka on yhtenäisen strategian puutteen kanssa mitä ilmeisemmin johtanut sisällönhallinnan käytäntöjen ja dokumentointikulttuurin eriytymiseen eri alueiden välillä. Liian monet tallennuspaikat aiheuttavat tiedot hajaantumista eri järjestelmiin, joka taas vaikeuttaa tiedon löytämistä. Osasta tallennuspaikkoja puuttuvat myös versionhallinta, käyttöoikeuksien rajoittamisen työkalut, tai molemmat.

Selvityksissä on ehdotettu mm. seuraavia muutosehdotuksia:

- Tarvitaan laitoksenlaajuinen strategia sisällönhallintaan
- Tarvitaan yhteinen sisällönhallinnan käsitteistö
- Tarvitaan kattava ohjeistus tiedonhallinnan käytännöistä
- Päällekkäisistä järjestelmistä pyritään luopumaan
- Drupalista luovutaan, koska sille on vaikea saada resursseja
- Ns. ATK Konna Wikistä luovutaan, ja siirrytään kokonaan Confluenceen. Selvityksissä ei ollut selkeää mainintaa siirtymisen syystä, mutta yksi mahdollinen syy on sovellusten ominaisuuksien päällekkäisyys.
- Intra- ja extranet siirretään toimimaan samalla alustalla

Suurimmasta osasta selvityksessä mainituista asioista ei löytynyt tarkempaa tietoa, onko niiden kanssa edetty. Poikkeuksena on Confluenceen siirtyminen, joka on ainakin kirjaamissovellusten alueella kokonaan toteutettu. Selvityksissä huomautettiin, että Confluencen käyttöä ei laajenneta MML:ssä, ennen kuin mahdollisuus sen käytöstä intranetin alustana on selvitetty. Intranetin uudistamisesta on maanmittauslaitoksen sisällä käynnistetty projekti, jossa selvitetään eri mahdollisuuksia intranetin uudistamiseen. Huomautettiin myös, että Confluencen käytössä on huomioitava ohjelmiston lisenssikustannukset ja niiden kasvu, mikäli Confluencen käyttöä laajennetaan – joskin

selvityksissä ei täsmennetty, onko Confluencen lisenssikustannuksissa jotain erityistä, vai onko kyseessä yleinen muistutus lisenssimaksujen olemassaolosta.
Tämän opinnäytetyön kannalta merkittävä seikka on intranetin sisällönhallinnan uudistamisprojekti, joka on alkanut opinnäytetyön kanssa samoihin aikoihin, ja tulee kestämään usean vuoden. Projektista on saatavilla tietoa, mutta sen suurempi integroiminen tähän projektiin ei sovi aikabudjettiin. Intranetin uudistuksen merkittävin vaikutus opinnäytetyölle on kuitenkin se, että projektin takia MML:llä on vahvasti suositeltu, että MML:n Confluencen käyttöä ei laajenneta, ennen kuin sen mahdollinen käyttö intranetin alustana on selvitetty. Tämä luonnollisesti vaikuttaa Confluencen sopivuuteen HTJ-projektissa, jossa ratkaisua saatetaan kaivata, ennen kuin intranet-projekti on päättynyt. Myös Drupalista on tarkoitus luopua, jolloin sekään ei ole hyvä vaihtoehto.

5 Tarpeiden kartoitus

Koska MML:n on suuri organisaatio, on uusien ohjelmistojen käyttöönotto suhteellisen raskas prosessi. Eri sidosryhmillä voi olla myös keskenään ristiriitaisia toiveita ohjelmiston suhteen. Näistä syistä projektin jäsenten tarpeiden kartoitus etukäteen on erittäin tärkeää.

Tarpeiden kartoitus toteutettiin haastattelemalla henkilöstöä HTJ, KVP ja KIRRE-projekteista aikavälillä 25.5.2020 – 27.6.2020. Haastattelut toteutettiin etäyhteydellä, ja haastatteluista tehtiin muistiot MML:n käyttöön. Haastateltavia oli yhteensä 15 henkeä. Haastatteluihin kutsuttiin HTJ-projektista testaajia, scrum-master, vaatimusmäärittelijä sekä kehittäjiä. KVP- ja KIRRE-projekteista haastateltiin molempien projektien tuoteomistajia. Näiden lisäksi haastateltiin henkilöitä MML:n hallinnosta, joiden työalue liittyy sovellusarkkitehtuuriin tai sovellusten ylläpitoon. Haastateltavilta kysyttiin, millaisia yleisiä vaatimuksia heidän mielestään CMS-sovelluksilla tulee olla, mitä CMS-ohjelmistoja he käyttivät työssään ja mitä muita CMS-ohjelmistoja he tietävät MML:llä olevan käytössä. Lisäksi kartoitettiin, millainen sisällön muokkaamisen prosessi on tällä hetkellä, ja kuinka paljon sisällönmuutokset vievät kehitysajasta. Lopuksi haastateltavia pyydettiin kuvailemaan mikä nykyisessä sisällönhallinnassa "ei toimi".

6 Haastattelulöydökset

6.1 MML:n yleiset vaatimukset

MML on osa julkishallintoa, joten MML:llä käytössä olevilla teknologioilla on paljon vaatimuksia, joista osa tulee lainsäädännöstä. Mikäli ohjelmistolla pääsee esimerkiksi katselemaan arkaluontoista tietoa tietokannasta, tulee siitä arkistolain mukaisesti jäädä kuusi vuotta säilyvä loki, joka tulee säilyttää lain edellyttämällä tavalla, ja jossa käy täsmällisesti ilmi, kuka tietoa katseli ja mitä hän näki (Arkistolaki 23.9.1994/831). Näin ollen ohjelmistossa täytyy olla täydellinen katseluloki, loki täytyy olla mahdollista lähettää keskitettyyn lokijärjestelmään, ja käyttäjän henkilöllisyyden pitää olla ehdottomasti tiedossa. MML on koostanut vaatimuksensa ns. teknologiakäsikirjaksi, joka on yli 100-sivuinen MML:n sisäinen dokumentti – laajuutensa vuoksi sen sisältöä ei referoida tarkemmin tässä opinnäytetyössä.

MML ylläpitää ns. perusrekistereitä, jotka ovat viranomaisen ylläpitämiä hallinnollisia rekistereitä (Tieteen termipankki 2020). Koska perusrekisterissä oleva tieto toimii useiden muiden rekistereiden perustana, on tiedon eheys ensiarvoisen tärkeää. Tieto on myös mahdollisesti hyvin arkaluontoista, joten tietoon pääsyä pitää hallinnoida huolellisesti. Tiedon ja sitä tarjoavien palveluiden pitää myös olla saavutettavissa, eli yhden palvelimen kaatuminen ei saa viedä mukanaan koko palvelua.

Haastateltavat nostivat vaatimuksista esiin erityisesti muutoshistorian, lokituksen ja käyttövaltuushallinnan tärkeyden. Muutoshistoria auttaa ylläpitämään tiedon eheyttä ja on jatkuvuuden edellytys, ja erityisesti arkaluontoisten tietojen muokkauksen ja katselun lokittaminen auttaa jäljittämään väärinkäytöksiä. MML:n sisäisiin tietoihin käsiksi pääsevien henkilöiden tulee olla vahvasti tunnistettuja - käytännössä sovellusten tulee olla yhteensopivia MML:n käyttövaltuushallinnan kanssa, jolloin vahvaa tunnistautumista ei tarvitse tehdä joka sovelluksessa erikseen.

On myös huomioitava, että sovellusten kriittinen sisältö, kuten termistö, ei saa muuttua yllättäen. Toiminnan tulee “näyttää lain kirjaimen mukaiselta”, eli käytettävien termien tulee aina vastata lainsäädännössä käytettyjä termejä - esimerkiksi huoneiston käyttötarkoitus tulee olla “lasten päiväkoti”, eikä pelkkä “päiväkoti”. Sisällönhallinnan prosessi tulee suunnitella nämä seikat huomioiden.

6.2 CMS-ohjelmistot

Kirjaamissovelluksissa on yleisessä käytössä Confluence ja Jira. Näiden lisäksi käytetään kohtuullisesti asianhallintasovellus ASHA:a, ja dokumenttihallintasovellus ASHA-DM:ää, jotka ovat MML:n ulkopuolta ostamia räätälöityjä sovelluksia. Mikään näistä sovelluksista ei ole varsinaisesti CMS-ohjelmisto sovelluskehitysmääritelmän mukaan (kts. 2.1 Sisällönhallinnasta yleisesti), mutta ne yleisesti mielletään MML:n sisällä CMS-sovelluksiksi.

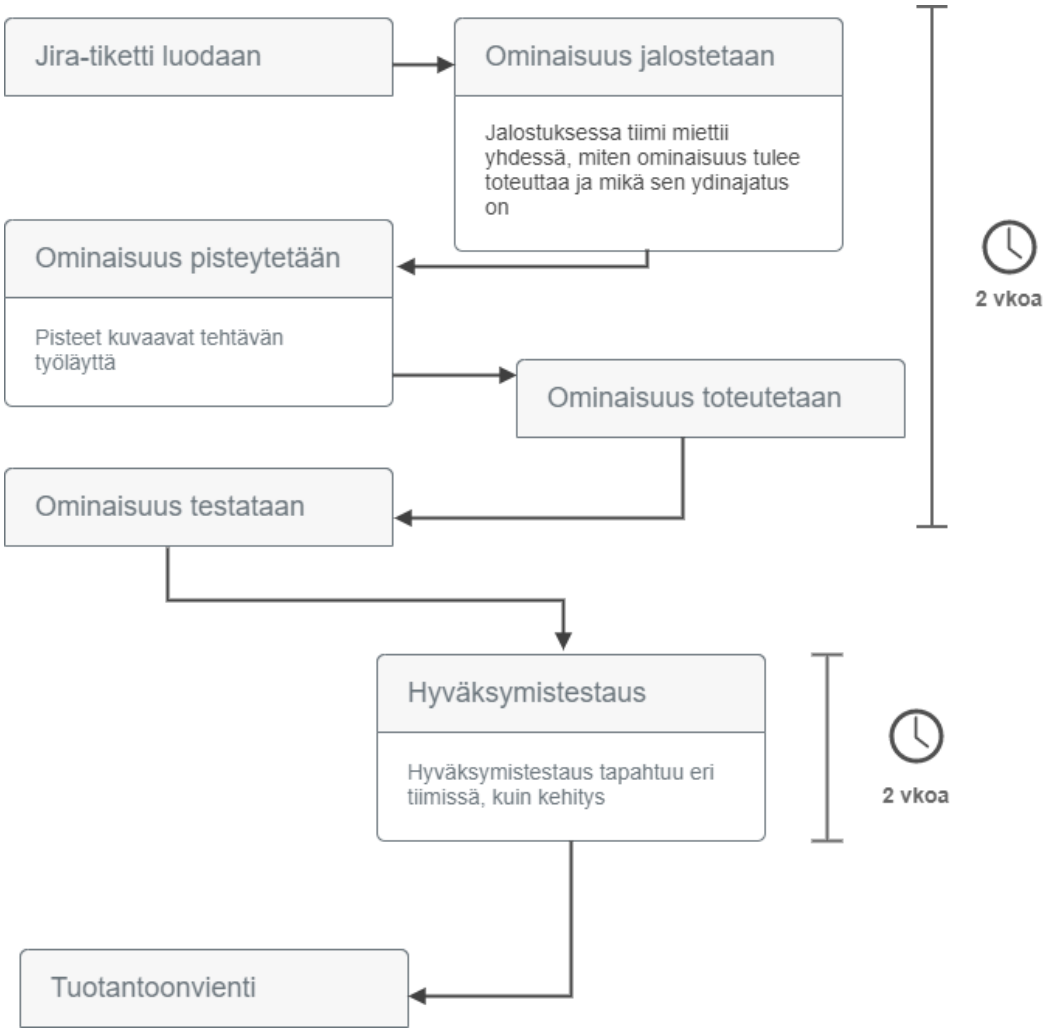
Varsinaisia sovelluskehityksen CMS-ohjelmistoista MML:llä oli käytössä vain Drupal, joka on käytössä intranetin ja ulkoisten verkkosivujen alustana.

Yllä mainittujen ohjelmistojen lisäksi KVP- ja KIRRE-projekteissa oli molemmissa käytössä itsetehdyt admin-sovellukset, nimiltään KVP-admin ja KIRRE-admin. Näistä etenkin KVP-admin sisälsi headless-CMS:ssää muistuttavia ominaisuuksia. HTJ-projektissa ei ollut käytössä mitään vastaavaa.

6.3 Sisällön muuttamisen prosessi

Kun puhutaan sovelluksen sisällöstä, tarkoitetaan sovelluksen staattisia tekstejä, kuten esimerkiksi otsikoita, ohjetekstejä ja käyttöehtoja, ja toisaalta myös tietokannassa säilytettävää tietoa kuten koodiarvoja. CMS-ohjelmistolla voidaan tyypillisesti hallita myös kuvasisältöä, mutta HTJ-, KIRRE- ja KVP-projektien kehittämissä sovelluksissa ei ole juurikaan kuvia.

Kaikissa kolmessa projektissa projektia hallitaan Jirassa, joten sisällönmuutoksen prosessi alkaa Jira-tiketillä. Projekteissa käytetään scrum-mallia kehittämiseen - tiketti jalostetaan, pisteytetään, otetaan sprintille, toteutetaan ja testataan kehitystiimin sisällä. Sprintin jälkeen tiketti siirtyy hyväksymistestaukseen, jossa ominaisuutta testataan lisää - vasta tämän jälkeen ominaisuus on valmis vietäväksi tuotantoon (Kuvio 1). Tuotannonviennin ajankohta vaihtelee projektien välillä - KVP- ja HTJ-projekteissa sovellus päivittyy aina jokaisen hyväksymistestauksen lopussa, KIRRE-projektissa taas neljännesvuosittain.

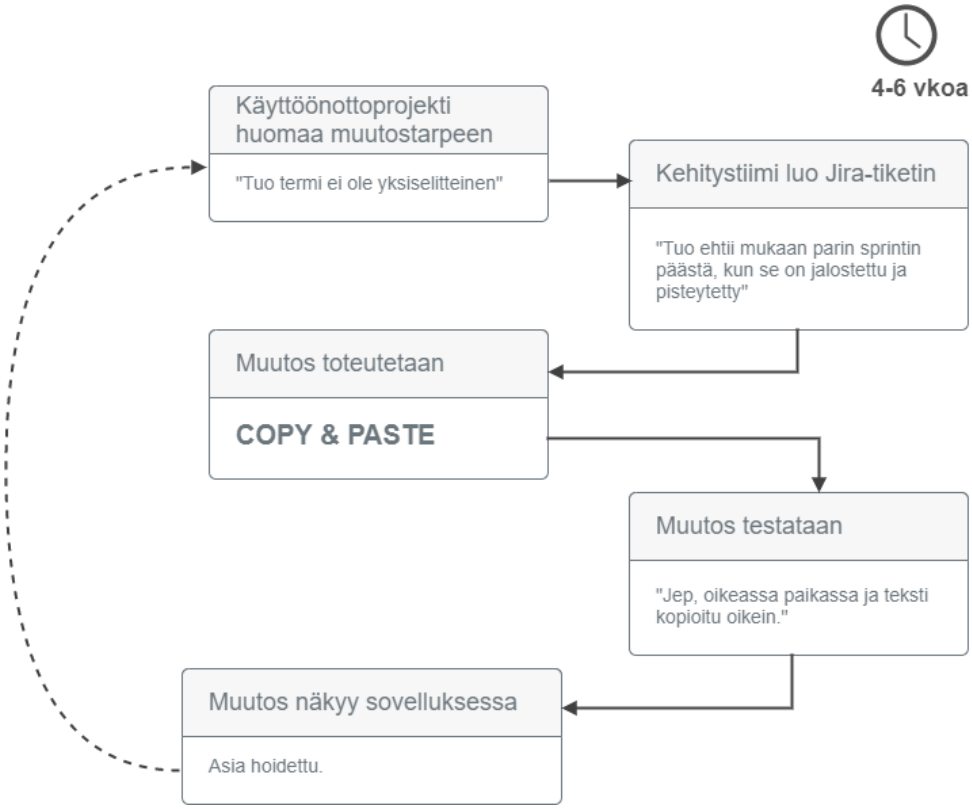


Kuvio 1. Sovelluskehityprosessi MML:lla

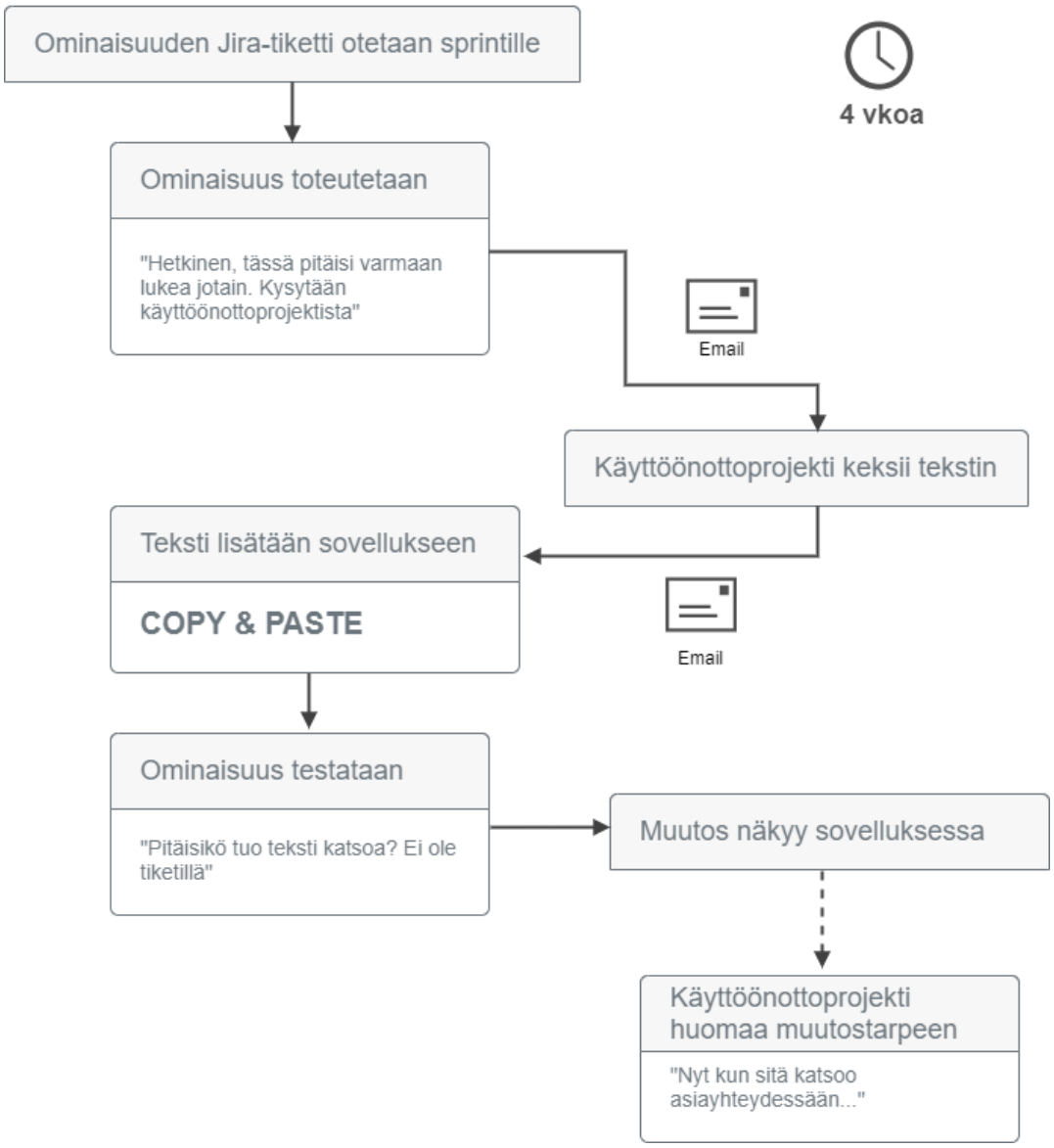
KVP- ja KIRRE projekteissa tekstimuutos voidaan yleensä toteuttaa kummankin sovelluksen omasta admin-sovelluksesta, ja muutoksen pystyvät tekemään asiakasvastaavat tai kirjaamisprosessin henkilöt. Muutoksen testaus kehitystiimissä päätetään tapauskohtaisesti testaajien toimesta. HTJ-projektissa sovelluksen sisältöä ei voi muokata ulkopuolelta, joten tekstimuutoksen tekevät kehittäjät ja muutokset testataan samaan tapaan kuten muutkin kehitystiketit. HTJ-projektissa on myös käytössä runsaasti automaattitestejä, jotka takaavat sovelluksen laatua.

HTJ-projektissa sisällönhallintaan osallistuu kaksi sidosryhmää: itse HTJ-projekti kehittäjineen (täsmällisemmin HTJ-kehitysprojekti) ja HTJ-käyttöönottoprojekti, joka on rinnakkainen projekti, jonka vastuualueena on kehitettävän sovelluksen sujuva käyttöönotto ja viestintä loppukäyttäjien kanssa. Sisällönhallinnan prosesseja on niin ikään kehittynyt kaksi: sisältömuutokset pistetään alulle joko käyttöönottoprojektista, kun tarvetta sisältömuutoksiin tulee (kuvio 2), tai sitten kehityspuolella luodaan jokin ominaisuus, johon erikseen pyydetään tekstejä käyttöönottoprojektista (kuvio 3). Mikäli aloite tulee

käyttöönottoprojektista, on sisältö dokumentoitu Jira-tiketille, mutta jos aloite tulee kehitystiimistä, hoidetaan asia sähköpostitse eikä pysyvää dokumentaatiota synny. Molemmissa tapauksissa sisältö siirtyy sovellukseen kopioimalla.



Kuvio 2. Käyttöönottoprojektista alkunsa saava sisällönhallintaprosessi. Prosessikuvausta on yksinkertaistettu luettavuuden vuoksi.



Kuvio 3. Kehitysprojektista alkunsa saava sisällönhallintaprosessi. Prosessikuvausta on yksinkertaistettu luettavuuden vuoksi.

KIRRE- ja KVP-projekteissa olemassa olevaa sisältöä muokataan vain vähän, "säännöllisen epäsäännöllisesti". HTJ-projektissa sisältömuutoksia on keskimäärin 1–3 kappaletta per sprintti.

6.4 CMS-ohjelmiston tarve

KVP- ja KIRRE-projekteissa oltiin enimmäkseen tyytyväisiä sisällönhallinnan tilaan. HTJ-projektissa sen sijaan erityisesti siirtosovelluksen parissa työskentelevät kehittäjät kokivat, että CMS-ohjelmisto helpottaisi heidän työskentelyään mahdollistamalla työn siirtämisen suoraan sisällön tuottajille. HTJ-projektin testaajien ja scrum-masterin työskentelyyn CMS-ohjelmisto ei vaikuttaisi suuntaan tai toiseen, paitsi mikäli ohjelmiston käyttöönotto muuttaisi kehitysprosessia merkittävästi. Useat haastateltavat kehitystiimeissä ja hallinnossa kokivat, että esimerkiksi vikatiedotteiden tekemiseen ja levittämiseen koko MML:n laajuinen CMS olisi hyödyllinen.

6.5 Ongelmakohdat

6.5.1 Sisällönhallinta MML:ssä

Yleisellä tasolla ongelmaksi koettiin se, että MML:n sisällönhallinnasta ei ole kokonaiskuvaa, jonka takia on vaikea suoralta kädeltä sanoa mitä kaikkea MML:llä on käytössä. Erilaisten käytössä olevien sisällönhallintajärjestelmien määrä eri laitoksen osissa on iso. Haastateltavat totesivat myös, että MML:llä sisällönhallinta ei ole osa prosessia, jonka takia kehitysratkaisuja ei tehdä sisällönhallintaa silmällä pitäen. CMS joudutaan täten integroimaan jälkijunassa, kun tuotantokäytössä huomataan sisällönmuutostarpeita. Kenties juuri tämän takia MML:llä on myös useita, ominaisuuksiltaan samankaltaisia itsekehitettyjä admin-sovelluksia käytössä, joiden kehittämiseen käytetty aika olisi voitu kenties käyttää järkevämmin, jos sisällönhallinta otettaisiin paremmin huomioon kehittämisessä.

6.5.2 Sisällönhallinta HTJ:n siirtosovelluksessa

Erityisesti HTJ-projektissa koettiin, että sisällönhallinnan roolitus ontuu - kehittäjät joutuvat koodaamaan tekstimuutoksia, joiden tekeminen sisällönhallintaohjelmistolla olisi triviaalia ja helposti jonkun ei-kehittäjän, esimerkiksi tekstin tuottajan itsensä, toteutettavissa. Toisinaan uudet ominaisuudet ovat hyvin tekstipitoisia, mutta niiden sisältöä ei ole määritelty, jolloin kehittäjät joutuvat kyselemään tekstejä käyttöönottoprojektista tai keksimään niitä itse. Koska sisältömuutokset ovat kehittäjien tehtävänä, menevät ne koko normaalin kehityssyklin läpi hyväksymistestauksineen. Tästä taas seuraa se, että yksinkertaisenkin kirjoitusvirheen korjaaminen tuotantoversioon asti vie 4 viikkoa.

Vastaavasti käyttöönottoprojektin puolella koetaan turhautumista pitkästä muutosprosessista, koska tekstien iteroiminen on hidasta ja hankalaa - kehittäjien jatkuva "vaivaaminen" tekstimuutoksilla koetaan epämiellyttäväksi. Tekstit eivät myöskään voi referoida käyttöliittymää liian yksityiskohtaisesti, koska kirjoittaja ei voi olla aivan varma, miltä käyttöliittymän yksityiskohdat näyttävät tekstin päätyessä tuotantoon asti. Hidas muutosprosessi estää myös ajantasaisen reagoinnin ongelmiin, joita esim. huonosti muotoiltu ohje voi saada aikaan.

Lähes kaikki HTJ-projektin haastateltavat kokivat jossain määrin, että kehittäjiä vaivataan liikaa asioissa, joiden ei pitäisi kuulua heille. Koettiin, että sovelluksen pitäisi olla sellainen, että tuotantokäyttäjät pystyvät käyttämään sitä itsenäisesti eräajojen ja sisällön muokkaamisen osalta.

6.5.3 Dokumentaatio

Kaikilla haastateltavilla ongelmana nousi esiin Confluenceen tallennetun tiedon vaikea löydettävyys ja tiedon hajanaisuus eri järjestelmissä. Tiedon löydettävyyttä haittaa jo itsessään tiedon suuri määrä, mutta myös Confluencen huonot hakutoiminnot vaikeuttavat löytämistä. Osasyynä on myös Confluencen sivujen puurakenne, joka voi muodostua ajan kanssa epäloogiseksi, jos Confluecen käyttöön ei ole selviä sääntöjä.

Erityisesti HTJ-projektissa sovelluksen dokumentointikäytäntöjä ei ole lyöty lukkoon, jonka vuoksi sovelluksen dokumentaatio on jäänyt monilta osin puutteelliseksi. Dokumentaatio on Confluencessa jokseenkin hajallaan, ja osa asioista on dokumentoitu pelkästään Jira-tiketeille.

KVP- ja KIRRE-projekteissa dokumentaation kattavuutta pidettiin hyvänä, joskin ominaisuudet on dokumentoitu hyvin teknisellä kielellä, eikä niiden sisältö välttämättä avaudu ulkopuoliselle.

6.5.4 Kantamuutokset

HTJ-projektin sovelluksissa on paljon kannassa sijaitsevaa, käyttöliittymässä suoraan näkyvää sisältöä kuten koodistoja ja käyttöehtoja. Tämän ansiosta kyseistä sisältöä voidaan muokata tekemättä sovelluspäivitystä. Muokkaukseen ei kuitenkaan ole työkalua, vaan se tehdään SQL-lauseita tuotantokantaan ajamalla, missä on suuri inhimillisen virheen riski.

Työkaluttomuus myös sitoo kehitystiimiä sovelluksen ylläpitoon, koska loppukäyttäjät eivät pysty mm. käynnistämään eräajoja koodistojen vuotuisia päivittämisiä varten, tai tekemään rutiininomaisia käyttöehtojen päivityksiä itsenäisesti.

6.6 Muut huomiot

Käyttöönottoprojektin puolella CMS-ohjelmistoon suhtauduttiin positiivisesti, ja haluttuina ominaisuuksina CMS:ssä nousivat esiin:

- Samanaikainen muokkaus
- Mahdollisuus muotoilla tekstiä
- Linkkien tyylittely
- Ohjelmiston toimintavarmuus
- Hyvä ja selkeä käyttöliittymä, missä perustoimintoja ei tarvitse etsiä
- Mikäli ohjelma antaa ilmoituksia, niistä pitää tulla sähköpostia

Toiveissa oli, että sisällöntuottoprosessi irrotettaisiin muusta kehittämisestä kokonaan, ja että tekstiä voisi muuttaa julkaisusyklistä riippumattomasti. Muutoksia ei testattaisi, vaan tekstintuottaja olisi vastuussa tekstistään.

6.7 Haastattelulöydösten yhteenveto

CMS toisi ilmeistä hyötyä HTJ-projektin ylläpito- ja kehitysprosesseihin. On mahdollista, että KVP- ja KIRRE-projekteissakin saavutettaisiin hyötyjä CMS-ohjelmiston käyttöönotolla, mutta siirtymisestä saatavat hyödyt eivät välttämättä kata kustannuksia.

Koska HTJ-projektissa kehitettävät web-sovellukset ovat luonteeltaan dynaamisia ja niillä on jo olemassa oleva front-end, on nimenomaan headless-CMS sopiva vaihtoehto projektin tarpeisiin. Koska HTJ-projektissa joudutaan tekemään paljon kantamuutoksia suoraan SQL-lauseilla, on projektissa tarvetta myös jonkinlaiselle admin-sovellukselle. Mikään MML:llä käytössä oleva kaupallinen sovellus ei vastaa näihin tarpeisiin, ja itsekehitetyt ratkaisut ovat liian räätälöityjä kopioitaviksi HTJ-projektiin.

Haastatteluissa nousivat esiin eri sidosryhmien keskenään ristiriidassa olevat tarpeet sisällönhallinnan suhteen. Testaajat ja hallinto pitivät tärkeänä, että sovelluksen sisältö ei muutu yllättäen kehitysprosessin ulkopuolelta, mutta sen sijaan sisällöntuottajat pitivät tärkeänä, että sisältöä pystyttäisiin muokkaamaan mahdollisimman nopeasti ja itsenäisesti. Tämä tarkoittaa, että CMS:n onnistunut käyttöönotto vaatii myös mahdollisia prosessimuutoksia jompaankumpaan suuntaan.

Eräs malli voisi olla se, että käyttöönottoprojektille annetaan autonomia sisällön suhteen - sovelluksesta määriteltäisiin ns. vapaasti muokattava sisältö, jota voisi muokata vapaasti CMS-ohjelmistolla. Jos sisältöä muokataan, pysyisi prosessi täysin käyttöönottoprojektin sisällä. Tekstin kirjoittaja vastaisi luomastaan tekstistä, ja nykyinen testausprosessi automaattitesteineen irrotettaisiin sisällön tarkastelusta. Kriittisempää sisältöä, kuten koodistoja, voitaisiin myös liittää CMS-ohjelmistoon, mutta muokkausoikeudet pitäisi rajata käyttäjänhallinnalla oikeille ihmisille, ja muokkaukseen pitäisi luoda oma prosessinsa.

Toisessa mallissa käyttöönottoprojekti voi vapaasti muokata sisältöä CMS-sovelluksella, mutta muutokset eivät tulisi voimaan välittömästi, vaan ne kulkisivat normaalin kehitysprosessin kautta. Tässä mallissa kehittäjä voisi esimerkiksi laittaa tekstiavaimen ilman sisältöä ohjelmistoon, ja laittaa sisällöntuottajalle herätteen asiasta. Prosessi olisi yhtä hidas kuin ennenkin, mutta kehittäjien ei tarvitsisi tehdä tekstimuutoksia ja testauksessa säilyisi varmuus sisällöstä.

7 CMS-ohjelmistojen vertailu

7.1 Tekninen vertailu

Koska haastatteluissa kävi ilmi, että mikään MML:llä jo käytössä oleva sovellus ei vastaa kirjaamissovellusten CMS-tarpeisiin, keskityttiin ohjelmistovertailussa puhtaasti kaupallisiin sovelluksiin. Mahdollisuutta käyttää Confluencea headless-CMS sovelluksena tutkittiin, mutta koska mitään konkreettista esimerkkiä confluencen taipumisesta tähän käyttöön ei löytynyt, pudotettiin se pois jo ennen vertailun aloittamista.

Haastatteluiden perusteella koostettiin lista ominaisuuksista, joita CMS-ohjelmistolla tulisi olla. Tekniseen vertailuun päästäkseen ohjelmiston täytyi täyttää seuraavat kriteerit:

- Headless CMS, tai toiminnallisuudeltaan vastaava
- Kypsä sovellus, sovellusversio vähintään 2.0
- Vähintään 3 yritysreferenssiä

Kuten luvussa 6.7 haastattelulöydösten yhteenveto todettiin, on headless CMS kirjaamissovellusten tarpeeseen sopiva CMS-ohjelmistotyyppi. Koska ohjelmistojen käyttöönotto on MML:llä viraston suuruuden ja byrokratian takia raskas prosessi, on ohjelmiston oltava korkealaatuinen. Käytännössä tämä tarkoittaa, että ohjelmiston tulee olla luotettavien tahojen säännöllisesti ylläpitämä, johon yritysreferenssit antavat osviittaa. Tämän lisäksi sovelluksen tulisi olla hiottu ja vakaa, jonka tavoittelemiseksi sovellusversiota rajattiin.

Tekniseen vertailuun otettiin näillä kriteereillä Directus, Squidex, Contentful, Ghost, Sanity, Butter CMS, Strapi ja Forest Admin. Vertailusta koostettiin excel-taulukko (liite 1), joka dokumentoitiin sellaisenaan MML:n Confluenceen.

Teknisen vertailun perusteella tarkempaan vertailuun otettiin Directus, Squidex, Contentful ja Sanity. Muut kandidaatit pudotettiin pois kunnollisen SSO-tuen puuttumisen vuoksi, tai Forest Adminin tapauksessa Oracle DB -yhteensopimattomuuden vuoksi.

7.2 Directus

Directus Suite on RANGER Studiosin kehittämä sovellus, joka on käytössä muun muassa AT&T:llä, Pradalla ja ACLU:lla. Directus Suite on eräänlainen admin-sovelluksen ja headless CMS:n yhdistelmä. Directus yhdistetään olemassa olevaan tietokantaan, jonka jälkeen tietoja pystytään muokkaamaan helposti web-pohjaisen käyttöliittymän kautta. Tämän lisäksi directus luo rajapinnat, joilla tietokannan tietoa voidaan lukea ja muokata, kuten headless CMS-ohjelmistossa. Tämä mahdollistaa ohjelman käyttämisen CMS:nä

projekteihin, joissa muokattava sisältö on jo tietokannassa, ilman että sisältöä tarvitsee siirtää CMS-ohjelmiston omaan dataformaattiin. Toisaalta ohjelmaa voi käyttää pelkkänä admin-sovelluksena projekteissa, joissa sisällönhallintaa ei tarvita. Directuksen omat tiedot sijaitsevat ohjelman omassa tietokannassa, jolloin varsinaiseen tietokantaan ei tarvitse tehdä muutoksia ohjelman käyttöönotossa, eikä sinne jää jälkiä, mikäli ohjelma poistetaan käytöstä. Directus ei myöskään estä perinteisen back-ending yhdistämistä kantaan samanaikaisesti, joten sitä voidaan käyttää rinnan nykyisten back-end -ratkaisujen kanssa. (RANGER studio 2020.)

Ohjelmisto osaa lukea tietokannan rakennetta, ja käyttöliittymässä on oletuksena näkyvillä kaikki tietokannan taulut. Tauluja on mahdollisuus piilottaa käyttöliittymästä käyttöoikeuksien mukaan - tai kokonaan, jos kyseessä on esimerkiksi aputaulu. Taulun sisällön esitystapaa voi muokata ohjelmistossa hyvin monipuolisesti, ja sisältö voidaan esittää taulukon lisäksi esimerkiksi lomakkeena tai galleriana. Tämä mahdollistaa käyttöliittymän muokkauksen tapauskohtaisesti sopivaksi sekä admin-sovelluksena käytettäväksi, että monipuolisempaa tekstinmuokkausta vaativaan sisällöntuotantokäyttöön. (RANGER studio 2020.)

Yhdellä asennuksella on mahdollista luoda useita projekteja, jotka voidaan yhdistää eri tietokantoihin (RANGER studio 2020). Näin ollen yhden henkilön on mahdollista helposti siirtyä projektista toiseen tai työskennellä useissa projekteissa.

7.2.1 Laajennettavuus

Directuksessa on mahdollista rakentaa lisäosina käyttöliittymäkomponentteja, moduuleja, kustomoituja rajapintoja ja ”hookkeja”. Käyttöliittymäkomponenteilla voidaan muokata monipuolisesti datan esitystapaa. Moduulit ovat isompia visuaalisia kokonaisuuksia, kuten esimerkiksi dashboard-näkymä, tai eräajon käynnistävää rajapintaa kutsuva nappi. Hookeilla taas voidaan luoda automaatiota, esimerkiksi lokitus ulkopuoliseen järjestelmään voidaan tehdä aina kun tiettyyn tauluun luodaan rivi. Hookit voi myös kytkeä ulkoisiin palveluihin webhookkien avulla, esimerkiksi Rocketchat-integraatiota varten. (RANGER studio 2020.)

7.2.2 Käyttäjänhallinta

Directuksessa on mahdollisuus luoda kustomoituja käyttäjärooleja, joiden avulla voidaan vapaasti kontrolloida pääsy- ja muokkausoikeuksia tietoihin. Ohjelmistossa voidaan myös luoda tietokantariveille "status"-tieto, jolloin esimerkiksi statuksella "tuotanto" olevaa tietoa eivät voi muokata muut kuin "tuotantovastaava"-roolilla olevat käyttäjät. Ohjelmistossa on myös public-rooli, jota voidaan käyttää, jos johonkin tietoon halutaan antaa pääsy autentikoimattomille käyttäjille. (RANGER studio 2020.)

7.2.3 Lokalisointi

Directuksessa on mahdollista hallita sisällön käännöksiä käännöstaulujen avulla, tai jo olemassaolevien "teksti_FI"/"teksti_SVE" -tyylisten tietokantakenttien avulla. (RANGER studio 2020.)

Directus itsessään on lokalisoitu useille kielille, joskin suomi ei vielä ole yksi niistä. Directusta käännetään Crowdin-palvelun kautta, joten tarvittaessa palvelu voidaan kääntää/käännättää suomeksi. (RANGER studio 2020.)

7.2.4 Hinta

Directuksesta on saatavilla ilmainen on-premise versio, ja maksullisia pilvilisenssejä (taulukko 1), joiden hinta määräytyy säilöttävän datan määrän mukaan. Maksullisiin lisensseihin kuuluu ilmaisena 5 käyttäjää / lisenssi. Ylimääräiset käyttäjät ovat 10 $/kk. (RANGER studio 2020.)

Tyyppi	Ominaisuudet	Hinta
Self-Hosted	Ei rajoituksia, täydet ominaisuudet, Community support	0 $ /kk
Basic-pilvilisenssi	10 datatyyppiä, 5 000 riviä, Standard support	29 $ /kk
Standard-pilvilisenssi	25 datatyyppiä, 10 000 riviä, Standard support	99 $ /kk
Professional	50 datatyyppiä, 25 000 riviä, Standard support	249 $ /kk
Enterprise	Ei rajoituksia, Premium support	Neuvoteltavissa

Taulukko 1. Directuksen hinnasto (RANGER studio 2020)

7.2.5 Vaaditut resurssit vastaan hyödyt

Vaaditut resurssit:

- Käyttöoikeuksien määritys, statuksien määritys
- Ohjelmiston asentaminen palvelimelle, mikäli halutaan self-hosted ratkaisu
- Lisenssimaksut, jos halutaan cloud-lisenssi
- Ohjelmiston kytkeminen MML:n SSO:iin
- Mikäli halutaan tekstinmuokkaustoimintoja täysiveriseen CMS-käyttöön, käyttöliittymän kustomointi vie aikaa ja vaatii osaamista
- Mikäli halutaan monimutkaista businesslogiikkaa, moduulien ja webhookkien kustomointi vie aikaa ja vaatii javascript/php -osaamista
- Ylläpito

Hyödyt:

- Tietokantaan päästään käsiksi käyttöliittymästä, jolloin kantamuutoksia ei tarvitse tehdä virhealttiisti suoraan SQL:llä
- Helppo ottaa käyttöön ja poistaa
- Yhdellä asennuksella voidaan hallita useita projekteja
- Voidaan käyttää rinnan nykyisten admin-sovellusten ja back-endien kanssa, jolloin Siirtyminen voidaan tehdä vaiheittain
- Granulaarisella käyttöoikeushallinnalla sisältö voidaan piilottaa niiltä, joiden ei kuulu sitä nähdä
- Sisällönhallinta voidaan antaa sisällöntuottajien käsiin, jolloin kehitystyötunteja säästyy
- Ylläpitotoimintoja varten ei tarvitse tehdä omia sovelluksia
- MML:llä ei välttämättä enää tarvitse tehdä uusia admin-sovelluksia

Puutteet:

- Ei tukea useammalle ympäristölle samassa projektissa, ympäristöjen pitää olla omia projektejaan
- Tekstinmuokkaustoiminnot eivät ole täysiverisen CMS:n tasolla
- Yhtäaikainen tekstinmuokkaus ei mahdollista

7.3 Squidex

Squidex on samannimisen yrityksen kehittämä vapaan lähdekoodin headless CMS-ohjelmisto, joka on käytössä muun muassa Matmatch:lla, Civicplussalla ja Fortress Reklamebyrålla. CMS-ohjelmistolle tyypillisesti sisällön rakenne on vapaasti kustomoitavissa, ja sitä voidaan tarjoilla ulospäin REST- ja GraphQL APIen kautta. Sekä ohjelman sisäiset tiedot että käyttäjän luoma sisältö sijaitsevat ohjelmiston omassa tietokannassa, joten sisällönhallinta on täysin irrotettu kehitettävästä sovelluksista. (Squidex 2020.)

Sisällön luonti tapahtuu käyttöliittymästä käsin - sisältötyypille annetaan nimi, kuten "infolaatikko" ja siihen lisätään halutut kentät, kuten "otsikko", "leipäteksti" ja "header-kuva". Sisältö voidaan esittää Squidexin käyttöliittymässä tapauksen mukaan eri tavoin, esimerkiksi muokattavana taulukkona, listana tai lomakkeena. Jokaiselle kenttätyypille on

valittavissa useita eri editoreja, joista voidaan valita kyseisen sisällön kannalta järkevin ratkaisu. (Squidex 2020.)

Squidexin tekstinmuokkausominaisuudet ovat monipuoliset, ja tekstiä voidaan muokata esimerkiksi raakatekstinä, markdown-editorilla tai rich text -editorilla. Ohjelmistossa on myös mahdollisuus rakentaa kokonaan kustomoituja editoreja HTML5:ä käyttäen. Työn organisoimiseksi sisällölle on mahdollista asettaa tila ("stage"), joka voi olla esimerkiksi "published", "odottaa käännöstä" tai "luonnos". (Squidex 2020.)

Ohjelmistossa on täysi muutosten lokitus ja sisällön versiointi, joka mahdollistaa muutosten helpon vertailun ja palauttamisen tarvittaessa (Squidex 2020). Yhdellä asennuksella on mahdollista hallita useita projekteja (Squidex 2020), jolloin MML:n henkilöstö voi helposti työskennellä useassa projektissa.

7.3.1 Laajennettavuus

Squidexissä ei ole mahdollisuutta varsinaisiin laajennuksiin, mutta ohjelmisto voidaan intergroida useiden ulkoisten palveluiden kanssa. Tämän lisäksi ohjelmistossa on mahdollisuus luoda "triggereillä" kustomoitua logiikkaa, esimerkiksi lähettää sähköposti-ilmoitus, kun sovellukseen lisätään käännöstoimistoon lähetettävää sisältöä. Triggerit voidaan myös yhdistää webhookkeihin kustomoidun toiminnallisuuden suorittamiseksi, esimerkiksi lokitukseen. (Squidex 2020.)

7.3.2 Käyttäjänhallinta

Ohjelmistossa on mahdollista luoda kustomoituja käyttäjärooleja, joilla voidaan hyvin granulaarisesti hallita pääsy- ja muokkausoikeuksia sisältöön. Ohjelmistossa voidaan asettaa sisällölle myös tila ("stage"), joka käyttäjärooleihin yhdistettynä mahdollistaa esimerkiksi sen, että vain tuotantovastaava voi siirtää sisältöä tuotantotilaan ja siten näkyville internettiin. (Squidex 2020.)

7.3.3 Lokalisointi

Squidexissä on tuki sisällön lokalisoinnille. Jokaiselle tietokentälle voidaan asettaa käännös halutulla määrällä kieliä, jonka lisäksi kentälle voidaan asettaa ns. ”fallback language”, eli kieli, jolla sisältö palautetaan, jos API-pyynnössä pyydettyä kieltä ei löydy. (Squidex 2020.)

Squidex itsessään on saatavilla vain englanniksi.

7.3.4 Hinta

Squidexista on saatavilla täysin pilvipohjainen SaaS-versio kahdelle, 10, 20 ja 50 käyttäjälle, sekä rajoittamaton-premise / self-hosted versio (taulukko 2). Squidexin pilvilisenssin hinta määräytyy käyttäjien lisäksi dataliikenteen mukaan, ei säilöttävien rivien mukaan – paremmalla lisenssillä saa tietysti enemmän muuta säilytystilaa esim. kuville.

Tyyppi	Ominaisuudet	Hinta
Self-hosted, Community	Täydet ominaisuudet	0 €/kk
Self-hosted, Enterprise	Täydet ominaisuudet, asennustuki, support SLA	Neuvoteltavissa
Pilvilisenssi, Starter	20 000 API kutsua /kk, 2GB dataliikennettä /kk, 500mb säilytystilaa, 2 käyttäjää	0 €/kk
Pilvilisenssi, Basic	100 000 API kutsua /kk, 20GB dataliikennettä /kk, 5 GB säilytystilaa, 10 käyttäjää	19 €/kk, 190 €/vuosi
Pilvilisenssi, Professional	500 000 API kutsua /kk, 50GB dataliikennettä /kk, 25 GB säilytystilaa, 20 käyttäjää	49 €/kk, 490 €/vuosi
Pilvilisenssi, Business	1500 000 API kutsua /kk, 100 GB dataliikennettä /kk, 100 GB säilytystilaa, 50 käyttäjää	99 €/kk, 990 €/vuosi

Taulukko 2. Squidexin hinnasto (Squidex 2020)

7.3.5 Vaaditut resurssit vastaan hyödyt

Vaaditut resurssit:

- Ohjelmiston asentaminen palvelimelle, jos halutaan self-hosted ratkaisu
- Koska MML:llä on käytössä Red Hat Linux, joudutaan ottamaan docker käyttöön
- Datamigraatio projektin tietokannasta ohjelmiston tietokantaan
- Sitoutuminen ohjelmistoon, kun sisältö on ohjelmiston tietokannassa
- Käyttöoikeuksien määritys, statuksien määritys
- Ohjelmiston kytkeminen MML:n SSO:iin
- Lisenssimaksut, jos halutaan cloud-lisenssi
- Mikäli halutaan monimutkaista businesslogiikkaa, triggerien kustomointi vie aikaa ja vaatii osaamista
- Ylläpito

Hyödyt:

- Sisällönhallinta voidaan antaa sisällöntuottajien käsiin, jolloin kehitystyötunteja säästyy
- Sisällön muokkaustapa on mukautettavissa, jolloin monipuolista sisältöä voidaan käsitellä luontevasti.
- Yhdellä asennuksella voidaan hallita useita projekteja

Puutteet:

- Ei toimi linuxilla ilman dockeria
- Ei mahdollisuutta käyttää olemassaolevaa tietokantaa
- Ei yhteensopiva Oracle-tietokannan kanssa
- Yhtäaikainen tekstinmuokkaus ei mahdollista
- Sovelluksen tietojen muokkaukseen tarvitaan yhä admin-sovellus

7.4 Contentful

Contentful on samannimisen yrityksen kehittämä suljetun lähdekoodin pilvipohjainen headless CMS-ohjelmisto, joka on käytössä muun muassa Nikellä, The British Museumilla ja Aldolla. Headless CMS -ohjelmistolle tyypillisesti sisällön rakenne on vapaasti kustomoitavissa, ja sitä voidaan tarjoilla ulospäin REST- ja GraphQL-rajapintojen kautta (Contentful 2020). Koska kyseessä on pilvipalvelu, kaikki data tallennetaan yrityksen palvelimille ja sisältödata on täysin irrotettu MML:n infrastruktuurista (Contentful 2020). Tämä vähentää ylläpitoon vaadittavia resursseja, mutta toisaalta datan saavutettavuus ja turvassapito tulee riippuvaiseksi ulkopuolisesta toimijasta.

Ohjelmistossa on mahdollisuus luoda useita projekteja, ja lajitella projekteja organisaatioiden alle. Tämä mahdollistaa sovelluksen käytön useissa projekteissa helposti, kuitenkaan tekemättä kokonaisuudesta sekavaa. Ohjelmistossa on myös mahdollista määritellä projektille erilliset tuontanto- ja kehitysympäristöt, jolloin esimerkiksi datamalleja ja sovellusintegraatiota on mahdollisuus testata turvallisesti ennen julkaisua. (Contentful 2020.)

Sisällön luonti tapahtuu käyttöliittymästä käsin - sisältötyypille annetaan nimi, kuten "infolaatikko" ja siihen lisätään halutut kentät, kuten "otsikko", "leipäteksti" ja "header-kuva". Tekstinmuokkausmahdollisuudet ovat monipuoliset, ja valittavina ovat mm. markdown- ja rich text-editorit. Muokkausnäkymää on mahdollista kustomoida haluamakseen, ja myös kokonaan omien komponenttien teko on mahdollista. Julkaistun sisällön lisäksi Contentfulissa on mahdollisuus luoda luonnoksia ja arkistoida sisältöä, jolloin sitä ei voi enää muokata. (Contentful 2020.)

Ohjelmistossa on automaattinen sisällön versiointi, joka mahdollistaa muutosten helpon vertailun ja palauttamisen tarvittaessa. (Contentful 2020.)

7.4.1 Laajennettavuus

Contentfuliin on mahdollista rakentaa lisäosia itse, tai asentaa niitä Contentful marketplacesta. Ohjelmisto tarjoaa rajapinnat myös ulkoisten integraatioiden tekoa varten, sekä webhook-tuen. (Contentful 2020.)

7.4.2 Käyttäjänhallinta

Ohjelmistossa on mahdollista luoda kustomoituja käyttäjärooleja, joilla voidaan hyvin granulaarisesti hallita pääsy- ja muokkausoikeuksia eri sisältötyyppeihin. Sovellukseen sisältyvät Enterprise-versiossa oletuksena roolit admin, editor, author, translator ja freelancer. (Contentful 2020.)

7.4.3 Lokalisointi

Contentfulissa on tuki sisällön lokalisoinnille. Jokaiselle tietokentälle voidaan asettaa käännös halutulla määrällä kieliä, jonka lisäksi voidaan määrittää ns. fallback language, eli kieli, jolla sisältö palautetaan, jos API-pyynnössä pyydettyä kieltä ei löydy. Ohjelmistossa on myös mahdollista julkaista sisältöä vain yhdellä kielellä kerrallaan, mikäli esimerkiksi suomenkielinen versio halutaan julkaista, ennen kuin ruotsinkielinen käännös on valmis. (Contentful 2020.)

Contentful itsessään on saatavilla vain englanniksi.

7.4.4 Hinta

Ohjelmistosta on saatavilla ilmaisversio viidelle käyttäjälle, Team-versio 25 käyttäjälle, ja Enterprise-versio rajoittamattomalle määrälle käyttäjiä (taulukko 3).

Tyyppi	Ominaisuudet	Hinta
Comminity	Community support, Max. 200 000 0 API-kutsua /kk, 48 datatyyppiä, 25 000 riviä, 5 käyttäjää	0 $/kk
Team	Standard support, 200 000 0 API-kutsua /kk + lisämaksu rajan ylimenevistä, 48 datatyyppiä, alkaen 25 000 riviä, 25 käyttäjää	Alkaen 498 $/kk
Enterprise	Enterprise support, kaikki rajat kustomoitavissa	Neuvoteltavissa

Taulukko 3. Contentfulin hinnasto (Contentful 2020)

7.4.5 Vaaditut resurssit vastaan hyödyt

Vaaditut resurssit:

- Datamigraatio projektin tietokannasta ohjelmiston tietokantaan
- Sitoutuminen ohjelmistoon, kun sisältö on ohjelmiston tietokannassa
- Sitoutuminen ulkopuoliseen toimijaan
- Käyttöoikeuksien määritys.
- Ohjelmiston kytkeminen MML:n SSO:iin
- Lisenssimaksut

Hyödyt:

- Sisällönhallinta voidaan antaa sisällöntuottajien käsiin, jolloin kehitystyötunteja säästyy
- Sisällön muokkaustapa on mukautettavissa, jolloin monipuolista sisältöä voidaan käsitellä luontevasti
- Yhdellä lisenssillä voidaan hallita useita projekteja
- Organisaatiot, projektit ja ympäristöt pitävät sisällönhallinnan kasassa
- Pienet ylläpitokustannukset

Puutteet:

- Ei on-premise-mahdollisuutta
- Ei mahdollisuutta käyttää olemassaolevaa tietokantaa
- Ei kustomoituja statuksia
- Yhtäaikainen tekstinmuokkaus ei mahdollista
- Sovelluksen tietokannan muokkaukseen tarvitaan yhä admin-sovellus

7.5 Sanity

Sanity on samannimisen yrityksen kehittämä osittain avoimen lähdekoodin headless CMS-ohjelmisto, joka on käytössä mm. Cloudflarella, Invisionilla ja Eurostarilla. Headless CMS -ohjelmistolle tyypillisesti sisällön rakenne on vapaasti kustomoitavissa, ja sitä voidaan tarjoilla ulospäin REST- ja GraphQL-rajapintojen kautta. Sanity koostuu teknisesti ottaen kahdesta ohjelmasta, Data storesta ja Sanity Studiosta - Data Store sisältää nimensä mukaisesti sisältödatan, joka sijaitsee sisältörajapinnan kanssa Sanityn palvelimilla, kun taas Sanity Studio on vapaan lähdekoodin CMA-sovellus. Sanity Studio voidaan haluttaessa asentaa on-premise. (Sanity 2020.)

Koska kyseessä on funktionaalisesti pilvipalvelu, kaikki data tallennetaan yrityksen palvelimille ja sisältödata on täysin irrotettu MML:n infrastruktuurista. Tämä vähentää ylläpitoon vaadittavia resursseja, mutta toisaalta datan saavutettavuus ja turvassa pito tulee riippuvaiseksi ulkopuolisesta toimijasta. Sanity tarjoaa valmiita työkaluja datan tuomiseksi Data Storeen. (Sanity 2020.)

Sisällön luonti tapahtuu käyttöliittymästä käsin - sisältötyypille annetaan nimi, kuten "infolaatikko" ja siihen lisätään halutut kentät, kuten "otsikko", "leipäteksti" ja "header-kuva". Sanity Studio on rakennettu erityisesti kustomointia silmällä pitäen, ja datan näyttö- ja muokkaustapa on pitkälle kustomoitavissa valmiiden ja itsetehtyjen komponenttien avulla - tai jopa tekemällä ohjelmasta oman version. Ohjelmiston ulkoasua on myös mahdollista muokata MML:n teemaan sopivaksi. (Sanity 2020.)

Sanity Studion tekstinmuokkausominaisuudet ovat monipuoliset, ja muista vaihtoehdoista poiketen ohjelmistossa on myös mahdollisuus saman dokumentin yhtäaikaiseen muokkaukseen. Sisällöntuotannon prosessia on tämän lisäksi mahdollista selkeyttää kustomoiduilla statuksilla ja toiminnoilla, joiden avulla voidaan näyttää esimerkiksi "ehdota muutoksia" -nappi sisällössä, joka on "odottaa katselmointia" -tilassa. (Sanity 2020.)

Sanity tallentaa täydellisen muutoshistorian määräajaksi - ilmaisversiossa muutokset tallennetaan 3 päivän ajalta, Enterprise-versiossa vähintään vuoden ajalta. Sisällön voi myös kätevästi palauttaa edelliseen versioonsa, mikäli tarvetta on. (Sanity 2020.)

7.5.1 Laajennettavuus

Sanity Studioon on mahdollista lisätä valmiita tai itsetehtyjä plugineita, joilla on mahdollista tehdä integraatioita ulkoisiin palveluihin tai suorittaa muuta logiikkaa. Data

store tukee myös webhookkeja, joilla voidaan suorittaa toimintoja automaattisesti datan muuttuessa. (Sanity 2020.)

7.5.2 Käyttäjänhallinta

Ohjelmistossa on Enterprise-versiossa mahdollista luoda kustomoituja käyttäjärooleja, joilla voidaan hyvin granulaarisesti hallita pääsy- ja muokkausoikeuksia eri sisältötyyppeihin. Sovellukseen sisältyvät Enterprise-versiossa oletuksena roolit ”admin”, ”read&write”, ”read”, ja ”public”. (Sanity 2020.)

7.5.3 Lokalisointi

Sanityssa on mahdollista hallita käännöksiä "localeString" -sisältötyypin avulla, joka on linkitetty varsinaiseen käännettävään sisältöön. (Sanity 2020.)

Sanity Studio itsessään on saatavissa vain englanniksi.

7.5.4 Hinta

Ohjelmistosta on saatavilla ilmaisversio kolmelle käyttäjälle, Advanced-versio 20 käyttäjälle, ja Enterprise-versio 50 käyttäjälle (taulukko 4). Lisäkäyttäjät maksavat 10 $/kk. Hinta määräytyy API-kutsujen ja dataliikenteen mukaan.

Tyyppi	Ominaisuudet	Hinta
Standard	3 käyttäjää, 100 000 API-kutsua /kk, 10GB dataliikennettä /kk, 5GB säilytystilaa	0 €/kk
Advanced	20 käyttäjää, 3M API-kutsua /kk, 1TB dataliikennettä /kk, 200GB säilytystilaa	199 $/kk
Enterprise	20 käyttäjää, 15M API-kutsua /kk, 3TB dataliikennettä /kk, 1TB säilytystilaa	Neuvoteltavissa

Taulukko 4. Sanityn hinnasto (Sanity 2020)

7.5.5 Vaaditut resurssit vastaan hyödyt

Vaaditut resurssit

- Datamigraatio projektin tietokannasta ohjelmiston tietokantaan
- Sitoutuminen ohjelmistoon, kun sisältö on ohjelmiston tietokannassa
- Sitoutuminen ulkopuoliseen toimijaan
- Käyttöoikeuksien määritys.
- Ohjelmiston kytkeminen MML:n SSO:iin
- Lisenssimaksut

Hyödyt

- Sisällönhallinta voidaan antaa sisällöntuottajien käsiin -> Kehitystyötunteja säästyy
- Sisällön muokkaustapa on mukautettavissa, jolloin monipuolista sisältöä voidaan käsitellä luontevasti.
- Granulaarisella käyttöoikeushallinnalla sisältö voidaan piilottaa niiltä, joiden ei kuulu sitä nähdä.
- Yhtäaikainen muokkaus tekee yhteystyöstä sujuvaa

Puutteet

- Dataa ei säilötä on-premise
- Ei mahdollisuutta käyttää olemassaolevaa tietokantaa
- Ei vielä virallista tukea useammalle projektille tai eri ympäristöille, mutta ne voidaan kytkeä "experimental feature":na päälle
- Sovelluksen tietokannan muokkaukseen tarvitaan yhä admin-sovellus

7.6 Vertailun tulos

Sopivin vaihtoehto on Directus Suite. Se toimittaa sekä headless CMS:n että admin-sovelluksen virkaa, vastaa MML:n teknisiä vaatimuksia, ja sopii HTJ-projektin käyttötarpeeseen. Tämän lisäksi ohjelman käyttöä on haluttaessa mahdollista laajentaa muihinkin projekteihin, jolloin voitaisiin jopa kokonaan luopua erinäisistä projektikohtaisista admin-sovelluksista, ja säästää rahaa niiden ylläpito- ja kehityskustannuksissa.

7.6.1 Perustelut

Mikäli HTJ-projektissa otettaisiin jokin headless CMS käyttöön, tulisi se todennäköisesti ensimmäisenä käyttöön ns. osakeluettelon siirtosovellukseen, joka on julkinen asiointipalvelu huoneistotietojärjestelmälle. Tällöin siirtosovellus pyörisi omalla palvelimellaan, ja ainakin osa sisällöstä tulisi siirtosovellukseen rajapinnan kautta toiselta palvelimelta. Koska kyseessä on julkinen palvelu, on tärkeää, että sivusto ja sen sisältö ovat saavutettavissa - esimerkiksi sisältöpalvelimen tekninen vika tai muu palveluntarjoajan oikku ei saa viedä sivulta käyttöehtoja. Tämän lisäksi palvelimeen voi potentiaalisesti kohdistua niin suuri pyyntömäärä, että kaupalliselta toimijalta ei välttämättä löydy tarpeeksi suurta lisenssiä.

Nämä seikat puoltavat sovellusta, joka voidaan asentaa MML:n (käytännössä Valtorin) palvelimille. Vaikka ratkaisu tuo ylläpitokustannuksia, voidaan sillä varmistaa, että data, tiedonsiirtokaista ja ongelmatilanteet ovat MML:n omissa käsissä.

Tämä on-premise vaatimus tiputtaa kilpailusta pois Contentfulin ja Sanityn. Vaikka Sanity on on-premise/cloud -hybridi, on nimenomaan datapääty vierailla palvelimilla, eikä on-premisen etuja saada.

Jäljelle jäävät Directus Suite ja Squidex. Näistä kahdesta Squidexissa on miellyttävämpi käyttöliittymä, ja kehittyneemmät lokalisointiominaisuudet. Se kuitenkin vaatii pyöriäkseen Windows Serverin, jollaista MML:llä ei ole käytössä. Ongelman voi kiertää kontteja käyttämällä, mutta kontit eivät taas ole yhteensopivia tietoturvatiimin monitorointiohjelmistojen kanssa. Luonnollisesti rautaa on mahdollista päivittää ja tietoturvakäytäntöja uudistaa, mutta Directuksessa näitä ongelmia ei ole alkuunkaan, joten se on Squidexia helpompi ratkaisu.

Directuksessa on huonojen puolien puuttumisen lisäksi hyviä ominaisuuksia, jotka tekevät siitä erityisen hyvän juuri HTJ-projektiin. HTJ:ssä sisältödata sijaitsee jo tietokannassa - Directus voidaan asentaa olemassaolevan tietokannan päälle, jolloin sisältödataa ei välttämättä tarvitse migratoida. Tämän lisäksi Directusta voidaan käyttää yleisesti admin-sovelluksena kannan katseluun ja muokkaamiseen, ja Directuksen laajennusmahdollisuuden mahdollistavat myös mm. eräajojen käynnistämisen, sekä integraatiot Jiraan ja Confluenceen – molemmat ovat ominaisuuksia, joista on hyötyä HTJ-projektissa.

Sekä käyttöönottoprojekti että kehitystiimi pitävät Directusta hyvänä sovelluksena esittelyn perusteella.

7.6.2 Mitä ongelmia tällä ratkaistaisiin?

Tarpeiden kartoituksessa paikallistettiin kaksi pääongelmaa HTJ-projektissa - sisällönhallinta puuttuu sovelluskehityksestä, ja kehittäjät on sidottu tekemään ylläpitomaisia tehtäviä, jotka voisivat yhtä hyvin kuulua tuotannolle tai käyttöönottoprojektille.

Koska sisällönhallinnan prosessia ei ole määritelty, on projektiin kehittynyt kaksi rinnakkaista prosessia: yhdessä prosessissa kehittäjät kehittävät Jira-tiketin pohjalta ominaisuuden ja kyselevät sen jälkeen sähköpostilla sisällöntuottajilta sille sisältöä, ja

toisessa prosessissa sisällöntuottajat kehittävät sisällön ensin Confluenceen, ja kehitystiimi tekee niiden pohjalta itselleen Jira-tiketit. Molemmissa prosesseissa kehittämisen *ydinongelma* on sama: itse sisällön päivitys tapahtuu manuaalisesti kopioimalla tietoa sähköpostista, Jirasta tai Confluencesta tietokantaan. Näin sisältömuutosten historia jää katkonaiseksi, tai sähköpostin tapauksessa jopa puuttumaan, ja uuden sisällön vertailu vanhan kanssa on vaikeaa. Confluence ja Jira eivät myöskään tue rakenteellista sisältöä, jolloin kehittäjä joutuu itse lisäämään esimerkiksi HTML-tagit käyttöehtoihin, ennen kuin ne voidaan tallentaa tietokantaan. Tiedon siirtäminen järjestelmästä toiseen lisää aina virheiden riskiä, varsinkin jos sitä joudutaan muokkaamaan matkalla. Koska tiedon kopioimisen joutuu tekemään nimenomaan kehittäjä, menevät pienetkin muutokset koko kehityssyklin läpi, ja pienikin korjaus voi kestää jopa kuukauden. Sisällöntuottajat myös kokevat "vaivaavansa" kehittäjiä, koska eivät voi tehdä muutoksia itse. Uuden ohjelman käyttöönotto ei luonnollisesti poista prosessin ongelmia, mutta pelkkä prosessin muuttaminen ei poista työkalujen puutteesta johtuvaa ydinongelmaa.

Directus Suiten käyttäminen Headless CMS:nä korjaisi ongelmaa antamalla työkalun, jolla sisällön muutokset voidaan viedä sovellukseen automaattisesti. Koska sisältö syötettäisiin vain yhteen paikkaan, eli Directukseen, jäisi siitä katkeamaton muutoshistoria ja kaikki sisältö löytyisi samasta paikasta. Directuksen ominaisuuksiin kuuluu myös mahdollisuus palauttaa tehdyt muutokset, jolloin mahdolliset virheet voidaan korjata nopeasti. Sisältö olisi rakenteistettua, ja esimerkiksi HTML-tagit voitaisiin asettaa paikalleen automaattisesti. Kun tämä ydinongelma olisi korjattu, voitaisiin prosessia muokata oman maun mukaan mihin tahansa suuntaan. Riippuen siitä, miten paljon prosessissa sisällöntuottajille halutaan antaa autonomiaa, voidaan sisällönhallinta ulkoistaa lähes kokonaan sovelluskehityksen ulkopuolelle. Tällöin sisällöntuottajat voisivat itsenäisesti viedä muutokset suoraan tuotantoon asti, tunneissa. Ainoa asia, johon kehittäjiä ehdottomasti tarvitaan, on integraation rakentaminen täysin uuden sisällön kohdalla.

Sisällönhallintaominaisuuksien lisäksi Directuksessa tulee mukana admin-sovellusominaisuuksia, jolla voidaan samalla vaivalla korjata myös muita kehityksen ongelmia. Tällä hetkellä mm. eräajojen käynnistäminen on kehitystiimillä, kun taas esimerkiksi KVP-projektissa se voidaan tehdä KVP-administa asiakasvastaavien toimesta. Tämän lisäksi kantakorjaukset joudutaan tekemään suoraan kantaan SQL-lauseilla, jotka lokitetaan käsin Confluenceen - työtapa, joka on erittäin altis inhimilliselle virheelle. Mikäli Directus tulisi käyttöön, voitaisiin eräajojen käynnistämistä varten rakentaa lisäosa, aineistokorjailut voitaisiin tehdä graafisesta käyttöliittymästä käsin automaattisen

lokituksen kera, ja virheet voitaisiin kumota napin painalluksella. Samalla säästettäisiin rahaa, kun HTJ-projektille ei tarvitsi kehittää omaa admin-sovellusta.

7.6.3 Mitä ongelmia tällä ei ratkaista?

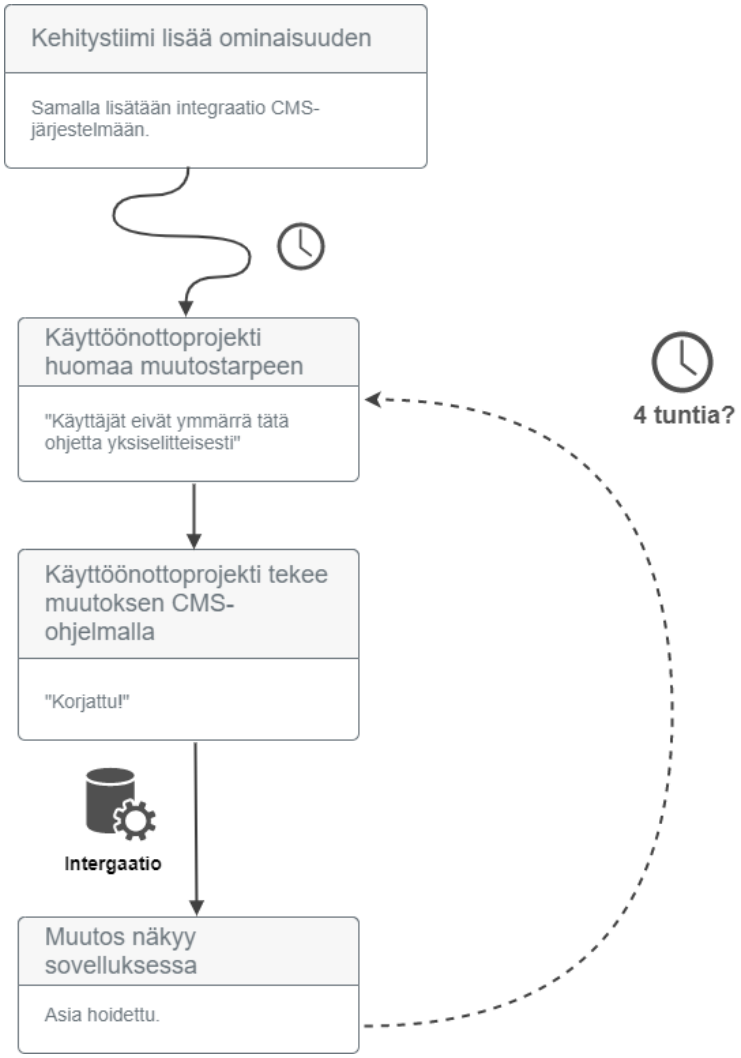
Ohjelmiston käyttöönotto ei itsessään korjaa prosessia, tai poista huonon prosessin tuomia ongelmia. Kuitenkin mikäli ohjelmisto otetaan HTJ-projektissa käyttöön, voitaisiin toimiva prosessi matkia esimerkiksi KVP-projektista, jossa KVP-admin toimittaa headless CMS:n virkaa. Directuksessa on hyvin monipuolinen käyttäjähallinta ja monia työnteon hallinnointia helpottavia ominaisuuksia, jolloin sovellus ei rajoita sitä, mihin suuntaan prosessia voi viedä.

Directuksen käyttöönotto ei myöskään ratkaise haastatteluissa havaittuja ongelmia Confluencen käytön kanssa tai dokumentoinnissa - nämä on ratkaistava käytäntöjä muuttamalla. Directuksen käyttöönotto ei myöskään itsessään suoranaisesti ehkäise tiedon siroamista eri järjestelmiin, mutta sopivilla Confluence ja Jira-integroinneilla tilanne pysyy hallinnassa.

8 Tulokset ja niiden hyödyntäminen

8.1 Kehitysehdotuksia

CMS-ohjelmiston käyttöönotto avaa monia mahdollisuuksia sisällönhallinnan prosessin kehittämiselle kirjaamissovelluksissa. Mikäli sovellus otettaisiin MML:ssä käyttöön, voisi sisällönhallinnan haluttaessa ulkoistaa täysin käyttöönottoprojektille - tällaisessa mallissa kehittäjiä tarvittaisiin vain sisällönhallinnan integraation rakentamiseen uuden ominaisuuden kohdalla (kuvio 4). Koska sisältömuutos menisi tuotantoon suoraan integraation välityksellä, sisällön muokkaukseen kuluva aika olisi viikkojen sijaan tunteja.



Kuvio 4. Mahdollinen malli sisällönhallinnan prosessille CMS-ohjelmiston käyttöönoton jälkeen

Mikäli prosessi ulkoistettaisiin kehitystiimin ulkopuolelle, pitäisi nykyisiä testauskäytäntöjä muuttaa. Tällä hetkellä HTJ-projektissa on runsas määrä automaattitestejä, jotka muun laadunvarmistuksen ohella tarkastavat, että sovelluksessa lukee oikeita asioita – mikäli

teksti muuttuu, antaa testi hälytyksen (ammattislangilla ”menee rikki”). Kehityksessä tämä ei ole ongelma, koska muutos on hyvin etukäteen tiedossa - mutta mikäli sisällönmuutoksen pystyisi tekemään käyttöönottoprojektista suoraan kehitysprosessin ohi, olisivat automaattitestit todennäköisesti rikki *jatkuvasti* ennakoimattomien muutosten takia. Ongelma voidaan kiertää yksinkertaisimmin poistamalla sisällön testaus kokonaan, jolloin sisällöntuottaja vastaa täysin tekstinsä oikeellisuudesta.

Käyttöönottoprojektin ja kehitysprojektin välille voidaan myös luoda yhteinen sisällöntuotantoprosessi, jonka hallinnoimiseksi Directus Suitesta löytyy monipuoliset työkalut. Esimerkki: sisällöntuottajat voisivat muokata sisältöä CMS-ohjelmiston kautta, ja muokkaus loisi kehitystiimille automaattisesti tiketin, jonka perusteella muutos testattaisiin. Muutoksen vieminen tuotantoon olisi yhtä hidasta, mutta prosessi olisi hallitumpi ja sisällöntuottajien ei tarvitsisi ”vaivata” kehittäjiä.

8.2 Opinnäytetyön onnistuminen

Opinnäytetyö sujui aikataulussa, ja valmistui itseasiassa hieman odotettua aikaisemmin. Pidän opinnäytetyötä henkilökohtaisesti hyvin onnistuneena: tarpeet saatiin kartoitettua, ja sopiva CMS-ratkaisu löydettyä. Haastavinta opinnäytetyössä oli nimenomaan tarpeiden kartoitus, koska suurin osa haastateltavista ei tiennyt mikä CMS-ohjelmisto on, saati osannut kokea mitään tarvetta sellaiselle. Kysymysten suunnittelussa yritin ottaa tämä huomioon muotoilemalla kysymykset mahdollisimman avoimiksi. Haastatteluissa yritin myös hyödyntää myös edellisten haastateltavien kanssa esiin nousseita pointteja – ”Miten teillä muutetaan sisältöä? Edellisessä haastattelussa ilmeni ongelmaa tämän kanssa, oletteko kokenut samaa…?”

Selvitystyössä toteutettiin ensin haastattelut, sitten aikaisempiin selvityksiin perehtyminen – syynä oli tarve saada haastattelut valmiiksi ennen kesälomakautta. Jos tekisin opinnäytetyön uudestaan, tekisin ne eri järjestyksessä, koska selvityksissä ilmeni paljon samoja asioita kuin haastatteluissa. Olisin voinut kysyä tietyistä asioista haastateltavilta lisää, tai jättää joitain selvityksissä läpikäytyjä asioita kysymättä.

Koska toiminnallisen opinnäytetyön tarkoitus on tuottaa jotain hyödyllistä toimeksiantajalle, on ”asiakastyytyväisyys” tärkeää. Tässä tapauksessa MML oli hyvin tyytyväinen tulokseen, ja organisaatiossa tullaan käsittelemään, miten opinnäytetyössä esitetty ratkaisu voitaisiin ottaa käyttöön. Kynnyskysymyksenä on, miten Directus Suiten ylläpito järjestetään.

Lähteet

AlternativeTo 2020. AlternativeTo – Crowdsourced software recommendations. Luettavissa: https://alternativeto.net/. Luettu: 15.9.2020.

Arkistolaki 23.9.1994/831

Burgy, P. 2020. A brief history of the content management system. Luettavissa: https://opensource.com/article/20/7/history-content-management-system. Luettu: 23.8.2020.

Cequens 2018. Cloud vs. on-premise solutions: the pros and cons. Luettavissa: https://www.cequens.com/story-hub/cloud-vs.-on-premise-solutions-the-pros-and-cons. Luettu: 10.9.2020.

Ciprlani, J & Rosenblatt S. 2015. Two-factor authentication: What you need to know. Luettavissa: https://www.cnet.com/news/two-factor-authentication-what-you-need-to-know-faq/. Luettu: 10.9.2020.

Contentful 2020. Contentful data model. Luettavissa: https://www.contentful.com/developers/docs/concepts/data-model/. Luettu: 28.8.2020.

Contentful 2020. Contentful features. Luettavissa: https://www.contentful.com/features/. Luettu: 1.9.2020.

Docker 2020. What is a container? Luettavissa: https://www.docker.com/resources/what-container. Luettu: 10.9.2020.

Heslop, B. 2018. History of content management systems and rise of headless CMS. Luettavissa: https://www.contentstack.com/blog/all-about-headless/content-management-systems-history-and-headless-cms. Luettu: 22.8.2020.

IBM 2010. What is batch processing? Luettavissa: https://www.ibm.com/support/knowledgecenter/zosbasics/com.ibm.zos.zconcepts/zconc_whatisbatch.htm. Luettu: 15.9.2020.

Jyväskylän yliopisto 2015a. Laadullinen tutkimus. Luettavissa: https://koppa.jyu.fi/avoimet/hum/menetelmapolkuja/menetelmapolku/tutkimusstrategiat/laadullinen-tutkimus. Luettu: 15.9.2020.

Jyväskylän yliopisto 2015b. Vertaileva tutkimus. Luettavissa: https://koppa.jyu.fi/avoimet/hum/menetelmapolkuja/menetelmapolku/tutkimusstrategiat/vertaileva-tutkimus. Luettu: 15.9.2020.

Jyväskylän yliopisto 2015c. Tapaustutkimus. Luettavissa: https://koppa.jyu.fi/avoimet/hum/menetelmapolkuja/menetelmapolku/tutkimusstrategiat/tapaustutkimus. Luettu: 15.9.2020.

Kentico Software 2018. State of headless CMS market in 2018. Kentico Software. Luettavissa: https://assets.kenticocloud.com/4e9bdd7a-2db8-4c33-a13a-0c368ec2f108/1812d59a-814e-4d56-88b4-8a272e20f845/headless-cms-report.pdf. Luettu: 28.8.2020.

Maanmittauslaitos 2019. Esittelykalvot 2019. Intranet. Esittelykalvot. Luettu: 5.5.2020.
Maanmittauslaitos 2020. Tietoa Maanmittauslaitoksesta. Luettavissa: https://www.maanmittauslaitos.fi/organisaatio. Luettu: 1.9.2020.

Merriam-Webster 2020. On-premises. Luettavissa: https://www.merriam-webster.com/dictionary/on%20premises. Luettu: 10.9.2020.

RANGER studio 2020. Directus documentation & guides. Luettavissa: https://directus.io/resources.html. Luettu: 1.9.2020.

Red Hat 2020. REST vs SOAP. Luettavissa: https://www.redhat.com/en/topics/integration/whats-the-difference-between-soap-rest. Luettu: 10.9.2020.

Rouse, M. 2020. Single sign-on. Luettavissa: https://searchsecurity.techtarget.com/definition/single-sign-on. Luettu: 10.9.2020.

Sanity 2020. Sanity docs. Luettavissa: https://www.sanity.io/docs. Luettu: 1.9.2020.

Sengrid 2017. What’s a webhook? Luettavissa: https://sendgrid.com/blog/whats-webhook/. Luettu: 10.9.2020.

Sininen meteoriitti 2020. Ketteryys haltuun: Scrum pähkinänkuoressa. Luettavissa: https://meteoriitti.com/2013/06/06/ketteryys-haltuun-scrum-pahkinankuoressa/. Luettu: 15.9.2020.

Squidex 2020. Squidex features. Luettavissa: https://squidex.io/features. Luettu: 1.9.2020.

Sturgeon, P. 2017. GraphQL vs REST: overview. Luettavissa: https://phil.tech/2017/graphql-vs-rest-overview/. Luettu: 10.9.2020.

Tieteen termipankki 2020. Perusrekisteri. Luettavissa: https://tieteentermipankki.fi/wiki/Avoin_tiede:perusrekisteri. Luettu: 31.8.2020.

Pastorino, M. s.a. Frontend vs. backend: What's the difference? Luettavissa: https://www.pluralsight.com/blog/software-development/front-end-vs-back-end. Luettu: 15.9.2020.

Vaxter, S & Vogt L. 2002. Content management system, US6356903B1. United States Patent and Trademark Office. Luettavissa: https://patents.google.com/patent/US6356903B1/en Luettu: 21.8.2020.

W3Techs 2020. Market share trends for content management systems. Luettavissa: https://w3techs.com/technologies/history_overview/content_management. Luettu: 27.8.2020.

Wikipedia 2020a. Comparison of open-source and closed-source software. Luettavissa: https://en.wikipedia.org/wiki/Comparison_of_open-source_and_closed-source_software. Luettu: 10.9.2020.

Wikipedia 2020b. Service-level agreement. Luettavissa: https://en.wikipedia.org/wiki/Service-level_agreement. Luettu: 10.9.2020.

Wikipedia 2020c. WYSIWYG. Luettavissa: https://en.wikipedia.org/wiki/WYSIWYG. Luettu: 10.9.2020.

Wordpress 2020. Wordpress codex: Database description. Luettavissa: https://codex.wordpress.org/Database_Description. Luettu: 23.8.2020.

Liitteet

Liite 1. CMS-ohjelmistojen tekninen vertailu

	Directus	Strapi	Ghost	Contentful	Sanity	Forest admin	Butter CMS	Squidex
Vapaa lähdekoodi	x	x	x	ei	osittain	osittain	ei	x
Ilmaisversio saatavilla	Self-hosted	Self-hosted	Self-hosted	5 käyttäjää	3 käyttäjää	5 käyttäjää	ei	x
Enterprise-paketti & SLA	x	premium supportin kanssa	x	x	x	x	x	Enterprise-paketti, ei SLA
2FA	x	pluginina	x	x	x	x	x	x
SSO	self-hosted & enterprise edition	"tulossa", vain enterprise edition	plugineilla / custom fork	vain enterprise edition	vain enterprise edition	vain google sso	ei	x
SSO tyyppi	OAUTH 1.0 & 2.0, SCIM	-	-	kustomoitavissa	kustomoitavissa	google SSO	-	OIDC
Self-hosted / On-premise	x	x	x	ei	on-premise + cloud -hybridi	on-premise + cloud -hybridi	ei	x
Mahdollisuus käyttää olemassaolevaa tietokantaa	x	ei	ei	ei	ei	x	ei	ei
Oracle db -yhteensopiva	x	ei	x	-	-	ei	-	ei
Docker-yhteensopiva	x	x	x	-	-	x	-	**toimii linuxilla vain dockerissa**
CLI	x	x	x	x	x	x	x	x
Useampi projekti yhdellä asennuksella	x	"tulossa"	x	x	ei	ei	x	x
Käyttäjäroolit	Admin, public, itsemääritellyt	vain enterprise edition	owner, admin, editor, author, contributor	admin & editor ilmaisversiossa, enterprise-versiossa itsemääritellyt	admin, read&write, read, public, enterprise-versiossa itsemääritellyt	vain premium & enterprise	admin, publisher, author	itsemääritellyt
Eri ympäristöt (production, tuotanto...)	toteutettavissa projekteilla ja statuksilla	ei	ei	x	toteutettavissa statuksilla	x	x	toteutettavissa workflow'lla
Workflow-apuvälineet	itseluotavat statukset, esim "waiting for review", "tuotanto"	statukset: draft, published, archive	drafts	statukset: draft, published, archived	itseluotavat statukset, esim "waiting for review", "tuotanto"	"approval workflow"	kustomoitava workflow	kustomoitava workflow
Kustomoitu businesslogiikka	webhookeilla ja laajennuksilla	webhookit	integraatiot muihin palveluihin	webhookit	webhookeilla ja laajennuksilla	"smart actions"	webookit	"triggers"
Muutoshistoria	x	x	ei	x	3 pv ilmaisversiossa, 365+ pv enterprise-versiossa	x	x	x
Muutosten palautus	x	???	ei	x	x	???	x	x
Yhtäaikainen muokkaus	ei	ei	google docs -integraatiolla	ei	x	ei	ei	ei
WYSIWYG-editori	laajennuksella	"tulossa"	x	x	x	x	x	x
REST-API	x	x	x	x	x	ei	x	x
GraphQL-API	x	x	ei	x	x	ei	ei	x
Asset-managerointi	x	x	x	x	x	x	x	x
Datamallin kustomoitavuus	Erinomainen	Hyvä	Kohtalainen	Hyvä	Hyvä	Erinomainen	Hyvä	Hyvä
Lokalisointituki	x	"tulossa"	x	x	x	ei	x	x