Dokumentoinnin ja henkilökohtaisen tietämyksenhallinnan kehittäminen konsulttityössä

Frans Reinikka
Haaga-Helia ammattikorkeakoulu
Tradenomin tutkinto
Amk-opinnäytetyö
2024

Tiivistelmä

Tekijä(t)
Frans Reinikka

Tutkinto
Tradenomi

Raportin/Opinnäytetyön nimi
Dokumentoinnin ja henkilökohtaisen tietämyksenhallinnan kehittäminen konsulttityössä.

Sivu- ja liitesivumäärä
41 + 0

Päiväkirja-tyyppinen opinnäytetyö käsittelee IT-alan konsultin työtä jatkuvan kehitys- ja ylläpitotyötä vaativan projektin kehityksessä. Opinnäytetyössä seurataan tekijän ammatillista kehitystä seuranta-aikana, erityisesti dokumentaation laatimisen, henkilökohtaisen tietämyksenhallinnan osalta ja laajojen kielimallien hyödyntämisestä, mitkä kuvataan peittomatriisitaulukossa. Seuranta ja kehityksen analyysi suoritettiin 26.02 – 19.4.2024.

Opinnäytetyössä kuvataan tekijän työnantaja, asiakas, sidosryhmät ja konteksti, missä työ suoritetaan. Tekijän kompetenssi ja motivaatiot kehitykseen esitellään seuranta-ajan alussa. Lisäksi esitellään tekijälle mahdolliset vuorovaikutustilanteet, tietoperustaa opinnäytetyölle, sekä keskeiset käsitteet.

Tekijä kirjaa päiväkirjamerkintöjä työstään kahdeksan viikon ajan, mitkä liittyvät valittuihin ammatillisiin kehityskohteisiin. Viikoittain tekijä analysoi toimintaansa ja käsitteitä mitä on opittu tai otettu käyttöön projektissa seurantaviikon aikana. Analysoinnissa hyödynnetään lähdekirjallisuutta, tukevia sidosryhmiä ja projektissa saatua kokemusta.

Seuranta-ajan päätyttyä tekijä tiivistää oppimansa ja analysoi ammatillista kehitystään itsereflektion avulla. Lopuksi tekijä arvioi kehityksensä ja mahdolliset tulevat kehityksen kohteet.

Asiasanat
Dokumentaatio, Henkilökohtainen tietämyksenhallinta, laajat kielimallit

Sisällys

1 Johdanto

Tämä opinnäytetyö toteutetaan päiväkirjamuotoisena keväällä 2024, 26.2. ja 19.4. välisenä aikana. Opinnäytetyöprojektin tekijä toimii IT-alan konsulttiyrityksessä, Solita Oy:ssä, IT-konsultin toimessa. Solita palvelee lukuisia asiakkaita yksityisellä ja julkisella sektorilla Pohjoismaissa ja Pohjois-Euroopassa.

Asiakkaan projektissa tekijällä on yli vuoden kokemus IT-konsulttina. Projektin päivittäinen työ organisoidaan itseohjautuvasti ja korkeamman tason päämäärät kehittyvät tuoteomistajan ja sidosryhmien kanssa vuorovaikutuksessa, priorisoiden yhteisiä tavoitteita ja sidosryhmien tuottamien toiminnallisuuksien fasilitointia. Päivittäisten työtehtävien suorittaminen vaatii osaamista seuraavilta osa-alueilta:

- Vuorovaikutustaidot ja kommunikaatio
- Ohjelmointitaidot
- Työn organisointi ja koordinointi
- Jatkuva oppiminen ja tiedonhankinta
- Riskienhallinta
- Käyttöliittymäsuunnittelu
- Tietokantaosaaminen
- Pilvipalvelut

Opinnäytetyö käsittelee pääasiassa tekijän ammatillista kompetenssia ja kehittymistä. Opinnäytetyön ammatillisen osaamisen kehityskohteiksi on valittu seuraavat teemat:

- Ohjelmiston dokumentaation tuottaminen, jalostaminen ja katselmointi
- Henkilökohtaisen tietämyksenhallinnan prosessien kehittäminen
- Laajojen kielimallien hyödynnettävyys dokumentaation tuottamisessa

Dokumentaatio on opinnäytetyön keskeinen teema, johon opinnäytetyössä keskitytään eniten. Dokumentaatiolla tarkoitetaan kuvausta ohjelmistosta, sen toiminnoista, ominaisuuksista, käyttötarkoituksesta ja käyttötavoista. Opinnäytetyössä pyritään huomioimaan dokumentaation roolia kommunikoinnissa eri sidosryhmien kanssa ja dokumentaation puutteellisuuden tuomia haittoja. Henkilökohtaisella tietämyksenhallinnalla tarkoitetaan tietotyöläisen henkilökohtaista tiedonhallintaa ja tiedon jalostamista, jota hyödynnetään tietotyöläisen ammatilliseen kehitykseen, tehokkuuteen ja päätöksentekoon. Henkilökohtainen tietämyksenhallinta on avustavana teemana yleiselle kehitykselle tietoalan työtekijänä. Laajat kielimallit ovat tekoälyjärjestelmätyyppi, joka on suunniteltu vastaamaan annettuihin syötteisiin ihmismäisellä tavalla. Laajojen tietomallien ymmärtäminen ja hyödyntäminen on osa tietotyön kehitystä ja soveltuu siksi opinnäytetyön teemaksi.

Taulukko 1. Peittomatriisi, jossa kuvataan tietoperustan ja ammatillisen kehityksen yhteys

Ammatillisen kehityksen tavoitteet	Kehityksen käsittely raportissa	Seurantaviikko	Ammatillisen kehityksen tulos
Miten voidaan kehittää dokumentaation tuottamista ja kehittämistä projektin kontekstissa?	3.1, 3.2, 3.3, 3.4, 3.5, 3.6, 3.7, 3.8	1, 2, 3, 4, 5, 6, 7, 8	4.0
Mitä henkilökohtaisen tietämyksenhallinnan tekniikoita voidaan soveltaa tukemaan tietotyötä ja tiedon tuottamista?	3.1, 3,2, 3.4, 3.5	1, 2, 4, 5	4.0
Miten laajoja kielimalleja voidaan hyödyntää dokumentaation tukena?	3.2, 3.4	2,4	4.0

Opinnäytetyön ulkopuolelle rajataan organisaatiotason tietämyksenhallinnan käsittely, tämä on aiheena dokumentaatiota sivuava teema mutta liian laaja käsiteltäväksi dokumentaation ohessa. Laajojen kielimallien toimintaperiaatetta ei käsitellä opinnäytetyössä. Opinnäytetyössä laajojen kielimallien rooli rajautuu niiden hyödynnettävyyteen dokumentaation tuottamisessa ja parantamisessa. Dokumentaation ja dokumentointitaitojen kehittäminen tapahtuu Solita Oy:n työyhteisön kontekstissa ja siten rajautuu palveltavan asiakkaan ja projektityöyhteisön tarpeiden ja rajoitteiden mukaan.

2 Lähtötilanteen kuvaus

Opinnäytetyön tekijällä on yli viisi vuotta kokemusta IT-alalla ohjelmistokehittäjänä ja konsulttina. Kokemuksen lisäksi tekijän tieto perustuu opintoihin Haaga-Helia ammattikorkeakoulussa. Projektikohtaista kokemusta on yli vuosi. Projektin dokumentaatiossa on havaittavissa puutteita ja niiden parantaminen mahdollistaisi joustavampaa toimintaa projektityössä.

2.1 Lähtötilanteessa tekijän taitojen arviointi

Lähtötilanne ammatillisen kehityksen seurannan alkaessa on vaihteleva. Tekijä on toteuttanut dokumentaatiota aikaisemmin, mutta se on ollut toissijainen työtehtävä. Seurantajakson ajan tavoitteena on nostaa dokumentaation laatimisen nykyistä tasoa ja arviointikykyä dokumentaation katselmoinnissa, siten että dokumentaatio kommunikoi kattavasti ohjelmiston ominaisuuksista. Tällöin asiakasta ja sidosryhmiä voidaan palvella aiempaa korkeammalla tasolla. Tärkeätä on myös osata arvioida milloin asioita ei kannata dokumentoida, sillä kulujen optimointi on osa konsultin toimenkuvaa.

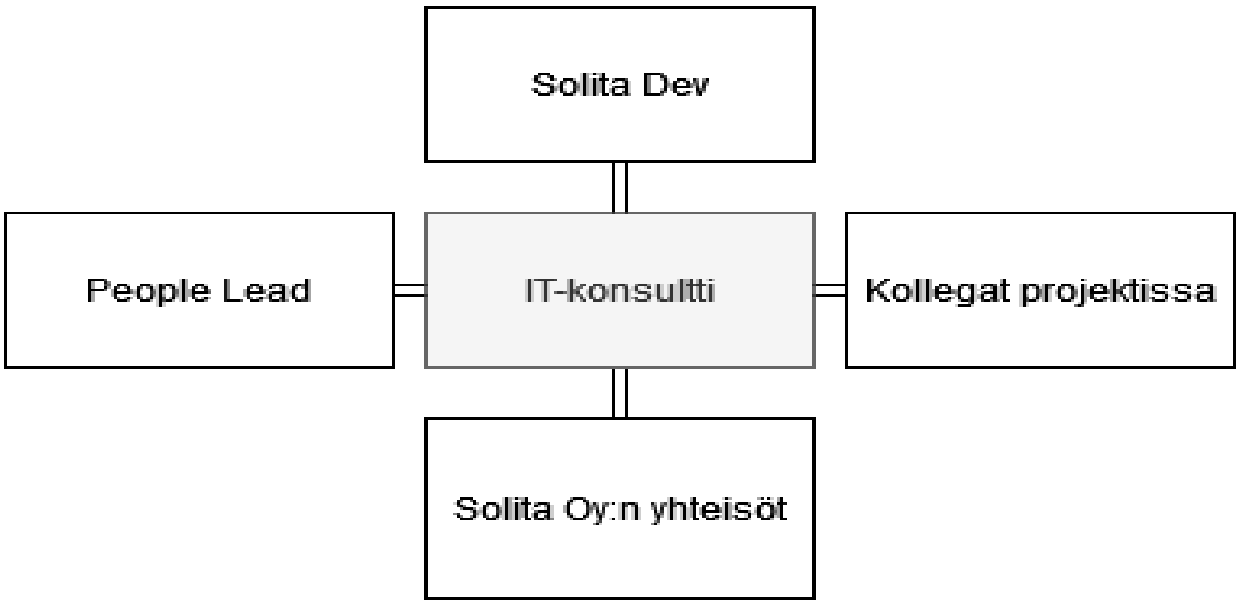
Henkilökohtaisen tietämyksenhallinnan lähtökohta on vaatimaton. Tekijällä ei ole aiemmin ollut tietoista prosessia henkilökohtaisen tietämyksenhallinnan harjoittamiseen. Prosessi alkaa perusteista ja työkalujen valinnasta, minkä jälkeen voidaan alkaa kehittää syvällisempää tietoa ja perusteellisempaa työprosessia tietämyksenhallintaan.

Konkreettisia työtehtäviä on projektissa lukuisia ja ne vaihtelevat laadunvarmistuksen, tietokantatyöskentelyn, full-stack ohjelmoinnin, käyttöliittymäsuunnittelun, pilvipalvelujen hyödyntämisen, asiakkaan tarpeiden selvittämisen, suunnittelun, dokumentoinnin, sekä vianselvittelyn välillä. Koska on osattava lukuisia eri taitoja, niiden kehittäminen vaatii paljon resursseja. Taitotaso vaihtelee työtehtävittäin mutta keskimäärin tekijällä on ymmärrys toteutettavasta tehtävästä ja kyky toteuttaa se. Joskus ilmenee työtehtävä, joka menee mukavuusalueen ulkopuolelle. Tällaisessa tilanteessa täytyy tukeutua kollegoiden neuvoihin, mutta omien rajojen tunteminen ja koettelu on osa kompetenssia ja kehitystä.

Projektissa on toisinaan tilanteita, joissa sovellusarkkitehdin taidot olisivat tarpeen ja näiden taitojen osaaminen olisi hyödyllistä projektityössä. Lisäksi tekijällä olisi parannettavaa tuoteomistajan ja sidosryhmien haastamisessa, jotta pystyttäisiin paremmin tuottamaan ohjelmiston ominaisuuksia mitä tarvitaan, eikä mitä halutaan. Dokumentaation saralla tekijällä on kehitettävää dokumentaation tuotannossa ja katselmoinnissa.

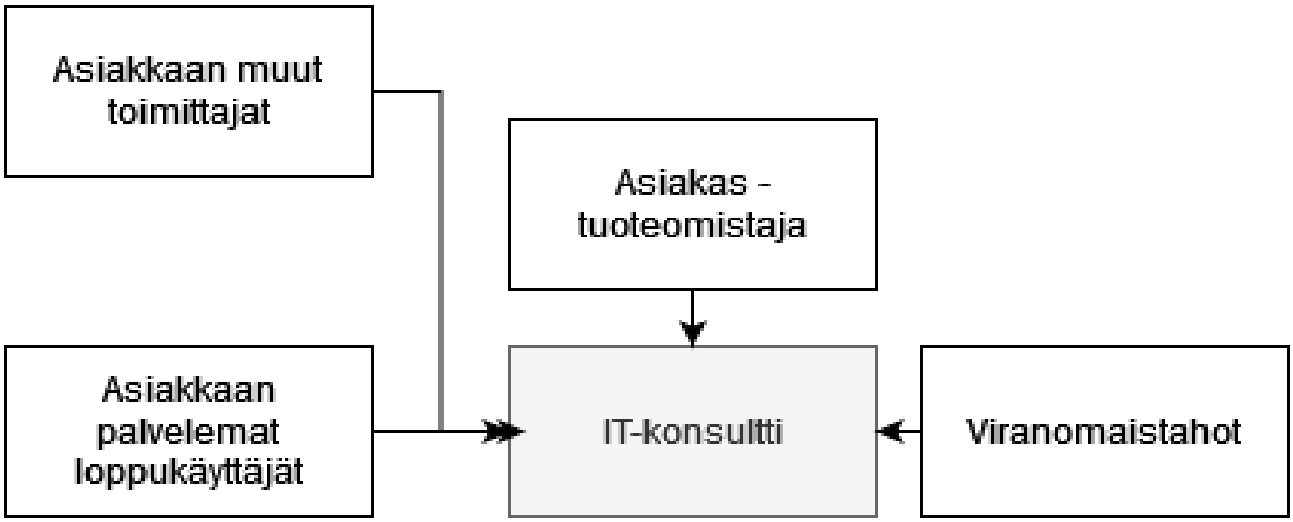
2.2 Sidosryhmät

Sidosryhmät kuvataan Solita Oy:n konsultin näkökulmasta. Sisäiset sidosryhmät käsittävät projektissa työskentelevät kollegat, Solitan tukiyhteisöt, Solitan ohjelmistokehittäjien osaston ja konsultin lähimmän esihenkilön, joka toimii myös tukihenkilönä.



Kuva 1. Sisäiset sidosryhmät (kuva tuotettu draw.io -palvelulla)

Ulkoiset sidosryhmät koostuvat asiakkaasta, eritoten tuoteomistajasta, heidän muista toimittajistaan ja toimittamamme ohjelmiston loppukäyttäjistä. Tärkeimmät intressit ovat tuoteomistaja, joka ohjaa työn kehityssuuntaa ja asiakkaan muut toimittajat, jotka ovat läheisessä yhteistyössä kanssamme. Viranomaistahot voivat vaikuttaa projektin toimintaan.



Kuva 2. Ulkoiset sidosryhmät (kuva tuotettu draw.io -palvelulla)

2.3 Vuorovaikutustilanteet

Vuorovaikutustilanteet projektissa ovat monimuotoisia, valtaosa niistä on työtovereiden kanssa keskustelu kasvotusten toimistolla. Tyypillisesti kyseessä on neuvonanto, yhteinen suunnittelu, informaation jakaminen tai vinkki ohjelmistototeutusta varten. Tämä on yksi tärkeimpiä vuorovaikutustilanteita ammatillisen kehittymisen ja yhteisöllisyyden kannalta. Lisäksi Solita Oy:n Slack-kommunikointialustan yhteisöt muodostavat ammattilaisten verkoston, johon pystyy tukeutumaan tarvittaessa. Solita Oy tukee kompetenssin kehittämistä ja tekijä ottaa osaa lukupiiriin, jossa osaamisen kehitystä toteutetaan yhteisillä keskustelu- ja analyysi-istunnoilla. Tämä on tehokas keino jakaa tietoa ja kokemuksia aiemmista projekteista.

Asiakkaan työyhteisön kanssa kommunikointi toimii asiakkaan Zulip-kommunikointialustan ja sähköpostin kautta. Palveltavat sidosryhmät ovat saavutettavissa täältä. Asiakas suosii myös sähköpostijakelua viestintään, Teams-palaverien lisäksi. Dokumentaatio ja ohjelmiston informaatiojakelu on saavutettavissa pääasiallisesti asiakkaan hallinnoimasta Confluence-palvelusta. Merkittävin vuorovaikutustilanne on poikkeuksetta työpäivät asiakkaan tiloissa ja Teams palaverit, missä tyypillisesti päästään yhteisymmärrykseen. Erikoisena vuorovaikutuspaikkana toimii lounasruokailut sidosryhmien henkilöstön kanssa, mikä on osoittautunut hyväksi tilanteeksi muodostaa tiiviimpiä suhteita sidosryhmien välille.

2.4 Tietoperustaa, menetelmiä ja viitekehyksiä

Confluence dokumentaation hallinnassa

Confluence on tyypillinen ratkaisu keskitetylle dokumentaation hallinnalle. Erityisesti projektit, jotka hyödyntävät Jira-projektinhallintatyökalua, käyttävät usein Confluencea. Molemmat ovat Atlassian-yrityksen tuotteita ja ne ovat hyvin integroitu keskenään. Tämä ratkaisu mahdollistaa oikeuksienhallinnan dokumentaatioon ja ohjelmistokehitykseen. Esimerkiksi voidaan asettaa, että vain ohjelmistokehittäjät voivat luoda dokumentaatiota, kirjautuneet normaalikäyttäjät voivat lukea dokumentaatiota ja henkilöt, jotka eivät ole omista oikeuksia oikeasta aihepiiristä eivät näe dokumentaatiota lainkaan. Tämä on käytännöllistä hallinnollisessa mielessä, vaikkakaan ei täysin ongelmatonta. Esimerkkitapauksena dokumentaatiota on hankala etsiä, jos ei tiedä sen olemassaolosta.

Tiedon piilottaminen täytyy arvioida dokumentaation arkaluontoisuuden mukaan. Voi olla erittäin hyviä syitä olla jakamatta ohjelmistoratkaisun tarkkaa kuvausta suurelle yleisölle, esimerkkeinä tietoturva ja turvaluokitellut tiedot. Confluencessä dokumentaatio tallennetaan sivuina, jotka muodostavat hierarkkisen rakenteen, täten dokumentoijan vastuulle jää jäsentää dokumentaatio ja katseluoikeudet parhaaksi katsomallaan tavalla.

Confluenceen kirjoitettavat sivut muodostavat hierarkkisen puurakenteen, jonka avulla dokumentaatio jäsennellään. Olen itse huomannut eri projektien Confluencessa hallinnoiduissa dokumentaatioissa sen, että tiedon tallettajat mieluummin lisäävät dokumentaatiota olemassa olevien sivujen alle sen sijaan, että tehtäisiin uusi sivu ja tieto olisi helpommin havaittavissa puurakenteessa. Tätä rakennetta voidaan käyttää väärin esimerkiksi lisäämällä saman ohjesivun alle useamman eri konseptin tai ominaisuuden dokumentaatio, mikä hankaloittaa tiedonhakua ja lisää väärinymmärryksen riskiä.

2.5 Keskeiset käsitteet

Henkilökohtainen tietämyksenhallinta. Eroaa tietämyksenhallinasta skaalassa ja tavoitteissaan. Tällä tarkoitetaan tietotyöntekijän henkilökohtaista kasvavaa tietovarastoa, jolla hän tukee työtään ja oppimistaan.

TypeScript. JavaScript kielten päälle rakennettu ohjelmointikieli, joka lisää ominaisuuksia JavaScriptiin. Tärkeimpinä muutoksina ovat tyypitykset funktioille, muuttujille ja luokille. Tämä yleensä parantaa koodin ylläpidettävyyttä.

Refaktorointi. Koodin uudelleen järjestäminen ja kirjoittaminen niin, että alkuperäinen toiminto pysyy muuttumattomana.

Koodikatselmointi. Prosessi, jonka avulla suoritetaan koodin laadunvarmistus. Prosessissa toinen ohjelmoija tarkistaa onko tuotettu koodi ymmärrettävää, onko koodiin jäänyt virheitä ja olisiko siihen tehtävissä parannuksia.

Lokitus. Prosessi, jossa ohjelmistossa tapahtuneiden operaatioiden tapahtumisesta pidetään lokitietoja. Lokitusta hyödynnetään vikatiloissa selvittämään ohjelmistovirheitä.

Front-end. Tällä viitataan ohjelmointiin, jossa toteutetaan käyttöliittymiä, ulkoasuja ja käytettävyyttä.

Back-end. Tällä viitataan ohjelmointiin, jossa toteutetaan ohjelmiston toimintalogiikkaa ja rajapintoja.

Full-stack. Tällä viitataan ohjelmointiin, joka käsittää sekä front-end, että back-end ohjelmointia.

Regressio. Koodimuutos, joka heikentää ohjelmiston suorituskykyä aiempaan versioon verrattuna.

NodeJS. JavaScript pohjainen ajoympäristö, jolla voidaan toteuttaa ohjelman Back-end.

OpenApi: Rajapinta, jolla kuvataan http-ohjelmointirajapintoja, standardisoitu ja yleinen.

3 Seurantajaksot viikoittain

3.1 Seurantaviikko 1

Viikon tavoitteena on kuvata päiväkirjamuotoisesti viikon työt peittomatriisissa esiteltyjen valittujen osaamisten kautta. Prosessi alkaa havainnoimalla ongelmia projektin nykyisessä tilanteessa. Prioriteettina olisi havainnoida millä tavoin nykyinen dokumentaation tila vaikuttaa projektityöhön, sidosryhmien työhön ja loppukäyttäjän ymmärrykseen ohjelmistosta. Päivän aikana tehtäväni priorisoituvat niiden kiireellisyyden mukaan, kesken päivää voi ilmentyä vikatilanne, joka minun on selvitettävä oitis. Ensimmäisen seurantaviikon analyysi on tavanomaista laajempi ja esittelee useamman käsitteen ja työkalun, minkä avulla aloitetaan projektin tilan analysointi ja itsereflektio ammatilliselle kehitykselle.

Maanantai 26.02.2024

Työtehtävänäni oli selvitellä ja korjata käyttöliittymässä ilmenneitä ja raportoituja bugeja, eli ohjelmistovirheitä. En käy niitä tarkemmin läpi, koska niissä ei ilmene dokumentoinnin kannalta kiinnostavia asioita.

Henkilökohtaisen tietämyksenhallinnan kehittämisen ensimmäinen haaste on työkalujen valinnassa. Vaihtoehtoja on monia ja on mielestäni tärkeämpää aloittaa prosessi kuin jäädä valitsemaan täydellisintä työkalua. Kollegani suosittelemana ja lyhyen selvitysprosessin jälkeen päädyin kokeilemaan Obsidian -nimistä muistiinpanosovellusta.

Dokumentaatiota kirjoittaessa ei aina ole samanlaista vapautta työkalujen suhteen, tyypillisesti konsulttityössä asiakas määrittelee minne ja missä muodossa dokumentaatio on tuotettava. Projektini tapauksessa tämä tarkoittaa Atlassian tuoteperheen Confluence -yhteistyöalustaa. Tässä on olennaista tiedon tallentaminen eri tiloihin, mikä määrittää dokumentaation näkyvyyden, joka on asiakkaan hallinnoitavissa.

Tiistai 27.02.2024

Sidosryhmä, joka käyttää kehittämäämme palvelua kysyi kysymyksen ohjelmistomme käyttöliittymän hakutoiminnon integroitavuudesta. Kyseessä oli asiakirjojen hakutoiminto parametrisoituna url-osoitteena. Tätä haluttiin käyttää toisesta palvelusta suorana linkkinä. Kysymys herätti keskustelua ja kollegani ehdotti ongelman ratkaisua indirektiota hyödyntäen.

Keskiviikko 28.02.2024

Päivän aikana otin työstettäväksi muutaman pienemmän kokonaisuuden, lähinnä pienen prioriteetin bugikorjauksia, dokumentoinnin näkökannalta vähemmän kiinnostavia.

Päivän aikana ilmeni tarve lisätä uusia konekäyttäjiä eräälle sidosryhmälle, automatisoituja palvelumme rajapintaa käyttäviä ohjelmia tuotantoympäristömme hyväksyttyjen käyttäjien listalle. Tämän nostan erityisesti kiinnostavana dokumentaation kannalta.

Tätä työtehtävää suorittaessani kohtasin epäselvyyttä ja hankaluutta oikeiden Amazon web services -tuotteiden ja välilehtien löytämisessä olemassa olevan dokumentaatiomme perusteella. Tämä otettiin korjaukseen lähdemateriaalin kuvaaman ’Friction log" käsitteen ja korjausprosessin mukaisesti. Tässä tapauksessa minun ei tarvinnut haastatella muita käyttäjiä löytääkseni tämän epäselvyyden.

Torstai 29.02.2024

Päivän aikana korjasin ja optimoin käyttöliittymätestejä, mitkä hyödyntävät React-testing kirjastoa. Testejä kirjoittaessa on ollut epäselvyyksiä, milloin kuuluu käyttää findBy, queryBy tai getBy alkuisia funktiota. Kirjaston dokumentaatio ei selkeästi kerro, mikä ero ja tavoite eri funktioilla on. Tämä johti siihen, että kaikki vaiheet kirjoitettiin asynkronisena operaationa, vaikka tämä ei ollut tarpeellista. Aikaisemmin tänä vuonna käyttöön otettu apukirjasto ohjasi oikeiden funktioiden käyttämisessä riippuen käytettiinkö koodissa asynkronista kutsua vai ei.

Perjantai 01.03.2024

Työtehtäväni oli jatkaa bugien korjaamista ja sen jälkeen työstää käyttöliittymässä olevan valikon refaktorointia, koodin uudelleen jäsentämistä tavalla, joka ei muuta sen käyttäytymistä. Viikon päätteeksi on hyvä nostaa esiin, että miltei joka päivä projektimme uudet ominaisuudet ja muutokset laitetaan koodikatselmoitavaksi. Tässä prosessissa käydään läpi mahdolliset ajatusvirheet ja tehdään ehdotuksia korjauksia varten.

Solitalla tuetaan ammatillista kehittymistä vapaaseen malliin ja otan osaa toimiston lukupiiriin, joka kokoontuu perjantaisin ja luemme paraikaa ohjelmistoratkaisujen dokumentaation periaatteista ja tuottamisesta.

’Koodikahvit’ ovat kahden viikon välein pidettävä Solitan sisäinen tapahtuma, missä vapaasti kuka tahansa voi esitellä työtänsä tai kiinnostavaa teknologiaa mitä on oppinut. Tämä tarjoutui oivaksi

tilanteeksi tehdä muistiinpanoja uusista käsitteistä ja työkaluista ja harjoitella tietovaraston laajentamista.

Viikkoanalyysi

Viikon aikana tein lukuisia havaintoja, miten dokumentaation puute tai tietämättömyys dokumentaation olemassaolosta vaikuttaa omaan toimintaamme ja sidosryhmiemme toimintaan. Tutkin keinoja parantaa omaa dokumentaation tuotantoani ja vaikutusmahdollisuuksiani, jota käyn läpi seuraavaksi. Viikon analyysi käsittelee peittomatriisin ensimmäistä ja toista kehityskohdetta.

Obsidian

Pintapuolisesti ohjelmisto muistuttaa muita Markdown merkintäkieleen pohjautuvia ratkaisuja. Merkittävänä erona muihin ohjelmistoihin on eri muistiinpanojen linkittäminen toisiinsa. Tällä Obsidianissa voidaan rakentaa miellekarttoja ja asiakokonaisuuksia. Lisäksi nämä muistiinpanot voidaan tallentaa käyttäen Git -versionhallintajärjestelmää. Kehittäjänä minulle valmiiksi tutut menetelmät riittävät mielestäni perusteluksi Obsidianin käyttöönottoon sen helppouden takia. Mikäli seurantajakson aikana ilmenee parempia työkaluja, harkitsen niihin siirtymistä tilaisuuden tarjoutuessa.

Koodikahvit ja lukupiiri osoittautuivatkin Obsidianin käyttöönoton tulikokeeksi, jonka pohjalta huomioin, että asiakokonaisuuksien linkittäminen kokonaisuuksiksi vaatii oman aikansa. Kun katselmoin viikon loppuvaiheen tuloksia näin, että ne olivat irrallisia eivätkä muodostaneet eheitä kokonaisuuksia. Aloin korjaamaan tätä rakentamalla näille hierarkkista rakennetta linkittämällä asioita yhdistäviä välisivuja, jolloin nämä asiat rakentuvat kokonaisuudeksi helpottaen muistiinpanojen navigointia. Ymmärrän käyttöönoton jälkeen, että henkilökohtaiset tietämysvarastot vaativat dokumentoinnin kaltaisia prosesseja, jotta lopputulos on käytettävissä ja korkealaatuinen. Henkilökohtaisiksi tarkoitettujen muistiinpanojen kirjoittamisessa ei kuitenkaan ole samankaltaista tiedonjakamisen epävarmuutta, kun dokumentaation kirjoittamisessa, sillä kirjoittaja toimii myös lukijana.

Tietovaraston rakentaminen

Obisdian ei ole ensimmäinen, eikä varmasti viimeinen työkalu mitä rakennetaan henkilökohtaisten muistiinpanojen järjestelyä varten. Obisidiania edeltää useampi ratkaisu, esimerkiksi TiddlyWiki ja Roam. Tutustun tarkemmin seuranta-aikana Obsidianin mahdollistamiin tiedonhallinnallisiin tekniikoihin, tällä hetkellä opetteluni keskittyy ohjelman käytön rutinoitumiseen. Ymmärrykseni mukaan Obsidian ei sido käyttäjää ennalta ajateltuun ratkaisuun, vaan jättää käytettävän tiedonhallintatekniikan käyttäjän päätettäväksi.

Ensimmäiseksi suunnittelin kokeilevani Niklas Luhmannin popularisoimaa Zettelkasten -menetelmää, joka koostuu pienistä muistiinpanoista ja tiedon pätkistä, jotka lokeroituvat ja linkittyvät aihepiireittäin. Nämä linkitetyt muistiinpanot muodostavat kokonaisuuksia ja ajatusketjuja. Kattavia ajatusketjuja on mahdollista hyödyntää tietotyössä.

Obsidian mahdollistaa muistiinpanojen ristiin linkittämisen, sekä muistiinpanoista ulkoisiin sivuihin linkittämisen. Todennäköisesti tällä voisi myös mallintaa dokumentaation henkilökohtaiseksi lähteeksi ennen Confluenceen vientiä, sillä Obsidian ja Confluence molemmat hyödyntävät Markdown merkintäkieltä, tällöin on mahdollista muotoilla dokumentaatiota Obsidianin avulla ennen lopullista julkaisua.

React-testing library

Oli varsin hankalaa saada selville mitä funktiota minun kuului käyttää testien rakentamisessa. QueryBy, GetBy ja FindBy alkuiset funktiot React-testing kirjastossa eroavat toisistaan implementaation osalta, mutta dokumentaatiosta se ei käy selkeästi ilmi (Testing Library, Soares, Dodds). Funktioilla haetaan virtualisoidusta käyttöliittymästä elementtejä, joita käytetään hyödyksi testien ajossa. Tällä on merkitystä testien suoritusaikoihin, riippuen käytetystä funktiosta. Tämän puuttuvan tiedon voisi lisätä ReadMe dokumentaatioon, jos projektitiimiin tulee uusia tekijöitä, mutta asia jäi vielä harkintaan, sillä loppujen lopuksi väärän funktion käyttö ei ole merkittävä haitta. Kyse on arviolta muutamasta millisekunnista, toisaalta vähäinenkin epäoptimaalinen funktiokutsu voi kertautua tuhansia kertoja yhdessä testiajossa, mutta testejä ei ole vielä tarpeeksi, että tämä olisi merkittävä hidaste projektin CI-palvelussa.

Olen pohtinut, miten dokumentaatiossa tuodaan ilmi eri asioita, sillä kaikki dokumentaatiota lukevat eivät etsi samaa tietoa. Halusin neuvoja kirjaston optimaaliseen käyttöön, mitkä jouduin etsimään ulkoisista blogikirjoituksista (Sbai 17.07.2023). Dokumentaatiossa haluttiin näyttää funktioiden käyttötavat ja funktioiden mahdolliset parametrit, mikä on toki ymmärrettävää dokumentaation päämääräisen tavoitteen kannalta.

Friction log

Bhatti, Corleissen, Lambourne, Nunez ja Waterhouse (2021, luku 3) esittelemää friction-log käsitettä sovellettiin olemassa olevan dokumentaation täydentämiseen. Käytännössä se on menetelmä, jolla havainnoidaan prosessien tai tehtävien kitkapisteet, jotka hidastavat tai estävät jonkin toiminnon toteuttamista. Kirjassa käsite esiteltiin käyttäjähaastattelun muodossa, minkä perusteella ohjelmiston käyttöliittymää kehitettiin. Sovelsin tätä menetelmää dokumentaation perusteella suoritettavaan työtehtävään.

Huomasin että olin hämmentynyt kohdassa, missä konekäyttäjiä lisättiin sallituiksi ulkoisiksi käyttäjiksi. Tätä hämmennystä hyväksikäyttäen kirjoitin friction-log:in paperille, sillä olin itse käyttäjänä. Friction-log:in avulla oli helpompaa muokata dokumentaatiota käsittämään yksityiskohtaisemman prosessin konekäyttäjän lisäämiseksi, niissä kohdissa missä minulla oli eniten vastoinkäymisiä.

Indirektio

Asiakkaan muiden toimittajien tekemät kyselyt toivat esiin useamman ongelman. Asiakirjojen hakemista url-osoitteita hyödyntäen implisiittisenä rajapintana ei ollut suunniteltu ominaisuus, mutta se on kuitenkin laajassa käytössä. Yleistä dokumentaatiota toiminnolle ei ollut, sillä osoitekenttä täyttyi käyttöliittymästä muutettavien asetusten mukaan, eikä sitä tarvinnut dokumentoida käyttöliittymän käyttäjille. Toimintaa emme voi täysin taata, sillä palvelumme vaatii autentikointia. Toimittajien tavoitteena oli linkittää verkkolinkkejä palveluumme omasta verkkotunnuksestaan. Tämä mitä todennäköisimmin toimii, kun käyttäjällä on molemmissa palveluissa kertakirjautumisistunto mutta emme voi taata sitä.

Eräs ratkaisu tähän olisi, että toteuttaisimme dokumentaation tälle implisiittiselle rajapinnalle. Kollegani mielestä tämä ei olisi hyvä ratkaisu, sillä se rajoittaa käyttöliittymän muuttamista ja ottaisimme vastuun rajapinnan toiminnasta, koska se on dokumentoitu toimivan tietyllä tavalla. Olisimme tällöin sitoutuneet toiminnallisuuden ylläpitoon. Tällä hetkellä muut toimittajat ovat itse päättäneet käyttää implisiittistä rajapintaa ilman että olemme sitoutuneet varmistamaan sen toiminnan ja muuttumattomuuden.

Parempana ratkaisuna tähän ongelmaan olisi toteuttaa indirektio ja luoda uusi ohjelmointirajapinta, joka siten hyödyntäisi kyseistä olemassa olevaa toimintoa. Tällöin tämä ominaisuus saadaan testattavaksi ja siten voimme taata sen toiminnan käyttöliittymän tai hakutoiminnon muuttuessa.

Indirektio oli minulle uusi termi mutta tuttu konsepti. Yksinkertainen selitys indirektiolle on: Ohjelmiston toiminnon toteuttaminen välivaiheen kautta (Elston, 09.10.2011). Indirektio-rajapinnan käytön tukemiseen kehittäjätiimi voi sitoutua ja sen avulla sidosryhmät voisivat käyttää toimintoa paremmalla varmuudella.

Konsulttityössä meidän täytyy myös kustannusarvioida toteutettava ominaisuus. Tässä tapauksessa kustannukset vaikuttivat niin korkeilta, että rajapinnan toteuttaminen ei olisi kannattavaa. Emme olleet saaneet merkittäviä havaintoja siitä, että ulkoisista verkkotunnuksista linkittäminen ohjelmiston käyttöliittymän hakutoimintoihin ei toimisi. Tästä syystä päätimme, että asiaan voi palata myöhemmin, mikäli käyttöä haittaavia ongelmia ilmenee.

Kuluneen viikon havainnot ovat kiinnittäneet huomioni puutteelliseen dokumentaatioon ja miten se lopulta vaikuttaa omaan ja sidosryhmien työhön. Tulevien viikkojen aikana on tavoitteena korjata ja kehittää henkilökohtaista prosessia dokumentaatiotyöhön, sekä edistää projektin dokumentoinnin tasoa ja laatua.

3.2 Seurantaviikko 2

Viikon tavoitteenani haluaisin jatkaa edellisen viikon teemaa ja havainnoida nykyisen dokumentaatiomme tilaa ja kokeilla onko joitakin nopeita ratkaisuja mitä voisin implementoida.

Maanantai 04.03.2024

Työtehtävinäni oli lokituksen lisääminen, jotta voimme kerätä metriikkaa ohjelmistokutsuista ja sitä kautta parantaa sovelluksen laatua. Mekanismi tälle on seurata kutsujen toteutumista ja nopeutta, minkä avulla voimme seurata sovelluksen toimintaa.

Kerättyä metriikkaa voi hyödyntää esimerkiksi, jos ilmenee jokin tietty hidas prosessi tai tutkia koodimuutoksen mahdollista sivuvaikutusta prosessinopeuteen. Tässä on hyvä pitää mielessä, että käyttäjät eivät tyypillisesti halua odottaa verkkosivuilla. Toteuttamamme palvelun luonteen takia, käyttäjät eivät turhaudu toimintojen hitaudesta, vaikka pyrimmekin mahdollisimman nopeaan lopputulokseen.

Toisena tehtävänä minulla oli toiminnon elinkaaren lisäys väliaikaisesti tallennetuille tiedostoille. Tämä oli erään sidosryhmän toivoma laajennus olemassa olevaan rajapintaan. Ohjelmistossamme on olemassa OpenApi standardin mukainen kuvaus, jolla kommunikoimme sidosryhmillemme rajapintojemme validaation hyväksymiä datamuotoja. Tietotyyppien lisäksi kuvataan myös mahdolliset raja-arvot ja niiden käsittelytavat.

Tiistai 05.03.2024

Kiinnostavana työtehtävänä dokumentoinnin kannalta nostaisin selvittelypyyntöihin vastaamisen sidosryhmän automatisoitujen prosessien käyttöönottoa varten. Rajapintamme tietomalli, jota kuvasimme swagger-työkalun avulla, ei ollut tarpeeksi selkeä siinä, miten tarjoamamme sivutus toimii. Tämä johti epäselvyyteen rajapinnan tuottamissa tuloksissa ja käytössä. Seuraavaksi oli selvitettävä, että miten voimme ratkaista ongelman, tapahtuuko se rajapintamuutoksella vai ratkaiseeko sidosryhmä sen eri tavalla.

Selvittelyn ja viestinvaihdannan jälkeen perimmäisenä ongelmana oli, että rajapinta kutsussa palautamme enintään 60 mittaisen listan heidän haluamiaan olioita JSON muodossa ja heillä oli tarve

useammalle. Ratkaisuvaihtoehtoja tarjoutui useampi, mutta heidän päättämänsä ratkaisu ei lopulta tullut tietooni, ilmeisesti ratkaisu päätettiin toteuttaa olemassa olevan ratkaisun tukemalla tavalla.

Keskiviikko 06.03.2024

Työtehtäväkseni valitsin käyttöliittymän valintapalkin refaktoroinnin. Kyseiset muutokset tehtiin fasilitoimaan asiakirjojen tyypitysten tasalaatuisuutta. Tämä sai minut yleisellä tasolla miettimään käyttöliittymämme käytettävyyttä, eli miten ohjaamme käyttäjää onnistumaan tehtävässään, ilman että hän joutuu prosessoimaan ylimääräisiä ohjeita.

Oman kokemukseni mukaan on hyvin tyypillistä, että käyttöliittymiä dokumentoidaan teknisellä tasolla harvoin. Esimerkiksi React-kirjastoa käytettäessä komponenttikohtaisia kuvauksia on vähän. Tähän vaikuttaa myös projektin koko, vain tarpeeksi suuri tai ylläpitoon siirtyvä projekti tarvitsee teknistä kuvausta käyttöliittymän toteutuksesta. Tällä hetkellä arvioisin, että projektimme ei ole siinä kokoluokassa, missä sellaisen dokumentaation tuottaminen olisi tarpeellista. Projekti on pyritty pitämään mahdollisimman yksinkertaisena ja jaettuna React-komponentteihin tavalla, jota on helppo seurata kehitysympäristön avulla.

Torstai 07.03.2024

Projektissamme tietokokonaisuuksia käsitellään asiakirjoina, joille asetetaan metatietoja ja yksi näistä metatiedoista on asiakirjan tyyppi. Tyyppiä käytetään asiakirjan luokitteluun ja käyttöoikeuksien määrittämiseen.

Otin tehtäväkseni edistää asiakirjatyyppiemme tyyppiturvallisuutta, jotta voimme paremmin hyödyntää TypeScriptin tuomia laajennoksia. Tyyppiturvallisuuden puute ei ole sinällään akuutti ongelma mutta asian korjaamatta jättäminen jättää mahdollisuuden bugien muodostumiselle. Asiakirja on olio, jolla on tunnisteena yksi uuid-merkkijono. TypeScriptin avulla saamme tämän eksplisiittisemmin tyypitettyä, niin että koodissa voidaan seurata paremmin tiedon kulkua. Tämä on myös askel kohti mahdollisuutta sovittaa sidosryhmiemme käyttämiä ja sisäisiä asiakirjatyyppejämme keskenään yhteensopivimmiksi, tällä hetkellä ne ovat valitettavan erilaiset. Tämäkään ei ole suoranaisesti ongelma mutta nostaa todennäköisyyttä käsittää asia väärin.

Perjantai 08.03.2024

Tänään tehtävänäni oli toteuttaa uuden asiakirjatyypityksen päivitys. Toisin sanoen asiakas toteuttaa haluamansa muutokset, jotka toimitetaan projektitiimille. Tätä operaatiota varten on olemassa prosessi ja dokumentaatio, miten asia toteutetaan, jota seuraten itsekin opin miten asia hoituu käytännössä. Normaalitapauksessa tämä on nopea ja helppo prosessi, sillä tätä varten on luotu skripti,

joka käy läpi ja tutkii annetun tyypityksen ja lopuksi muuttaa sen JSON muotoon, mitä ohjelmistomme käyttää.

Tällä hetkellä asiakirjojen tyyppien yhdistämisessä on käytössä yksi vähän työläs prosessin osa – asiakirjatyyppien yhdistäminen, mitä yritän keventää ja automatisoida. Valitettavasti en ehtinyt saada sitä valmiiksi asiakirjatyypitystä varten. Tätä prosessin osiota ei tarvita usein mutta se on tällä hetkellä työläs ja automatisoitavissa. Tyyppien yhdistäminen toimii SQL-lauseella, jota varten tarvitaan uuden ja vanhan tyypin tunniste, sillä tietokannassa pystymme muuttamaan asiakirjan tyyppiluokitusta vaihtamalla uusi vanhan tunnisteen tilalle. Tämä viedään tietokantaan migraation avulla ja toisinaan nämä migraatiot voivat olla hyvin pitkiä. Ongelmaksi osoittautuu inhimillisten virheiden livahtaminen SQL-lauseseen, sillä kaikki tunnisteet ovat uuid-merkkijonoja.

Lukupiirissä käytiin läpi dokumentaation laatimisen tärkeätä ominaisuutta: Luettavuutta. Väärinymmärrykset dokumentaation tulkinnassa ovat haitallisia ohjelman käytettävyyden kannalta. Kirjoittaja voi itse tulkita dokumentaation hyvin eri tavalla kuin lukija. Tämä on esitetty vapaasti käännettynä ’tiedon kirous’ -nimisenä konseptina, joka on hyvä pitää mielessä dokumentaatiota laatiessa (Bhatti ym. 2021, luku 1). Lukupiirissä herännyt keskustelu käsitteli myös muuta kirjoittamista ja sitä kuinka tärkeätä on, että kirjoitetun tekstin katselmointi on välttämätön vaihe ehkäisemään eri tulkintojen muodostumista dokumentaatiosta.

Osa lukupiiriin osallistujista mainitsi käyttävänsä laajoja kielimalleja ensimmäisenä katselmoijana. Ensimmäiset vedokset olivat heidän mukaansa käytännöllistä syöttää kielimallille, minkä jälkeen he saivat välitöntä palautetta. Palaute ei välttämättä ollut korkealaatuista. He myös kertoivat, että annettu kritiikki kannattaa ottaa vastaan pitäen mielessä kielimallien rajoitteet, sillä suuremman syötteen kanssa kielimallin hallusinaation riski kasvaa merkittävästi. (IBM)

Viikkoanalyysi

Kuluneella viikolla dokumentointityöt eivät olleet keskeisessä asemassa, mutta se ei tarkoita, ettei tehdyillä tehtävillä olisi arvoa dokumentaation tukena tai dokumentointitarpeen kartoittamisessa. Viikon analyysi käsittelee peittomatriisin ensimmäistä ja toista kehityskohdetta.

Esimerkkinä: kuinka moni sidosryhmistämme tietää, että ohjelmistomme tukee väliaikaista tiedostojen tallentamista. Useampi sidosryhmämme palvelee loppukäyttäjää tavalla, joka aloittaa prosessiin, jossa luodaan asiakirja, mutta asian käsittelyä ei välttämättä suoriteta loppuun ja asiakirja ei siten edisty pois luonnostilasta. On siis todennäköistä, että se voisi olla ominaisuus mitä useampi sidosryhmä käyttäisi hyödykseen, jos tämä olisi yleisessä tiedossa. Ominaisuuden toteutushetkellä väliaikaisen tiedoston tallennusmahdollisuus selviää nopeasti nopeasti ohjelmistomme OpenApi kuvauksesta, joten kirjoitetun dokumentaation vajavaisuus ei ole merkittävä ongelma.

Koska tämä on kohtuullisen nopeasti selvitettävää tietoa, olisiko tämän kirjoittaminen erikseen liiallista dokumentaation tuottamista, jos tämä lisätään omaksi Confluence aiheekseen. Asiasta tiedottaminen on haastavaa, sillä emme tiedä kaikkia sidosryhmiä, joita asia kiinnostaa. Tiedotuskanavina omaamme vain Zulip-viestintäalustan, missä sidosryhmämme liittyvät aihepiireittäin kanaville, kun he tarvitsevat vastauksia ohjelmistomme toiminnasta ja kun koordinoimme integraatioiden toteutusta.

Sama ajatus pätee mielestäni käyttöliittymän dokumentaatioon. Ellei suunnitteluasiakirjoja tai muita vastaavia ole jo olemassa, on niitä turha tuottaa jälkikäteen. Nykyisille kehittäjille siitä ei ole merkittävästi hyötyä. Sidosryhmien ja ohjelmiston käyttöliittymän käyttäjien ei tarvitse niistä välittää. Tälle muodostuu tarve, kun projektin kehittäjät vaihtuvat mutta se ei ole tällä hetkellä ilmeinen muutos. Päätavoite olisi varmistaa, että nykyinen dokumentaatio on tehty korkeatasoisesti tukeutuen lähdekirjallisuuteen.

Front-end Dokumentaatio

Front-end dokumentaatiota mihin itse olen törmännyt ovat olleet käyttöliittymien suunnitteluasiakirjat, wireframe-mallit ja suunnitteluun käytetyt muotoiluohjeet. Projektissamme ei tämänkaltaisia suunnitteluasiakirjoja ole projektin alkuvaiheen aikataulupaineiden takia. Toisaalta uskon tämän olevan hyväksyttävä asia, sillä ohjelmistomme käyttöliittymän React-komponentit ovat pieniä ja tarkoituksellisesti pidetty yksinkertaisina. Mielestäni hyödyllisin lisä olisi käyttäjätarinoiden lisääminen. Ohjelmistokehittäjille on selvää mitä lopputuloksia käyttäjien kuuluisi saavuttaa, mutta emme ole täysin varmoja mitä kaikkia keinoja he käyttävät niiden saavuttamiseen, ja mitä byrokraattisia prosesseja he käyvät läpi ohjelmiston ulkopuolella. Käyttäjätarinat parantaisivat kehittäjien kokonaiskuvaa ohjelmiston käytöstä, ja sitä voitaisiin hyödyntää eri ratkaisuvaihtoehtojen arviointiin.

Dokumentoinnin tarve

Päällimmäisenä prioriteettina uskoisin, että olisi tärkeätä tunnistaa yleisö ja heidän tietotarpeensa. Tämä on helpompaa, kun sidosryhmät kysyvät dokumentaatiosta tai jostain ohjelmistomme ominaisuudesta. Voimme myös arvioida mitkä osiot olisivat dokumentoinnin tarpeessa, perustuen sidosryhmiemme tuleviin tarpeisiin. Tämä vaatii luonnollisesti kartoitusta sidosryhmien nykytilasta ja tavoitteista ja korkeamman tason näkemystä.

Arvion kautta voidaan helpommin määritellä, onko ominaisuuden dokumentointi tarpeen, sillä konsultin toimessa on tärkeätä tuottaa arvoa asiakkaalle. Tämä ominaisuuksien priorisointi voi johtaa dokumentaation tuottamisen niukkuuteen, etenkin jos projektin vaihe vaatii ominaisuuksien tuottamista ripeässä tahdissa. Tällä hetkellä arvioisin, että projektimme on vaiheessa, jossa voimme asettaa resursseja myös dokumentaation tason parantamiseen ja laajentamiseen.

En tule tekemään lopullisia ratkaisuja yksin vaan yhdessä muun projektiryhmän kanssa. Arvioin kokemustasoni olevan vielä sen verran vajavainen, että riski ylidokumentointiin on merkittävä ja tämä on omalla tavallaan vahingollista. Jos dokumentaatiota on paljon, tiedon välittäminen lukijalle hankaloituu. On pidettävä mielessä, että lukijalla on tietty tavoite dokumentaatiota lukiessaan ja se on etsimänsä tiedon paikallistaminen ja hyödyntäminen ongelmanratkaisuun. Liiallinen yksityiskohtaisuus hankaloittaa lukijan tavoitteen toteuttamista, joten on syytä keskittyä ratkaisemaan lukijan tavoitteita kuten Carey, M., McFadden Lanyi, M., Longo, D., Radzinski, E., Rouiller, S. ja Wilde E (2014, luku 3) painottavat. Dokumentoinnin tuloksen katselmointi auttaa tämän tavoittelussa kuten myös dokumentaation sisällön validoinnissa.

Kuluneen viikon aikana itse dokumentaatiotyötä toteutettiin vain vähän mutta kuten huomaamme, niin voimme kartoittaa nykytilan puutteita ja tavoitetilaa projektillemme ja siten voimme saada kattavamman tietämyksen dokumentaatiomme tasosta. Ohjelmiston ominaisuuksien toteuttaminen usein priorisoidaan dokumentaatiotyön kustannuksella, tällöin dokumentaatiotyöhön käytettävä aika ja resurssit on valikoitava harkiten.

3.3 Seurantaviikko 3

Viikon tavoitteena on siirtyä puutteiden havainnoinnista dokumentaatioratkaisujen suunnitteluun ja toteuttamiseen. Aikaisempina viikkoina olemme havainneet dokumentaation olevan vajavaisempaa kuin toivottua ja tällä on haittavaikutuksia. Pyrin ottamaan dokumentaation tuottamisen osaksi viikoittaista toimintaa vähentääkseni dokumentaatiovajetta.

Maanantai 11.3

Ohjelmiston tuotantoympäristössä havaittiin ongelma, mikä ilmeni eräässä ohjelmiston hyödyntämässä PDF käsittelijässä. Tätä ongelmaa ei ollut kehitysympäristössä mahdollista toistaa, täten tuli debugaus, bugien selvittäminen, tehdä Docker-ohjelmiston avulla. Prosessi itsessään ei ole vaikea, mutta Docker-konttien käsittely on jotain mitä olen päässyt tekemään kovin vähän. Tässä tilanteessa hyödynsin tietämysvarastoani, sillä olin kirjoittanut muutaman komennon muistiin selityksineen, joiden avulla pystyin toteuttamaan ohjelmistovirheen toistokaavan. Siitä seurasi havainto, että epätäydellisten PDF-tiedostojen sivumäärää ei saatu selville, joten sitä seuraavat operaatiot epäonnistuvat. Tähän kollegani ehdotti, että toteuttaisin suunnitteludokumentin, jossa kuvailisin ongelman ja toivotun ratkaisutilanteen sekä mahdolliset ratkaisut ongelmaan.

Ratkaisuvaihtoehtoja keksin loppujen lopuksi neljä ja nämä kustannusarvioitiin. Suunnitteludokumentaation voisi sellaisenaan ottaa osaksi projektin laajempaa dokumentaatiota, mutta se ei välttämättä ole tarpeen, mikäli sidosryhmät eivät tarvitse niitä toimintaansa. Kyseinen ominaisuus muuttaa olemassa olevan ominaisuuden toteutuksen yksityiskohtia mutta ei toimintaperiaatetta, eikä

siitä ilmene lainkaan muutoksia sidosryhmäkäyttäjille. Tämä lisättiin tietovarastoon, ettei se ole täysin hukassa, jos sitä joskus tarvitaan tai halutaan lisätä projektin Confluenceen.

Tiistai 12.3

Kävin kollegani kanssa läpi tulevaa ominaisuutta työstämästämme integraatiosta. Hän oli muodostanut Flow-kaavion - prosessikuvauksen, jossa kaikki mahdolliset tilat ja tilasiirtymät on otettu huomioon ja kuvattu täysin. Tämän avulla hän kuvasi tulevaa ohjelmiston ominaisuutta, millä laajennamme palvelujamme sidosryhmillemme.

Refaktoroitiin epäselvää käyttöliittymän rakennetta selkeämmäksi. Lisäksi teimme tuotantopäivityksen, joka epäonnistui. Tämä ei suinkaan johtanut ohjelman vikatilaan: käyttämämme CI-palvelu, eli jatkuvan integraation ohjelmisto, ei sallinut sen tapahtuvan. Tätä seurasi luonnollisesti julkaisuvirheen selvittely, jotta ymmärtäisimme ongelman juurisyyn ja jotta voisimme välttää vastaavan ongelman tulevaisuudessa.

Keskiviikko 13.3

Työpäivä alkoi koodin palauttamisella takaisin toimivaan versioon koska ilmeni, että käyttämämme CI-palvelu, eli jatkuvan integraation palvelumme, ei tukenut uusinta Kustomize-kirjastoa, jota käytämme pilvi-infrastruktuurikoodin päivittämiseen. Puutteellisen kirjaston takia, ei ollut mahdollista toteuttaa tuotantopäivitystä.

Kävi myös ilmi, että tarvitsemme lisää dokumentaatiota, jos muut projektit haluavat käyttää hyödykseen projektimme Docker-imageja, eli tiedostoja millä jaamme toteuttamamme ohjelmiston riippuvuuksineen ja ajoympäristöineen. Tämä on käytännöllistä, jos sidosryhmämme haluavat kehittää integraatiota täysin omassa kehitysympäristössään. Docker-imagejen käyttöönotto ei onnistunut ja huomasimme "friction log" kohdan Aws-Vault ohjelmiston käytössä. Löydetty ongelma lisättiin dokumentoitavaksi, kun huomattiin, että tämän Docker-imagen käyttöönotto ei ollutkaan mahdollista sidosryhmillemme.

Torstai 14.3

Toteutetaan aikaisempi suunnitelma skriptistä, jolla voidaan generoida migraatiotietokantalauseita perustuen asiakirjatyyppien päivitykseen. Dokumentoin skriptin tarkoituksen ja käytön sen valmistuessa, jotta vältetään dokumentaation vanhentuminen ennen koodin valmistumista. Tämä on aina riskinä, kun dokumentoidaan toteutusta ennen kuin se on valmistunut.

Dokumentaation tallennuskohteina toimisi projektin ReadMe ja wikisivu, jolle prosessi on aiemmin kuvattu. Tarkoituksena on vähentää inhimillisten virheiden mahdollisuutta, sillä nykyinen prosessi on manuaalinen mutta automatisoitavissa.

Perjantai 15.3

Jatkoin edellisen päivän töistä, eli generointiskriptin kirjoittamista. Haasteena ilmeni, että olemassa olevat tekstin analysointifunktiot eivät käsitelleet vanhentuneita asiakirjatyyppejä odotetulla tavalla. Koodista ei ilmennyt suoraan, miten vanhentuneita tiedostoja tallennetaan lopullisiksi asiakirjatyyppiluetteloiksi ja miten ne vanhenevat ja poistuvat käytöstä. Skriptin dokumentointi on tarpeen tehdä tavalla, jotta käyttäjän ei tarvitse erikseen selvittää toteutuksen yksityiskohtia.

Lukupiirissä kävimme läpi koodin käyttämistä osana dokumentaatiota ja siihen liittyviä riskejä: Virheet dokumentaatiossa saavat käyttäjät menettämään luottamuksen dokumentaatioon. Menetetty luottamus on vaikea voittaa takaisin. Koodin ymmärrettävyydellä on samanlaisia riskejä, kuten Bhatti ja muut toteavat (Bhatti, ym, 2021).

Koodikahveilla eräs kollegani esitteli miten dokumentaation puute erikoislaitteiston kanssa vaikuttaa negatiivisesti työn edistymiseen, sillä jokainen vikatilanne johtaa merkittävään selvittelytyöhön, ei pelkästään hänelle itselleen vaan myös loppukäyttäjälle. Kollegani etsivät paraikaa tapaa ratkaista ongelmaa. Yksi tapa olisi käyttää kuluttajakäyttöön tarkoitettuja tuotteita, sillä tyypillisesti kuluttajatuotteissa on vakaampi ja kattavampi dokumentaatio.

Viikkoanalyysi

Dokumentoinnin tuottamisen lisääminen osaksi kehitysprosessia näyttää onnistuneen tämän viikon osalta, vaikka parannusta vaativia kohtia löytyy vielä useita. Osittain luulen sen johtuvan siitä, että huomioin ongelmakohtia enemmän kuin aiemmin. Käsittelen seuraavaksi kuluneella viikolla edistettyjen töiden merkitystä yksityiskohtaisemmin. Viikon analyysi käsittelee peittomatriisin ensimmäistä ja toista kehityskohdetta.

Dokumentaatio osana suunnittelua

Valitettavasti uusien ominaisuuksien tuottaminen priorisoidaan usein niiden dokumentoimisen edelle. Sama pätee nykyisessä projektissakin. Poikkeus tälle on, jos dokumentaatio syntyy suunnittelun yhteydessä, kuten viikolla totesimmekin. Emme koe tällä hetkellä aikataulupainetta, mikä mahdollistaa maltillisemman lähestymistavan ratkaisuihin ja mielestäni tämä johtaa parempaan lopputulokseen.

Pelkkä laatiminen ei ole tuotetun dokumentaation elämänkaaren loppu, vaan nyt sitä on päivitettävä läpi ohjelmiston toiminnon elinkaaren ja tämän takaaminen voi olla perin hankalaa. Koodissa

itsessään ei ole viitettä dokumentaatioon, jolla voitaisiin viestiä, että dokumentaatio kuuluisi päivittää toiminnallisuuden päivittyessä. Projektinhallinnassa tehtävä tiketöinti on myös hyvä tilaisuus ottaa dokumentaation päivitys osaksi projektinhallinnan prosessia, jotta dokumentaation ylläpito ei jää tekemättä.

Flow-kaaviot

On hyödyllistä luoda flow-kaavioita ohjelman toiminnasta ja lisätä ne osaksi dokumentaatiota, sillä ne havainnollistavat ominaisuuden tiloja, dataa ja prosesseja nopeammin kuin tekstikuvaus. Optimitilanne olisi, jos flow-kaaviot ovat selkeitä ja yksiselitteisiä. Vaarana on myös, että kuvaaja hämmentää ja harhauttaa lukijaa. Se voi olla liian iso tai kuvata tarpeettomia asioita. Kuvaajien lisääminen vaatii tarkkuutta ja tarkoituksenmukaisuutta siten, että lukija uskoo lukemansa dokumentaation hyödyllisyyteen. Riskinä on tilanne, jossa lukija selvittää ongelmansa muulla tavalla, mikä tuodaan ilmi lähdekirjallisuudessa (Bhatti ym, 2021, luku 2).

Saavutettavuus on myös aspekti, mikä tulisi pitää mielessä, kun dokumentaatioon lisätään kuvaajia. Keinoja parantaa saavutettavuutta on lukuisia ja ne mukailevat muita saavutettavuuden tekniikoita, joita hyödynnetään käyttöliittymien kehittämisessä. W3C Web Accessibility Initiative (WAI) tarjoaa kattavan mallin, hyvin dokumentoituna. Sen mukaan toteutettu käyttöliittymä kattaa valtaosan saavutettavuushaasteista.

Ohjelmiston ulkoiset ominaisuudet osana dokumentaatiota

Aiemmin viikolla toteutin skriptiä, jolla muodostin SQL-migraatio lauseita, lyhyesti selitettynä se muodostaa tietokantapäivityksiä skriptille syötetystä datasta. Skriptiä voidaan ajatella ohjelmiston ulkoisena toimintona, vaikka sitä tyypillisesti säilytetään muun ohjelmistokoodin rinnalla. Vaikka skriptin käyttäjä näkee koodin mitä ajetaan, näiden tarkemmalla dokumentoimisella vältetään niiden muuttumista ‘mustiksi laatikoiksi,’ joiden sisäistä toimintaa ei ymmärretä ja niiden ylläpitoa laiminlyödään. Tämä vaikuttaa erityisesti uusien ohjelmoijien ymmärrykseen ohjelmistosta ja sen kontekstista.

Kerratakseni, ei ole ainutlaatuinen tilanne, että projekteissa on vajavainen dokumentoinnin taso. Tätä voidaan parantaa eri menetelmin ja voin retrospektiivisesti havaita, että otamme askelia dokumentaation tuottamiseen osana kehitystä. Tätä prosessia voisi tukea päivillä, jolloin työtehtävänä olisi pelkkää olemassa olevan koodin dokumentointia mutta sellaisia mahdollisuuksia on tyypillisesti varsin vähän.

3.4 Seurantaviikko 4

Tavoitteena tälle viikolle on jatkaa ohjelmistomme dokumentointia uusien ominaisuuksien implementoinnin lomassa. Tämä on hyvä tilaisuus kokeilla edellä mainittujen lähteiden ehdottamia tekniikoita dokumentaation tuottamiseen ja arviointiin.

Maanantai 18.3

Erään asiakirjan tallennus testiympäristössä aiheutti virheitä sidosryhmän rajapintakutsuissa. Tätä selviteltiin ja ilmeni, että kyseessä oli järjestelmämme bugi: Asiatyyppiä ei ollut asetettu validiksi tyypiksi rajapinnan validaatioon kaikissa tarvittavissa paikoissa. Tämä johti kutsun hylkäämiseen ja virheeseen. Dokumentoin tämän virheen ja pyrin dokumentoimaan sen kattavasti, että tulevat tyyppilaajennokset eivät kaadu vastaavaan ongelmaan. On hyvä, että ongelmasta saatiin kiinni testiympäristössä mutta haluamme välttää pienetkin virheet, jottei niihin tuhlata aikaa tulevaisuudessa. Arvioin riittäväksi tarkastuslistamaisen dokumentaation, jossa mainitaan kaikki lisättävät kohdat ja ohjataan lukija Git-commit viesteihin. Tämä vaikuttaa olevan toistaiseksi riittävä ratkaisu, jota voidaan laajentaa uuden projektijäsenen tutustuttamisen yhteydessä tarvittaessa.

Tiistai 19.3

Toteutamme ohjelmistoomme laajennosta, jolla mahdollistamme sidosryhmillemme ja käyttäjillemme suorapostin lähettämisen, aikaisempi toteutuksemme vaati postitettavilta kohteilta henkilötunnusta. Dokumentaatiota muutoksesta on muodostettu edeltävällä viikolla suunnittelun yhteydessä. Toteutin tästä käyttöliittymäosiota, jota varten emme erikseen tehneet suunnitelmaa tai suunnitteludokumentaatiota, sillä visuaalisen käyttöliittymän kannalta muutos on varsin rajattu: lisätään vain yksi lisävalinta käyttäjälle. Käyttöliittymän logiikassa ja jäsentelyssä oli enemmän toteutettavaa ja vanha toteutus muutettiin kompleksisemmaksi. Tämän takia merkitsin toteutetun koodin yhteyteen kommentteja, jotka selventävät miksi tietyt ratkaisut tehtiin. Tein tämän itseäni varten, sillä suurempi kompleksisuus vaatii tietorakenteiden refaktorointia, jota ei otettu vielä toteutettavaksi. Tästä tiesin, että toteutukseen palataan vielä tulevaisuudessa ja näin ollen olen dokumentoinut koodin lomaan itselleni tai kollegalleni apuja ja kontekstia tulevaa refaktorointia varten kommenttien muodossa.

Keskiviikko 20.3

Edellisen päivän aikana aloitetun käyttöliittymäominaisuuden toteutus jatkui. Seurattavan kehityksen kannalta ei merkittävää muutosta.

Seurannan kannalta kiinnostavana työtehtävänä nostaisin kollegani tuottaman ohjelmointirajapinnan ja ohjelmistomme prosessin dokumentaation katselmoinnin. Tarkoituksenani oli lukea kollegani tuottama dokumentaatio ja arvioida sen tarkkuus ja ymmärrettävyys. Käytin lukemani lähdemateriaalin, tässä tapauksessa documents for developers -kirjan väliotsikoita muistilistana, jonka avulla arvioin dokumentaatiota ja sen kattavuutta (Bhatti, ym 2023).

Torstai 21.3

Toteutetaan suorapostituksen käyttöliittymä loppuun. Laadunvarmistus, eli käytännössä testien kirjoittaminen vei valtaosan työpäivästä. Kiinnostavia nostoja dokumentaation kannalta ei ilmennyt.

Perjantai 22.3

Sain palautetta tehdyn postitusominaisuuden käyttöliittymästä käyttäjiltä, minkä implementoin kirjattuani sen Jira-palveluun bugikorjauksena. Lukupiirissä kävimme läpi dokumentaation julkaisuprosessia, sekä miten koodikatselmoinnin periaatteet pätevät tähän ja myös yleisesti kirjoittamiseen. Huomioimme, että dokumentaation automaattinen testaus olisi kiinnostava aihepiiri tutkia. Sitä voisi kokeilla laajojen kielimallien kautta, mutta sen toteuttaminen olisi työlästä ja hankala saata luotettavaksi, sillä laajojen kielimallien ’hallusinointi’ riski on edelleen suuri. Tällä tarkoitetaan virheitä, joita laajan kielimallit tekevät, kun ne tuottavat vastauksen, joka ei vastaa mallissa käytettyä dataa. kuten IBM verkkosivuilla asia kuvataan (IBM). Tästä syystä on hankalaa saada normalisoitua tulosta ja ennen tämän ongelman ratkeamista, ei ole syytä käyttää laajoja kielimalleja dokumentaation generoinnissa.

Viikkoanalyysi

Viikko oli odotettua kevyempi dokumentaation tuottamisessa mutta pääsin tarkastamaan ja arvioimaan miten kollegani tuottama dokumentaatio vastasi omaa käsitystäni ohjelmasta. Samalla tuli testatuksi myös lähdemateriaaleissa esiintyneet konseptit kertauksena. Tällä viikolla käsiteltiin peittomatriisin ensimmäistä, toista ja kolmatta kehityskohdetta

Dokumentaation arviointi

Katselmointiprosessi vaikutti hyvältä prosessilta, vaikka minulla oli kenties liikaa tietoa ohjelmistosta ja juuri koodikatselmoimastani ominaisuudesta. Liiallinen tieto saattaa johtaa aikaisemmin mainittuun ’tiedon kiroukseen’ - asymmetriseen tietoon ohjelmistosta, mikä hankaloittaa arviointia käyttäjän näkökulmasta. Toisaalta projektissa ei ole olemassa henkilöä, joka olisi täydellinen vastine käyttäjillemme.

Tukeuduin lähdemateriaaleissani esitettyihin suosituksiin, päämääräisesti arvioin dokumentaation tehtäväkeskeisyyttä, eli pystytäänkö dokumentaation avulla suorittamaan haluttu toiminto.

Dokumentaation ymmärrettävyys ja tarkkuus oli helppo arvioida, olin juuri koodikatselmoinut tuotoksen. Näitä arvioidessa tietoasymmetria sokaisee pahiten mahdollisille epäjohdonmukaisuuksille. Löydettävyyden ongelmia on osaltamme hankala ratkaista, dokumentaatio löytyy sieltä, minne se julkaistaan. Voimme tiedottaa asiasta mutta se ei ole hyödyllistä sidosryhmille, jotka eivät ole kiinnostuneet asiasta tällä hetkellä. Sidosryhmämme lopulta kertovat olemmeko onnistuneet vaiko epäonnistuneet tehtävässämme ja korjaamme dokumentaatiota tarvittaessa.

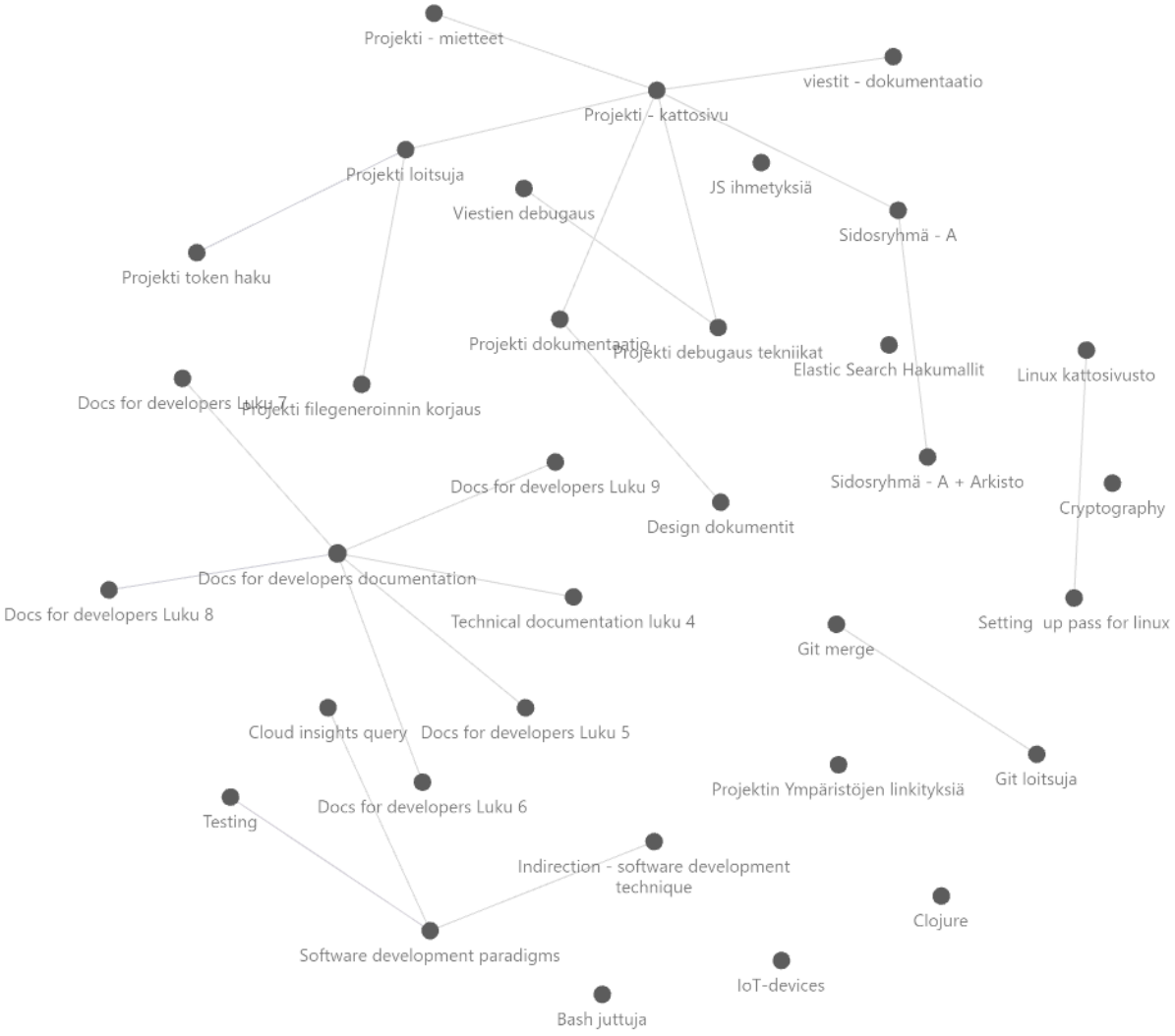
Dokumentaation korjaamisesta herää kysymys sen ylläpidosta ja palautteenhankinnasta. Meille palaute ja korjausehdotukset tulevat mitä ilmeisimmin suoria keskustelukanavia pitkin, sillä projektimme ei ole niin suuri, että olisi järkevää kerätä metriikkaa tai muodostaa käyttäjäkyselyjä, kuten Bhatti ja muut (2021, luku 8 ja luku 9) ehdottavat. Nämä ovat käytännöllisempiä työkaluja suurempien tuotteiden tai palveluiden kanssa. Lähdekirjallisuus kuitenkin kuvaa myös projektiimme soveltuvia ratkaisuja, esimerkiksi arviointiraatien muodostamista. Tämä on tehtävä sidosryhmiemme kanssa yhteistyössä, sillä he parhaiten ymmärtävät omat tarpeensa.

Viimeisenä ajatuksena aiheesta on dokumentaation virheiden ja vanhenemisen käsittely bugeina työjonossa. Tyypillisesti dokumentaation ylläpitoa ei pidetä järin tärkeänä työtehtävänä mutta virheellinen ja vanhentunut dokumentaatio ei ole luotettava lukijan näkökannalta ja on siten arvotonta, sillä hän ei voi arvioida mikä osa on luotettavissa ja mikä ei. Täten yksittäiset virheet dokumentaatiossa voivat vahingoittaa sitä kokonaisuutena, mikäli lukija ei kykene uskomaan dokumentaation virheettömyyteen. Tässä korostuu dokumentaation ylläpidon tärkeys ja työprosessina se vertautuisi tärkeydessään bugien korjauksen tasolle. Dokumentaatiossa ilmenevät virheet ja vanhentumisen kriittisyys eheyden kannalta on tulkittavissa, jota (Bhatti ym. 2021, luku 8) kehottaa priorisoimaan niiden vakavuuden, toistettavuuden ja korjattavuuden mukaan.

Zettelkastenin kehitys

Henkilökohtaisen tietämyksenhallinnan saralla tein muutoksia nykyiseen prosessiini, lisäsin tekemiini muistiinpanoihin linkityksiä, jotta ne muodostavat järkevän verkoston. Aiemmin lisäsin uusia muistioita, mikäli aihe sattui kiinnostamaan ja halusin pitää sen mielessäni, nyt lisäsin yhdistäviä koostesivuja eri aihepiireittäin. Näin pystyn hahmottamaan paremmin kokonaisuuksia ja havaitsen eri konseptien yhteyksiä paremmin. Lisäämäni koostesivut kokosivat yhteen esimerkiksi projektin sidosryhmien yhteyksiä, projektin toteutusyksityiskohtia sekä käyttöjärjestelmän operointiin tai asetuksiin liittyviä yksityiskohtia.

Tämä valitettavasti tekee muistiinpanoprosessista työläämmän mutta mielestäni tämä luo verkkomaisen rakenteen tiedolle ja on helpommin navigoitavissa, etten joudu tukeutumaan pelkästään Obisdian-sovelluksen hakuominaisuuksiin.



Kuva 3 Tekijän anonymisoitu muistiinpano. (kuva kaappaus tekijän Obsidian ohjelmasta)

Henkilökohtaisesti kirjoitin vähemmän dokumentaatiota kuin odotin, mutta osoittautuu, että pelkällä katselmoinnilla on positiivinen vaikutus dokumentaation tuottamiseen ja laadun parantamiseen.

3.5 Seurantaviikko 5

Tämän viikon tavoitteeni on parantaa Zettelkasten-menetelmän käyttöäni, sillä en koe, että käytän menetelmää oikein enkä saa siitä toivottua hyötyä. Samalla opettelen Obsidian ohjelman korkeatasoisempaa käyttöä, tämän pitäisi sujua käsi kädessä Zettelkasten-menetelmän opettelun lomassa. Tällä viikolla käsiteltiin peittomatriisin ensimmäistä ja toista kehityskohdetta

Maanantai 25.3

Tavoitteena on toteuttaa erään vanhemman bugi-ilmoituksen edellyttämä korjaus PDF-tiedostojen lukemiseen. Tietyt PDF-tiedostomuodot eivät menneet prosessiemme läpi ilman erillistä korjausta. Tämän selvittäminen oli perin hankalaa, sillä käyttämämme PDF työkalun versiolla oli merkittävä vaikutus onnistumiseen ja debugaus vaati muistiinpanoihin turvautumista. Tässä mielessä Obsidian kokeilu on osoittautunut hyödylliseksi. Prosessin uudelleen ajattelu olisi ollut vaivalloisempaa kuin sen etsiminen muistiinpanoistani.

Tällä viikolla toteutamme myös asiakirjatyypityksen asiakkaan toimittamien tietojen pohjalta. Huomasin, että tietoa asiakirjatyypitysten eriytymisestä tuotanto- ja testityypityksiin ei ole dokumentoitu selkeästi. Tämä ei sinänsä ole ongelma mutta mahdolliselle uudelle kehittäjälle se tulisi yllätyksenä. Dokumentoin tämän projektin ReadMe-tiedostoon, mikä on tavanomainen paikka kehittäjien väliselle kommunikoinnille. Lisäksi päivitin asiakirjatyypityksen prosessidokumentaatioon eri ympäristöjen päivitysohjeet. Nämä olivat helposti ymmärrettävät muutokset, joten en tehnyt erillistä tarkastuskierrosta.

Tiistai 26.3

Päivän tavoitteena on implementoida PDF-tiedostojen puhdistus rajapintojen ja käyttöliittymän kautta saapuviin PDF-tiedostoihin, joihin lisätään laadunvarmistus yksikkötestien avulla. Haasteena ilmeni toteutettaessa se, että osassa yksikkötesteissä oli käytetty Mock-implementaatiota oletuksena, eli niiden vastaus oli tekaistu. Tämä aiheutti hankalasti selvitettäviä virheitä implementaation toteutuksessa. Yksikkötesteissä käytetään usein oikoteitä, mikäli kyseinen toiminnallisuus ei ole testattavana. Yllättävää oli, että mockaukset olivat oletuksena päällä, ilman erillistä mainintaa. Olisin saanut sen selville, jos olisin selvittänyt testipalvelun toteutuksen tarkasti, mutta ehdin kysyä kollegalta ennen sitä. Päätin dokumentoida nykyisen oletusasetuksen testien ReadMe-tiedostoon.

Keskiviikko 27.3

Päivän tehtäväkseni otin projektin tyypitysten parantamisen. Tämä toteutetaan, jotta ohjelmiston funktioiden käyttämät tietorakenteet saadaan kuvattua koodissa paremmin. Tämä selkeyttää ohjelmistomme luettavuutta. Samalla odotamme sidosryhmiemme uuden toiminnallisuuden, suorapostituksen päälle kytkemistä. Olimme toteuttaneet lähetystoiminnallisuuden etukäteen ja siten jouduimme hieman odottamaan sidosryhmämme toteutusta, jotta palvelu toimii halutulla tavalla. Päivän tehtäväkseni otin pienempiä toteutettavia kokonaisuuksia, käyttöliittymäparannuksia ja muutostoiveita. Muutokset oli helppo toteuttaa, eivätkä ne vaatineet muutoksia testeihin, joten ne olivat vähän kuormittavia.

Alan lukemaan enemmän henkilökohtaisesta tietämyksenhallinnasta töiden jälkeen, eritoten Zettelkasten -menetelmästä. Tähän asti minulla on ollut vain pinnallinen tietämys aiheesta ja olisi siten hyvä kuulla kokeneempien mielipiteitä. Hyvin nopeasti löysinkin useamman yhteisön ja lähteen, missä aihetta käsitellään ja analysoidaan, jotka avaan tarkemmin viikkoanalyysissä.

Torstai 28.3

Päivän tavoitteena toteuttaa asiakirjatyyppipäivitys, mikä on rutiinitoiminto. Olemme ehdottaneet asiakkaalle, että tekisimme vastedes päivityksiä useammin mutta ne olisivat pienempiä. Tämä on mielestäni hyvä ratkaisu, sillä pienemmät päivitykset ovat helpompi käsitellä, ja tämä puolestaan pienentää inhimillisten virheiden riskejä. Lisäksi pienemmät päivitykset on helpompi korjata virheiden sattuessa. Aiemmat päivitykset ovat olleet suurempia ja niistä ilmenneet virheet ja tarkastuskierrokset ovat olleet kuormittavia ohjelmoijalle, tämä ei ole kustannustehokasta käyttöä konsulttityölle.

Toisena tavoitteena on suunnitella ja valmistella kuormitustestejä ohjelmistollemme. Niiden avulla saamme paremmin selville, että miten järjestelmämme skaalaus oikeastaan toimii. Tästä on vain valistunut arvaus, sillä vaikka tiedämme infrastruktuurikoodin tasolla missä kohtaa Kubernetes-palvelua ohjeistetaan skaalaaman resursseja enemmän, emme kuitenkaan voi olla varmoja, että resurssit skaalataan sillä vauhdilla mitä toivomme. En ehtinyt päästä itse kuormitustestaukseen mutta käsittelen sen mahdollisia hyötyjä viikkoanalyysissä tarkemmin.

Lukupiirissä kävimme läpi dokumentaation julkaisuprosessia, mukaan lukien mitä kanavia meillä on käytössä, miten päivitykset saavuttavat haluamamme yleisön ja miten julkaistu dokumentaatio pysyy ajan tasalla. Monet lähdemateriaalissa esitetyt ehdotukset eivät oikein soveltuneet meidän kontekstiimme, vaan näyttivät soveltuvat enemmän tuotteita kehittäville yrityksille tai palveluntarjoajille. Tämä oli kuitenkin hyvä kontrasti siihen mitä työssä kohtaa päivittäin ja tarjosi perspektiiviä arkityöhön.

Perjantai 29.3

Pitkäperjantai

Zettelkasten-menetelmä avattuna

Tutkittuani enemmän mitä kokeneemmat käyttäjät kertovat Zettelkasten-menetelmän käytöstä, olen muodostanut kattavamman ymmärryksen aiheesta ja todennut, että sen tuomat hyödyt ovat päämääräisesti suuremman työnteon takana ja tätä työtä kuuluisi jatkaa pitkäjänteisesti. Zettelkasten-menetelmä vaatii toimiakseen kategorisointia, joka vaatii kirjoittajalta lisätyötä ajatusten linkittämisen ja muistiinpanoja yhdistävien rakenteiden avulla. Kategorisointi tehdään, jotta säilytetään muistiinpanojen laajempi konteksti ja luettavuus. Organisoituna muistiinpanoihin voidaan palata samanlaisessa kontekstissa kuin se alun perin ajateltiin. (Arcrenas 2021, Atlassian, Sascha).

Olen käyttänyt Obisidiania muistiinpanojen kirjaamiseen lukupiirien keskusteluista, siihen se soveltuu hyvin mutta huomaan, että tekemäni muistiinpanot ovat liian epämääräisiä. Perusteellisempaan Zettelkasten-menetelmään kuuluu myös ajatusten kirjaaminen tai liittäminen aiemmin kirjoitettuun muistiinpanoon, mitä Luhmann ehdottaa käännöksen mukaan (Luhmann, 1981). Esimerkkinä hyvästä yhteysketjusta olisi yhdistää lukupiirissä käsittelemämme kirjan luku ajatukseen, joka käsittelyn aikana syntyy, vaikkapa flow-kaaviot ja oma projektimme.

Täydellisenä esimerkkinä asiayhteyksistä: Aihe ”ilmastonmuutos” voitaisiin yhdistää uusiutuvaan energiaan, joka puolestaan voidaan yhdistää vesivoimaan tai tuulivoimaan. Näin muodostuu verkkomainen rakenne käsiteltävän aiheen ympärille.

Zettelkasten-menetelmän perusidea esitellään seuraavasti:

- Zettelkasten on *hypertekstuaalinen*, eli jokainen muistiinpano on linkitettävissä tai on linkitettynä johonkin olemassa olevaan ajatukseen tai muistiinpanoon.
- Tehdyt muistiinpanot ovat *atomisia*, eli ne käsittävät vain yhden muistiinpanon tai ajatuksen.
- Kokonaisuus mikä muodostuu, on *henkilökohtainen*, eli ajatuksia mitä kirjoitetaan ylös ei tule sensuroida tai yrittää kohdistaa suurempaa yleisöä varten.

Tavoitteena on muodostaa tietovarasto, joka on mukautettu tekijän ajatusmalleihin ja tapaan ymmärtää ja jäsentää tietoa. Huomaan, että on olemassa dokumentaation kirjoittamisen tekniikoita, joita voi soveltaa tähän kuten yleisön tunnistaminen ja käyttäjätarinat. Erikoinen kirjoittamisen muoto tuntuu olevan vielä haasteellinen käsite, sillä pituus ja ympäröivä konteksti missä päädytään muistiinpanon muodostamiseen, tulisi myös kirjoittaa muistiin. (Atlassian, Sascha)

Zettelkasten-menetelmän käytössä erotaan dokumentaation tuottamisesta merkittävästi, esimerkiksi Zettelkasten-menetelmän avulla tuotetuissa muistiinpanoissa on tärkeää kirjoittaa ajatus tai kuvata konsepti omin sanoin, ei suuremman yleisön ymmärrettäväksi, koska usein ainoa arvioija on käyttäjä itse. Samankaltaisuuksia löytyy myös. Muistiinpanoja on ylläpidettävä ja niille on annettava ymmärrettävä konteksti, jotta ne voivat olla hyödyllisiä.

Huomaan omissa muistiinpanoissani olevan hyödyttömiä, kontekstittomia muistilappuja, joita ei ole linkitetty mihinkään suurempaan käsitteeseen. Tavoitteena olisi nyt luoda näille kontekstia ja yhdistää ne ajatuksiin ja prosesseihin. Irrallista tietoa on hankala hyödyntää ilman konkreettisia esimerkkejä ja sovelluksia. Etenkin pitemmän ajan kuluttua muistot alkuperäisestä ajatuksesta ovat hataria.

Zettelkasten-menetelmä ei siis sovi yrityksen tai organisaation tasolla tietämyksenhallinnan työkaluksi. Sitä on ajateltu käytettävän henkilökohtaisena tietämyksenhallinnan työkaluna tukemaan ajattelua, kirjoittamista, tiedonluontia, jäsentelyä ja tiedonhakua.

Kuormitustestaus

Tyypillisesti kuormitustestaukseen käytetään työkaluja, kuten Locust tai JMeter-testiautomaatiotyökaluilla. Meidän tarpeisiimme ne eivät sovellu, sillä tiedämme, että pullonkaulana on ohjelmistomme prosessointikapasiteetti ja täten voimme keskittyä testaamaan pelkästään pitkäkestoisia prosesseja, kunnes ohjelmiston kipuraja saavutetaan.

Tiedämme infrastruktuurikoodistamme skaalausrajat, eli missä resurssikäyttöasteessa rinnakkaisia prosessoreja lisätään käyttöön. Mitä emme tiedä on, että kuinka montaa rinnakkaista prosessointityötä se kykenee ylläpitämään. Kuormitustestimittausten kautta pystymme kartoittamaan järjestelmämme niin, että tiedetään miten kauan tai nopeasti sovelluksen pitäisi toimia milläkin käyttöasteella ja miten se vaikuttaa muuhun ohjelmiston toimintaan. Tämä kyseinen parannus voidaan siten kirjata ylös osaksi dokumentaatiota ja pystymme näin seuraamaan muutoksiemme vaikutusta ohjelmiston toimintoon pitemmällä aikavälillä.

Parempi metriikka auttaa tulevissa suunnitelmissa ja päätöksissä, sillä pystymme mittaamaan ja arvioimaan muutosten aiheuttamia kustannuksia paremmin. Esimerkiksi prosessi, joka hidastuu 0,2 sekuntia, ei kuulosta paljolta mutta mikäli se toistuu 100 000 kertaa päivässä, prosessointiaikaa onkin kulunut 5,5 tuntia enemmän. Tämä voi vaatia enemmän ohjelmallista prosessointia, mikä puolestaan tarkoittaa suurempia kustannuksia asiakkaalle. Tämä on merkittävä arvo mitata, minkä tärkeys mielestäni tulee tiedostaa yleisessä dokumentaatiossa ja seurata pitemmällä aikavälillä.

3.6 Seurantaviikko 6

Viime viikolla oli tarkoituksena alkaa kehittämään kuormitustestejä ja keräämään tilastoa SQL-tietokannasta, sillä olemme lisänneet uusia aikaleimoja ohjelmistoprosessiemme eri osiin. Tavoitteena tälle viikolle olisi saada metriikkaa, jonka perusteella voisimme tehdä infrastruktuurikoodisäätöjä.

Maanantai 01.04

Pääsiäispäivä ei merkintöjä

Tiistai 02.04

Päivän tehtävänä on kuormitustestien alustaminen ja kirjoittaminen, tämä sujuu hieman hitaalla tahdilla, sillä valitsin toteutuskieleksi Bashin, sillä sen avulla voimme hyödyntää Linux järjestelmän ohjelmia. Tämä osoittautui hyväksi tilanteeksi kasvattaa muistiinpanokokoelmaani, jossa liitin kokeiluni ja testini osaksi Bash-skripti kokoelmaani.

Kiinnostavampaa oli kollegani tekemä tilastojen tutkiminen prosessien SQL-rivien aikaleimojen perusteella, jonka toteutimme osin parikoodauksena. Tilastoinnin perusteella pystyimme selvittämään miten kauan yksittäiset operaatiot kestävät ja kuinka kauan ne joutuvat odottamaan. Tällä tavoin saamme lisäarvoa tuotannon datasta ja yhdistettynä kuormitustesteihin näillä pystytään mittaamaan koodimuutosten vaikutusta kokonaisuuteen.

Keskiviikko 03.04

Tavoitteenani on saattaa kuormitustestaus käyttökelpoiseen kuntoon jatkaen edellisen päivän työstä.

Testitulokset eivät toteutuneet kuten ajattelin. Synkroniset kutsut eivät kuormita järjestelmäämme toivotulla tavalla, tarvitaan paljon merkittävämpi määrä rajapintakutsuja jonoon. Täytyy selvittää niiden lopullinen tila mikä simuloi kutsujen ’ruuhkautumista’ paremmin. Ruuhkautumisen avulla pystymme seuraamaan miten Kubernetes-ohjelma, käyttämämme kontinhallintajärjestelmä, luo uusia resursseja ja sulkee niitä sitä mukaa kun resursseja ei tarvita. Näin pystymme arvioimaan kuinka voimme muuttaa infrastruktuurin koodia ohjelmistomme kannalta mahdollisimman tehokkaaksi.

Torstai 04.04

Kuormitustestit näyttävät tuottavan paremman tuloksen, jos kutsut tehdään rinnakkain, eikä sarjassa. Synkroninen Bashillä toteutettu implementaatio muutetaan asynkroniseksi TypeScript implementaatioksi. TypeScriptiin siirtymällä luovutaan Linux-ympäristön työkaluista, joita aiemmin käytettiin hyödyksi tulosten tulkintaan. Hyötyinä saavutetaan rajapintakutsujen ajaminen rinnakkain, jolloin on mahdollista asettaa suurempi kuorma järjestelmälle ja siten testata sen skaalautuvuutta joustavammin, mikä on merkittävämpää kuin statistiikan keräämisen vaivattomuus.

Perjantai 05.04

Päivän tehtävänä jatkettiin toteutusta asynkronisesta kuormitustestauksesta. Lukupiirissä käsittelimme miten mitata dokumentaation onnistumista ja tuloksia ja sattumoisin päädymme pohtimaan tilastotieteen metodeja ja kuinka niitä voidaan hyödyntää dokumentaation laadun mittaamisessa. Sama pätee myös koodin muutoksen merkittävyydessä, etenkin kun ohjelmoinnissa saman kutsun tai funktion ajaminen kerta toisensa jälkeen muodostaa tyypillisesti normaalijakauman, täten siihen voidaan soveltaa tilastollisia menetelmiä. Teoreettisesti dokumentaation palautteesta voitaisiin myös soveltaa tilastollisia menetelmiä, mikäli annettu palaute muodostaa normaalijakauman. Tämä asettaa tiettyjä vaatimuksia palautteelle: Palautteessa on oltava numeroarvosana ja se vaatii merkittävän määrän palautteita eri versioista, mielellään samoilta henkilöiltä mutta se ei ole välttämätöntä.

Viikkoanalyysi

Kuluneella viikolla aloitettiin prosessi ohjelmiston toimintojen mittaamiseen. Tämä on merkittävämpi askel sovelluksen kyvykkyyksien kartoituksessa ja tulee parantamaan kehitystiimin arviointikykyä, kun ohjelmistoon tehdään muutoksia. Viikolla käsiteltiin peittomatriisin ensimmäistä kehityskohtaa.

Palaute

Palaute on toisaalta aina hyvä asia vastaanottaa, oli se numeraalinen, sanallinen tai minimalistinen hyvä tai huono arvostelu. Tässä on hyvä pitää mielessä, että minkälaista palautetta käyttäjä on valmis antamaan ja kuinka auliisti. Monet eivät ole välttämättä halukkaita jättämään palautetta, jos palautteenantamisprosessi kestää liian kauan. On myös pidettävä mielessä, millaista on hyödyllinen palaute, jonka avulla kehittäjä voi parantaa dokumentaatiota, kuten Bhatti ja muut huomauttavat (2021, luku 9).

Kuormitustestien metriikka

Merkittävimpänä hyötynä kerätystä kuormitustestien metriikasta on mahdollisuus automatisoida se ja suorittaa vertailuanalyysiä eri ohjelmaversioiden välillä. Mitatut tulokset on mahdollista dokumentoida ja pitemmällä aikavälillä on mahdollista muodostaa tilastoja. Tilastoja vasten testaaminen helpottaa mahdollisten koodiregressioiden löytämistä, mikä ehkäisee kulujen kasvamista. Tilastojen säilyttäminen on tärkeätä myös tilanteissa missä halutaan optimoida ohjelmiston suorituskykyä ja tunnistaa pullonkauloja. Skaalautuvuus olisi mahdollista silloin määritellä reagoimaan paremmin äkillisiin kutsuryöppyihin. Hyvällä metriikalla olisi mahdollista tunnistaa myös ohjelmiston kohtia, joissa käytetään liikaa resursseja tai jos resursseja skaalataan liian nopeasti.

3.7 Seurantaviikko 7

Viikon tavoitteena on saattaa kuormitustestaukseen liittyvät tehtävät loppuun ja suunnitella prosessi, joka auttaa hyödyntämään kuormitustestauksen tuloksia. Prosessiin kuuluisi metriikan keräystiheys, varastointi, analysointi ja mahdollinen automatisaatio. Tämä prosessi tukisi laadunvarmistusta ja yhdessä asiakkaan määrittämien KPI-mittarien kanssa voidaan metriikan avulla arvioida eri muutosten vaikutusta ohjelmiston kehitykseen. Toteutettavat kuormitustestit ja niiden tulokset vaativat dokumentointia, täten prosessiin voidaan soveltaa dokumentoinnin tekniikoita.

Maanantai 08.04

Päivän tavoitteena on dokumentoida edellisellä viikolla toteutettu kuormitustestiprosessi. Tätä varten pyrittiin hälventämään mahdollista ’tiedon kirousta’ ja tiedustella kollegoilta etukäteen mitä he haluaisivat kirjoitettavan muistiin. Haluttiin dokumentoitavan ainakin olennaisimmat tiedot: Kuinka testit ajetaan, mitä parametrejä on asetettava, että testit voivat saada yhteyden järjestelmiin, millä infrastruktuurimuutoksilla testi on ajettava saadaksemme aitoja tuloksia, mitä testi tuottaa tulokseksi ja minne se tallennetaan? Kollegoiden etukäteen esittämiä kysymyksiä voidaan käyttää selventämään lukijan tavoitteita dokumentaatiota laadittaessa ja päästä siten tarkempaan lopputulokseen.

Dokumentaatio toteutettiin seuranta-ajan aikana opittujen käytäntöjen mukaisesti, lisäksi parannettiin dokumentaatiokokonaisuuden hahmotusta linkittämällä testien ReadMe -tiedostosta projektin Confluence-sivustolle, minne testitulokset on suunniteltu talletettavan ja Confluence-sivulta viitataan ohjelmiston ReadMe -tiedostoon. Dokumentaatiolähteiden linkityksellä pyritään mahdollistamaan dokumentaation helppo ylläpito, sillä kehittäjä näkee kaikki mahdolliset päivitettävät kohteet, kun yhtä kohtaa muutetaan.

Tiistai 09.04

Päivän tehtäväksi suunnitellaan ajaa kuormitustestejä ja toteuttaa korjauksia havaittuihin bugeihin ohjelmiston muissa osissa.

Kollegani koki kuormitustestin käyttöönotossa haasteita, mikä johti mahdollisuuteen hyödyntää ’friction-log’ metodia olemassa olevan dokumentaation parantamiseen. Parannukset koskivat asetettavia muuttujia, sekä sitä mistä käytetyt ohjelmistosalaisuudet ja testitilit löydetään pilviympäristössä.

Infrastruktuurikoodin hallinta osoittautui myös haasteelliseksi. Testit eivät ole hyödyllisiä, ellei niitä suoriteta tuotantoympäristön kaltaista infrastruktuuria vasten. Oletusasetuksena testiympäristössä ei ole kustannussyitten takia käytössä samoja resursseja. On siis tarpeen tehdä infrastruktuurikoodimuutos, että testejä on järkevää ajaa. Lisäksi testien ajon jälkeen infrastruktuurikoodimuutos olisi peruttava. Näillä rajoitteilla testejä on hankala automatisoida, mikä puolestaan rajoittaa testien käytettävyyttä. Korkeatasoisempana tavoitteena on testien automaattinen ajo ja niiden tuottamien tuloksien vaivaton tai automaattinen dokumentointi.

Keskiviikko 10.04

Päivän tavoitteena on saada ensimmäiset kuormitustestit toteutettua ja niiden tulokset kirjattua dokumentaatioon. Ensimmäisen onnistuneen ajon jälkeen on mahdollista tehdä rajauksia sovelluksen skaalautuvuuden tavoitteista. Kuormitustestit ajettiin testiympäristössä mutta ennen sitä asetettiin testiympäristöön tuotantoympäristöä vastaavat asetukset, jotta skaalautuvuus ja prosessointiteho eivät anna harhaanjohtavia tuloksia.

Testikuormaksi valittiin kymmenen rinnakkain ajettavaa PDF-luomisprosessia, joita ajetaan 30 minuutin ajan. Prosessien lukumäärä, aloitus ja prosessointiaika tallennetaan väliaikaiseen tulostiedostoon testin edetessä ja lopuksi viedään projektin Confluence-sivulle. Testiä ajettaessa pyrittiin välttämään muiden sidosryhmien häiritsemistä ajamalla ne päivän päätteeksi. Kuormitustestauksen aiheuttamat jonot haittaisivat muuten muuta testiympäristössä toteutettavaa toimintaa. Ongelmalliseksi osoittautui kuormitustestauksen pysähtyminen, kun ajettava tietokone siirtyi lepotilaan tai näyttö lukittiin. Nämä havainnot dokumentoitiin ja kuormitustestien onnistuttua, palautettiin testiympäristö alkuperäisiin asetuksiin.

Torstai 11.04

Päivän tavoitteena on jatkaa ohjelmiston tyypityksen parantamista ja tutustua siihen, miten kerättyä dataa voitaisiin hyödyntää paremmin. Amazon palveluista löytyy valmiina ratkaisuna Amazon

Managed Grafana-tuote. Tuote on tehty Grafana-monitorointipalvelusta, joka on avointa lähdekoodia. Nopean arvion ja kollegoiden kanssa keskustelun perusteella päätettiin, että saadaan suurempia hyötyjä, vasta kun kuormitustestauksesta on kerätty enemmän dataa.

Perjantai 12.04

Päivän aikana ei ollut juurikaan dokumentaatiokeskeisiä tehtäviä. Dokumentointia käsiteltiin kuitenkin viikoittaisessa lukupiirissä, eritoten dokumentaation ylläpidon saralla. Luvussa avataan jo aiemmin havaittuja ongelmia, kuten vanhentuneen dokumentaation haitallisuutta. Kiinnostavana aihenostona ilmeni vanhentuneen dokumentaation rinnastaminen ohjelmistobugiin ja sen käsittely ohjelmistovirheenä. Perustelu sille on sama, se vaikuttaa ohjelmiston käyttöön haitallisesti, se on korjattavissa ja priorisoitavissa (Bhatti ym, 2021, luku 11). Tämä edellyttäisi dokumentaatiovirheiden seurantaa projektinhallinnassa ja vian kriittisyyden perusteella voidaan määritellä, kannattaako virhe korjata heti vai myöhemmin.

Viikkoanalyysi

Viikolla ajettiin onnistuneesti aiemmin suunniteltuja ja toteutettuja kuormitustestejä ja niiden tulokset dokumentoitiin. Tuloksien karttuessa projektin kehityssuuntaa pystytään paremmin seuraamaan ja estämään regressioiden livahtaminen tuotantoon.

Kuormitustestien dokumentointiprosessi

Toteutettu dokumentaatio on ensimmäinen, joka on luotu seurantajakson aikana saatujen oppien pohjalta. Siinä kuvataan kuormitustestien tavoite yksittäisenä operaationa sekä kokonaisuutena, sekä sen esittelyssä priorisoidaan kuormitustestien tuloksia ja niiden ajamista. Dokumentaatiossa myös mainitaan tämänhetkiset puutteet ja haasteet: Kuormitustestit eivät ole automaattisesti ajettavia ja ne ovat riippuvaisia kehittäjän työlaitteesta. Testiympäristön infrastruktuurikoodi on muutettava tuotantoympäristöä vastaavaksi oikeanlaisia tuloksia varten. Toistaiseksi tulosten dokumentointi on manuaalinen operaatio, ideaalitilanteessa se ei kuormittaisi kehittäjää. Dokumentaatiota päivitetään iteratiivisesti muun ohjelmistokehityksen ohessa.

Kuormitustesteihin jäi selviä kehittämisen kohteita, jotka vaativat pilvipalveluiden käyttöönottoa suuremmalla skaalalla. Toistaiseksi projektissa on hyväksytty, että lisäkehitys odottaa parempaa ajoitusta. Prosessimetriikan kerääminen toisaalta mahdollistaa ohjelmiston KPI-mittarien muodostamisen tarkemmalla tasolla, minkä avulla kehittäjät voivat asettaa ohjelmiston suorituksen alarajat ja tavoitteet.

Virheet ja epäonnistuneet prosessit ovat myös kiinnostavaa dataa, jota kuormitustestit on asetettu keräämään. Havaittujen virheiden avulla voidaan nähdä mitkä prosesseja alkavat pettämään ylikuormitustilanteessa ja mitä korjauksia kehittäjien kannattaa ennakoivasti toteuttaa. Esimerkiksi ohjelmiston asynkronisia prosesseja yritetään uudelleen tietyn aikarajan jälkeen. Kuormitustestit muodostivat niin suuren prosessijonon, että uudelleen yrittämisen aikaraja umpeutui yksittäisille testeille ennen kuin prosessi oli ehtinyt valmistua. Tätä ei ollut tapahtunut tuotannossa mutta testien avulla bugi saatiin havaittua ennen kuin siitä ilmeni ongelma tuotantokäytössä.

Viikon aikana käsiteltävät asiat keskittyivät päämääräisesti dokumentointiin ja sen tuottamiseen. Muutama ohjelmiston ominaisuus, joka ilmeni kuormitustestaamisen aikana, lisättiin muistiinpanoihin henkilökohtaisen tietämyksenhallinnan kehittämiseksi.

3.8 Seurantaviikko 8

Viikon tavoitteena on jatkaa projektin olemassa olevan dokumentaation parantamista ja tarkastaa dokumentaation tila. Tarkastuksessa katsastettaisiin, onko dokumentaatio vanhentunut tai tarpeellista, sillä edellisviikoilla projektin ominaisuudet ovat muuttuneet, mikä on mahdollisesti tehnyt osasta dokumentaatiota tarpeetonta.

Maanantai 15.04

Päivän tehtävänä oli laajentaa suorapostitusratkaisua. Laajennus mahdollistaisi lisätittelien ja tarkemman osoitteen asettamisen postituskohteille. Dokumentoitavia osioita ei ollut, sillä muutokset koskivat vain ohjelman käyttämiä rajapintoja ja käyttöliittymiä, joita käytämme, ei rajapintoja joita tarjoamme.

Tiistai 16.04

Päivän tavoitteena oli tehdä kattavampia testejä edellisen päivän laajennoksen pohjalta. Kiinnostavana nostona: Tuotantoympäristössä ilmeni bugi, joka ilmeisesti syntyi, kun ohjelmistossa siirryttiin uusimpaan Ghostscript-ohjelman versioon tai aiemmassa versiopäivityksessä. Bugi liittyy PDF-tiedostojen automaattiseen kääntämiseen ja sen selvittäminen suhteessa ongelman vaikutukseen on hyvin kallista. Ongelman selvittäminen vaatisi kattavaa tietoa Ghostscriptistä ja sen toiminnasta tai merkittävää dokumentaation tutkimista, joka on suuritöinen urakka, sillä Ghostscriptin dokumentaatio on kattava ja laaja (Artifex, 2023). On todennäköistä, että ongelman ratkaisu löytyy Ghostscriptin merkittävän kokoisesta dokumentaatiosta mutta sen etsiminen kävisi niin kalliiksi, että toteutettu korjaus ei maksaisi itseään takaisin. Ongelma pyritään kiertämään jollain muulla keinolla, jos tämä ei onnistu, niin puute dokumentoidaan ja korjaus priorisoidaan asiakkaan toivomalla tavalla.

Keskiviikko 17.04

Päivän tehtävinä on koodin refaktorointia ja koodikatselmointia. Katselmoitavana oli koodia, joka oli poistumassa ohjelmistosta. Tämä osoittautui hyväksi tilanteeksi tarkistaa vaikuttavatko muutokset myös dokumentaatioon. Tällöin päästään merkitsemään dokumentaatioita vanhentuneeksi, kuten lähdekirjallisuudessa neuvotaan (Bhatti ja muut, luku 11). Tärkeänä nostona on, ettei dokumentaatiosta luovuta vaivihkaa, vaan viestitään sidosryhmille selkeästi, milloin mistäkin ohjelmiston osasta luovutaan. Projektimme kokoluokassa tämä ei liene suuri ongelma, sillä suurin osa aktiivista käyttäjistämme ovat tiedossa ja voimme viestiä heille suoraan.

Torstai 18.04

Päivän tavoitteena on ohjelman käyttämien viestien tyypitysten parantaminen, tällä hetkellä funktioiden datan kulku on epäselvä ja implementaatio nojaa vahvasti TypeScriptin rakenteelliseen tyypitykseen (TypeScript, 2024). Tämä on toimiva ja täysin TypeScriptin käyttötarkoituksen mukainen mutta olisi parempi käytäntö, jos ohjelmistossa olisi selkeästi nimetyt tietotyypit. Työn ohella kehitettiin myös henkilökohtaista tietämyksenhallintaa TypeScriptin ominaisuuksista lisäämällä ne alati kasvavaan muistiinpanoverkostoon.

Perjantai 19.04

Lukupiirissä käytiin läpi metodologioita, joilla pidetään dokumentaatiota ajan tasalla. Yksi varteenotettava keino, joka voisi toimia projektissamme, olisi asettaa jokaiselle dokumentaatiosivulle oma vastuuhenkilö. Henkilön tulisi päivittää tai ainakin tarkastaa sivun ajankohtaisuus aina, kun kyseistä dokumentaatiosivua koskevaa koodia päivitetään. Tätä varten on mahdollista tehdä työkaluja, jotka hälyttäisivät vastuuhenkilöä automaattisesti koodin muuttuessa, mutta projektissamme tämä ei ole tarpeen, sillä pieni tiimimme tarkastaa kaikki muutokset koodikatselmointien avulla. Tämä on suhteellisen vaivaton rutiiniprosessi, mikäli koodimuutokset eivät ole merkittäviä dokumentaation kannalta. Mikäli ilmenee paljon dokumentaatiopäivityksiä, lienee paras tehdä niistä oma työtehtävänsä.

Lisäksi toteutettiin tiimin kesken projektiauditoinnin katselmointi. Tämä on eräänlainen dokumentaation tyyppi, jolla seurataan projektille annettujen auditointikriteerien täyttymistä ja niiden etenemistä.

Viikkoanalyysi

Kuluneen viikon aikana toteutettiin dokumentaation ylläpitoon liittyviä toimintoja, jotka ovat merkityksellisiä ohjelmiston dokumentoinnin kannalta. Tällä varmistettiin, että tuotettu dokumentaatio ei ole vanhentunutta eikä haitallista. Menetelmiä käydään läpi yksityiskohtaisesti aihepiireittäin.

Dokumentaation tarkastus

Ylläpitovaihe tarjoaa mahdollisuuden tarkastella luodun dokumentaation ylläpitoprosessia. Ensiksi tarkastetaan kuvaako tuotettu dokumentaatio nykyhetken toteutusta, tässä vaiheessa mahdolliset muutokset on helppo löytää ja korjata. Vaikka toteutuksen yksityiskohdat ovat voineet muuttua, niitä ei tarvitse erikseen mainita dokumentaatiossa, mikäli se ei muuta ominaisuuden käyttöä tai tuloksia. Toinen harkinnanvarainen lisä dokumentaatioon voisi olla ominaisuuden syy ja tarkoitus. Koodista harvoin ilmenee mistä syystä koodi on tehty, sillä se on projektinhallinnan ja asiakkaan sisäisen ohjauksen huoli. Tästä huolimatta, voi olla hyödyllistä lisätä dokumentaatioon konteksti ja syy prosesseille, mikäli se auttaa ohjelmiston kokonaiskuvan hahmottamisessa.

Jos dokumentaatiossa on rakenteellisia puutteita, esimerkiksi, siitä puuttuu flow-kaavioita tai niistä etsittävä tieto on hankala etsiä, on ne korjattava. Nämä korjaukset saattavat olla kuitenkin suhteellisen suuritöisiä, eivätkä ole koodimuutosten yhteydessä tehtäviä rutiinitarkastuksia, joten tällaiset saattaa olla parempi suorittaa omina tehtävinään projektihallinnan kautta.

Koodimuutosten yhteydessä lähdekirjallisuudessa ehdotetaan dokumentaation ylläpidon automatisointia osittain. Käytetyt työkalut riippuvat julkaistavasta dokumentaatiosta ja missä julkaisu tehdään. Automaatiotyökalut on myös syytä valita tarkkaan, jotta dokumentaation laatu ei kärsi pitemmällä aikavälillä. Dokumentaatiota voidaan generoida, esimerkiksi toisella seurantaviikolla esitellyllä Swagger-työkalulla, tällä tavoin säästetään kehittäjän työaikaa, jos generoitua dokumentaatiota uudelleen generoidaan rajapinnan päivityksien yhteydessä. Toinen suositeltu prosessi on dokumentaation tuoreuden seuraaminen, jonka avulla lukija pystyy arvioimaan, onko dokumentaatio mahdollisesti vanhentunutta. (Bhatti ja muut, 2021, liitteet)

Ilmeisen tärkeä prosessi on myös vastuuhenkilöiden määrittäminen dokumentaatiolle. Tällä vältetään epäselvyys dokumentaation päivittäjästä ja samalla nimetään yhteyshenkilö, jolle dokumentaation lukija voi ilmoittaa epäselvyyksistä ja virheistä. Jaettu vastuu dokumentaation päivittämisessä nostaa riskiä sille, että dokumentaatio jää täysin ilman päivityksiä.

Yhtenäinen dokumentaatiomalli

Lähdekirjallisuudessa kehotetaan myös käyttämään hyödyksi mallipohjia. Mallipohjilla voidaan helpottaa uuden dokumentaation tuottamista ja luettavuutta (Lakatos 2023, luku 4). Mallipohjien omaksuminen ja yhteisen dokumentaatiotyylin valinnat ja valintojen perustelut on hyödyllistä dokumentoida myös tulevia kehittäjiä varten, jotta hyvät käytännöt eivät unohdu tekijöiden vaihtuessa. Tällaisen käytännön ylläpitämien muodostaa eräänlaisen metadokumentaation projektille. Tätä metadokumentaatiota voidaan sitten hyödyntää sidosryhmien kanssa kommunikoinnissa ja ohjelmistosuunnittelussa.

Lähdemateriaalissa esitellään useita, yleisessä käytössä olevia mallipohjia, joiden käyttöönottoa tai soveltamista harkitaan projektissa. (Lakatos 2023, liitteet). Näiden pohjien kuvaaminen ja lisääminen osaksi henkilökohtaista tietovarastoa sekä soveltaminen dokumentaation tuottamisessa lienee riittävä ratkaisu toistaiseksi, sillä projektin koko ei edellytä lukuisien henkilöiden koordinointia dokumentaation tuottamisessa.

4 Pohdinta

Seuranta-aikana toteutettu osaamisen kehitys on tuottanut tuloksia useammalla osa-alueella, mikä on johtanut korkeatasoisempaan osaamiseen projektityössä. Osaamisen kehittymisellä on myös vaikutusta projektin ulkopuolisessa ammatillisessa osaamisessa esimerkiksi myyntityössä.

Seuranta-ajan alussa ei ollut olemassa järjestäytynyttä prosessia, jonka mukaan dokumentaatiota tai henkilökohtaista tietämyksenhallintaa toteutettaisiin. Seuranta-ajan aikana on tutustuttu merkittävään määrään prosesseja ja tekniikoita, joiden hyötyjä on analysoitu ja otettu käyttöön. Henkilökohtaisen tietämyksenhallinnan tekniikoihin on tutustuttu ja Zettelkasten-menetelmä on otettu käyttöön henkilökohtaisen tietovaraston kartuttamiseen. Laajojen kielimallien hyödyntämisessä saavutettiin vain vaatimattomia tuloksia, sillä niiden kattavampi hyödyntäminen olisi vienyt muilta osa-alueilta huomiota, joten tämän aiheen käsittelyä vähennettiin.

Dokumentaation tuottamiseen ja arviointiin liittyviä menetelmiä on otettu käyttöön projektissa onnistuneesti ja dokumentaation ylläpidon prosesseja on implementoitu päivittäiseen työhön. Esimerkkeinä onnistuneista menetelmien toteutuksista ovat friction-log, dokumentaation mallipohjien soveltaminen yhtenäistä dokumentaatiotulosta varten ja käyttäjähaastattelut dokumentaation luomisvaiheessa.

Henkilökohtaisen tietämyksenhallinnan menetelmänä harjoiteltu Zettelkasten-menetelmä on otettu käyttöön mutta siitä saadut hyödyt ovat vielä rajatut. Menetelmä vaikuttaa lupaavalta ja sen käyttöä jatketaan seuranta-ajan jälkeen. Lähdekirjallisuuden merkitys sekä lähteisiin viittaaminen muistiinpanoissani työn tukena on kasvanut merkittävästi seurantajakson aikana. Laajojen kielimallien hyödyntämistä on rajatusti kokeilu henkilökohtaisessa tietämyksenhallinnassa, esimerkiksi oikolukijana ja katselmoijana Zettelkasten-menetelmää käytettäessä.

Opinnäytteen kirjoittamisen aikana huomattiin useamman kerran, että ensimmäinen vedos yleensä kehittyy katselmointikierrosten myötä. Samanlainen ilmiö toistuu ohjelmoinnissa ja dokumentaation luomisessa. Käsitteiden ja ajatusten kirjoittaminen kokonaisiksi loogisiksi ajatuksiksi on auttanut hahmottamaan näitä paremmin. Tämä on siten korostanut itsereflektoinnin tärkeyttä ja oman työn uudelleen lukemista.

Toisena havaintona todettiin katselmoinnin tärkeys. Opinnäytetyötä kirjoittaessa on ollut tärkeätä vastaanottaa palautetta, jotta välttyy eräänlaiselta tiedon kiroukselta. Samanlainen ilmiö tuntuu toistuvan missä tahansa luovassa työssä, mukaan lukien dokumentaation luomisessa, ohjelmoinnissa ja muistiinpanojen tekemisessä.

Lähdekirjallisuuteen tutustuminen opinnäytetyön aikana on selkeyttänyt dokumentaation tavoitteellisuutta, havainnollistavat esimerkit ja yksityiskohtaiset prosessit ovat osoittaneet, kuinka tärkeätä dokumentaatio on kommunikoinnin kannalta. Dokumentaation laatiminen osoittautui monitasoiseksi ja luovaksi prosessiksi, jonka yllättävä kiinnostavuus auttoi ylläpitämään motivaatiota opinnäytetyön jatkamiseen.

Zettelkasten-menetelmän opettelu henkilökohtaisen tietämyksenhallinnan työkaluna oli myös kiinnostava. Erityisesti sen kokeilu pitkällä aikavälillä vaikuttaa projekilta, jolla voi saavuttaa merkittävää ammatillista ja henkilökohtaista kehittymistä. Oli kiinnostava huomata miten tiedon kirous vaikuttaa myös muistiinpanojen kirjoittamisessa, mikä johti myös aikaisempien muistiinpanojen katselmointiin ja iteroimiseen. Lisäksi muistiinpanot alettiin kirjottamaan tavalla, että henkilö, joka ei ole koskaan kuullut asiasta, ymmärtäisi sen. Tällä ehkäistään muistivirheitä ja kerrataan konsepti parempaa lopputulosta varten. Osittain tähän voidaan käyttää hyväksi laajoja kielimalleja tarkastajana. Esimerkiksi syöttämällä muistiinpano ja lisäten syötteen: ”Kuvaako annettu teksti konseptia ja mitä muutoksia voidaan tehdä lopputuloksen parantamiseksi?”

Dokumentaatiotyön analysointi itsenäisesti on hankalaa, sillä arviointikriteerit dokumentaatiolle ovat osittain tulkinnanvaraisia. Solitan kulttuuri rakentavan palautteen antamiselle on ollut kehittymisen kannalta hyvin tärkeätä. Kollegat ovat mielellään kertoneet kuinka luotua dokumentaatiota voi parantaa tai mitä he etsisivät dokumentaatiosta ja perustelevat sen johdonmukaisesti. Dokumentaation luomisen yhteydessä saatu kokemus on parantanut myös itsearviointikykyä sen laatimisen suhteen, mutta lähdekirjallisuudessa esiintyvät arviointikriteerit ovat olleet päämääräinen apuväline tämän arviointikyvyn kehittämisessä. (Carey ja muut, 2014, liitteet)

Lähdekirjallisuuden lukeminen on osoittautunut sopivaksi tavaksi käsitellä uusia menetelmiä ja tulee olemaan osana osaamisen kehittämistä. Zettelkasten-menetelmä tulee olemaan osana osaamisen kehittämisprosessia, sillä menetelmä toimii parhaiten pitkällä aikavälillä. Laajat kielimallit voivat olla osana osaamisen kehitystä mutta tällä hetkellä ne eivät ole tarpeeksi luotettavia, että niiden antamiin tietoihin voi uskoa tietoja tarkistamatta.

Ammatillisen kehittymisen mahdollisuudet ovat varsin laajat. Olisi mahdollista jatkaa käsiteltyjä teemoja syvemmällä tasolla sekä lisätä organisaation tietämyksenhallinnan kehitettäviin teemoihin, sillä se tukisi dokumentaatiota, tiedonjakoa ja kommunikaatiota paremmalla tasolla kuin opinnäytetyössä käsitelty Zettelkasten-menetelmä. Käytännönläheinen kehittämisen kohde olisi teknisen tai luovan kirjoittamisen harjoittelu.

Dokumentaation laadun kehittäminen käyttäjäpalautetilastojen kautta vaikuttaa kiinnostavalta, vaikka sen hyödyntäminen dokumentoinnissa ei ole kovin todennäköistä, sillä dokumentaation

käyttöasteet ovat varsin matalat kaikissa Solitan projekteissa ja siten metriikan keräys riittävän suurissa määrissä ei ole todennäköistä. Ohjatut koulutukset ja kurssit voivat olla tärkeä kehittymisen työkalu, sillä järjestelmällisessä koulutuksessa on selkeitä tavoitteita, joita seurata ja näitä voidaan käyttää virstanpylväinä kehityksen mittaamisessa.

Yhteenvetona seuranta-aikana tehty osaamisen kehittäminen onnistui dokumentaation laatimisen kehittämisen ja henkilökohtaisen tietämyksenhallinnan saralla. Laajojen kielimallien hyödyntäminen osoittautui odotettua suuremmaksi aiheeksi, joten sen prioriteettia laskettiin ja seuranta siirtyi tiiviimmin dokumentaatioon ja henkilökohtaiseen tietämyksenhallintaan. Kokonaisuutena seuranta-aikana toteutettu osaamisen kehittyminen on ollut palkitseva prosessi, jonka aikana on opittu merkittävästi ammatillisen kehittämisen kohteista, sekä myös opinnäytetyön kirjoitusprosessista.

Lähteet

Arcenas, J. A Beginner’s Guide to the Zettelkasten Method. Zenkit suite blog. Luettavissa: https://zenkit.com/en/blog/a-beginners-guide-to-the-zettelkasten-method/ Luettu: 01.04.2024

Artifex. Ghostscript and the PostScript Language. 2024. https://ghostscript.readthedocs.io/en/gs10.0.0/Language.html Luettu: 19.4.2024

Atlassian, Build your own second brain with the Zettelkasten method. Luettavissa: https://www.atlassian.com/blog/productivity/zettelkasten-method Luettu: 22.04.2024

Bhatti, J., Corleissen S., Lambourne, J., Nunez, D. ja Waterhouse H. Docs for Developers: An Engineer’s Field Guide to Technical Writing. Apress. New York City. E-Kirja. Luettu 21.02.2024.

Carey, M., McFadden Lanyi, M., Longo, D., Radzinski, E., Rouiller, S. ja Wilde E. Developing Quality Technical Information: A Handbook for Writers and Editors, Third Edition. IBM Press. Indiana. E-Kirja. Luettu: 2.03.2024

Diana Lakatos, , Crafting Docs for Success: An End-to-End Approach to Developer Documentation 2023, Apress, New York City. E-Kirja Luettu 16.04.2024

Elston, J., 09.10.2011, “What is the difference between 'layer of abstraction' and 'level of indirection'?” StackExchange. Luettavissa: https://softwareengineering.stackexchange.com/questions/111756/what-is-the-difference-between-layer-of-abstraction-and-level-of-indirection Luettu: 24.03.2024

IBM. What are AI hallucinations. Luettavissa: https://www.ibm.com/topics/ai-hallucinations. Luettu: 01.04.2024.

Luhmann, N. Manfred Kuhenin kääntämänä englanniksi. Communicating with Slip Boxes – alkuperäisesti: Kommunikation mit Zettelkästen. Luettavissa: https://zettelkasten.de/communications-with-zettelkastens/ Luettu: 22.04.2024

Sbai, A. 17.07.2023, “What's the difference between getByText vs findByText vs queryByText in testing-library?” StackOverflow. Luettavissa: https://stackoverflow.com/questions/76705164/whats-the-difference-between-getbytext-vs-findbytext-vs-querybytext-in-testing Luettu: 24.03.2024

Soares, L., Dodds, K. Testing Library About Queries. Luettavissa: https://testing-library.com/docs/queries/about Luettu: 29.02.2024.

Sascha 27.10.2020. Introduction to Zettelkasten. Zettelkasten. Luettavissa: https://zettelkasten.de/introduction/ Luettu: 22.04.2024

TypeScript. Type Compatibility. 2024. https://www.typescriptlang.org/docs/handbook/type-compatibility.html. Luettu: 20.4.2024