Haaga-Helia
ammattikorkeakoulu Oy

Ohjelmistokehityksen konsultointi finanssialalla

Aleksi Uusitalo

Opinnäytetyö
Tietojenkäsittelyn koulutusohjelma
2020

Haaga-Helia ammattikorkeakoulu Oy

Opinnäytetyö

28.4.2020

Tekijä(t)
Aleksi Uusitalo

Koulutusohjelma
Tietojenkäsittelyn koulutusohjelma

Opinnäytetyön otsikko
Ohjelmistokehityksen konsultointi finanssialalla

Sivu- ja liitesivumäärä
84 + 3

Opinnäytetyön otsikko englanniksi
Software consulting in finance industry

Ohjelmistokehityksen konsultointi on ala, jossa työskentelevien on kyettävä nopeasti omaksumaan uuden projektin työtehtäviin liittyviä uusia käsitteitä ja työskentelytapoja. Tässä opinnäytetyössä kuvaillaan finanssialan ohjelmistokehityksen parissa työskentelevän konsultin työn sisältöön liittyviä työskentelymalleja, esiin nousevia haasteita ja niiden ratkaisemista.

Opinnäytetyö on päiväkirjamuotoinen tutkielma ja sen tarkoituksena on avata konsultin työn sisältöä ja ohjelmistoprojektin etenemiseen liittyvää oman osaamisen kehittymistä. Työssä esitellään konsultin työympäristöä, työssä käytettäviä teknologioita ja yhteistyötapoja niin oman työnantajayrityksen kuin myös asiakasyrityksen vastuuhenkilöiden kanssa. Opinnäytetyön päiväkirjaosuus ajoittuu keväällä 2020 aikavälille 10.2. - 24.4.2020.

Työssä havainnollistetaan konsultin jatkuvaa henkilökohtaisen osaamisen kehittymistä ja päivittäisissä työtehtävissä esiin nousevien haasteiden ratkaisumalleja niin henkilökohtaisella tasolla kuin myös hyödyntäen projektiin liittyviä sidosryhmiä. Asetetut aikataulutavoitteet ovat projektin etenemisen kannalta tärkeää huomioida, koska projektin hidastuminen jollakin osa-alueella voi aiheuttaa viivästystä muille samassa projektissa työskenteleville.

Eräs tärkeimmistä havainnoista ilmenee ohjelmistokehittäjän vastuusta työn sisällön monipuolisessa hahmottamisessa ja kehittämisessä sekä merkityksellisen asiakaskokemuksen tuottamisessa. Oman työn menestyksellinen hoitaminen tuottaa tällöin lisäarvoa niin asiakasyritykselle kuin myös omalle työnantajayritykselle. Yritysmaailmassa tyytyväinen asiakas on aina tärkeä meriitti tulevissa asiakaskontakteissa.

Asiasanat
Konsultointi, ohjelmistokehitys, oppiminen, projektityöskentely

Sisällys

1 Johdanto

Tässä opinnäytetyössä tulen tarkastelemaan päivittäistä työnkuvaani finanssialalla ohjelmistokehityksen konsultin ammatin näkökulmasta. Opinnäytetyön seurantaviikot sijoittuvat aikavälille 10.2.2020-27.4.2020.

Työnantajani on konsulttiyritys Siili Solutions Oyj (jäljempänä Siili), jonka vahvuuksia ovat asiansa osaavat ammattilaiset. Siili tuottaa kokonaisvaltaisia ratkaisuja digitaalisiin tuotteisiin, palveluihin ja liiketoiminnan kehittämiseen. Siili on perustettu vuonna 2005 ja sillä on nykyään toimistoja Suomen lisäksi Puolassa, Saksassa ja Yhdysvalloissa. Vuonna 2012 Siili listautui Helsingin Pörssin First North -listalle (NASDAQ OMX NORDIC 2019). Siilin päätoimialana on it-konsultointi eli ohjelmistokehityksen asiantuntijoiden välittäminen asiakasyrityksille. Siilin suurimpiin asiakasyrityksiin kuuluvat myös Suomen suurimpiin yrityksiin kuten esimerkiksi Nokia, KONE ja Maanmittauslaitos. Konsulttien työtehtävät vaihtelevat ohjelmoinnista ja palvelumuotoilusta aina järjestelmäarkkitehtuurin ja agile coachin tehtäviin. Siilissä on ollut käynnissä suuri organisaatiomuutos, jonka takia yritys on erottanut tietyt liiketoimintayksiköt omiksi tytäryhtiöiksi. Näitä ovat muun muassa Siili_Auto, joka on erikoistunut pelkästään autoteollisuuteen ja Siili_one, joka välittää kaikista kokeneimpia asiantuntijoita yksittäisiin projekteihin (Siili Solutions Oyj 2018).

Siilin työllistää tällä hetkellä yli 700 työntekijää. Yrityksestä löytyy kokeneita tekijöitä koko kehitettävän tuotteen tai palvelun elinkaarelle ja tällä saavutetaan paras mahdollinen lopputulos asiakkaalle, kun samalla ymmärretään asiakkaiden tarpeita. Siili pyrkii organisaationa olemaan avoin ja tasainen, ilman suurempia byrokratian portaita. Jokainen liiketoimintayksikkö on itseohjautuva ja vastuussa omasta toiminnastaan. Tämä saavutetaan ottamalla enemmän vastuuta asiakkuuksien kehittämisestä, jotta asiakkaita voidaan palvella kokonaisvaltaisemmin tehokkaana kumppanina. Siilillä suurin osa työntekijöistä työskentelee asiakkaan tiloissa, jolloin projektin avoimuus näkyy asiakkaalle paremmin. Samalla ollaan lähempänä asiakkaan edustajia, jolloin myös päätöksenteko nopeutuu ja ongelmiin saadaan keksittyä ratkaisuja nopeammin.

Siili on tunnettu laadukkaasta koodista ja erinomaisten palvelujen rakentamisesta, millä mahdollistetaan hyvien asiakassuhteiden säilyvyys ja menestyksekäs liiketoiminta. Konsulttina työskennellessä on pyrittävä aina parhaaseen mahdolliseen tulokseen. Työskentelen itse ohjelmistokehityksen konsulttina ja tässä työssä pitää pystyä olemaan erittäin dynaaminen ja kyettävä mukautumaan tarpeen mukaan. Olen esimerkiksi käynyt Scrum Master -kurssin sekä suorittanut Scrum.orgin myöntämän Scrum Master -sertifikaatin, jotta voin tarvittaessa työskennellä myös siinä roolissa. Työssäni toimin full stack -

kehittäjänä eli työskentelen sekä frontendin, backendin, että devopsin puolella. Konsulttina työskennellessä projektien aikaväli voi vaihdella yleensä muutamasta kuukaudesta noin kahteen vuoteen. Tässä tosin saattaa olla poikkeuksia, jos projekti on ollut esimerkiksi kokeiluluontoinen konseptin testaus projekti, jota halutaan lähteä jatkokehittämään. Konsultin pitää pystyä oppimaan uutta nopeasti, sillä seuraavassa projektissa käytettävät teknologiat ja toimintatavat saattavat poiketa täysin aiemmin olleista projekteista.

1.1 Tietoperusta

Työtehtävissäni käytän tiedon lähteenä teoksia ja kursseja, jotka käsittelevät ohjelmointikieliä, ohjelmointikäytänteitä sekä eri teknologioita ja tekniikoita. Työpaikaltani löytyy kattava kirjasto ohjelmistokehitykseen liittyvää kirjallisuutta, jota pystyn hyödyntämään oman kehittymiseni kannalta sekä työn yhteydessä.

Ensimmäinen teos, joka mielestäni jokaisen ohjelmistokehittäjän tulisi lukea, on Robert C. Martinin (2009) kirjoittama Clean Code. Teos on julkaistu vuonna 2009 ja se jakautuu kolmeen osaan. Ensimmäinen osa käsittelee hyvän koodin periaatteita, kuvioita sekä käytäntöjä. Toinen osa käsittelee tapaustutkimuksia, kuinka huono koodipohja muutetaan tehokkaaksi ja hyväksi. Viimeinen osa käsittelee heuristiikkoja tapaustutkimuksia tehdessä. Kirja on jo yli 10 vuotta vanha, mutta sitä pidetään silti edelleen yhtenä ohjelmoinnin tärkeimpinä ja ajankohtaisimpana teoksena. Toinen alaan liittyvä tärkeä teos, jota käytän tässä työssä tietoperustana, on The Pragmatic Programmer: Your Journey to Mastery, 20th Anniversary Edition (Thomas & Hunt 2019). Kyseisestä kirjasta on tehty ensimmäinen painos vuonna 1999, mutta vuonna 2019 siitä otettiin uusittu painos, sillä käytänteet ovat hieman muuttuneet kahdessakymmenessä vuodessa. Kirjassa on runsaasti kuvauksia, kuinka hyvää koodia kirjoitetaan ja se on hyvä tietolähde alaa opiskeleville ja alalla työskenteleville edellä mainitun Clean Code -kirjan lisäksi.

1.2 Keskeiset käsitteet

Opinnäytetyö sisältää teknisiä termejä, lyhenteitä ja ammattisanastoa. Keskeisimmät käsitteet on avattu liitteessä (Liite 1).

1.3 Luottamuksellisuus

Tässä opinnäytetyössä käyttämieni henkilöiden, tuotteiden, tiimien ja yritysten nimet ovat kuvitteellisia luottamuksellisuuden takaamiseksi. Ainoastaan oman työnantajayritykseni nimeä Siili Solutions ei ole muutettu.

2 Lähtötilanteen kuvaus

2.1 Oman nykyisen työn analyysi

2.1.1 Työtehtävien kuvailu

Ohjelmistokehittäjänä työtehtäviini kuuluu pääasiassa ohjelmointi niillä teknologioilla, joita asiakkaalle kehitettävässä sovelluksessa ja yrityksen teknologisessa ympäristössä käytetään. Asiakasprojektissa kehitämme yrityksille tarkoitettua taloudenhallintasovellusta, jonka nimi olkoon **Kassis** tässä opinnäytetyössä. Kassis on uusi sovellus, jonka kehittämisen olemme aloittaneet täysin tyhjästä. Projektissa käytettävät pääteknologiat ja ohjelmointikielet ovat Typescript, React, Java, Jenkins, AWS ja Robot Framework. Näistä teknologioista Typescript ja Jenkins olivat minulle täysin uusia, eli niitä en ollut käyttänyt koskaan aiemmin.

Projektissa työskentelen nimikkeellä full stack -kehittäjä, eli kehitän sovellusta backendista pilvipalveluihin saakka. Tiimimme koostuu useista samalla nimikkeellä työskentelevistä kehittäjistä, mutta jokaisella on myös oma osa-alue, jonka hallitsee parhaiten. Itse työskentelen kaikista eniten frontendin parissa. Kehitän paljon käyttöliittymiä, teen niille logiikkaa, kuinka rajapinnoista palautuva data näytetään sekä pyrin saamaan kaikki komponentit ja elementit asettumaan sivulle CSS:n avulla kuten on suunniteltu.

Asiakas, jonka nimi olkoon tässä työssä **Pankki Oy**, osti Siililtä neljän hengen tiimin alun perin joulukuussa 2018, jolloin projekti virallisesti alkoi. Tällöin aloimme kehittämään asiakkaan legacy-järjestelmiä ja lisäämään sinne muutamia uusia ominaisuuksia, jonka jälkeen oli tarkoitus aloittaa Kassiksen teko. Legacy-järjestelmien kehittämisessä kuitenkin meni hieman odotettua pidempään, joten aloitimme Kassiksen kehittämisen täydellä teholla vasta marraskuussa 2019.

Konsulttina työskennellessä pitää pystyä olemaan eräänlainen kameleontti. Konsultin pitää pystyä olemaan dynaaminen ja valmis oppimaan uutta nopeasti, sillä teknologiat ja työtehtävät saattavat vaihtua nopeasti. Tämän asiakasprojektin tehtävissä se on tullut tutuksi, sillä työssäni vaaditaan vankkaa asiantuntemusta sekä taitoa oppia uusia teknologioita nopeasti. Alun perin legacy-järjestelmiä kehitettäessä lähes kaikki siinä käytettävät teknologiat olivat entuudestaan tuntemattomampia minulle. Näitä olivat mm. JSP, jQuery ja Java 6. Näistä olin käyttänyt ainoastaan Javaa, mutta Java 6 ja modernissa Javassa on aika suuria eroja metodien ja kirjastojen välillä, mikä teki sen käyttämisestä haastavampaa. Uusien teknologioiden määrä on siis ollut suhteellisen suuri koko projektin aikana.

Konsulttina työskennellessä vuorovaikutustaidot ovat myös olennaisessa osassa. Konsultin pitää pystyä olemaan hyvin oma-aloitteinen sekä tekemään omaa selvitystyötään. Päivät koostuvat monesti palavereista, jossa suunnitellaan tulevia töitä aina arkkitehtuuriselta tasolta käyttäjäkokemukseen ja on tärkeää, että jokainen tuo oman näkemyksensä esiin. Tämä korostuu varsinkin finanssialalla, joka on tarkasti säännelty ja jossa uusien toiminnallisuuksien vaatimusmäärittely saattaa olla monien sivujen mittainen.

Tässä projektissa olen oppinut paljon uusia teknologioita, ohjelmointikäytänteitä ja kuinka finanssi- ja pankkialan ohjelmistot yleisesti toimivat Suomessa. Tiimissä on nykyään minun lisäksi seitsemän muuta kehittäjää, joista kahdella on monen vuoden kokemus ohjelmistokehityksestä tällä alalla. He ovat hyvin avuliaita ja aina valmiita auttamaan, jos jokin asia tuntuu vaikealta. Pidämme tarvittaessa palaverin, jossa käymme läpi, mitä työtehtävät sisältävät ja määrittelemme, mitä niiden tekeminen vaatii. Näin pystymme yhdistämään tietämyksemme ja taitomme, jotta pääsemme parhaaseen mahdolliseen lopputulokseen.

2.1.2 Oma osaaminen suhteessa työtehtäviin

Kuten aiemmin olenkin maininnut, työtehtäväni vaatii laaja-alaista asiantuntijuutta ja monia taitoja, jotta tehtävässä pärjää kunnialla. Tässä opinnäytetyössä kuvattavan sovelluksen lanseeraus olisi tarkoitus suorittaa toukokuun lopussa, joten projektissa on myös aikataulullisia paineita. Kassiksen rakentamisessa mielenkiintoista on, että saamme rakentaa täysin alusta koko sovelluksen ja täten valita itse ne teknologiat, joita tulemme käyttämään. Tällöin meillä ei ole vanhaa koodipohjaa taakkana. Toisaalta haasteellista on, että rakennamme tosiaan kaiken tyhjästä. Kaikkia niitä integraatioita, mitä tarvitsemme, ei ole tehty eikä meillä ole myöskään mitään tietoa, kuinka vastaavan tyyppisiä toimintoja mahdollisesti muualla toteutettu. Uuden sovelluksen rakentaminen tuotantovalmiiksi noin puolen vuoden aikana vaatiikin tekijöiltä paljon.

Koen, että olen tehtävissäni tällä hetkellä aloittelevan toimijan ja taitavan suoriutujan välillä. Full stack-kehittäjänä teknologioiden skaala on niin suuri, että on mahdotonta osata kaikkia teknologioita täydellisesti. Ohjelmointitaitoni ovat kuitenkin sen verran hyvät, että pystyn kehittämään ja parantamaan sovellusta itsenäisesti. Palavereissa pystyn ottamaan kantaa arkkitehtuurillisiin päätöksiin ja suosittelemaan, mitä työkaluja ja kirjastoja voimme hyödyntää työssämme. Teen projektissa myös aktiivisesti koodikatselmointeja muiden kehittäjien pull requesteista versionhallintaan ja tarkkailen, että niissä tehdään myös luettavaa, siistiä ja koodikäytänteidemme mukaista koodia.

Tunnen tarvitsevani enemmän ohjausta ainoastaan backendin puolella, kun teen rajapintaintegraatioita. Koska pääpainoni on frontendissä, niin koen, että en ole vielä niin kokenut Java-kehittäjä. Kun on kyseessä web-palvelimen kehittäminen Springin avulla, voi minulla olla haasteita syntaksin tai Javan metodien kanssa.

2.1.3 Kehittyminen ammatillisesti

Aloitin työni Siilillä toukokuussa 2018, jolloin olin juuri saanut ensimmäisen kouluvuoteni päätökseen Haaga-Heliassa tietojenkäsittelyn koulutusohjelmassa. Taidoiltani olin silloin täysi aloittelija, sillä olin ehtinyt käymään vain ohjelmoinnin peruskurssin ja hieman harjoitellut yleisesti ohjelmointia vapaa-ajallani. Siilillä pääsin Mestari & Kisälli-ohjelmaan, jossa 2,5 kuukauden aikana meille opetettiin uusimmat teknologiat mitä asiakasprojekteissa käytetään ja kuinka konsultin töitä tehdään Siilillä. Ohjelman aikana teknologinen osaamiseni ja kuinka sovelluskehitystä oikeasti tehdään, syveni huomattavasti. Koin olevani edelleen taidoiltani juniortason koodaaja, mutta käytin myös kaiken vapaa-aikani erilaisten verkkokurssien ja opetusmateriaalien opiskeluun, minkä avulla pystyn kehittymään ohjelmoijana.

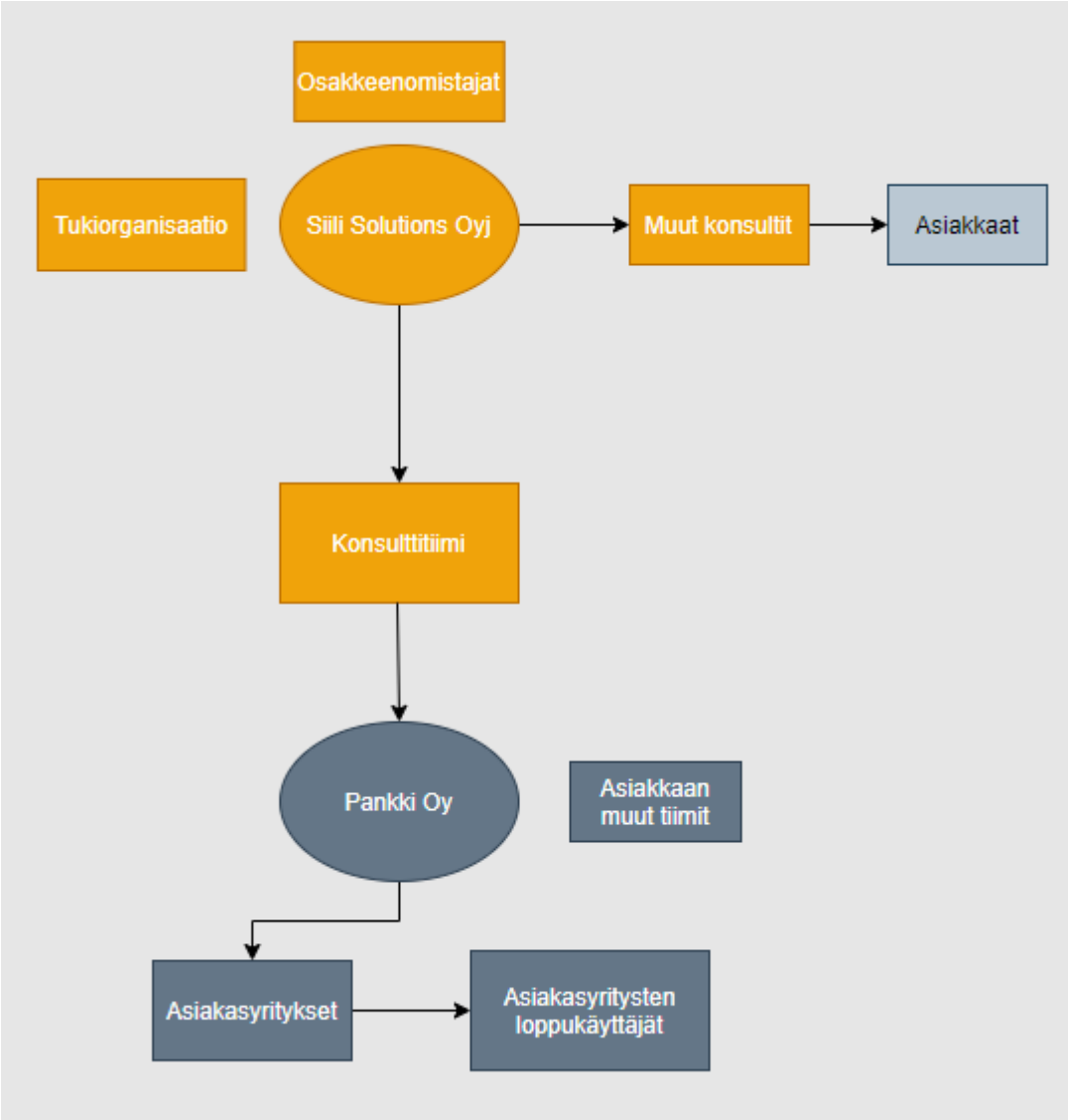
Mestari & Kisälli-ohjelman jälkeen minun työpanokseni myytiin nykyiseen projektiini Pankki Oylle, jossa olen ollut siitä asti. Projektin aikana olen kehittynyt huomattavasti sekä ohjelmointi- että konsulttitaidoiltani. Alussa tarvitsin hieman ohjausta ja saatoin kysellä paljon kysymyksiä, mutta nykyään pystyn kehittämään sovellusta paljon itsenäisemmin. Osaan huomattavasti paremmin löytää itse vastauksia kysymyksiin. Olen myös oppinut ohjelmoinnin periaatteet sen verran hyvin, että uusien teknologioiden opiskelu on nykyään paljon helpompaa. Koen, että alan olemaan nykyään keskiverto ohjelmistokehittäjän tasolla. En ole enää juniortasoinen, mutta en ole vielä myöskään ihan niin kokenut, että olisin senior-kehittäjä, vaikka varsinaista konsultin työkokemusta alkaa olemaan kohta 2 vuotta takana.

Ammatillisesti haluan jatkaa kehittymistä full stack -kehittäjänä. Vapaa-aikana kehitän mielelläni omia projektejani ja testailen täysin erilaisia asioita, mitä teen asiakasprojektissani. Olen harjoitellut muun muassa mobiiliohjelmointia, peliohjelmointia ja IoT-laitteiden kehittämistä. Kaikki nämä ovat todella mielenkiintoisia ja tuntuukin, että olen oikealla alalla, sillä pidän vapaa-ajan projekteja enemmänkin harrastuksena kuin työntekona. Mutta jotta kehittyisin nopeammin ammatillisesti, minun pitäisi tarkentaa fokustani hieman. On parempi osata muutamia asioita todella syvällisesti kuin monia asioita vähän. Minun pitäisi

siis keskittyä joihinkin tiettyihin osa-alueisiin tarkemmin sen sijaan, että kokeilisin vähän kaikkea. Edellä oleva määritelmä tuleekin suoraan yrityksemme Siilin nimestä ja siilikonseptista. Kyseisessä konseptissa on keskiössä siili ja kettu. Kreikkalaisrunoilija Arkhilokhoksen mukaan kettu osaa monta asiaa, mutta siili osaa yhden asian todella hyvin. Tästä syystä mekin pyrimme hallitsemaan jonkin tietyn asian todella hyvin, sillä on parempi olla mestari jossain, kuin osata hieman kaikenlaista.

2.2 Sidosryhmät työpaikalla

Työhöni liittyy kaksi selvästi erillään olevaa sidosryhmää: työnantajayritykseni Siili Solutions sekä asiakasyritys ja sen sidosryhmät (kuvio 1).



Kuvio 1. Keskeiset sidosryhmät työssäni

Siilin tukiorganisaatio on tiimimme tärkeimpiä sidosryhmiä työnantajayrityksen suunnalta. Tukiorganisaatioon kuuluu koko muu Siilin organisaatio muiden konsulttien lisäksi kuten

myynti, IT-tuki, rekrytointi sekä johto. Jokainen näistä on jollakin tapaa tekemisissä työmme kanssa tai joihin työmme vaikuttaa. IT-tuki pitää huolen, että meillä on tarvittavat työvälineet ja työkalut tehdä työtämme, myynti pitää meidät työllistettynä projekteissa, rekrytoijien avulla saamme mahdollisesti lisää osaamista tiimeihin, sekä johto seuraa projektimme edistymistä ja muita mittareita. Siilin muut konsultit ovat myös tärkeä sidosryhmä, sillä tapaamme toisiamme viikoittain ja keskustelemme, kuinka projektimme edistyvät. Pystymme myös pyytämään toisiltamme apua tarpeen tullen, jos tarvitsemme jonkin asian ratkaisemiseen jotain sellaista asiantuntijuutta, jota meidän tiimistämme ei välttämättä niin paljoa löydy.

Asiakasyrityksen tärkeimmät sidosryhmät ovat muut tiimit, sillä teemme tiivistä yhteistyötä heidän kanssaan. Kolme muuta tiimiä rakentaa rajapintoja, joita me voimme hyödyntää Kassiksessa. Heidän kanssaan teemme todella tiivistä yhteistyötä, jotta molemmat tiimit pysyvät kartalla tulevista töistä ja tarpeista. Tärkeä sidosryhmä on tietenkin myös asiakasyrityksen omat asiakkaat, jotka tulevat olemaan sovelluksen lopullisia käyttäjiä. Heiltä saamme palautetta, miten sovellusta voisi kehittää asiakkaan edustajien välityksellä. Käyttäjiä on myös haastateltu ennen projektin alkua, millaisia toiveita heillä on kehitettävän sovelluksen osalta, jotta saamme rakennettua mahdollisimman hyvän asiakaskokemuksen.

2.3 Vuorovaikutustaidot työpaikalla

Konsulttina työskennellessä vuorovaikutustilanteet kuuluvat olennaisena osana päivittäisiin työtehtäviini. Teemme ohjelmistokehitystä Scrumin-käytänteillä, joten joka päivä on vähintään lyhyt tiimin sisäinen Scrum Daily-tapaaminen, jossa käymme läpi mitä olemme saaneet viimeisen vuorokauden aikana valmiiksi ja mitä teemme seuraavaksi.

Haastetta projektissa aiheuttaa jonkin verran se, että eri tiimien jäsenet voivat olla hyvin monikansallisia. Tämä voi aiheuttaa joissakin tapauksissa haasteita, kun työkulttuurit voivat olla hyvin erilaisia. Kieliongelmia ei ole mielestäni juurikaan esiintynyt, sillä kaikki ovat puhuneet hyvin englantia. Aksentin vuoksi saattaa tosin joskus olla ymmärtämisvaikeuksia. Työssä pitää pystyä olemaan dynaaminen ja mukautumaan erilaisiin tilanteisiin. Oman tiimimme työntekijät ovat kaikki suomalaisia, mutta kun teemme tiivistä yhteistyötä toisten tiimien kanssa, on välillä huomioitava kulttuuriset erot. Työpäiviin sisältyy myös monia palavereita, joissa pitää pystyä ilmaisemaan näkemyksiään ja mielipiteitään. Tiimillämme on lähes päivittäin palaveri, jossa suunnittelemme jotain tulevia töitä. Keskustelemalla avoimesti käsiteltävästä asiasta pääsemme parhaaseen lopputulokseen, joka tyydyttää yleensä kaikkia.

Työpaikalla pitää myös muistaa, että kaikki palavereihin osallistuvat eivät välttämättä hallitse kaikkia teknisiä käsitteitä. Varsinkin asiakkaan liiketoiminnan edustajien kanssa käytävissä keskustelussa ei voi käyttää välttämättä niin teknistä termistöä, kuin mitä käyttäisi omien tiimiläisten kanssa, sillä he eivät välttämättä ymmärrä mitä tarkoitat. Tämä tuo omat haasteensa, mutta asiat on yleensä aika helppo kuvailla havainnollisesti, kunhan vain muistaa näin tehdä.

3 Päiväkirjaraportointi

3.1 Seurantaviikko 1: 10-14.2.2020

Maanantai 10.2.2020

Aamu alkaa siitä, mihin olin perjantaina jäänyt. Oli tutkittava mistä syystä osa automaatiotesteistämme eivät mene läpi. Aloitan tutkimalla Jenkinsistä viikonlopun aikana epäonnistuneet testit ja luen niiden raportit läpi, jotta saan selville, missä kohtaa ne ovat menneet vikaan. Vaikuttaisi siltä, että kaikki ulkomaan maksuja testaavat testit hajoavat kaikki samaan kohtaan.

Käytän Putty -sovellusta, jotta saan SSH-yhteyden testiympäristömme tietokoneeseen, jonne kaikki virheet lokittuvat. Laitan lokien jäljityksen päälle ja teen manuaalisesti ulkomaanmaksun samalla testidatalla, jota automaatiotesteissä käytetään. Huomaan, että käyttämämme validointirajapinta palauttaa virheen, jonka mukaan pankin nimi ei täsmää sallittuihin merkkeihin. Bugi on sinällään mielenkiintoinen, sillä maksussa ei syötetä pankin nimeä. Kirjaan tästä bugin JIRA-tiketöintijärjestelmäämme, jotta joku muu voi tutkia kyseistä bugia. Minun pitäisi aloittaa API clientin rakentamista frontendiin, jotta voimme sen avulla kutsua backendia.

Olimme käyneet läpi edellisellä viikolla mahdollisia kirjastoja, jotka voisivat olla hyviä tähän tarkoitukseen. Päädyimme käyttämään React Query -nimistä kirjastoa, jolla saamme dataa talletettua välimuistiin helposti, jotta rajapintakutsuja ei tarvitsisi tehdä joka vaiheessa. Pystymme myös tekemään jokaiselle haulle oman custom hookin, joka on yksi parhaimpia ominaisuuksia modernissa Reactissa.

Aloitin tutkimalla kirjaston dokumentaatiota ja sen toimintaa, jonka jälkeen lähdin rakentamaan siitä yksinkertaistettua versiota. Tällä versiolla pystyisin testaamaan yksinkertaisia kutsuja näennäistä JSON-serveriä vasten, jotta näen mitä dataa ja missä muodossa kirjasto sitä palauttaa. Ongelmia tulee eteen, kun yritän tyypittää, mitä tämä funktio ottaa sisään ja mitä se palauttaa. Päivä alkaa tulemaan päätökseen ja clientin tyypityksen ongelmat jäävät seuraavaan päivään.

Tiistai 11.2.2020

Tiistait ovat meillä yleensä päiviä, joille yritetään mahduttaa mahdollisimman paljon palavereita, jotta muina päivinä saisimme keskittyä mahdollisimman paljon itse ohjelmointiin. Pyrin päivän aikana jatkamaan mahdollisimman paljon edellisenä päivänä kesken jäänyttä clientin tekoa frontendiin, jotta pääsisin testaamaan sitä keskiviikkona.

Päivä alkaa, kun asiakkaan Product Owner pyytää minulta apua selvittämään legacy järjestelmän lähdekoodista, millä tavalla eräs arvo määräytyy. Selvitän logiikan lähdekoodista ja selitän asiakkaalle, miten kyseinen logiikka toimii.

Heti dailyn jälkeen tiedossa on koko tiimin kesken grooming-palaveri, jossa Scrumin käytänteiden mukaisesti käydään läpi backlogiamme. Tavoitteena on käydä kaikki epäselvät user storyt ja bugit läpi, sekä järjestellä ne tärkeysjärjestykseen. Aloitamme storyjen läpikäymisen keskustelemalla mitä niissä pitää tehdä, minkä jälkeen pohdimme teknisiä ratkaisuja. Tässä päätämme mm. mikä palvelu tekee validoinnin ja palauttaa tilitiedon käyttäjän syöttämästä tilinumerosta, jotta tiedämme mihin rajapintaan meidän pitäisi tehdä integraatio. Saamme kaikki käsiteltävät asiat valmiiksi ja priorisoitua, joten palaveri oli onnistunut.

Lounastauon jälkeen on tiedossa demo. Demossa esittelemme asiakkaalle, mitä olemme saaneet viimeisen kahden viikon aikana valmiiksi. Suurin esiteltävämme asia on, kuinka testiympäristöstä pääsee nyt kirjautumaan uuteen Kassis-palveluun, joka pyörii AWS:ssä. Olimme taistelleet tämän kanssa jo pari viikkoa lukuisten palomuuri- ja yhteensopivuusongelmien takia, joten tämä oli merkittävä saavutus.

Demon päättymisen jälkeen on aika aloittaa tiimiretro, jossa myös Scrumin mukaan pohdimme, mikä oli kahden edellisen viikon aikana mennyt hyvin ja missä voitaisiin parantaa. Saamme muutamia merkintöjä tehtyjä missä voitaisiin parantaa ja palaamme näihin taas kahden viikon kuluttua katsomaan, onko tilanne parantunut.

Retron jälkeen ehdin vielä hetken jatkaa clientin tekoa. Katsomme yhdessä vanhemman kehittäjän kanssa, mikä clientin tyypityksestä mahdollisesti on vikana ja päädymme tutkimaan käytettävän kirjaston dokumentaatiota, sekä tutkimme Typescriptista tämän tyypitysmahdollisuuksia. Saamme ongelman ratkaistua juuri ennen työpäivän loppumista, joten keskiviikkona pääsen heti aamulla testaamaan, saako clientilla haettu rajapinnasta dataa.

Keskiviikko 12.2.2020

Keskiviikkona päivä jatkuu siitä mihin tiistaina jäin. Tavoitteena olisi saada testattua edellisenä päivänä valmiiksi tullut API client, jotta ehtisin aloittaa vielä yleiskäyttöisen tililista komponentin ja saada sen valmiiksi ennen perjantaina alkavaa hiihtolomaa. Lähden testaamaan API clientia käynnistämällä JSON-server kirjastolla mockatun rajapinnan, joka palauttaa /accounts endpointista kovakoodatun tililistan. Client toimii kuten pitääkin ja dataa saadaan lokitettua konsoliin onnistuneesti ja vieläpä oikeassa muodossa.

Seuraavaksi lähden toteuttamaan tililistaa Reactilla. Tililistan yksityiskohdat on valmiiksi kirjoitettu JIRAan kyseisellä user storylle, joten lähden ensimmäisenä selvittämään mitä toiminnallisuuksia kyseisessä komponentissa pitää olla. Työ etenee ilman sen suurempia ongelmia, mutta jäi hieman kesken ennen päivän päättymistä.

Kokonaisuudessaan keskiviikko oli onnistunut päivä, sillä sain API clientin testattua toimivaksi ja tehtyä siitä pull requestin versionhallintaan. Tililista-komponentti eteni myös hyvää vauhtia, joten saan sen toivon mukaan huomenna valmiiksi.

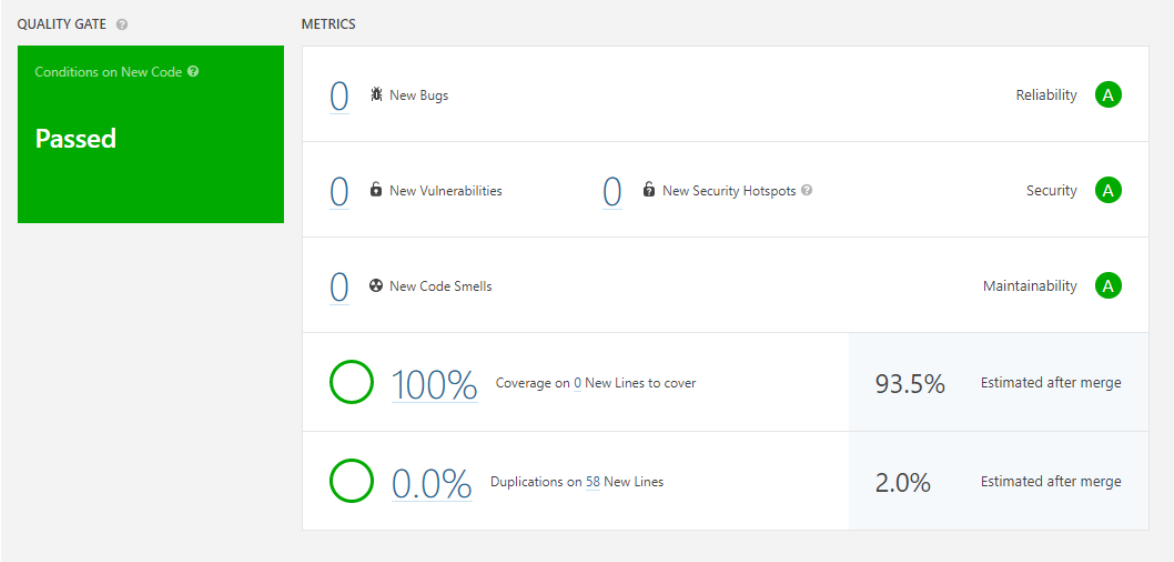
Torstai 13.2.2020

Torstaisin päivä alkaa UX-weekly palaverilla. Palaverissa käydään läpi käyttöliittymän asioita käyttäjäkokemuksen näkökulmasta ja hiotaan yksityiskohtia näiden osalta. Esittelen hieman eilen aloittamaani tililista-komponenttia ja käymme läpi muutamia tapauksia, kuinka lista muuttuu riippuen siitä, kuinka monta tiliä käyttäjällä on. Tarvittavat päätökset saadaan tehtyä ja päivän tavoitteena onkin nyt saada tililista valmiiksi, ennen huomenna alkavaa hiihtolomaa.

Tililista on muuten valmis, mutta kirjoitan sille vielä yksikkötestit. Käytän tässä Reactissa yleisesti käytettyä Jest-kirjastoa, jolla testaan komponentin logiikkaa renderöidä eri näkymiä. Olemme määritelleet, että testikattavuus pitää olla 85 %, joten yksikkötestien pitää testata lähes kaikki tapaukset. Tililista tulee valmiiksi jo voin avata myös siitä pull requestin.

Loppupäivän käytän muiden tiimiläisten pull requestien läpikäymiseen. Tarkistan itse koodin ja annan siihen muutosehdotuksia, jos löydän jotain optimoitavaa. Jenkins suorittaa automaattisesti projektin rakennuksen ja ajaa yksikkötestit läpi, josta näen heti, jos jokin näistä kohdista epäonnistuu. Käytössämme on myös SonarQube-kirjasto, joka tarkistaa koodin laatua ja tietoturvallisuutta. Onnistuneen Jenkins ajon jälkeen se tuottaa kätevän

raportin, josta näkee nopealla silmäyksellä, onko koodissa mitään sen poimimia virheitä (Kuva 1).



Kuva 1. SonarQuben tuottama raportti koodin laadusta ja testikattavuudesta.

Päivän tavoite saada tililista valmiiksi tuli täyteen ja ehdin myös katselmoida auki jääneitä pull requesteja. Ehdin käymään myös sähköpostini läpi, jotta asiat eivät jää kesken ennen huomenna alkavaa hiihtolomaa.

Perjantai 14.2.2020

Perjantaina työpäivä jää lyhyemmäksi hiihtoloman alkamisen vuoksi. Tänään en tule tekemään mitään kehitystä asiakasprojektin suhteen, vaan aamupäivän tarkoituksena on opiskella AWS Solutions Architect-sertifikaattia varten.

Olen hankkinut sertifikaatin opiskelua varten kurssin Cloud Guru -nimiseltä sivustolta, joka tarjoaa valmentavia kursseja sertifikaattien suorittamiseen jokaiselle suurelle pilvipalvelu alustalle (AWS, Google Cloud Platform, Microsoft Azure). Kurssissa itsessään on noin 12 tuntia videomateriaalia, joka on jaoteltu osiin AWS:n palveluiden mukaan. Jokaisen osan lopussa on monivalintakysymyksiä kyseisessä osassa käydyistä asioista, jotta ne jäisivät paremmin mieleen. Kurssin lisäksi olen ostanut harjoituskokeita, jotka simuloivat oikeaa sertifikaattikoetta, jotta koe menisi varmemmin läpi. Koska 80:stä kysymyksestä 70 % pitää olla oikein, niin liian moneen virheeseen ei ole varaa.

Viikkoanalyysi

Viikon aikana työtehtäväni koostuivat pitkälti Typescript-pohjaisesta frontendin rakentamisesta sekä testiautomaatiosta. Typescriptilla koodista saadaan helpommin luettavaa ja ymmärrettävää, sekä myös helpommin debugattavaa, sillä kaikki koodissa on tyypitetty, jolloin on helpompi saada kiinni mahdolliset tyypitykseen liittyvät bugit, mitä Javascriptissa usein esiintyy (Sharma 2018). En ole itse vielä kovin kauaa kehittänyt Typescriptilla, joten tyypitysten kohdalla minulle tulee vielä välillä ongelmia, kun teen mukautettuja tyyppejä. Typescript on onneksi sen verran käytetty kieli, että netistä löytyy yleensä hyvin samantyyppisiä tapauksia, joita voi sitten soveltaa omassa koodissaan.

Viikon suurin haaste olikin rakentaa API client frontendiin, jotta pystyisimme alkaa käyttämään oikeaa dataa kovakoodatun datan sijaan. React Query-kirjasto, jota kyseisen clientin rakentamisessa päätettiin alkaa käyttämään, oli minulle täysin vieras, enkä ole aiemmin toteuttanut mitään vastaavaa Typescriptilla. Kirjaston dokumentaatio oli onneksi todella hyvin kirjoitettu, joten käyttöönotto ei ollut kovin hankalaa. Vaikeuksia tuli, kun clientille piti tehdä mukautetut tyypitykset, jotta data mitä frontendiin palautuu, on varmasti oikeassa muodossa. Myös yksikkötestien rakentaminen clientille oli oma haasteensa, sillä yksikkötestien ajon aikana ei voida hakea dataa oikeasta lähteestä. Tämä tehtävä oli yleisesti todella opettavainen, sillä tiedän nyt, miten pystyn jatkossa tekemään vastaavanlaisen clientin sekä miten voin testata sen käyttöä. Sain tehtyä clientista myös yleiskäyttöisen, parametrisoimalla mihin pyyntö lähtee, millä metodilla ja minkälaista dataa se palauttaa. DRY-periaatteen mukaan, kun koodia pystytään käyttämään myös muualla, niin sen luettavuus ja ylläpidettävyys paranee, eikä jokaiseen komponenttiin tarvitse tehdä samaa asiaa aina uudestaan (Thomas & Hunt 2019).

Viikon aikana pääsin tekemään myös hieman DevOps-tyyppistä kehittämistä, kun selvitin miksi automaatiotestimme ovat epäonnistuneet. Automaatiotestimme on rakennettu Robot Frameworkilla, jossa olen jo sen verran kokenut, että pystyn opettamaan sitä myös meidän testaajillemme, jotka testaavat manuaalisesti eri ominaisuuksia, jotta he pystyvät mahdollisesti kirjoittamaan testejä, jotka testaavat automaattisesti yksinkertaisesti asioita, jotta niihin ei tarvitse kuluttaa työaikaa. DevOps onkin semmoinen asia missä haluan syventää osaamistani entistä enemmän, minkä vuoksi myös opiskelen aktiivisesti AWS-sertifikaattia varten, jotta pystyn tarpeen tullen tekemään arkkitehtuurisia ratkaisuja siitä, mitä ja miten voimme hyödyntää eri AWS:n palveluita hyödyntää rakennettavassa Kassisjärjestelmässä.

3.2 Seurantaviikko 2: 24-28.2.2020

Maanantai 24.2.2020

Palasin toimistolle hiihtoloman jälkeen hieman aikaisemmin, jotta ehdin käydä läpi viikon aikana kertyneet sähköpostit ja keskustelut tiimien chat-kanavilla. Päivän tavoite on saada ensimmäisenä selvyys, mitä viikon aikana on tapahtunut ja mitä tehdään seuraavaksi. Saan käytyä sähköpostit läpi ennen kuin muut tiimiläiset saapuvat paikalle ja päätänkin keskustella heidän kanssaan, mitä on saatu edistettyä sinä aikana, kun olen ollut lomalla. Backendiin oli rakennettu integraatioita, joten pystyn hakemaan käyttäjän tilejä oikeista rajapinnoista ja tehty palvelu mikä hoitaa valuuttojen kurssimuunnokset.

Rakentamaani API-clientia ei ollut otettu vielä käyttöön, mutta se on nyt mahdollista backendiin rakennettujen integraatioiden myötä, joten otankin päivän tavoitteeksi ottaa sen käyttöön käyttöliittymässä, jotta kovakoodattu mockattu data saadaan korvattua oikealla rajapinnoista tulevalla datalla. Tiistaina olisi demo, jossa esittelemme taas asiakkaalle mitä kahden viikon aikana olemme saaneet aikaiseksi, joten työllä on hieman kiire. Huomaan, että tälle työlle ei ole tehty taskia JIRAan, joten teen ensimmäisenä sen, jotta myös muut tiimiläiset näkevät, että mitä minulla on tällä hetkellä työn alla.

Lähden miettimään, missä tilien data olisi paras hakea. Yksi mahdollisuus olisi hakea suoraan sovelluksen juuressa, jolloin data saataisiin helposti siirrettyä muille komponenteille. Tämä ei kuitenkaan ole ehkä paras idea, sillä dataa pitäisi siirtää muutaman komponentin läpi, jotka eivät sillä mitään tee, vaan siirtävät sitä vain eteenpäin. Päätänkin tehdä datan haun koko tilien näyttämisen päätason komponentissa, jotta tilidata on suoraan siellä missä sen pitääkin olla. Saan datan haettua onnistuneesti niin, että se näkyy tilinäkymässä kuten pitää, mutta huomaan, että testikäyttäjämme datassa on puutteita, joten käyttöliittymä ei renderöi kaikkia tapauksia. Sinällään toiminnallisuus toimii kuten pitääkin, mutta jotta voin varmistaa, että kaikki käyttöliittymän komponentit toimivat kuten oletettu, niin testidataa pitää hieman laajentaa. Keskustelen tästä Scrum Masterimme kanssa ja sovimme palaverin huomiselle, jossa käymme testidatan läpi ja mitä siihen pitäisi lisätä.

Päivä itsessään onnistui ilman suurempia ongelmia. Koska olin itse rakentanut clientin, niin tiesin myös tarkalleen, miten sitä tulisi käyttää ja clientissä käytetyn kirjaston dokumentaatiosta oli helppo löytää asioita, millä sain konfiguroitua esimerkiksi sen, kuinka kauan haettu tilidata pysyy selaimen välimuistissa ennen kuin se haetaan uudestaan. On myös hienoa, että pystymme pikkuhiljaa korvaamaan kovakoodatun datan oikealla datalla, jotta pääsemme näkemään, että Kassis-järjestelmän bisneslogiikka toimii kuten pitääkin.

Tiistai 25.2.2020

Tiistai on jälleen palavereita täynnä oleva päivä. Aamupäivän tavoitteena on yrittää saada tilidata palautumaan ja näkymään oikein käyttöliittymässä. Aamulla tiimimme Scrum Masterin kanssa käydyssä palaverissa huomaamme, että eri valuuttaiset tilit eivät jostain syystä palaudu kutsuissa. Lähden debuggaamaan tätä kutsumalla Postmanilla suoraan tätä rajapintaa, joka palauttaa tilidatan. Minun on osattava rajata ongelmaa, onko vika meidän vai heidän päässään. Huomaan, että kutsussa kaikki tilit palautuvat ongelmitta, joten bugin täytyy olla meidän koodissamme.

Grooming-tilaisuus ehtii alkamaan, joten joudun hieman multitaskaamaan, jotta ehdin saada bugin korjattua, sillä heti lounaan jälkeen on demon aika. Groomingissa käydään läpi erilaisia maksuun liittyviä käyttöliittymiä ja mitä mahdollisia erikoisuuksia kussakin on. Pystyn seuraamaan keskustelua ja samalla käymään koodiamme läpi backendissä, joka kutsuu tilejä palauttavaa rajapintaa. Huomaan, että eräs roolikoodi filtteröi ne kokonaan pois, joten poistan tämän ja vien muutokset versionhallintaan, jotta pystyn testaamaan korjaustani käytännössä testiympäristössämme. Muutosten viennin jälkeen AWSn CodePipeline pyörähtää käyntiin, mikä pyörittää testit ja tekee sovelluksestamme tuotantoon menevän version ja päivittää sen meidän testiympäristöömme. Tässä kuluu aikaa noin 15 minuuttia, joten pystyn hyvin keskittymään jälleen täysillä meneillään olevaan palaveriin.

Lounastauko ehtii alkamaan, joten pääsen näkemään korjaukseni toimivuuden vasta lounaan jälkeen. Ja se toimii kuin toimiikin. Eri valuuttaiset tilit palautuvat nyt myös onnistuneesti ja ne renderöityvät käyttöliittymään kuten pitääkin. Pääsemme näyttämään ominaisuutta demotilaisuudessa ja asiakkaan edustajat ovat hyvin tyytyväisiä etenemiseemme.

Demon jälkeen on jälleen retrospektiivin aika, jossa käymme läpi Agile Manifeston 12 periaatetta ja miten ne toteutuvat tiimissämme. Jokaiseen periaatteeseen jokainen tiimiläinen sai vastata kyllä/ei riippuen siitä, toteutuuko kyseinen periaate omasta mielestä. Olimme monesta asiasta suhteellisen yksimielisiä, mutta esimerkiksi periaate ”Toimitamme versioita toimivasta ohjelmistosta säännöllisesti, parin viikon tai kuukauden välein ja suosimme lyhyempää aikaväliä” jakoi tiimin aika lailla kahtia. Tämä oli hieman tulkinnanvarainen kysymys, sillä emme tosiaan tällä hetkellä toimita mitään versiota tuotantoon, joka oikeasti näkyisi käyttäjille. Noin kahden viikon välein toimitamme asiakkaalle uusimman version, jota demoamme kuten tänäänkin, mutta se ei ole vielä valmis tuotantoon.
Retron jälkeen työpäivä alkaa olemaan lopussa ja ehdin keskustelemaan muutaman tiimiläisen kanssa, miten voisimme jakaa nämä groomingissa käydyt käyttöliittymien työt, niin ettei liian monta henkilöä koodaa lähes samalla alueella.

Tämä päivä oli oikein hyvä, sillä onnistuin itselleni asettamassa päivän tavoitteessa saada tilit näkymään oikein ennen demoa ja opin siinä samalla muutaman asian. Tästä esimerkkinä opin, miten Postman piti konfiguroida, jotta pystyn lähettämään kutsuja suoraan rajapintaan, sillä kyseinen rajapinta on autentikoinnin takana, eikä siihen pystynyt lähettämään suoraan pyyntöjä pelkällä url-osoitteella, josta tiedot haetaan.

Keskiviikko 26.2.2020

Tavoitteena tälle päivälle olisi saada aikaan mahdollisimman valmiiksi käyttöliittymä, jossa käyttäjä voi siirtää rahaa omien tiliensä välillä, koska asiakas haluaisi esitellä edistymistämme myös muille tiimeille, jotka työskentelevät samalla osa-alueelle kuten mekin.

Käyn tarvittavat käyttöliittymämuutokset läpi ja huomaan, että kyseinen työtehtävä ei ole hirveän iso, sillä minun pitää vain lisätä muutama kenttä ja muokata hieman koko lomakkeen ulkonäköä CSS:n avulla. Lisään ensimmäisenä puuttuvat kentät lomakkeelle, jonka jälkeen lähden miettimään koko lomakkeen asettelua. Tilit, josta käyttäjä voi valita miltä tililtä mille tilille ovat tällä hetkellä rinnakkain, kun ne pitäisi olla allekkain. Selaan hieman tyyliopastamme ja huomaan, että meillä on käytössämme valmis komponentti, joka rivittää kyseiset tilivalikot nätisti allekkain sopivalla etäisyydellä. Käytämme muutenkin paljon näitä valmiita komponentteja sovelluksessamme, jotka pääasiassa designereista ja frontend kehittäjistä koostuva tiimi on tehnyt kaikkien käyttöön, jotta sovelluksemme pysyisi tyylillisesti mahdollisimman paljon linjassa muun sivuston kanssa.

Saan tehtävän valmiiksi ja lopputulos vaikuttaa toimivalta. Pystyn esittelemään asiakkaalle vielä huomenna kyseistä käyttöliittymää ja kysyä hänen mielipidettään siitä, jotta tiedän pitääkö siihen tehdä vielä jotain pieniä muutoksia. Jos ei tarvitse, pystyn huomenna jatkamaan suoraan tekemään uutta käyttöliittymää, mikä mahdollistaa käyttäjän tekemään EU:n sisäisiä maksuja.

Torstai 27.2.2020

Päivän tavoitteena on saada EU:n sisäisten maiden maksuihin liittyvä käyttöliittymä valmiiksi, jotta voin jatkaa siitä ulkomaisten maksujen käyttöliittymään. Päiväni alkaa kuitenkin siitä, että käyn tiimimme Product Ownerin kanssa läpi eilen valmiiksi tullutta oman tilien välisen siirron käyttöliittymää. Häneltä tulee vihreää valoa ja kaikki elementit ovat paikallaan, kuten pitääkin. Tyylittelyssä on vielä hieman hiomista, mutta siihen ei tarvitse juuri nyt keskittyä.

Aamupäivä menee yleisesti palavereissa, jossa käymme jälleen hieman käyttökokemuksen näkökulmasta käyttöliittymiä läpi ja teemme pieniä viilauksia niihin. Tämän palaverin jälkeen olin varannut palaverin koko tiimin kehittäjille, jossa kävisimme läpi kaikki säännöt mitä meillä on TS-linterissä käytössä. TS-lint tarkkailee, että koodimme noudattaa tiettyjä sääntöjä, jotta koodimme olisi mahdollisimman yhtenäistä sekä puhdasta. Linterin säännöt laitettiin projektin alussa hyvin tiukoiksi sillä idealla, että niitä voidaan ajan mittaan löysentää. Palaverissa käymme läpi konfiguraatiotiedostoamme ja jokaista sääntöä yksitellen keskustellen mitä sille pitäisi tehdä. Suurin osa säännöistä saa pysyä ennallaan, mutta paljon keskustelua herätti funktioiden pituus. Tällä hetkellä yhden funktion pituus saa maksimissaan olla 60 riviä. Tämä kuitenkin saattaa ylittyä isoissa komponenteissa, koska Reactin JSX on loppuen lopuksi hyvin HTML:n kaltaista koodia. Toisaalta komponentit pitäisi pitää pieninä, jotta ne ovat helpommin luettavia. Toisaalta jos pilkkoo ihan kaikkea, niin pian meillä on todella paljon tiedostoja, josta on vaikea löytää mitään. Päätämme pitää funktion pituuden säännön 60 rivissä vielä tällä hetkellä, mutta johon voimme palata myöhemmin.

Iltapäivällä pääsen viimeinkin aloittamaan EU:n sisäisiin maksuihin liittyvän käyttöliittymän työstämisen. Ensimmäiseksi siirrän sen ’Work In Progress’ sarakkeeseen JIRAssamme, minkä jälkeen käyn läpi mitä kyseiseen tehtävään liittyy tiketin kuvauksesta. Tehtävä vaikuttaa yksinkertaiselta, pitää tehdä vain komponentti, johon käyttäjän on mahdollista lisätä saajan nimi ja osoite, jotta tiedetään mihin maksu ohjataan. Aloitan käyttöliittymän työstön eikä siinä mene kovin kauaa, sillä pystyn hyödyntämään valmiiksi tehtyjä tekstinsyöttökenttiä hyvin tässä. Kyseisessä käyttöliittymässä ei ole mitään logiikkaa, joten kirjoitan sille yksinkertaisen yksikkötestin, joka tallentaa siitä snapshot-version. Snapshotin avulla Reactissa käytettävä Jest-kirjasto pitää huolen, että emme vahingossa muuta käyttöliittymää ei-toivotulla tavalla.

Päivän tavoite tulee täyteen ja samalla päivän tavoite tulee valmiiksi. Saan tehtyä EU:n sisäisiin maksuihin liittyvän käyttöliittymän valmiiksi ja teen siitä pull requestin versionhallintaamme, jossa joku muu kehittäjä voi käydä koodimuutokseni läpi.

Perjantai 28.2.2020

Perjantaiaamupäiväksi on sovittu paljon Siilin sisäisiä palavereita, joissa keskustelemme muun muassa tulevista opiskelijayhteistyötapahtumista ja suunnittelemme ohjelmaa uuden toimistomme tupaantuliaisiin, johon kutsumme opiskelijoita kaikista Helsingin korkeakouluista, joissa opiskellaan tietotekniikkaa.

Iltapäivällä ehdin avata työkoneen ja käydä kaikki sähköpostit läpi ja vastaamaan niihin. Muutama pull requesti on tullut myös versionhallintaamme, joten katselmoin ne myös vielä. Kaikki muutokset vaikuttavat hyviltä, joten hyväksyn ja yhdistän ne yhteiseen development-haaraamme.

Iltapäivällä jään seuraamaan kollektiivia toimistollemme, missä muutama Siilin kehittäjä on tehnyt Internet of Beer-nimisen laitteen, jota olisi tarkoitus esitellä ensi viikolla järjestettävässä Microsoft Tech Dayssa. Laite on oluthana, jonka oluttynnyri sijaitsee vaa'alla, joka on yhdistetty Arduinoon. Kyseinen laite lähettää painodataa Microsoftin Azure-pilvipalveluun, jossa tehdään laskentaa reaaliajassa aina, kun hanasta otetaan olutta. Sivulle päivittyy hyvin nopeasti, kuinka paljon olutta tynnyrissä on vielä jäljellä tämän jälkeen.

Päivän mielenkiintoisin asia oli nähdä tämä Internet of Beer ratkaisu, sillä kaikki IoT:hen liittyvät asiat, millä voidaan digitalisoida hyvin arkipäiväisiä asioita, ovat hyvin mielenkiintoisia mielestäni. Näistä saattaa saada inspiraatiota myös lähteä työstämään jotain omaa kokeilua, sillä käytettävät laitteet eivät maksa paljoa.

Viikkoanalyysi

Viikko meni lähes täysin jälleen frontendissä, kehittäessäni erilaisia käyttöliittymiä Typescript ja React teknologioilla. Viikon aikana kaikista mukavin työ oli rakentamani API clientin käyttöönotto, jotta sovellukseemme saataisiin ensimmäistä kertaa oikeaa dataa ja asiakkaan reaktio tästä oli myös hyvin innostunut. Nämä ovat juuri niitä pieniä hetkiä työssä, kun tietää, että on tehnyt jotain hienoa, josta myös muut ovat innoissaan. Työn tekeminen oli myös mukavaa vaihtelua jatkuvan frontendin työstämiseen, sillä sain käyttää tehtävässä kaikkia taitojani. Sain tutkia frontendin koodia, minkä tunnen aika hyvin ja myös sukeltaa backendin Javaan ja tutkia sen koodeja, mitä en ole hetkeen tehnyt, sillä olen keskittynyt aika lailla vain frontendin kehittämiseen. Toivon mukaan saisin tehdä myös mahdollisimman paljon full stack tehtäviä, eli saisin koodata sekä frontendin että backendin osuuden myös tulevissa projekteissa, sillä tunnen että niissä kehityn kaikkein eniten.

Viikon aikana Typescriptin osalta ei tullut mitään kovin haastavaa vastaan, mihin olisin jäänyt pidemmäksi aikaa jumiin. Huomaan, että kehityn siinä päivä päivältä ja pikkuhiljaa alan olemaan entistä sujuvampi sen syntaksin kanssa. Usein kun odotan jonkin valmistumista, niin pyrin käyttämään ajan hyödyksi. Tällä alalla jatkuva kehittyminen on pakollista ja koska koodin kääntymisessä tai koodin päätymisessä testiympäristöön menee aina

muutamia minuutteja. Mitään kirjaa ei siinä ajassa kannata aloittaa lukemaan, mutta olen huomannut, että muiden alalla työskentelevien blogipostaukset ovat juuri sopivia luettavaksi tuossa ajassa. Tällä viikolla keskityinkin lukemaan blogeja Typescriptista ja vastaan tuli blogipostaus Markus Hanslikilta. Kyseisen blogin viimeisessä kohdassa käytiin läpi, kuinka Typescriptille tarkoitettu TS-lint alkaa olemaan vanhentunut ja olisi aika siirtyä käyttämään ES-lintiä, joka on alun perin tarkoitettu Javascriptille, mutta nykyään siitä löytyy myös tuki Typescriptille. Kohdassa puhuttiin myös SonarJS:stä, mikä on mahdollista ottaa linterin kanssa käyttöön, joka tekee samanlaisia koodin tarkistuksia, kuin mitä Jenkins Pipelinessa käytettävä SonarQube tekee, mutta tämän avulla saisit palautteen jo koodia kirjoittaessa (Hanslik 2019). Valitettavasti törmäsin tähän blogiin vasta palaverin jälkeen, jossa kävimme TS-lintin sääntöjämme läpi. Molemmat asiat tulivat kuitenkin uutena tietona minulle. Tulen esittelemään näitä tiimille hyvin pian ja kyselen heidän mielipidettään, pitäisikö meidän siirtyä käyttämään näitä työkaluja.

Viikon aikana huomasin myös, että minun pitäisi kehittää taitojani CSS:n parissa. CSS tuntuu olevan aika iso mörkö monelle kehittäjälle, sillä jos sitä ei osaa hyvin, niin käyttöliittymän näpertelyssä saattaa mennä muutamia tunteja, jotta kaikki elementit saisi aseteltua kuten pitääkin. CSS ei kehity niin nopeasti kuin esimerkiksi frontendin frameworkit kuten React ja Vue.js, enkä löytänyt mitään kovinkaan hyödyllisiä blogiartikkeleita siitä. Monissa artikkeleissa tulevat esiin kuitenkin termit Grid ja Flexbox, joilla elementtien sijoittelua saa suoraviivaistettua huomattavasti (Habib 2018; Lotanna 2020). Löysin Udemysta kurssin, jossa käydään juuri nuo Flexbox ja Grid läpi kaiken muun ohella. Kurssi sisältää myös kiitettävät 28 tuntia materiaalia, joten luultavasti tulen hankkimaan tämän kurssin pian, jotta saan kehitettyä taitojani tämän parissa!

3.3 Seurantaviikko 3: 2-6.3.2020

Maanantai 2.3.2020

Työt jatkuvat siitä, mihin viime viikolla jäätiin. Viime viikon päätteeksi sain valmiiksi käyttöliittymän, minkä avulla käyttäjä pystyy tekemään EU:n sisäisiä maksuja. Huomaan, että se on myös hyväksytty versionhallintaamme, joten tälle päivälle otankin tavoitteeksi, että saisin ulkomaanmaksun käyttöliittymän mahdollisimman valmiiksi.

Tutkin JIRAsta, mitä kyseiseen taskiin pitäisi tehdä ja lähden työstämään käyttöliittymää. Työ on jälleen suhteellisen suoraviivaista ja etenee vauhdilla, kunnes on aika taas käydä viikoittaisessa käyttöliittymien suunnittelupalaverissa. Päivän agendana on käyttäjän navigointi ja mitä käyttöliittymässä pitäisi tapahtua, kun käyttäjä navigoi käyttäen selaimen takaisinnappia. Tämä monesti aiheuttaa ongelmia varsinkin meidän sovelluksessamme, missä lomakkeita lähetetään monella sivulla ja joka saattaa ohjata käyttäjän monen näkemälle ”Vahvista lomakkeen uudelleenlähetys”-sivulle. Eräs kehittäjä kommentoi, että pystymme mahdollisesti manipuloimaan käyttäjän historiatietoja sovelluksessamme, jotta pystymme navigoimaan käyttäjän sovelluksen etusivulle ja täten väistämään kyseisen vahvistussivun. Tämä oli uutta tietoa minulle ja päätämmekin, että tutkimme asiaa tarkemmin, miten tämä toimii käytännössä.

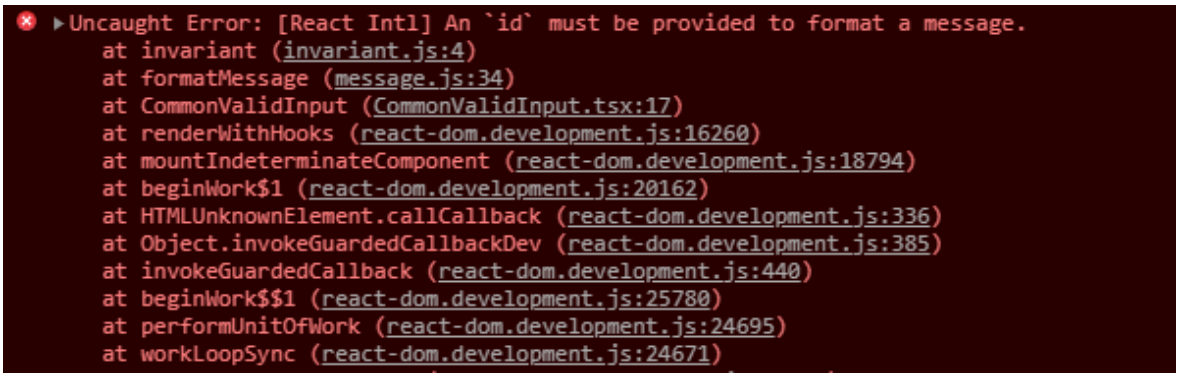
Iltapäivällä jatkan ulkomaanmaksun käyttöliittymän työstöä ja huomaan, että teen hyvin samanlaista koodia, mitä tein myös EU:n sisäisten maksujen käyttöliittymässä. Huomaan, että lähdin vähän liian nopeasti työstämään ratkaisua, enkä pohtinut tarpeeksi, kuinka se olisi paras tehdä. Keskeytänkin työt hetkeksi ja palaan suunnittelupöydän ääreen. Molemmissa käyttöliittymissä on täysin samoja kenttiä, joten voisin tehdä näistä kentistä yleiskäyttöisen komponentin, joka ottaa vaihtuvat asiat kuten kenttien id:n ja luokan nimen parametrina.

Päivä ehtii päättymään, mutta nyt tiedän, mistä jatkan huomenna. Joudun koodaamaan uudestaan jo valmiiksi tullutta EU:n sisäisten maksujen käyttöliittymää, sekä lähes valmista ulkomaisten maksujen käyttöliittymää, jotta saan vähennettyä toistuvan koodin määrää ja pysyttyä täten DRY-periaattessa. Tästä päivästä ainakin opin, että vanha sanonta ”hyvin suunniteltu on puoliksi tehty” pitää hyvin paikkansa, sillä jos olisin aluksi suunnitellut hieman, miten asiat kannattaisi tehdä, minun ei tarvitsisi nyt tehdä niitä uudestaan.

Tiistai 3.3.2020

Tavoitteena tälle päivälle on saada EU:n sisäisten maksujen sekä ulkomaanmaksujen yhdistävä käyttöliittymä komponentti valmiiksi, sekä saada ulkomaisten maksujen käyttöliittymä myös valmiiksi. Ehdin hetken tehdä töitä ennen kuin on viikoittaisen grooming palaverin aika. Groomingissa käymme läpi sekä frontendin ja backendin käynnissä olevien töiden tilannetta ja keskustelemme, mitä pitäisi tehdä seuraavaksi, jotta molemmat puolet etenisivät samaa vauhtia, eikä tulisi työnkeskeytyksiä sen takia, ettei jommankumman puolen työtä ole vielä aloitettu.

Palaverin jälkeen pääsen jatkamaan käyttöliittymien työstöä, kunnes ongelmia tulee vastaan. Yritän avata EU:n sisäisen maksun lomaketta ja koko käyttöliittymä kaatuu ja muuttuu valkoiseksi sivuksi. Avaan Chromen DevTools-konsolin, joka on täynnä virheitä. Virheiden seasta huomaan yhden, mikä saattaa olla kaatumisen syynä (kuva 2). Tehdessäni tätä yleiskäyttöistä komponenttia annan sille parametrina lokalisointi id:n, minkä perusteella käyttämämme lokalisointikirjasto osaa kääntää kyseisen kentän tekstin halutulle kielelle. Kaikkiin kenttiin kuitenkaan tätä ei tarvitse antaa, sillä teksti saattaa olla sama kaikilla kielillä (kuten tässä tapauksessa BIC). Korjaan virheen käyttäen Javascriptin Ternary operaattorilla, jolla tarkistan, annetaanko komponentille kyseistä id:tä vai ei. Jos sitä ei anneta, niin lokalisointia ei renderöidä, muuten se renderöidään annetun id:n perusteella.



Kuva 2. Kuva virheestä Chromen DevTools-konsolista.

Iltapäivällä kokoonnumme kehittäjien kanssa yhteiseen palaveriin, jonka sovimme groomingin aikana. Tarkoitus olisi keskustella datan liikkumisesta frontendin ja backendin välillä ja missä muodossa datan pitäisi tulla. Sovimme myös palaverissa, mitä logiikkaa teemme näiden osalta frontendin puolella ja sekä mitä backendin puolella.

Saan päivän päätteeksi molemmat käyttöliittymät valmiiksi, mutta olen aika varma, että jotain optimointia pitää tehdä, sillä olen tehnyt uutta koodia noin 400 riviä, joten jotakin on

varmasti jäänyt huomaamatta. Sovin kollegan kanssa, että avaan tästä pull requestin versionhallintaamme, jotta hän voi katselmoida koodin ja voimme aamulla keskustella voisiko jotain tehdä toisin.

Keskiviikko 4.3.2020

Tänään tavoitteena on, että saan mahdolliset korjaukset valmiiksi EU:n sisäisiin ja ulkomaistenmaksujen käyttöliittymiin. Jos saan ne nopeasti valmiiksi, niin sen jälkeen voisin lisätä lokalisointien määrää käyttöliittymissämme, sillä tällä hetkellä siellä on paljon kovakoodattuja tekstejä pelkästään suomeksi.

Aloitan päivän keskustelemalla tiimikollegan kanssa, joka katselmoi eilisen päivän muutokseni. Hän tosiaan löysi muutamia kohtia, joita voisi optimoida. Olen sekoittanut logiikkaa keskelle Reactin JSX:ää, mikä tekee koodista vähän huonommin luettavaa. Koodista löytyi myös muutama tyypitys, jotka ovat vähän pidempiä ja samaa tyyppiä käytetään muutamassa eri paikassa, joten näistä tyypeistä voisi tehdä yhden uudelleenkäytettävän tyypin. Molempien käyttöliittymien renderöintiä ohjataan myös pelkästään kovakoodatulla boolean-arvolla tällä hetkellä. Tälle voisi mahdollisesti tehdä jotain ja sovinkin, että keskustelemme tiimimme pääsuunnittelijan kanssa myöhemmin tänään asiasta.

Saan muutokset tehtyä, mutta jälleen minulla oli haasteita tyypitysten kanssa. Tein kaksi interface tyyppistä oliota ja yhden tyypin, joka laajentaa näitä interface olioita. Jostain syystä, kun yritän käyttää tätä yhtä päätyyppiä, niin koodi ei tunnista esimerkiksi yhden interfacen sisällä olevaa id-attribuuttia. Yritän käyttää Googlea ja löytää vastaavia toteutuksia, mutta mistään löytämästäni ei ole apua, joten päätän tehdä niistä vain erilliset tyypit nyt tällä hetkellä ja palata niihin mahdollisesti myöhemmin.

Iltapäivällä keskustelen pääsuunnittelijan kanssa EU:n sisäisten ja ulkomaanmaksun renderöintilogiikasta. Käymme läpi, missä tilanteissa kumpaakin käyttöliittymää tarvitaan ja mistä tämä tieto saadaan, minkä perusteella toinen näytetään. Tulemme siihen tulokseen, että frontendin pitäisi lähettää backendiin kutsu aina syötetystä tilinumerosta ja backendiin pitäisi tehdä logiikka, joka tunnistaa tilityypin annetun tilinumeron perusteella. Jos tili on esimerkiksi ulkomainen, niin renderöidään ulkomaisenmaksun käyttöliittymä. Tätä ei ole vielä toteutettu backendiin, joten teemme tästä tiketin JIRAan ja palaamme siihen luultavasti ensi viikon alussa.

Päivän tavoitteet tuli täyteen, kun sain EU:n sisäisten ja ulkomaistenmaksujen käyttöliittymät yhdistettyä versionhallinnassa päähaaraamme. Päivän aikana kompuroin jälleen tyy-

pitysten kanssa ja huomaan, että minun pitäisi löytää jotain materiaalia, josta voisin opiskella hieman lisää Typescriptin tyypityksistä ja miten niitä voisi laajentaa.

Torstai 5.3.2020

Tämä päivä tulee olemaan suhteellisen koodintäyteinen päivä, sillä päivälle ei ole varattu mitään muita palavereita päivittäisen Scrum Dailyn lisäksi. Päivän tavoitteena onkin siis saada korvattua kaikki kovakoodatut tekstit lokalisoiduilla teksteillä, jotta sovelluksemme taipuisi myös englanniksi ja ruotsiksi. Tämän pitäisi olla suhteellisen selvä tehtävä, sillä oikeat tekstit sekä suomeksi, ruotsiksi, että englanniksi pitää vain katsoa määrittelydokumentaatiosta ja lisätä ne koodiin. Lähden käymään läpi käyttöliittymiä yksi kerrallaan korvaten aina vastaantulevan kovakoodatun tekstin samalla, kun tarkkailen, onko koodissamme toistuvuutta, jota voisi mahdollisesti refaktoroida uudelleenkäytettäväksi komponentiksi.

Lokalisointien teko tulee valmiiksi ja teen siitä pull requestin versionhallintaan, jotta tiimikollegani saavat katselmoida muutokseni. Muutokset hyväksytään ja odotan, että ne päätyvät testiympäristöön. Kun tämä on valmistunut, testaan, toimiiko lokalisoinnit kuten pitääkin. Valitsen portaalista kieleksi englannin ja kirjaudun sovellukseemme. Huomaan, että kaikki on edelleen suomeksi, joten jossakin on vika. Ensimmäisenä tarkistan funktion millä haemme lokalisoinnin käyttöliittymästä. Koodin perusteella lokalisointi haetaan sivuston HTML-dokumentin ’language’-attribuutista. Tämä ei ole oikea paikka enää hakea sitä, sillä lokalisointitieto palautuu nykyään /self-endpointista. Refaktoroin funktion kutsumaan kyseistä endpointia ja nyt lokalisointi palautuu oikein ja sivu kääntyy käyttäjän valitsemalle kielelle.

Huomaan vielä, että sovelluksemme renderöi virheviestin ”Tilitietoja ei voitu hakea” hetkellisesti, kun sovelluksen aukaisee. Tiedän heti, mistä tämä johtuu. Renderöintilogiikassamme ei ole otettu huomioon sitä, että tilitietojen haussa kestää noin 1-2 sekuntia. Tämän hetken tilitiedot ovat tyhjät, joten käyttöliittymä renderöi yllä olevan virheviestin hetkeksi, kunnes tilit palautuvat rajapinnasta. Saan korjattua ongelman lisäämällä latausanimaation, jolloin se pyörii sen aikaa, kunnes tilitiedot ovat haettu onnistuneesti. Tähän onneksi on valmiina komponentti, jonka lisään käyttöliittymään. Muutan logiikan siten, että jos tilitietojen haussa palautuu jokin virheviesti, niin käyttäjälle näytetään kyseinen virhe. Muuten odotetaan hetki, että tilit latautuvat sivulle.

Asiakas haluaa huomenna taas esitellä sovellustamme muille yrityksen edustajille, joten tämän vuoksi tänään yritettiin tehdä sovelluksesta vielä hieman nätimpi. Toisaalta tämä on

hyvä juttu ja toisaalta ei. Näin monen muutoksen vieminen edellisenä päivänä testiympäristöön, josta sovellusta esitellään ei ole ehkä se viisain idea, sillä jos jotain menee rikki muutosten takia, niin siinä tulee pian hyvin kiire korjata viat, jos niitä edes saadaan korjatuksi tähän demotilaisuuteen mennessä. Tällä kertaa asiat näyttäisivät toimivan ilman suurempia ongelmia, joten toivotaan, että ne pysyvät siten myös huomisessa demossa.

Perjantai 6.3.2020

Tänään tavoitteena on refaktoroida lokalisoinnin hakeva funktio sekä keskittyä AWS-sertifikaatin opiskeluun. Tiimikollega oli illan aikana saanut valmiiksi API clientin refaktoroinnin. Käyttämämme React Query-kirjasto lähetti noin 20 sekunnin välein automaattisesti uuden pyynnön backendiin, sekä renderöi samalla kaikki komponentit käyttöliittymässä. Tämä vaikutti suuresti suorituskykyyn, joten olimme päättäneet, että siirrymme käyttämään Reactin natiivia contextia.

Lähden tutkimaan sekä Reactin dokumentaatiosta että tiimikollegani tekemästä integraatiosta, miten context toimii, sillä en ole koskaan aikaisemmin käyttänyt sitä. Reactin dokumentaatio on onneksi hyvin ajan tasalla sekä sisältää helppoja esimerkkejä, minkä perusteella lokalisointifunktion refaktorointi ei ole kovin iso työ. Saan tämän refaktoroinnin tehtyä, mutta olemme sopineet tiimin kesken, että tänään ei pusketa mitään testiympäristöön, jotta asiakas saa demota rauhassa sovellusta. Lokalisoinnin testaus jää siis maanantaille.

Päivä jatkuu AWS-sertifikaatin opiskelulla ja siihen kuuluvan kurssin materiaalin läpikäynnillä. Tänään vuorossa olisi osat: Tietokannat AWS:ssä sekä Route 53-palvelu. Molemmat osat ovat sisältöön osin tuttuja, sillä olen tehnyt AWS:ssä DynamoDB tietokantoja ennenkin, mutta esimerkiksi Redshift ja Elasticache ovat minulle vielä tuntemattomia. Route 53-avulla voi ostaa omia domaineja, minkä kautta olenkin ostanut jo muutaman domainin, joten tämä osa ei tarjoa paljoakaan mitään uutta.

Viikon viimeinen päivä oli jälleen onnistunut, sillä sain lokalisointifunktion refaktoroinnin korjattua sekä etenin mukavasti AWS sertifikaatin opiskelussa. Opin paljon uutta varsinkin tietokantojen osalta ja miten esimerkiksi Elasticachella saa tallennettua välimuistiin tavaraa Redisin avulla. Vielä kun pääsisin itse käytännössä testaamaan, miten nämä palvelut toimivat, niin ymmärtäisin vielä vähän syvemmin, miten niitä oikeasti käytetään.

Viikkoanalyysi

Tällä viikolla työstin jälleen enimmäkseen frontendiä. Viikon aikana pääsin tekemään paljon uusia käyttöliittymiä, mutta mukana oli myös paljon olemassa olevan koodin refaktorointia ja optimointia. Tällä viikolla opin, että minun pitäisi aina ennen työn aloittamista ottaa askel taaksepäin ja miettiä miten lähden haluttua ominaisuutta rakentamaan. Viikon alussa jouduin kirjoittamaan lähes kahden käyttöliittymän koodin alusta, sillä en ollut suunnitellut niitä tarpeeksi hyvin. Tässä meni suhteellisen paljon työaikaa hukkaan. Toinen asia, jota olisin voinut suunnitella paremmin, oli se, että en pilkkonut töitä tarpeeksi. Tiivistin näiden kahden käyttöliittymän refaktoroinnin sekä muutaman muun korjauksen yhdeksi isoksi kokonaisuudeksi. Lopputuloksesta tuli yli 400 riviä pitkä ja lähes 15 eri tiedoston pituinen pull request versionhallintaan, mitä kenenkään ei ole miellyttävää katselmoida. Tähän oli myös jäänyt virheitä, sillä kokonaisuus oli paisunut niin isoksi, että sen seasta oli hankala huomata yksittäisiä asioita, mitä voisi optimoida.

Luin artikkelin hyvän kehittäjän ajattelutavan perusteista, jossa yksi kohta käsitteli juuri edellä mainittuja ongelmia. Ennen koodauksen aloittamista pitäisi ensimmäisenä ymmärtää, mitä kyseisessä tehtävässä halutaan tehdä ja vaikka kirjoittaa ratkaistava ongelma auki. Tämän jälkeen pitäisi hetki rauhassa miettiä ja analysoida, miten tämä olisi paras ratkaista. Viimeinen kohta on tehtävän jakaminen osiin. Tällöin siitä saa pienempiä osia, jotka on helpompi yhdistää isommaksi kokonaisuudeksi (Yuruk 2019).

Viikon aikana onnistuin mielestäni hyvin löytämään kehityksen aikaiset bugit. Koska olen ollut projektin alusta alkaen rakentamassa frontendiä, alan tuntemaan koodin lähes läpikotaisin ja tiedän nopeasti, mikä komponentti tekee mitäkin. Tämä ehkä myös kertoo siitä, että kokemus alkaa näkymään. Monien rivien mittaiset virhe stack tracet eivät enää pelota, vaan alan pikkuhiljaa harjaantumaan siihen, että löydän rivien välistä virheen alkuperän ja tiedän, mistä kyseistä virhettä kannattaisi lähteä tutkimaan.

Viikon päätteeksi pääsin myös opiskelemaan ja oppimaan konkreettisesti lisää AWS:n palveluista. Saamme käyttää viikoittaisesta työajasta 15 % kehittymiseen ja onkin hienoa, että työnantaja kannustaa omia konsultteja kehittämään omaa osaamistaan, sillä onhan siitä loppuen lopuksi työnantajalle hyötyä, että he saavat entistä pätevämpiä työntekijöitä. Kunpa vain olisi enemmän aikaa vielä opiskella varsinkin DevOps-menetelmiä, sillä ne ovat viime aikoina kiinnostaneet minua entistä enemmän. Valitettavasti työn ja opintojen ohessa en ehdi kunnolla perehtymään edellä mainittuihin menetelmiin. Työn lomassa lyhyillä tauoilla voin lukea artikkeleita tai oppaita kiinnostaviin aiheisiin liittyen ja niihin voin sitten palata myöhemmin. Tällä hetkellä myös vapaa-aika iltaisin on hyvin vähissä, sillä

koulutehtävät vievät oman aikansa jokaisesta illasta. Valmistumisen jälkeen olisi kuitenkin tavoitteena herättää vähälle huomiolle jääneitä omia keskeneräisiä ohjelmistoprojekteja, kun vapaa-aikaa pitäisi olla enemmän.

3.4 Seurantaviikko 4: 9-13.3.2020

Maanantai 9.3.2020

Tämän päivän tavoitteena on testata perjantaina valmiiksi tullutta lokalisoinnin refaktorointia, jota en silloin päässyt testaamaan. Päivä on muuten täynnä palavereita, joten päivä kuluu pitkälti niiden parissa.

Huomaan ensin, että koodimuutokseni ei ole kääntynyt onnistuneesti Jenkins-työssä. Lokeja tutkimalla huomaan, että lokalisointifunktiossa käytetty muuttuja viittaa jostain syystä globaaliin muuttujaan, mitä koodissa ei pitäisi olla yhtään. Luen myös SonarQuben raportin ja huomaan, että kyseisen funktion yksikkötestissä ei ole 100 % kattavuus, vaan minulta on jäänyt testaamatta kokonaan yhden if-haaran testaus. Lähden tutkimaan ensin muuttujaa, joka oli mennyt rikki CI/CD-putkessa. Huomaan, että minulla on syntaksivirhe funktion parametreissa. Koska käytämme ES6-version myötä tulleita nuolifunktioita, niin funktion sisään tulevat parametrit täytyy ”tupla-tyypittää”, sillä tyypitämme sekä funktion että myös parametrit. Nyt olin tyypittänyt vain parametrin enkä funktiota (Kuva 3). Lisään yksikkötesteihin myös testin, joka testaa if-haaran, joka oli jäänyt testaamatta.

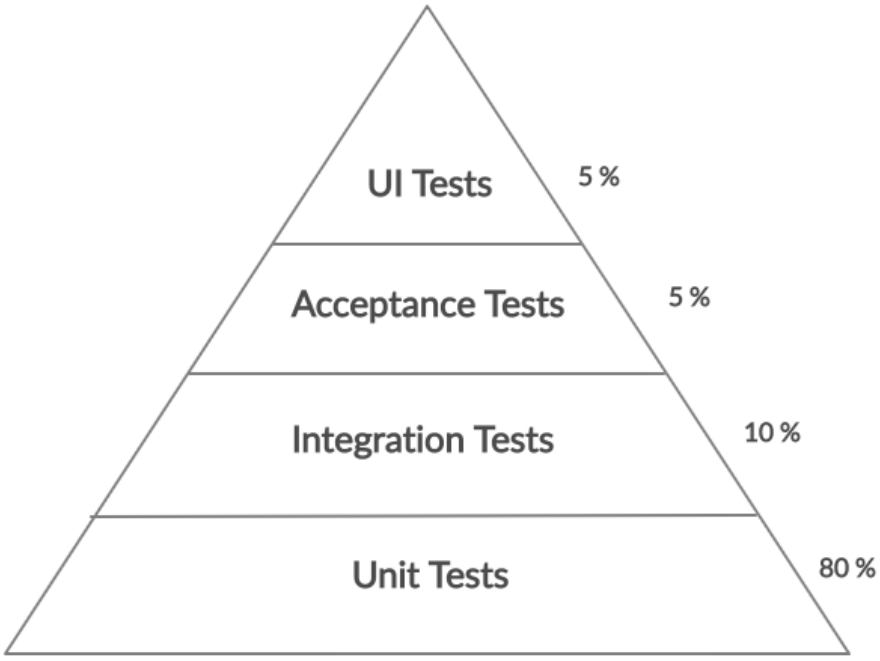
```
export const useLocale: (self: UseSelf) => string = (self: UseSelf): string => {
```

Kuva 3. Tuplatyypitys Typescriptissa.

Loppupäivä meneekin palaveriputkessa. Ensimmäisenä on palaveri, jossa käsittelemme käyttöliittymissä näytettäviä virheitä ja miten niitä tulisi näyttää. Keskustelua herättää, millä tasolla virheiden käsittely pitäisi olla sovelluksen tuotantoversiossa, sillä sovelluksen pitäisi olla valmis parin kuukauden kuluttua.

Seuraava palaveri käsittelee testauksen automatisointia. Lähdemme ensimmäisenä keskustelemaan siitä, mitä meidän tulisi automatisoida. Tiimiläinen alustaa keskustelun ja näyttää testipyramidia, mistä saamme näkemystä siihen, kuinka paljon testausta eri tasoilla pitäisi olla (Kuva 4). Käymme läpi eri integraatiot, mitä sovelluksessa on tai tulee olemaan, ja keskustelemme jokaisesta yksitellen, kuinka ne pitäisi testata. Testaajamme mielestä eri integraatioiden automatisointi ei ole tarpeellista juuri nyt, vaikkakin se vähentäisi hieman hänen työmääräänsä. Tämän tekeminen veisi huomattavan määrän jonkun kehittäjän työaikaa. Päätämme, että pidämme yksikkötestauksen edelleen vähintään 80 % kattavuudessa, sekä teemme käyttöliittymään Robot Frameworkilla automaatiotestit, jotka

testaavat end-to-end tyyppisesti. Minä lupaan ottaa näiden robot-testien kirjoittamisesta vastuun, sillä minulla on niiden kirjoittamisesta eniten kokemusta koko tiimistä.



Kuva 4. Testiautomaatio pyramidi ja kuinka monta prosenttia testikokonaisuudesta jokaisen osa-alueen pitäisi olla (mukaillen Vashishtha 2015).

Vaikka päivä oli palaverivoittoista, niin sain valmiiksi korjaukset lokalisoinnin refaktorointiin, jotta se meni onnistuneesti läpi testiympäristöömme. Palavereiden osalta saimme asioita hyvin päätettyä ja tiedämme jälleen suuntaviivat, kuinka sekä virheiden että testien automatisoinnin osalta edetään.

Tiistai 10.3.2020

Lupauduin eilen palaverissa ottamaan vastuun testiautomaatiosta, joten tänään tavoite on tehdä robottitestien infra valmiiksi ennen kuin iltapäivän palaveriputki alkaa. Tälle ei vielä ole tehty tikettiä JIRAan, joten aloitan siitä ja kirjoitan tiketin kuvaukseen kaiken, mitä pitää tehdä. Tehtäviin kuuluu mm. Git repositoryn luominen, robottitestien kansio- ja tiedostorakenteen luominen sekä yhden testin luominen, jolla voidaan todistaa, että robottitestien pohja toimii kuten pitää ja sitä pystytään laajentamaan siitä.

Aloitan työn tekemällä Git repositoryn, jonka jälkeen teen uuden kansion koneelle, johon lähden rakentamaan infraa. Robot Frameworkissa on yleensä yksi Resources.robot-

niminen tiedosto, jossa tuodaan käytettävät kirjastot projektiin. Tämän lisäksi luon yhden tiedoston, mikä sisältää sovelluksemme testisarjan. Teen myös tiedoston nimeltä variables.py, jonne tallennettaan kaikki testeissä käytettävät muuttujat. Tiedoston päätteestä voi myös huomata, että se on Python tiedosto. Tämä siitä syystä, että muuttujien sisällä pitää mahdollisesti tehdä logiikkaa, mikä onnistuu Pythonin avulla. Sille löytyy laajasti kirjastoja, joilla voi tehdä myös omia kompleksisempia metodeja, mitä ei välttämättä Robot Frameworkin omasta kirjastosta löydy.

Infran rakennus sujuu ilman ongelmia, mutta on jälleen viikoittaisen groomingin aika. Groomingissa käymme läpi tällä kertaa Hyväksymättömät maksut -sivun toiminnallisuutta. Käymme läpi sivun määrittelyä, jota tiimimme Product Owner esittelee ja keskustelemme vastaan tulevista asioista, jotka herättävät kysymyksiä. Huomaan, että määrittely on jälleen hyvin pitkä ja sitä on selvästi hiottu viimeiseen saakka. Tätä asiaa pitäisi mielestäni vielä työstää enemmän, sillä määrittelyn tekemiseen menee suunnattoman paljon aikaa, kun sitä yritetään hioa viimeiseen yksityiskohtaan saakka. Usein, kun toteutusta lähdetään koodaamaan, jokin asia toimii eri tavoin kuin on luultu, jolloin määrittelyä joudutaan muokkaamaan jälleen. Aion ottaa tämän aiheen puheeksi iltapäivän retrossa.

Groomingin jälkeen on demo, jossa tarkoituksena on esitellä EU:n sisäisten- ja ulkomaistenmaksujen käyttöliittymät ja lokalisointi asiakkaalle. Aloitan demon kirjautumalla asiakkaan palveluun valitsemalla kieleksi ruotsin ja navigoimalla sovellukseemme. Sovelluksemme kääntyy hienosti ruotsiksi ja asiakas on selvästi tyytyväinen. Esittelen myös nopeasti EU:n sisäisten ja ulkomaisten maksujen käyttöliittymiä, mihin oltiin yleisesti myös tyytyväisiä. Designer antoi palautetta muutamista tyylillisistä asioista, mutta kerroin, että voimme käydä näitä tarkemmin läpi lähempänä julkaisua. Asiat tulevat vielä luultavasti muuttumaan, joten niihin ei kannata juuri nyt käyttää liian paljoa aikaa.

Päivän viimeinen palaveri on tiimin retrospektiivi, missä käymme jälleen läpi mitä on tehty ja missä voisimme parantaa. Tänään retron aiheena on, mikä edistää ja mikä taas toisaalta hidastaa kehitystä, jotta sovelluksemme olisi julkaisuvalmis toukokuun lopussa. Nostan yhtenä hidasteena esille tämän liian pitkälle menevän määrittelyn. Kerron oman mielipiteeni siitä, kuinka paljon tämä hidastaa tulevien töiden aloittamista ja miten tämä muistuttaa enemmän entisajan Waterfall-kehitysmenetelmää kuin nykypäivän ketteriä menetelmiä. Tiimi on hyvin samaa mieltä kanssani ja Scrum Masterimme lupaakin ottaa asian puheeksi tiimin Product Ownerin kanssa, sillä he yleensä ovat ne henkilöt, jotka näitä tulevien töiden määrittelyjä tekevät.

Päivän päätteeksi jatkan vielä testiautomaation infran tekemistä valmiiksi, jotta voisin jatkaa huomenna seuraaviin tehtäviin. Teen robottitestin, mikä kirjautuu yrityksen portaaliin, navigoi sovellukseemme ja vahvistaa olevansa oikealla sivulla. Rakentamani infra toimii odotetusti, joten lisään kaiken versionhallintaan ja teen siitä pull requestin aikaisemmin aamulla tekemääni repositoryyn, jotta tiimikollegani saavat katselmoida sen.

Keskiviikko 11.3.2020

Tänään tehtävänä on selvittää, miten saan sulautettua vahvistuspalvelutiimin tekemän käyttöliittymän meidän käyttöliittymäämme. Tehtävä tulee olemaan varmasti yksi haastavimmista, mitä olen tähän mennessä tehnyt, sillä kyseinen käyttöliittymä on tehty JQuerylla, kun taas meidän käyttöliittymämme on React/Typescript. En ole koskaan tehnyt mitään vastaavaa, joten se tulee vaatimaan työaikaani.

Aloitan tehtävän tutkimalla vahvistuspalvelun dokumentaatiota. Asiakasyrityksessä kaikki tiimit käyttävät Confluence-palvelua dokumentaation säilömiseen, joten se on helppo löytää. Valitettavasti dokumentaatiota ei ole riittävästi, joten se ei riittävästi kuvaile, miten kyseinen käyttöliittymä toimii. Käytän dokumentaation tutkimiseen muutaman tunnin työaikaa, mutta sieltä ei valitettavasti löydy mitään apua, kuinka lähtisin ongelmaa ratkaisemaan. Keskustelen Scrum Masterimme kanssa asiasta ja sovimme, että hän ottaa selvää, olisiko joku muu tiimi olisi tehnyt vastaavanlaista tehtävää. Sillä aikaa, kun odotan tätä selvitystä, käyn läpi versionhallintaamme tulleita pull requesteja. Nämä ovat pääosin pieniä muutoksia, joten hyväksyn ja yhdistän ne päähaaraamme.

Iltapäivällä saan Scrum Masterilta yhteyshenkilön toisesta tiimistä, joka on tehnyt vastaavanlaisen toteutuksen. Pistän hänelle viestiä ja kyselen heidän toteutuksestaan. Hän lähettää linkin heidän lähdekoodiinsa, mutta minulla ei valitettavasti ole oikeuksia nähdä sitä. Hänkään ei niitä voi antaa, joten joudun odottamaan huomiseen, että näen lähdekoodista, miten toteutus on tehty. Loppupäivä menee eri dokumentaatioiden läpikäynnissä, sekä katson meidän legacy-järjestelmämme lähdekoodia, jotta ymmärtäisin vähän paremmin, miten kyseinen käyttöliittymä toimisi.

Päivä oli hyvin opiskelun täyteinen, kun yritin saada selvää, miten vahvistuspalvelun käyttöliittymän saisi implementoitua meidän sovelluksemme käyttöliittymään. Tehtävä vaikuttaa kyllä todella haastavalta ja jopa hieman suurelta yhdelle kehittäjälle, mutta ehkä huomenna asiat selkenevät hieman.

Torstai 12.3.2020

Tänään päivä alkaa pitkästä aikaa käynnillä Pasilan kampuksella, jossa käyn tekemässä muutamat kurssitehtävät. Konsultin työ on siitä mukavaa, että työaika on hyvin joustavaa. Jos tulee henkilökohtaisia menoja, niitä voi yleensä sovittaa hyvin omiin työaikatauluihin. Tärkeää on kuitenkin aina osallistua oman tiimin palavereihin.

Palattuani työpaikalleni tutkin edelleen, miten vahvistuspalvelun käyttöliittymän saisi integroitua meidän käyttöliittymäämme. Eilinen dokumentaation tutkiminen ei oikeastaan auttanut, joten lähden tutkimaan toisen tiimin yhteyshenkilöltä saatua lähdekoodia, kuinka he ovat toteuttaneet sen. Asiaa hankaloittaa se, että kyseinen toteutus on tehty vanhalla Reactilla ja Javascriptilla. Käytössä oleva React-versio käyttää vielä luokkia sekä Redux-kirjastoa, kun taas meidän käyttöliittymämme on modernia Reactia sekä Typescriptia. Pyrin kuitenkin ymmärtämään, kuinka tämä toteutus kääntyisi meidän sovelluksessamme käytettäville kielille.

Seuraavaksi selvittelen, kuinka kyseisen käyttöliittymän lisäys on tehty, joten lähden replikoimaan sitä meidän käyttöliittymäämme. Työ sujuu pääsääntöisesti hyvin, mutta pian tulee seinä vastaan. Huomaan, että tarvitsen backend-integraatiota myös, sillä backendin pitäisi kutsua vahvistuspalvelun rajapintaa, joka palauttaa käyttöliittymän sekä vahvistustapahtuman aina frontendiin saakka. Keskustelen tästä tiimimme pääsuunnittelijan kanssa ja sovimme, että pidämme maanantaina palaverin kaikkien kehittäjien kanssa ja käymme tämän läpi, koska työ on turhan iso yhdelle kehittäjälle.

Koska vahvistuspalvelun integraatio on nyt jäissä maanantaihin asti, tarkistan JIRAsta, mitä voisin tehdä seuraavaksi. Huomaan, että siellä on kaksi hyvin samantyyppistä tehtävää. Käyttäjästä palautuva tieto tulee vielä mockkidatasta ja se pitäisi korvata oikealla rajapinnoista palautuvalla datalla. Sama pitäisi tehdä myös maksuille, jonka jälkeen kaikki sovellukseen tuleva data pitäisi tulla rajapinnoista.

Tänään en enää niitä ehdi koodaamaan, mutta aloitan suunnittelun, kuinka nämä molemmat tehtävät voisi toteuttaa, jotta voin heti huomenna aloittaa työt. En tunnista kummassakaan tehtävässä mitään yhtäläisyyksiä, jotta voisin tehdä yleiskäyttöisiä komponentteja, sillä molemmat kutsuvat eri rajapintoja ja eri osoitteita. Molempien toteutus tulee kuitenkin olemaan hyvin samankaltainen kuin tilien haun yhteydessä, joten molempien tehtävien toteutuksessa ei pitäisi mennä kovinkaan kauaa.

Päivä tulee päätökseen ja suuntalinjat huomisen tehtäville ovat selvät. Huomaan, että olen kehittynyt hyvin työn suunnittelun osalta verrattuna viime viikkoon, jolloin aloin tekemään tehtävää lähes heti ilman kunnon suunnittelua, kuinka työ tulisi tehdä.

Perjantai 13.3.2020

Päivän tavoitteena on päästä tekemään mahdollisimman pitkälle eilen suunnittelemaani kovakoodatun datan korvaamista oikealla datalla. Koska sain eilen suunnitelman valmiiksi, pitäisi tänään tehdä pelkästään toteutus ja mahdollisesti testata sitä, mikäli saan kaiken valmiiksi.

Keväällä 2020 puhjennut koronaviruspandemia alkaa näkymään myös työarjessamme. Suurin osa tiimistä työskentelee sen vuoksi toistaiseksi etänä. Jotta töitä pystyy tekemään, pitää ottaa VPN-yhteys asiakkaan verkkoon. Asiakasyritys on ohjeistanut suurinta osaa henkilöstöä tekemään etätöitä, mikäli mahdollista, mikä aiheuttaa ruuhkaa VPN-verkkoon ja ylikuormittaa sen. Pääsen onneksi itse kirjautumaan ja pystyn aloittamaan työt. Käymme aamun Scrum Dailyssa kaikkien työt läpi ja keskustelemme tiimiläisten kanssa, miten koronavirus alkaa vaikuttamaan työhömme. Kaikki eivät ole päässeet paikalle, sillä osa tiimiläisistä ei ole saanut yhteyttä VPN:n kautta. Dailyssa huomaamme muutaman tiimiläisen kesken, että tehtävämme riippuvat toisistaan. Kollegani alkaa tekemään maksujen lähettämistä frontendista backendiin, mihin hän tarvitsee nyt aloittamaani maksujen hakua backendista. Minä taas tarvitsisin toisen kehittäjän muutoksia, jotka on katselmoitu ja joihin pitäisi tehdä pieniä korjauksia. Kyseinen kehittäjä ei tällä hetkellä pääse verkkoon, joten minun pitää hypätä hänen tilalleen ja korjata ongelmat koodikatselmoinnissa, jotta saamme koodin versionhallintaan päähaaraamme ja pääsen aloittamaan maksujen haun.

Saan tehtyä korjaukset suhteellisen nopeasti ja tiimikollega katselmoi ja hyväksyy ne. Pystyn viimeinkin aloittamaan maksujen haun backendista frontendiin. Tehtävässä minun pitää käyttää hyödyksi Reactin Contextia, mikä on minulle jo tuttu lokalisoinnista sekä tilien haun kautta. Teen maksujen haulle oman contextin, sekä kirjoitan sille yksikkötestit. Testaan myös, että koodi kääntyy sekä kaikki yksikkötestit menevät edelleen läpi. Ehdin saada tämän työn valmiiksi ja teen siitä pull requestin versionhallintaan, jotta tiimikollega pystyy aloittamaan maksujen lähettämisen työt.

Tänään en saanut ihan kaikkia työtehtäviäni valmiiksi, mitä olisin halunnut. Jouduin korjaamaan toisen kehittäjän koodissa olevia virheitä, sillä hän ei niitä itse päässyt korjaamaan. Vasta tämän jälkeen pääsin aloittamaan omat työni. Sain maksujen haun valmiiksi,

mutta en päässyt vielä testaamaan sitä testiympäristössä, enkä ehtinyt aloittamaan työtä, mikä hakee käyttäjän tiedot frontendiin. Mielenkiinnolla odotan, mitä maanantai tuo tullessaan ja miten koronaviruspandemia tulee vaikuttamaan ensi viikolla työskentelyymme.

Viikkoanalyysi

Viikko alkoi korjaamalla virheitä lokalisointifunktiossa. Haasteeksi kyseisessä tehtävässä tuli jälleen Typescriptin syntaksi. Sovelluksessamme on käytössä tiukka tyypitys, minkä vuoksi myös funktiot tyypitetään. Tämän on määritellyt eräs tiimimme sovelluskehittäjä rakentaessaan sovelluksen infrastruktuuria, ja hän ei valitettavasti enää ole tiimissämme. Yritin etsiä netistä materiaalia, mikä hyöty funktioiden tyypityksestä on, sillä enemmän meitä kiinnostaa se, mitä se palauttaa. Löysin ainoastaan samanlaisen esimerkin Typescriptin dokumentaatiosta, jossa kerrotaan, että funktiota ei ole tarve tyypittää, sillä Typescript osaa itse päätellä tämän perustuen palautuksen tyyppiin (Typescript Handbook). Yritän löytää lisää materiaalia tulevina viikkoina asiasta, jotta tietäisin, onko tämä tosiaan tarpeellista ja mikä tämän mahdollinen hyöty on.

Seuraavaksi pääsin rakentamaan testiautomaatiota sovelluksellemme täysin tyhjästä. Tämä oli todella mieluisa tehtävä, jossa tunnen olevani vahvoilla ja samalla opin myös uutta, kun saan rakentaa kaiken alusta. Käytin testiautomaation infran rakentamisessa viitteenä Robot Frameworkin Githubista löytyvää dokumentaatiota siitä, miten testejä tulisi rakentaa. Dokumentaatiossa kerrotaan muun muassa, kuinka testit tulisi dokumentoida ja minkälaisella rakenteella eri testit pitäisi laatia. Käytin tehtävän tukena myös Dale Emeryn kirjoittamaa opasta, jossa kuvataan, kuinka rakentaa ylläpidettäviä automaattitestejä. Sain oppaasta hyviä ideoita, kuinka pystyn tekemään uudelleenkäytettäviä keywordeja Robot Frameworkissa sekä miten pystyn pitämään muuttujat selkeinä, jotta koodi pysyy helposti ymmärrettävänä myös henkilölle, joka näiden parissa tulee jatkossa työskentelemään. (Robot Framework, Emery 2009)

Viikon vaikein työ oli ehdottomasti vahvistuspalvelun käyttöliittymän upottaminen sovelluksemme käyttöliittymään. Lähdin ensimmäiseksi tutkimaan dokumentaatiota ja suunnittelemaan, miten tämä olisi mahdollista suorittaa. Tiesin myös, että tehtävä tulee olemaan todella haastava, sillä en ollut koskaan tehnyt mitään vastaavaa. Olen tähän mennessä oppinut sen, että dokumentaatio on tärkeää pitää ajantasaisena. Vaikka käytin lähes kokonaisen työpäivän dokumentaation lukemiseen, en saanut siitä tarpeeksi ohjeita, miten tätä tehtävää olisi mahdollista lähteä toteuttamaan. Dokumentaation pitäisi olla aina sellaista, että kuka tahansa sitä lukeekin, pystyy sen perusteella lähteä tekemään toteutusta. On työlästä, jos pitää aina käydä lähdekoodista tutkimassa, miten mikäkin funktio toimii.

Aliarvioin aluksi myös työn vaatiman työmäärän. Ajattelin aluksi, että saan tämän työn luultavasti tehtyä yksin. Kun selvittelin asiaa enemmän, niin huomasin, että tarvitsen integraatioita backendiin sekä myös apua frontendin kanssa, että saan yhdistettyä tämän vanhan teknologian uuteen.

Tällä viikolla en valitettavasti ehtinyt työn ohella opiskelemaan CSS-kurssia taikka AWS-sertifikaattia varten. AWS on sen verran suuri palvelu ja sertifikaattia varten käydään läpi monia osa-alueita, että olisi tärkeää pitää yllä viikoittaista oppimista, jotta asiat eivät pääse unohtumaan. Ensi viikolla otankin tavoitteeksi kerrata viime viikolla käydyt tietokannat ja välimuistiin tallentamisen AWS:ssä, jotta ne palautuvat paremmin mieleen ennen kuin lähden opiskelemaan seuraavaa osa-aluetta. CSS-kurssin pyrin aloittamaan myös lähiviikkojen aikana ja olenkin päättänyt jo, että tulen rakentamaan oman portfoliosivuston kyseisen kurssin rinnalla, jotta pääsisin hyödyntämään heti oppimaani käytännössä. Pystyisin myös käyttämään AWS:n palveluita kyseisen sivuston rakentamiseen, jotta myös sertifikaattiin opiskeltavat asiat pysyisivät paremmin muistissa.

3.5 Seurantaviikko 5: 16-20.3.2020

Maanantai 16.3.2020

Sain viime perjantaina maksujen haun valmiiksi ja sitä olisi tarkoitus testata tänään. Päivälle on varattuna myös muutama palaveri, joten pyrin näiden välillä testaamaan maksujen haun toiminnallisuutta parhaani mukaan.

Ensimmäisenä lisään muutamia maksuja testidataamme, jotta voin testata, palautuuko maksut näennäiseltä JSON-serveriltä takaisin kuten pitääkin. Laitan lokaalin testiympäristön päälle ja laitan lokien seurannan päälle, jotta näen mitä backendista palautuu. Mitään maksuja ei tule näkyviin käyttöliittymään, joten jossain on ongelma. Huomaan, että lokeille tulee seuraavanlainen virheviesti: ”400 Missing the required parameter 'xRequestId' when calling getPayments”. Virhe on selkeä ja siitä ymmärtää heti, mistä on kyse. Rajapinta, josta haemme maksut odottavat saavansa xRequestId-nimisen parametrin, mitä ei lokaalissa testiympäristössä mene kutsun headereissa. Tämä on myös aika hankala lisätä lokaaleihin headereihin, sillä kyseinen parametri saadaan vain silloin, kun käyttäjä on kirjautuneena sovellukseen. Joudun miettimään hetken, jotta saan ongelman ratkaistua.

Seuraavaksi on vuorossa palaveri, jossa käymme läpi viime viikolla ongelmaksi muodostunutta vahvistuspalvelun käyttöliittymän upottamista omaan käyttöliittymäämme. Esittelen ensimmäisenä, mitä sain viime viikolla aikaan ja mitä sain selville dokumentaatiosta sekä lähdekoodista. Keskustelemme muiden kehittäjien kanssa, mitkä voisivat olla ratkaisuja ja kokeilemme palaverin aikana myös eri tapoja toteuttaa kyseinen implementaatio, mutta mikään niistä ei oikein toimi. Lupaan tutkia asiaa hieman myöhemmin, kun olen saanut muutaman muun työn alta pois ja palaan sen jälkeen asiaan.

Edellisen kokouksen aikana ehdin hetken miettiä, miten toteutan maksujen haun lokaalisti. Teen frontendiin funktion, mikä tarkistaa, onko kyseessä tuotanto- vai kehitysversio. Sen perusteella maksut haetaan joko oikeasta rajapinnasta tai palautetaan kovakoodatusta testidatasta. Teen muutoksista pull requestin versionhallintaan ja jään odottamaan, että se on testattavissa testiympäristössä.

Muutokset ovat nyt sisällä ja pääsen viimeinkin testaamaan maksujen hakua testiympäristössä. Teen maksun Postmanilla rajapintaan ja avaan käyttöliittymän huomaten, että koko sovellus kaatuu. Se ei ole hyvä juttu, sillä testiympäristöä käyttävät muutkin kehittäjät tiimissämme. Alan pikaisesti selvittämään, mitä virheitä konsoliin ilmestyy, jotta näkisin, mikä kaatoi sovelluksen. Konsolissa näkyy virhe: ”RangeError: invalid time value”. Haen

virhettä googlen avulla ja pian selviääkin, että jos Date-funktiolle antaa ”null” tai ”undefined” arvon, niin koko sovellus kaatuu. Käytämme eräpäivän formatoinnissa Date-funktiota, joten korjaan asian siten, että jos eräpäivää ei ole annettu, sille annetaan jokin kaukana menneisyydessä oleva päivämäärä. Lisään myös tälle yksikkötestin, jotta voin olla varma, että korjaukseni toimii kuten pitääkin eikä sovellus enää sen takia kaadu.

Päivä oli loppujen lopuksi onnistunut, sillä sain maksut näkymään testiympäristössä oikein. Kaikki tieto ei tullut kuitenkaan oikein backendista, joten sitä pitää selvittää myöhemmin. Sain joka tapauksessa todennettua, että frontendissä maksujen lajittelulogiikka toimii kuten pitääkin. Opin taas hieman uutta varsinkin bugien korjauksen osalta ja tuntuukin, että päivä päivältä osaan ratkoa niitä paremmin. Ymmärrän virheviestejä paremmin sekä tiedän, millä Googlen hakusanoilla internetistä voisi löytää apua.

Tiistai 17.3.2020

Tänään tarkoituksena olisi jatkaa eilistä tehtävää ja tutkia, miksi kaikki maksun tiedot eivät näy frontendissa, kun ne palautuvat backendista. Epäilen, että vika on backendissa. Rajapinnasta palautuvaa dataa ei lajitella oikeisiin JSON-kenttiin backendissa, joka tekee datasta yhden paketin, mikä palautetaan frontendiin. Tätä minun pitää testata testiympäristössä, että pystyn lokittamaan rajapinnan palauttaman datan sekä backendin palauttaman datan.

Ennen edellä mainitun työn aloittamista on kuitenkin viikoittaisen groomingin aika. Tämä on ensimmäinen päivä, kun kaikki työskentelevät etänä koronapandemian takia, kunnes toisin ohjeistetaan. Tulee olemaan mielenkiintoista nähdä, kuinka työskentelymme tulee muuttumaan nyt, kun kaikki työskentelevät kotoa käsin. Käymme groomingissa hyväksymättömien maksujen käyttöliittymän designia ja siihen liittyviä määrittelyjä läpi. Yksi tiimin kehittäjistä onkin aloittanut jo tämän työstämisen ja esittelee, mitä on saanut tähän mennessä aikaiseksi, ja mikä on muodostunut haasteeksi kehittämisen aikana. Käyttöliittymään pitäisi pian yhdistää minun aloittamani vahvistuspalvelun käyttöliittymän upotus ja kerronkin, että siinä on edelleen ongelmia, mutta voin tutkia huomenna sitä lisää.

Palaverin jälkeen lähden selvittämään aamulla tavoitteeksi otettua tehtävää. Yritän päästä testiympäristöömme, mutta vaikuttaa siltä, että se on jostain syystä kaatunut. Ilmoitan tästä tiimi-chatissamme ja lähdemme toisen kehittäjän kanssa tutkimaan lokeilta, mikä virheen mahdollisesti aiheuttaa. Yritämme myös käynnistää testiympäristön palvelinta uudestaan ja muuttaa sen konfiguraatiota siten, mikä on varmistettu toimivaksi ennen, mutta tämäkään ei auta. Lopulta löydämme virheen, että ympäristömme yrittää asentaa

jonkin riippuvuuden, mutta ei sitä jostain syystä löydä. Emme hallinnoi omaa testiympäristöämme, joten emme voi lähteä tätä selvittämään sen kummemmin. Kirjoitamme tiketin tiimille, joka hallinnoi kaikkia testiympäristöjä, jotta he voivat selvittää ja korjata tämän ongelman.

Tänään en saanut aamulla tavoitteeksi asetettua tehtävää valmiiksi testiympäristön ongelmien vuoksi. Toivottavasti ongelmat pystytään korjaamaan mahdollisimman pian, sillä testiympäristö on hyvin kriittinen osa testauksen kannalta. Sillä pystytään varmistamaan, että integraatiot ja datan liikkuvuus toimivat kuten pitääkin. Osaan nykyään selvittää virheviestien ja lokien avulla nopeammin, mistä bugit johtuvat. Tämänpäiväinen kuitenkin opetti, että ympäristötason bugit ovat haastavampia. Lokeja tulee tuhansia rivejä moniin eri tiedostoihin, joista pitää löytää se yksi virhe, mikä mahdollisesti on kaatanut koko ympäristön.

Keskiviikko 18.3.2020

Tänään olisi tarkoitus jatkaa vahvistuspalvelun käyttöliittymän tutkimista, kuinka sen saisi upotettua meidän käyttöliittymäämme. Maanantaina käydystä palaverista ei ollut hirveästi hyötyä työn edistämisen kannalta, joten tänään yritän löytää eri tapoja saada kyseinen käyttöliittymä toimimaan.

Löydän dokumentaatiosta repositoryn, mistä löydän käyttöliittymän lähdekoodin. Lähden tutkimaan tätä ja huomaan koodissa kommentin, mikä kertoo, että käyttöliittymä pitäisi importata Requirejs-kirjastolla. Lähden lisäämään tätä kirjastoa sovellukseemme ja lisään myös @types/requirejs-kirjaston, mikä tarvitaan ulkopuolisten kirjastojen tyyppien määrittelyyn. Tämän jälkeen yritän käynnistää sovellusta lokaalisti, mutta koodi ei käänny, sillä konsoliin tulostuu seuraava virhe: ”node_modules/@types/node/index.d.ts(5950,5): error TS2300: Duplicate identifier 'mod'.” Virheet node_modules kansiossa ovat yleensä todella hankalia selvittää, sillä kyseisen kansion kaikki tiedostot ovat ulkopuolisten kirjastojen tiedostoja ja niitä ei voi muokata.

Lähden googlettamaan virhettä ja tutkimaan, kuinka muut ovat ratkaisseet vastaavanlaisen ongelman. Löydänkin muutamia samankaltaisia virhetilanteita, mutta niiden ratkaisut eivät oikeastaan auta minua. Osassa ratkaisuista kyseiset henkilöt ovat loppuen lopuksi joutuneet ottamaan requirejs:n pois, sillä he eivät ole saaneet korjattua ongelmaa. Yksi ratkaisu olisi erottaa webpack-config tiedosto pois node_moduuleista, mutta tämä ei ole suositeltavaa, sillä sovelluksemme hallitsee kyseistä tiedostoa itse. Jos poistamme sen

sieltä, joudumme konfiguroimaan webpackin kaikki asetukset itse tulevaisuudessa ja se ei ole tarkoituksenmukaista.

Tutkin päivän aikana erilaisia ratkaisuja ja kokeilin eri keinoja saada Requirejs-kirjasto toimimaan, mutta mikään ei tuntunut auttavan. Tehtävä alkaa tuntumaan haastavalta, joten joudun ottamaan tämän huomenna Dailyssa puheeksi muiden kehittäjien kanssa ja keskustelemaan heidän kanssaan, kuinka asian kanssa voisi edetä.

Torstai 19.3.2020

Aamu alkaa tänään palaverilla tiimikollegan kanssa. Kerron ongelmista vahvistuspalvelun käyttöliittymän kanssa ja sovimme, että menemme maanantaina toimistolle parikoodaamaan ja yritämme saada sitä toimimaan silloin.

Aamun Scrum Dailyssa keskustelemme jälleen, mitä kukin kehittäjä on saanut aikaiseksi ja mitä tulee tekemään tänään. Kysyn Dailyssa myös, mitä voisin ottaa nyt työn alle ennen kuin palaan vahvistuspalvelutyöhön maanantaina. Seuraava työ olisi aloittaa maksujen validointien tekeminen backendiin. Tämän avulla maksuista voidaan tarkistaa eri tietoja, kuten viitenumeroita, tilinumeroita ja veloitettavia summia. Näillä voidaan varmistaa, että maksu lähtee käsittelyyn oikeanlaisilla tiedoilla. Työtä ei ole kuitenkaan vielä suunniteltu lainkaan, joten päätämme pitää heti pienen arkkitehtuurisen palaverin keskustellen siitä, kuinka validoinnit pitäisi tehdä ja mihin ne pitäisi rakentaa backendissa. Pääsemme palaverin päätteeksi hyvään yhteisymmärrykseen siitä, mitä pitäisi tehdä, jotta validointien tekoa pystyisi aloittamaan. Muutama kehittäjä lupaa ottaa vetovastuun tämän pohjan rakentamisesta ja alkaakin tekemään sitä heti, jotta me muut voisimme aloittaa validointien teon mahdollisimman pian.

Lähden itse käymään läpi versionhallintaan ilmestyneitä pull requesteja, sillä niitä on tullut aika paljon. Tämä vie aika paljon aikaa, sillä koodimuutoksista on yhteensä noin 1200 riviä uutta koodia ja 1300 riviä poistettua koodia. Käyn muutoksia läpi systemaattisesti yksi commit kerrallaan. Näin niitä on helpompi katselmoida pienemmän rivimäärän vuoksi eikä esimerkiksi 400 rivin koodina. Käytän tässä apuna SonarQuben muodostamia raportteja, joista näen mikä muutosten testikattavuus on ja missä niitä voisi parantaa. Raporttien ja oman tietoperustan perusteella kirjaan koodimuutoksiin kommentteja ja ehdotan parannuksia.

Tämä päivä oli hieman erilainen päivä, sillä en kirjoittanut riviäkään koodia, vaan päivä meni suunnittelun ja muiden kehittäjien toteutusten läpikäynnin parissa. Toisten koodia

lukemalla oppii paljon, kun näkee, kuinka toiset kehittäjät ovat ratkaisseet ongelmia. Samalla tulee itse pohdittua, miten kyseisen kohdan olisi itse ratkaissut ja samalla voi koodimuutoksiin ehdottaa parannuksia.

Perjantai 20.3.2020

Tänään tarkoituksena olisi tehdä legacy-järjestelmän osalta julkaisupaketti ja sen jälkeen jatkaa AWS-kurssin läpikäyntiä. Testiympäristömme ei näytä vieläkään toimivan, joten toivon mukaan se olisi jälleen maanantaina toiminnassa.

Asiakasyrityksellämme on käytäntö, että kerran kuussa on ”code freeze” -päivä, jolloin kaikki ominaisuudet, jotka halutaan viedä seuraavassa julkaisussa tuotantoon, pitää olla valmiita. Tämän jälkeen ne viedään kaikkien tiimien yhteiseen testiympäristöön, missä niitä testataan parin viikon ajan sisäisesti ja varmistetaan, että kaikki toimii kuten pitääkin. Tämä siitä syystä, että tuotannossa tapahtuvat virheet voivat aiheuttaa asiakkaalle todella suuria kustannuksia ja finanssialalla palveluiden toimivuus ympäri vuorokauden on välttämätöntä.

Julkaisupaketin tekeminen ei ole kovin vaikeaa, sillä se on hyvin automatisoitu. Ensin tarkistan, mihin kaikkiin järjestelmiin on tehty muutoksia viimeisen kuukauden aikana ja nostan niiden versionumeroita yhdellä. Tämän jälkeen käynnistän Jenkinsistä työn, joka tekee näistä maven-paketteja. Nämä ladataan automaattisesti artifactory-palveluun, josta niitä pystyy kuka tahansa lataamaan. Kun tämä on valmis, teen yhden pull requestin näistä muutoksista kaikkien tiimien yhteiseen testiympäristöön, missä julkaisuihin keskittyvä tiimi katselmoi nämä ja hyväksyy muutokset.

Iltapäivän käytän AWS:n opiskeluun ja sen palveluun nimeltä VPC (Virtual Private Cloud). Tämä on minulle täysin uusi palvelu, joten tämän osan opiskeluun pitää uppoutua kunnolla. VPC on virtuaalinen verkko, minkä voi jakaa IP-alueille ja sen perusteella lajitella sovelluksia ja järjestelmiä, jotka pyörivät joko julkisessa tai yksityisessä verkossa. Minulla on hyvin vähän kokemusta verkkoinfrastruktuurista tai infran konfiguroimisesta yleensäkään, joten tulen varmasti oppimaan tästä paljon uutta.

Päivän tavoitteet tulivat hyvin täyteen, kun sain tehtyä kuukauden julkaisupaketin valmiiksi sekä ehdin myös oppimaan uutta AWS:stä. Ensi viikolla pitäisi ottaa edistysaskelia viimeinkin vahvistuspalvelun suhteen, sillä sitä tarvittaisiin pian, jotta voimme testata koko maksun luonnin prosessia.

Viikkoanalyysi

Tämän viikon tehtävät olivat hyvin paljolti eri bugien selvitystöitä, palavereita ja koodikatselmointeja. Viikko alkoi frontendin työllä, kun yritin toteuttaa ja testata maksujen hakua käyttöliittymiin. Tässä ongelmaksi tuli ensin se, kuinka saan eroteltua toteutuksen. Maksut on haettava rajapinnasta silloin, kun ollaan testiympäristössä ja lokaalista datasta, kun kehitystä tehdään omalta koneelta. Tämän jälkeen törmäsinkin isompaan bugiin, joka hajotti koko sovelluksen testiympäristössä, kun Typescriptin Date-funktiota kutsutaan ”null” tai ”undefined” -arvolla. Tämä oli minulle uusi tieto, sillä monesti nämä ohjelmointikielen natiivit funktiot osaavat käsitellä tyhjiä arvoja, eivätkä kaada koko ohjelmaa. Mielestäni reagoin tähän ongelmaan kuitenkin hyvin, sillä tiesin heti, kuinka löytäisin tähän bugiin mahdollisen ratkaisun. Lisäsin myös tekemääni korjaukseen yksikkötestin, jotta pystyin varmistamaan sen toimivuuden ennen kuin vien muutokset uudestaan testiympäristöön.

Viikon haastavimpiin tehtäviin kuului ehdottomasti testiympäristön kaatamisen aiheuttaneen bugin selvittäminen. Testiympäristön infrastruktuuri on todella massiivinen, ja siihen kuuluu todella monia teknologioita. Tämä tarkoittaa sitä, että lokeja tulee myös todella paljon. Tehtävästä haastavamman teki myös se, että nämä lokit tulostuvat eri paikkoihin, kun muutokset menevät CI/CD-putkemme läpi. Lokeja tulostuu muun muassa testiympäristön palvelimelle, Jenkinsiin ja AWS:n palveluihin. Tätä onneksi sain selvittää vanhemman kehittäjän kanssa, mikä oli hyvin opettavaista, sillä nyt tiedän paremmin, mihin kaikkialle lokeja tulostuu ja minkälaisia virheitä niistä kannattaa etsiä.

Toinen haastava tehtävä viikolla oli jälleen vahvistuspalvelun käyttöliittymän implementointi. Kyseisessä tehtävässä koen ajoittain turhautumisen tunnetta enkä luota omiin taitoihini. Tämän seurauksena yritän kaikkia keinoja sattumanvaraisesti, mikä mahdollisesti toimisi. Maanantaina lähden selvittämään tätä uudella asenteella ja pyrin etenemään loogisesti. Apua tuo myös toinen kehittäjä, joka pystyy katsomaan ongelmaa uusin silmin ja näkemään ehkä jotain sellaista, mitä en ole itse huomannut.

Viikon aikana työhöni sisältyi paljon myös koodikatselmointeja. Tunnen, että olen kehittynyt myös tällä osa-alueella huomattavasti. Teoriataustan avulla tiedän hyvän koodin käytänteitä ja yritän löytää uusia ohjeita aina, kun teen katselmointia, jotta löydän mahdollisesti uusia näkökulmia. Tällä viikolla löysin käytänteet, joita Googlen työntekijät käyttävät tehdessään koodikatselmointeja. Näistä silmään pisti koodin monimutkaisuus ja koodikatselmoinnin kommenttien kirjoittaminen. Pilkkomalla asioita pienempiin osiin yleensä selkeyttää koodia, mutta jos sen vie liian pitkälle, niin koodi saattaa olla vaikeammin luettavampaa, kuin mitä se olisi isompana kokonaisuutena. Koodikatselmoinnissa voi myös

kirjoittaa positiivisia kommentteja, mitä en oikeastaan ole aiemmin tullut ajatelleeksi. Yleensä kirjoitamme korjausta tai parannusta vaativia huomioita, emmekä kehu kehittäjän hyvin toteutettuja muutoksia (Google 2020). Otankin tästä lähtien tämän enemmän huomioon ja pyrin kirjoittamaan kehuja niihin koodimuutoksiin, joissa on onnistuttu todella hyvin.

Vaikka viikon aikana kirjoitin vähemmän koodia kuin aikaisempina viikkoina, niin tunnen, että olen silti oppinut paljon. Bugien ratkominen, koodinkatselmoinnit sekä erilaiset selvitystyöt kehittävät yhtä lailla koodaajan ajattelutapaa kuin itse koodin kirjoittaminen. Jos osaa koodata esimerkiksi yhtä tai kahta kieltä todella hyvin, mutta ei ymmärrä ohjelmoinnin perusperiaatteita taustalla, niin on hankalaa kehittyä ohjelmistokehittäjänä. Haastavammat ohjelmistoprojektit vaativat yleensä kehittäjiltä suuremman kokonaiskuvan hahmottamista (Peck 2019).

3.6 Seurantaviikko 6: 23-27.3.2020

Maanantai 23.3.2020

Tänään tarkoituksena on saada viimeinkin vahvistuspalvelun käyttöliittymä implementoitua meidän käyttöliittymäämme. Olen varannut tehtävään koko päivän aikaa ja suoritamme sen parikoodauksena tiimin toisen kehittäjän kanssa.

Viime viikolla yritin kovasti saada RequireJS-kirjastoa toimimaan edellä mainittua tehtävää varten asentamalla sen NPM-moduulina. Tämä ei kuitenkaan onnistunut, vaan node-moduuleissa tuli jatkuvasti virheitä. Päätimme ettemme lähde yrittämään sen selvittämistä, vaan yritämme keksiä muun tavan asentaa kyseinen kirjasto. Luen toisen vastaavan toteutuksen lähdekoodia hieman enemmän ja huomaan, että he ovat tuoneet RequireJS:n staattisena tiedostona. Teemme samanlaisen toteutuksen meidän sovellukseemme sekä tuomme myös muutaman muun kirjaston samalla tavalla, jotka RequireJS tarvitsee. Viimeinkin edistymme tämän tehtävän suhteen.

Kohtaamme välittömästi uuden haasteen. Vahvistuspalvelun käyttöliittymä ei vielä näy, sillä se tekee POST-requesteja backendiin. Backendin pitäisi vastata tähän tietyllä datalla, joten tällä hetkellä näemme vain virheitä. Käytämme yleensä JSON-serveriä mockataksemme backendin kutsut, mutta POST-kutsun tapauksessa kyseinen kirjasto ei toimi. Se yrittää tallettaa vain tietoa, kuten POST-kutsussa yleensä tehdään eikä palauta mitään. Pohdimme hetken yhdessä, mikä olisi paras ratkaisu tähän ja päätämme, että voimme tehdä oman NodeJS:llä tehdyn serverin, mikä palauttaa tarvittavan datan, kun sitä kutsutaan.

Viimeinkin vahvistuspalvelun käyttöliittymä latautuu meidän käyttöliittymäämme, vaikkakin käyttöliittymän CSS-tyylit eivät vielä lataudu. Tämä on silti suuri askel eteenpäin. Nämä ovat niitä parhaita hetkiä kehittäjän työssä, kun jokin kauan aikaa ongelmana ollut asia saadaan ratkaistua ja työssä päästään viimeinkin eteenpäin. Päivä oli syväsukellus vähän vanhempaan tapaan tehdä kehitystä. En ollut tullut ajatelleeksi, että kirjastoja voisi tuoda staattisina Javascript-tiedostoina, eikä asentamalla NPM:n avulla, kuten nykyään aina tehdään. Auttoi myös huomattavasti, että koodasimme yhdessä kollegan kanssa. Pystyimme hänen kanssaan sparraamaan ideoita, kuinka tämä ongelma saataisiin ratkaistua. Opin, että on tärkeää lähestyä asiaa monesta näkökulmasta, eikä pitäytyä vain yhteen lähestymistapaan, kuinka jokin on aina tavattu tehdä.

Tiistai 24.3.2020

Tämä päivä on jälleen hyvin täynnä erilaisia palavereita. Palavereiden välissä olisi tarkoitus yrittää selvittää, mikseivät tyylit lataudu vahvistuspalvelun käyttöliittymään. Pidän nopean keskustelun tiimiläisten kanssa ja sovimme, että pidämme iltapäivällä pienen palaverin näiden tyylien latautumiseen liittyen, mikäli se ei ole selvinnyt ennen sitä.

Tämän jälkeen on viikoittaisen groomingin aika. Tiimimme muutama kehittäjä on selvittänyt kaikkien erilaisten maksujen prosessit ja validoinnit ja tehnyt niistä vuokaaviokuvaukset, jotka käymme läpi groomingissa. Tämän jälkeen kaiken pitäisi olla valmista työstettäväksi, eikä selvitettäviä asioita pitäisi enää olla. Päätämme pitää iltapäivällä retrospektiivin sijaan palaverin, jossa käymme läpi kaikkia avoinna olevia töitä, sillä kun kaikki ovat työskennelleet nyt jonkin aikaa etänä, ei töiden eteneminen ole niin läpinäkyvää.

Groomingin jälkeen ehdin hetken tutkia mikseivät CSS-tyylit lataudu odotetusti vahvistuspalvelun käyttöliittymään. Tyylit on tuotu meidän sovellukseemme samalla tavalla kuin toisessa sovelluksessa, jonka lähdekoodia olen käyttänyt tämän työn apuna. Uskon, että tässä syynä on joko se, että nämä tyylit eivät ole yhteensopivia meidän tyyliemme kanssa, sillä käytämme uudempaa versiota kyseisistä tyyleistä. Toinen vaihtoehto on, että meiltä puuttuu jokin ulkoinen kirjasto, joka auttaa näiden tyylien implementoinnissa. Kysyn toisen vastaavan sovelluksen kehittäjältä, oliko heillä vastaavia ongelmia. Tarvitsisin tarkempia suuntaviivoja, mihin minun pitäisi kiinnittää huomiota.

Siirryn tästä työstä suoraan demotilaisuuteen, jossa esittelemme asiakkaalle, mitä olemme jälleen saaneet aikaiseksi viimeisen kahden viikon aikana. Esittelemme muun muassa, kuinka tilityyppi tunnistetaan nyt oikein saajan tilinumeron perusteella sekä maksun luomista taustajärjestelmään.

Demon jälkeen siirrymme suoraan avoimien töiden palaveriin. Käymme palaverissa läpi kaikki kesken olevat tehtävät ja mitä pitää nyt priorisoida, jotta mikään ominaisuus ei estä muiden töiden etenemistä. Sovellus pitäisi olla valmis 8 viikon kuluttua, joten tekemistä on vielä paljon, jotta saamme sovelluksen valmiiksi siihen aikarajaan mennessä. Keskustelemme jäljellä olevat työt läpi ja samalla jaamme työt alustavasti eri kehittäjien kesken, jotta kaikilla töillä on alustavasti ainakin yksi vastuuhenkilö. Itse lupaudun olemaan päävastuullinen koko maksun vahvistuksen osalta, sillä olen työstänyt sitä nyt kaikista eniten koko tiimistämme.

Ehdin käydä tämän jälkeen vielä lyhyen keskustelun yhden tiimiläisen kanssa vahvistuspalvelun käyttöliittymän tyyliongelmista. Hän pitää myös todennäköisenä sitä, että vahvistuspalvelua varten tuodut CSS-tyylit eivät ole yhteensopivia meidän omien tyylien kanssa, mutta jäämme odottamaan vastausta toisen toteutuksen kehittäjältä.

Vaikka päivä meni pitkälti palavereissa, saimme vietyä asioita kuitenkin hyvin eteenpäin. Tiedän, kuinka lähden selvittämään CSS-tyyliongelmaa huomisaamuna ja päivän palavereiden perusteella tiedän myös seuraavat priorisoitavat työt, mitä pitää ottaa työn alle seuraavaksi.

Keskiviikko 25.3.2020

Tänään tavoitteena on selvittää bugi, joka aiheuttaa sen, että CSS-tyylit eivät lataudu vahvistuspalvelun käyttöliittymään kuten pitää. Bugi on suhteellisen haastava ratkaistava, sillä konsoliin ei tule mitään virhettä, kun sovelluksen käynnistää. Iltapäiväksi on varattu koko loppupäivän kestävä mob-programming sessio, joten yritän saada bugin selvitettyä ennen sitä.

Huomaan, että olen saanut viestin toisen tiimin kehittäjältä, jolta kysyin eilen apua tämän asian kanssa. Hän neuvoi lisäämään importatut CSS-tyylit webpack-konfiguraatioon, joka tekee koko frontendin koodista yhden paketin siinä vaiheessa, kun sovellus käynnistetään. Emme kuitenkaan voi muokata näitä webpack-konfiguraatioita, sillä käyttämämme React-versio pitää itse huolen webpackin konfiguraatiosta. Jatkan heidän koodin läpikäymistään ja vertailen heidän ja meidän käyttämiä ulkoisia kirjastoja, löytyisikö sieltä jotain eroavaisuuksia, minkä avulla tyylit saataisiin tuotua onnistuneesti.

En löydä heidän koodistaan mitään kummempia eroja, joten alan epäilemään virhettä siinä, miten tyylit tuodaan. Perinteiseen tapaan staattisia CSS-tiedostoja tuodaan HTML-tiedostossa <link> elementillä, mille kerrotaan, minkä muotoinen tiedosto on kyseessä ja mistä polusta tyylit haetaan. Huomaan, että tässä minulla on sattunut pieni kirjoitusvirhe tyylien polussa. Korjaan polun ja yritän käynnistää sovelluksen huomatakseni, että tyylit latautuvat ja vahvistuspalvelun käyttöliittymä näyttää siltä miltä pitääkin. Saan viimeinkin tämän muutaman viikon kestäneen työn päätökseen ja pääsen jatkamaan seuraaviin töihin. Käyttöliittymää pitää vielä myöhemmässä vaiheessa jatkokehittää, mutta ennen sitä pitää saada backendiin toiminnallisuus, joka oikeasti lähettää oikeanlaista dataa rajapinnasta, jota frontendissa tarvitaan.

Tämän jälkeen alkaa mob-programming sessio. Sessiossa koodaamme koko tiimin voimin maksun luomista varten tarvittavan toiminnallisuuden backendiin. Päätimme tehdä tämän yhdessä, sillä jokaisen tiimin kehittäjän pitäisi tietää, miten tämä prosessi toimii ja mitä kaikkea siihen kuuluu. Käymme läpi vaatimusmäärittelystä prosessin ja alamme koodaamaan ensimmäisestä askeleesta alkaen. Koodaamme siten, että jokainen kirjoittaa koodia 15 minuutin ajan, jonka jälkeen vaihdamme vuoroa. Se, joka koodaa, kirjoittaa koodia muiden ohjaamana. Näin kaikki ovat tilanteen tasalla siitä, mitä tapahtuu ja miksi.

Työpäivän päättyessä tunnen onnistuneeni. Sain viimeinkin valmiiksi vahvistuspalvelun käyttöliittymätyön ja pääsen keskittymään muihin töihin. Mob-programming oli myös hyvin opettavainen sessio, sillä en ollut koodannut hetkeen mitään backendissa. En ole myöskään koodannut Springillä hetkeen, vaikkakin olen palvelinohjelmoinnin kurssin käynyt Haaga-Heliassa. Oli helpompi virkistää muistia kyseisestä frameworkista koko tiimin voimin kuin tekemällä yksin.

Torstai 26.3.2020

Yksi kehittäjä toisesta tiimistä on tehnyt meille toteutuksen, missä käyttäjä pystyy tarkastelemaan omia tilitapahtumiaan. Tänään tehtäväni on tarkistaa kyseinen toteutus frontendin osalta ja korjata siinä mahdolliset virheet.

Aloitan tekemään koodikatselmointia ja huomaan, että hän käyttää erilaista syntaksia funktioiden kohdalla kuin mitä me yleensä käytämme. Hän on kirjoittanut funktion muotoon: *function useLocale(self: useSelf): string {},* kun taas me kirjoitamme yleensä *const useLocale: self: UseSelf) => string = (self: UseSelf): string => {}*. Eli käyttämällä tuota function sanaa ei tarvitse ”tuplatyypittää”, josta olen maininnut aikaisempina viikkoina. Emme halua kuitenkaan käyttää tuota vanhaa syntaksia, missä käytetään function-sanaa, vaan haluamme käyttää jatkossakin nuolisyntaksia.

Päätän, että nyt jos koskaan on aika selvittää, miksi meillä on tuo tuplatyypitys käytössä, sillä se tekee syntaksista ja koodista huomattavasti vaikeammin luettavaa. Lähden käymään läpi TS-lintin sääntöjä läpi heidän Githubistaan ja sieltä pistää silmään seuraava sääntö: ”typedef – Requires type definitions to exist”. Tämän alta löydän säännön "variable-declaration-ignore-function" jonka kuvaus kertoo, että kyseisellä säännöllä voidaan sivuttaa nuolifunktioiden tyypitys. Tämä on juuri se sääntö, mitä olen jo pitkään etsinyt. Lisään kyseisen säännön TS-lint konfiguraatioomme ja yritän muokata nuolifunktiota muotoon *const useLocale = (self: UseSelf): string => {}.* Huomaan, että TS-lint ei enää valita tuosta, joten sääntö toimii kuten pitää. Nyt funktiot näyttävät paljon enemmän perinteiseltä

Javascriptin syntaksilta ja ovat paljon luettavampia. Hyväksytän vielä tämän muutoksen tiimin muiden kehittäjien kesken ja he ovat samaa mieltä, että muutos on hyvä ja parantaa luettavuutta.

Loppupäivä minulla meneekin muuttaessani jokaisen frontendin funktion käyttämään tuota uutta syntaksia. Päivän loppuun mennessä saan kaikki korjaukset tehtyä myös tilitapahtumien toteutukseen ja teen tästä kaikesta pull requestin versionhallintaamme, jotta muut kehittäjät voivat katselmoida muutokseni. Viimeinkin pääsin eroon tästä tuplatyypityksestä, minkä kanssa olen kamppaillut monta aikaisempaa viikkoa. Tästä eteenpäin uskon, että Typescriptin kirjoitus sujuu huomattavasti paremmin, sillä syntaksi on paljon tutumpaa nyt.

Perjantai 27.3.2020

Tänään pyrin keskittymään oman osaamiseni kehittämiseen ja tuen asiakasprojektia tarvittaessa. Olen varannut toukokuun alkuun päivän AWS-sertifikaattikoetta varten, joten pyrin keskittymään sen opiskeluun. Sen ohessa pyrin myös opiskelemaan Typescriptia ja CSS:ää työstämällä omaa portfoliosivustoani. Joudun painottamaan kuitenkin enemmän AWS:n opiskelua, sillä kokeen yrittäminen maksaa aina 150 dollaria, joten sitä en halua monesti uusia, vaikkakin työnantajayritys tämän kustantaa.

Aloitan aamun tekemällä viikkoaikataulun. Viikkoaikatauluun lisään sen viikon aikana käytävät osa-alueet kurssista sekä myös harjoituskokeen, johon laitan tietyn pistemäärän tavoitteeksi. Tälle viikolle otan tavoitteeksi käydä AWS:n High Availability Architecture osa-alueen läpi ja pyrin saamaan harjoituskokeesta vähintään 75 % vastauksista oikein.

Lähden opiskelemaan kurssin materiaalia läpi. Siinä esitellään muun muassa load balancer, automaattinen skaalaus sekä AWS:n palveluita, joiden avulla datakeskuksia ja verkkosivustoja voidaan pitää käynnissä, jos tapahtuisi jotain yllättävää. Verkkosivustot voivat kaatua, jos monia käyttäjiä pyrkii sivustolle samaan aikaan. Kun kaikki kutsut tulevat yhdeltä palvelimelta, palvelin ruuhkautuu, mikä aiheuttaa sivun latautumisen hitautta tai jopa sen kaatumisen. AWS:n palveluiden avulla käytettävissä olevia palvelimia voi lisätä sitä mukaa, kun ruuhkaa tulee. Load balancerin avulla pystytään jakamaan liikennettä tasaisesti eri palvelimille, jotta liikenne ei ruuhkauttaisi vain muutamaa palvelinta.

Tämän jälkeen teen harjoituskokeen, missä käydään kaikkia osa-alueita läpi. Odotuksenani on, että kokeesta ei vielä kovin korkeita pisteitä tule, sillä en ole vielä käynyt kahta kokeen osa-aluetta läpi. Kokeen tehtävät ovat monivalintatehtäviä, jotka ovat hyvin sovel-

tavia ja oikean elämän tilanteisiin perustuvia. Osa kysymyksistä on todella haastavia ja vastausvaihtoehdot saattavat olla hyvin samantyyppisiä. Saan kokeesta tällä yrittämällä 70 % oikein, mikä on mielestäni ihan hyvä suoritus ottaen huomioon, että en ole vielä päässyt opiskelumateriaalissa loppuun asti.

Päivä oli todella opettavainen ja onnistunut. Tein selkeän suunnitelman, kuinka lähteä opiskelemaan seuraavan viiden viikon aikana AWS-sertifikaattia varten, jotta pysyn aikataulussa ja minulla olisi hyvät mahdollisuudet päästä kokeesta läpi. Opeteltavat asiat olivat todella mielenkiintoisia arkkitehtuurillisia asioita, joita en pääse työssäni hirveästi tekemään tällä hetkellä. Näiden asioiden opiskelu kuitenkin auttaa kehittymään, sillä vanhempana kehittäjänä pitää pystyä ottamaan kantaa suurempiin arkkitehtuurillisiin asioihin ja siihen, miten sovellusten arkkitehtuuri voitaisiin toteuttaa.

Viikkoanalyysi

Alkuviikosta sain viimeinkin ratkaistua vahvistuspalvelun käyttöliittymän upotuksen sovelluksemme käyttöliittymään, jota olen koittanut jo parin viikon ajan saada toimimaan. Päätimme työstää ongelman ratkaisun parikoodaamalla, jota teemme tiimissä aina välillä. Kun omat ideat alkavat loppumaan, on hyvä sparrata ideoita toisen kehittäjän kanssa tällaisessa tilanteessa. Jos jokin kehitettävä asia on laajuudeltaan suurempi, on hyvä ottaa toinen henkilö mukaan katselmoimaan koodia, jolloin koodista tulee siistimpää ja työ luultavasti valmistuu myös nopeammin. Parikoodaus onkin yksi tehokkaimmista tavoista ratkoa ongelmia kuten Robert Martin mainitsee kirjoittamassaan kirjassa The Clean Coder (Martin 2011, 164).

Lähdin selvittelemään tätä ongelmaa keskittymällä vain yhteen näkökulmaan. Tutkin koodia ja pyrin ymmärtämään, mitä se tekee ja miten se toimii, mutta en kiinnittänyt lainkaan huomiota laajempaan kuvaan. En ajatellut lainkaan, kuinka ulkoisia kirjastoja sovellukseen on mahdollisesti tuotu, enkä ajatellut, mitä eroja tällä vanhemmalla React-versiolla on verrattuna meidän sovelluksessamme käytettyyn versioon. Vasta, kun parikoodatessa keksimme katsoa, miten ulkoisia kirjastoja tuodaan projektiin, pääsimme oikeille jäljille ja pystyimme lähteä toteuttamaan vastaavaa meidän sovellukseemme.

Toinen isompi asia, minkä tällä viikolla sain ratkaistua, oli ”tuptatyypitys” Typescriptinfunktioiden kohdalla. Tässäkin ongelmana oli lähestyminen vain yhdestä näkökulmasta. Aiemmin olin koittanut löytää Googlen avulla vastaavia esimerkkejä, mutta vaikutti siltä, ettei kukaan muu kirjoita vastaavanlaista syntaksia, kuin mitä meillä on projektissa käytössä. En ollut ajatellut tutkivani sovelluksemme konfiguraatiota ja siellä määriteltyjä Ty-

pescript-sääntöjä, sillä tiimissämme työskennellyt vanhempi kehittäjä oli ne projektin alussa määritellyt. TS-lintillä on kohtuullisen hyvä dokumentaatio (Palantir 2020), mutta kaikista säännöistä ei ole olemassa esimerkkejä, kuinka ne toimivat käytännössä. Vasta nyt, kun syvennyin tutkimaan heidän dokumentaatiotaan, löysin kyseiset säännöt ja sain korjattua jo pitkään vaivanneen ongelman.

Tällä viikolla otin päämäärätietoisen tavoitteen AWS-sertifikaatin opiskelusta. Lähdin laatimaan opiskelusuunnitelmaa ja aikataulua seuraaville viidelle viikolle netistä löytämieni materiaalien perusteella. CBT Nuggets-niminen sivusto, joka tarjoaa myös valmennuskursseja sertifikaattia varten, suositteli aikataulun jakamista kymmeneen viikkoon. Jokaiselle viikolle on määritelty tietty käsiteltävä osa-alue, kuvattu sen sisältö, määritelty osaamisvaatimukset ja kuinka kauan opiskeluun pitäisi varata aikaa (CBT Nuggets 2019). Kedar Dabhadkarin kirjoittamassa artikkelissa on myös vinkkejä, miten kokeeseen tulisi valmistautua. Artikkelissa suositellaan, että AWS:n omia oppaita kannattaa opiskella myös muun materiaalin ohessa. Siinä annetaan myös vinkkejä eri osa-alueisiin ja mihin asioihin kannattaa painottaa erityistä huomiota (Dabhadkar 2019). Tämä tieto täydensi todella hyvin opiskelusuunnitelmaani ja auttoi minua varmistamaan, että minulla on nyt kaikki mahdollinen materiaali koetta varten. Nyt pelkästään toteutus on minusta itsestäni kiinni.

3.7 Seurantaviikko 7: 30-3.4.2020

Maanantai 30.3.2020

Aamu alkaa tänään tutkimalla JIRAa ja selvittämällä, mitä töitä seuraavaksi pitäisi tehdä. Huomaan tiketin, jossa tehtävänä on lisätä EU:n sisäisten ja ulkomaisten maksujen käyttöliittymiin muutama uusi kenttä ja nämä kentät pitäisi lisätä myös backendissa rajapintaan lähtevässä kutsussa. Pitkästä aikaa pääsen tekemään samassa tehtävässä sekä frontendia että backendia, joten otan tehtävän mielelläni vastaan.

Suoriudun työstä nopeasti, sillä kenttien lisäys on suhteellisen yksinkertaista. Tarkistan kuitenkin ennen työn aloittamista, mihin Java-luokkaan minun pitää maanantaina kyseinen muutos tehdä, sillä en ole tehnyt backendiin töitä pitkään aikaan. Aloitankin työn backendista ja lähden toteuttamaan työtä. Yritän tehdä työtä myös pitkästä aikaa TDD-metodilla eli kirjoitan ensimmäisenä yksikkötestit, joiden pitäisi mennä läpi sen jälkeen, kun toteutus on valmis backendissa.

Koska en ole kirjoittanut hetkeen JUnit-testejä, käytän hetken muiden vastaavanlaisten testien tutkimiseen, jotta JUnit-testin syntaksi muistuu mieleeni. Tämän jälkeen lähden rakentamaan testiä. Kirjoitan yksikkötestit siten, että ne testaavat sekä oikean että virhetilanteen. Kun olen saanut testin kirjoitettua, teen itse toteutuksen. Tähän ei mene kovinkaan kauaa, sillä lisään vain Java-metodiin olion, joka lähettää uudet kentät rajapintaan ja palauttaa sen jälkeen vastauksen frontendille. Myös kirjoittamani testit menevät läpi, joten toteutuksen pitäisi toimia oikein.

Iltapäivälle on varattu mob-programming sessio, missä pyrimme viimeistelemään uuden maksun luonnin. Siitä pitäisi lähteä tieto rajapintaan, jotta voisimme yhdistää frontendin tähän ja pääsisimme testaamaan toiminnallisuutta frontendissä. Työ etenee hyvin ja tuleekin valmiiksi vielä ennen päivän päättymistä. Huomenna pyrimme yhdistämään tämän frontendiin, jotta voimme testata ennen keskiviikon demoa tämän toimintaa.

Pääsin tänään pitkästä aikaan tekemään koodia backendissa Javalla. En ole koodannut kyseisellä kielellä hetkeen, enkä sitä yleensä valitse käytettäväksi kieleksi omissa projekteissani, mutta fakta on, että se on edelleen käytetyimpiä kieliä koko maailmassa, jonkavuoksi sitä on myös hyvä osata.

Tiistai 31.3.2020

Tänään tavoitteena olisi jatkaa uusien kenttien lisäämistä EU:n sisäisten ja ulkomaisten maksujen käyttöliittymiin. Eilen päivän päätteeksi toinen kehittäjä on saanut korjatuksi vian, jonka takia tilivalikon avaaminen kaatoi koko sovelluksen.

Tänään kärsin ensimmäistä kertaa VPN-yhteyden katkeilusta, koska verkko on kuormittunut koronapandemian aiheuttaman etätyövelvoitteen takia. En pääse liittymään aamun Scrum Dailyyn enkä groomingiin, sillä VPN-yhteys asiakkaan toimistolle on pakollinen, jotta Microsoft Teamsiin, Jiraan ja muihin työkaluihin saa yhteyden. Ongelmat vaikuttavat jatkuvan pitkään, joten päätän käyttää päivän hyödyksi AWS-sertifikaatin opiskelua varten, jotta koko päivä ei mene VPN-yhteyden korjautumisen odottelemiseen.

AWS-sertifikaatti kurssin seuraavana osa-alueena on serverless-arkkitehtuuri, mikä on suhteellisen tuttua minulle. Serverless-arkkitehtuuri ei nimestään huolimatta tarkoita palvelintonta-arkkitehtuuria, vaan se tarkoittaa sitä, että koko ”rautainfrastruktuuri” on ostettu palveluna. Tällöin esimerkiksi rajapinta, jota kutsutaan, pyörii jollakin AWS:n palvelimella, josta ulkopuolinen palveluntuottaja vastaa. Palvelun kustannukset tulevat yleensä siitä, kuinka kutsuja rajapintaan tulee ja kuinka paljon laskentatehoa nämä kutsujen toteuttamiset vaativat.

Teen tämän jälkeen myös uuden AWS-harjoituskokeen. Kysymykset ovat pääosin eri kysymyksiä kuin viime kokeessa, joten en voi hyödyntää aikaisempia vastauksiani. Saan tällä kertaa tulokseksi 75 %, mikä oli viime viikon tavoitteeni. Tälle viikolle olen asettanut tavoitteeksi 80 %, joten yritän uutta koetta luultavasti loppuviikosta, jotta saisin paremman tuloksen.

Päivä oli hieman erilainen, mitä alun perin suunnittelin. En päässyt lisäämään uusia kenttiä käyttöliittymiin, mutta toivon mukaan huomenna yhteydet toimivat, jotta pääsen jatkamaan töitäni. Päivä oli silti opiskelun kannalta onnistunut, sillä sain keskittyä lähes koko päivän AWS-sertifikaatin opiskeluun ja pääsin näin lähemmäksi tämän viikon tavoitteita opiskelun osalta.

Keskiviikko 1.4.2020

Eilinen työpäiväni kului pitkälti opiskelun parissa, joten tänään olisi tarkoitukseni palata töiden ääreen ja viimeistellä uusien kenttien lisääminen EU:n sisäisten ja ulkomaisten maksujen käyttöliittymiin.

Lähden tutkimaan vaatimusmäärittelystä, mihin nämä uudet kentät halutaan sijoittaa ja aloitan työn. Työ ei ole kovinkaan hankala, sillä minun pitää lisätä kentät vain käyttöliittymään oikeille kohdille, lisätä kyseisten kenttien arvot JSON-objektiin, joka lähetetään backendille sekä lisätä kentät yksikkötesteihin. Työ kuitenkin keskeytyy hetkeksi, sillä asiakas on pyytänyt ylimääräistä demoa tälle viikolle. Pystymme saamaan asiakkaalta nopeammalla syklillä palautetta valmistuneista töistä, jotta pystymme korjaamaan mahdolliset virheet heti.

Demossa esittelemme eilen valmiiksi tullutta maksun luomista, joka onnistuu nyt käyttöliittymän kautta. Tämän lisäksi pystymme esittelemään käyttöliittymää, jossa näkyy kaikki asiakkaan vahvistamattomat maksut ja johon käyttäjä ohjataan tehtyään uuden maksun. Jotta koko maksun luonnin prosessin on valmis, pitää meidän vielä tehdä valmiiksi aloittamani vahvistuspalvelun integraatio ja uusi käyttöliittymä, mikä näyttää vahvistetut maksut. Asiakas nostaakin nämä prioriteetiksi ja lupaamme ottaa nämä työn alle, kunhan nykyiset työt saadaan valmiiksi.

Demon jälkeen päätämme pitää vielä viikon viimeisen mob-programming session, jossa tarkoituksena olisi tehdä backendiin ulkomaisten maksujen mahdollistava ominaisuus. Tämä eroaa hieman normaalista maksusta, minkä saimme jo valmiiksi, sillä kutsumme eri rajapintoja sekä validoimme maksun hieman eri tavalla. Saamme valmiiksi rajapintoihin menevät kutsut ja yksikkötestit näille, mutta maksun validoinnit jäävät vielä kesken, sillä niitä on hyvin paljon. Päätämmekin tehdä näistä kaikista JIRAan omat tiketit, joita kuka tahansa kehittäjä voi sitten tehdä, kun ei ole kiireellisempää tekemistä.

Tämän jälkeen pääsen viimeinkin viimeistelemään aamulla aloittamani työn. Saan kentät lisättyä ja testattua myös, että ne lähtevät backendiin ja backend lähettää ne kutsuttavalle rajapinnalle. Kaikki näyttää toimivan kuten pitääkin, joten teen kaikista muutoksista pull requestin versionhallintaan ja ilmoitan testaajallemme, että kyseinen tehtävä on nyt valmiina testattavaksi.

Torstai 2.4.2020

Eilen ajaessani frontendin yksikkötestejä huomasin, että konsoliin tulostuu paljon erilaisia varoituksia. Niitä on jopa niin paljon, että se tekee testien seuraamisesta hankalaa, joten tänään tarkoituksena olisi korjata kaikki tämä tekninen velka, mitä on päässyt kertymään.

Varoitukset näyttävät olevan hyvin erityyppisiä. Osa varoittaa, että kutsuttavalle komponentille annettavan parametrin arvo ei vastaa sille määriteltyä tyyppiä. Osa on taas hyvin yksinkertaisia, kuten esimerkiksi samanlaisia id-tunnuksia, joiden perusteella React uudelleen renderöi elementtejä. Tämä johtuu luultavasti vain kopioi-liitä-toiminnolla tehdystä koodista, mitä testidatassa on tehty eikä tällöin ole huomattu vaihtaa näitä id-arvoja.

Olen käyttänyt tuoreinta versiota development-päähaarastamme ja huomaan, että muutamia testejä ei ajeta ollenkaan. Lähden tutkimaan syytä tähän ja huomaan, että näissä testeissä ei kutsuttaville komponenteille anneta kaikkia parametreja tai ne ovat väärässä muodossa. Tämä johtaa siihen, että testiä ei koskaan ajeta, koska koodi ei mene läpi. Tästä ei kuitenkaan tule mitään ilmoitusta mihinkään, vaan koko testi vain ohitetaan. Tämä vaikuttaa aika erikoiselta, joten kirjaan tämän muistiin myöhempää tutkimista varten ja jatkan muiden varoitusten korjausta.

Päivä meni pitkälti näiden testien korjaamisessa ja varoitusten vähentämisessä. Sain lähes kaikki varoitukset korjattua. Ainoa, mitä en saanut ratkaistua, oli tuo kutsuttavan komponentin parametrin tyyppitarkistus. Niiden tyyppitarkistus on integroitu Reactiin, enkä ainakaan vielä löytänyt dokumentaatiosta mitään, millä ne saisi korjattua tai ohitettua. Tämä jää selvitettäväksi vähän myöhemmälle, kuten tuo rikkinäisten testien ilmoitus.

Perjantai 3.4.2020

Tänään aloitan vahvistettujen maksujen käyttöliittymän työstämistä. Tavoitteena olisi opiskella ainakin vähän AWS-sertifikaatin materiaaleja myös tänään, jotta pysyn viikon tavoitteessa.

Sovimme ensiksi tiimimme Scrum Masterin kanssa, että käymme läpi vaatimusmäärittelyn ja keskustelemme vähän logiikasta ja siitä, milloin, mistä ja miten sivulle päätyy. Käymme läpi ensimmäisenä käyttöliittymän designin prototyyppiä, jonka jälkeen siirrymme logiikkaan. Käyttöliittymään voidaan päätyä monesta eri paikasta sovelluksessa ja maksuja voi olla eri tiloissa, jotka pitää ryhmitellä tilojen perusteella. Tila voi olla esimerkiksi hylätty tai hyväksytty. Tämä käyttöliittymä on varmaankin vaativin, minkä parissa olen tähän mennessä työskennellyt.

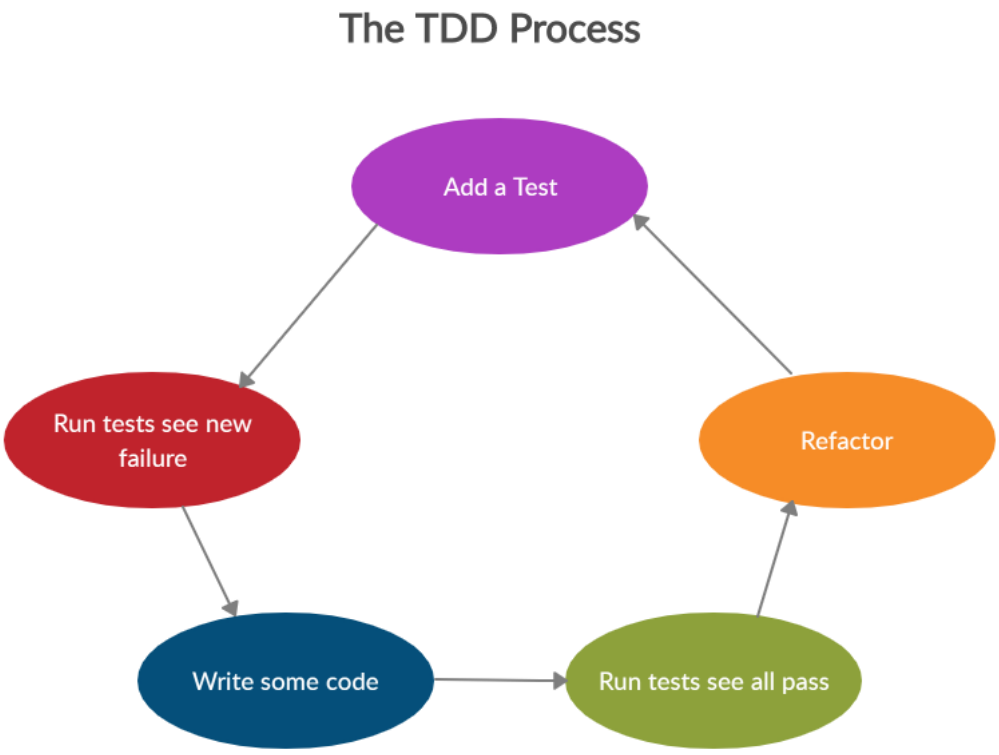
Palaverin jälkeen lähden työstämään yksinkertaistettua versiota käyttöliittymästä, jotta saan sen perustan valmiiksi ja voin lähteä sitä kehittämään. Käytän aluksi myös pelkästään kovakoodattua dataa, jotta näen, että kaikki elementit ovat oikeilla paikoillaan. Pystyn hyödyntämään hyvin uudelleenkäytettäviä komponentteja, jotka on tehty kaikille tiimeille

käytettäväksi, joten työ etenee hyvää vauhtia. Saan perustan rakennettua ilman suurempia ongelmia, jonka jälkeen päätän keskittyä vielä pari tuntia AWS sertifikaattia varten.

Loppupäiväksi päätän keskittyä pelkästään AWS:n omien ohjeiden ja dokumentaation lukemiseen, mitä viime viikon artikkelissa oli mainittu. Oppaat ovat noin 50-90 sivun mittaisia ja täynnä teknistä asiaa, joten niihin pitää perehtyä ihan ajatuksella. Pidin varsin hyvänä ”Overview of Amazon Web Services”-nimistä dokumentaatiota, vaikka se oli lähes sata sivuinen dokumentti. Kyseinen dokumentti oli kuin hieman pitkä tiivistelmä kaikista palveluista. Siinä käytiin läpi muun muassa, mitä AWS:ssä on tarjolla, mitä hyötyä niistä on ja mitä eroa eri pilvipalvelumalleilla on (IaaS, PaaS, Saas).

Viikkoanalyysi

Tällä viikolla mielenkiintoisin ja opettavaisin tehtävä oli ehdottomasti käyttöliittymien lisäys frontendiin sekä backendiin. Tehdessäni backendia yritin pitkästä aikaa tehdä TDD-metodilla, eli kirjoitin ensimmäiseksi yksikkötestit ennen kuin kirjoitin itse toteutuksen (Kuvio 2). Toteutusta tehdessäni pyrin pitämään mielessäni TDD:n kolme lakia, mistä Robert Martin kertoo kirjassaan. Koodia ei saa kirjoittaa ennen kuin olet kirjoittanut epäonnistuvan testin, testiä ei saa kirjoittaa enempää, mitä on tarpeellista saada se epäonnistumaan ja koodia ei saa kirjoittaa enempää, mitä on tarpeellista, että testi menee hyväksytysti läpi (Martin 2009, 122-123). Tällä tavoin testi tulee kattamaan jokaisen rivin koodista samalla, kun kirjoitat sitä.



Kuvio 2. Test Driven Development-metodin prosessi (mukaillen Ryan 2016).

Koodatessani huomasin, että yllä mainittujen lakien täsmällinen noudattaminen on hieman raskasta. Tuntui, etten päässyt missään vaiheessa oikeasti kunnolla keskittymään itse toteutukseen, sillä vaihtelin jatkuvasti testin ja itse toteutuksen välillä. Huomasin myös, että saatoin keskittyä enemmän siihen, että testi menee läpi enkä välttämättä siihen, että testi on laadullisesti hyvä. Samaa näkemystä tuo esille myös Charlee Li artikkelissaan, jossa hän pohtii TDD:n käytänteitä. Hän nostaa esiin, kuinka yksikkötestien pitäisi keskittyä implementaatioon, jota on hankala hahmottaa ennen kuin koko implementaatio on selkeä. TDD:ssa menee myös aikaa hukkaan jatkuvaan refaktorointiin, kun korjauksia pitää tehdä jatkuvasti sekä toteutukseen että testeihin (Li 2018).

Viikolla osallistuin useisiin mob-programming sessioihin. Nämä olivat tärkeitä ja opettavaisia sessioita tiimin kaikille kehittäjille, sillä kaikkien oli tärkeä tietää, miten koko maksun prosessi toimii ja mitä dataa rajapintoihin lähetetään ja mitä sieltä saadaan vastauksena näissä tapauksissa. Mob-programming sessioiden tuloksena pyritään vähentämään kommunikaatio-ongelmia, kun muut kehittäjät joutuvat kysymään, miten tämä toimii näin ja miksi tämä palauttaa tiettyä dataa. Tarkoituksena on myös pyrkiä mahdollistamaan mahdollisimman monen yhteinen näkemys siitä, miten tietyt asiat ja ongelmat pitäisi ratkaista, sillä yhden kehittäjän ratkaisu ei välttämättä ole niin hyvä kuin kahdeksan kehittäjän yh-

dessä miettimä ratkaisu (Agile Alliance). Maksun luominen on myös yksi sovelluksemme kriittisimpiä tehtäviä, joten on tärkeää, että kaikki osallistuvat tähän työhön.

3.8 Seurantaviikko 8: 6-10.4.2020

Maanantai 6.4.2020

Vapaapäivä töistä koulutehtävien vuoksi.

Tiistai 7.4.2020

Tämä päivä on pitkälti eri palavereita täynnä. Tänään on tiedossa viikottainen grooming ja demo, minkä jälkeen olisi tarkoitus pitää sessio, jossa aloitamme vahvistuspalvelun backend-integraation työstämisen.

Ensimmäisenä vuorossa on grooming, jossa käymme asiakkaan kanssa läpi kaikki käyttöliittymiin liittyvät tekemättömät työt. Sovelluksen julkaisu on puolentoista kuukauden päässä, joten keskustelemme palaverissa, mitä töitä voimme karsia, jos näyttää siltä, ettemme saa kaikkea valmiiksi julkaisupäivään mennessä. Saamme myös luvan ylityön tekemiselle oman tarpeen mukaan, jotta saisimme sovellukseen kaikki ominaisuudet, mitä asiakas on halunnut ja myös testattua ne kaikki kattavasti ennen julkaisupäivää. Koska tiimiläiset saattavat nyt tehdä töitä viikonloppuisin, päätämme jakaa jäljellä olevat työt kaikkien kehittäjien kesken. Tämä siitä syystä, että jos tekee viikonloppuisin työtä, ei välttämättä saa muihin tiimiläisiin yhteyttä ja näin ei välttämättä tiedä, mitä tulisi tehdä seuraavaksi.

Groomingin jälkeen siirrymme lähes suoraan demoon. Demossa meillä oli tarkoituksena esitellä EU:n sisäisten ja ulkomaisten maksujen luomista sekä esitellä uusia ominaisuuksia hyväksymättömien maksujen sivulla, mutta huomasimme juuri ennen demon alkamista ongelmia testiympäristössä. Rajapinta, jota kutsumme luodessamme maksuja, on päivitetty muutama tunti sitten, vaikka olimme tälle toiselle tiimille ilmoittaneet, että ei viedä muutoksia ennen demoa. Rajapintaan tulleet päivitykset aiheuttivat sen, että emme pystyneet tekemään maksuja, joten demossa esiteltävät asiat vähenivät huomattavasti. Asiakasta tämä ei kuitenkaan haitannut, vaan he ehdottivat, että he voisivat puolestaan esitellä, mitä he ovat valmistelleet sovelluksen lanseerausta varten.

Iltapäivän käytämme vahvistuspalvelun backend-logiikan toteuttamiseen toisen kollegan kanssa. Hän on koodannut paljon sovelluksemme backendia, joten pystymme hyvin yhdistämään hänen taitonsa backendin osalta ja minun tietoni vahvistuspalvelusta yhteen, jotta saamme tehtyä toteutuksen mahdollisimman nopeasti. Työ etenee ilman sen kummempia ongelmia siihen asti, että olemme lähes valmiita viemään ensimmäisen version

testiympäristöömme. Ainoa asia, mikä jää kesken on yksikkötestit, jotka tämä toinen kehittäjä pyrkii korjaamaan loppuviikon aikana.

Keskiviikko 8.4.2020

Ensi tiistaille on sovittuna demo, jossa asiakas haluaisi nähdä koko maksun luomisen prosessin. Minun pitäisi saada siihen mennessä sekä vahvistuspalvelun integraatio toimimaan kokonaisuudessaan että tehdä vahvistettujen maksujen käyttöliittymä valmiiksi. Työtä siis riittää huomiselle ja tälle päivälle paljon.

Aloitan päivän pitämällä palaverin tiimin Scrum Masterin ja pääsuunnittelijan kanssa. Keskustelemme siitä, millä logiikalla maksuja jaetaan eri taulukoihin vahvistettujen maksujen käyttöliittymässä sekä miten jaamme logiikan frontendin ja backendin välillä. Sovimme, että teemme backendiin logiikan, mikä odottaa maksurajapinnan vastausta maksimissaan viiden sekunnin ajan, jonka jälkeen backend palauttaa kaikki maksut takaisin frontendille tietyillä statuksilla ja minkä perusteella ne lajitellaan taulukoihin. Tämä helpottaa huomattavasti työtäni, sillä voin luottaa, että backendilta palautuva data on aina samanmuotoista ja voin tehdä lajittelulogiikan pelkästään tämän statuksen perusteella. Logiikan tekeminen käyttöliittymään etenee hyvin ja huomiselle jääkin testidatan teko sekä yksikkötestien kirjoittaminen, jotta pääsen testaamaan toteutusta käytännössä.

Tämän jälkeen siirrymme jälleen vahvistuspalvelun integraation pariin. Eilen keskityimme pelkästään backendin toteutukseen, joten tänään päätämme keskittyä frontendin pariin. Tätä ei ole edistetty sen jälkeen, kun olen onnistuneesti kokeillut upotetun käyttöliittymän toimintaa. Siirrämme vahvistuspalvelun käyttöliittymän sille määritettyyn paikkaan ja siivoamme hieman koodia. Eilen rajapintamuutosten myötä rikki mennyttä käyttöliittymää ei ole vielä saatu korjattua, joten joudumme korjaamaan myös sen virheitä samalla.

Päivän päätteeksi yritämme viedä koodimuutoksia testiympäristöömme, jotta voisimme yrittää kutsua vahvistuspalvelun käyttöliittymää testiympäristöstämme. Tähän mennessä sitä on kutsuttu vain lokaalisti kehittäessä. Emme kuitenkaan saa muutoksia CI/CD-putken läpi, sillä SonarQube ilmoittaa, että yksikkötestien kattavuus on liian pieni. Lupaan korjata nämä huomenna aamulla, jotta pääsisimme viimeinkin testaamaan vahvistuksen toimimista testiympäristössä.

Päivän aikana sain edistettyä paljon asioita. Vahvistettujen maksujen käyttöliittymä on pieniä viilauksia ja yksikkötestejä vaille valmis ja saimme vietyä vahvistuspalvelun käyttöliittymää frontendissä eteenpäin siten, että se on pian valmiina testattavaksi.

Torstai 9.4.2020

Tämä päivä tulee kulumaan vahvasti yksikkötestien parissa. Minun pitäisi saada valmiiksi yksikkötestien lisäys vahvistettujen maksujen käyttöliittymän osalta sekä korjata eilen vahvistuspalvelun käyttöliittymässä ilmenneitä yksikkötestien kattavuuksia. Asiakas on toivonut demoa molemmista ominaisuuksista ensi viikon tiistaille, joten molemmat ominaisuudet pitäisi saada siihen mennessä valmiiksi.

Aloitan korjaamalla vahvistuspalvelun käyttöliittymän yksikkötestejä. Avaan SonarQuben generoiman raportin, mikä kertoo, mitä rivejä ja ehtoja en ole kattanut yksikkötesteillä. Testien korjauksen aikana törmään ongelmaan. Yritän testata funktiota, joka tarkistaa, että maksun eräpäivä ei ole tyhjä. Testissä yritän antaa Date-funktiolle argumentiksi null-arvon, mutta Typescript ei sitä kelpuuta, sillä Daten on tyypitetty olevan merkkijono tai numero. Joudun käyttämään ”@ts-ignore” -kommenttia, millä saan ohitettua tyyppitarkistuksen. Huomaan tämän jälkeen seuraavan ongelman. Jos Date-funktiolle antaa null-arvon, niin Date-funktio asettaa oletuksena päivämäärän 1.1.1970, mikä on UNIX-järjestelmien standardi. Onneksi Typescriptissa pystyy käyttämään myös undefined-arvoa, minkä avulla pystyn suorittamaan testin onnistuneesti.

Testikorjausten jälkeen teen muutoksista pull requestin katselmoitavaksi. Toinen kehittäjä katselmoi muutokseni ja hyväksyy ne, mutta muutoksia ei saada yhdistettyä development-päähaaraamme, sillä syntyy merge conflict. Minun tekemien muutosten aikana tiimin toinen kehittäjä on korjannut ongelmia, jotka syntyivät tiistaisen rajapintapäivityksen takia. Nämä muutokset vaikuttavat maksujen eräpäiviin, joiden testejä juuri itse korjasin. Koska minulla on vanhempi versio päähaarastamme muutosteni pohjalla, niin Git pakottaa katselemaan nämä konfliktoivat muutokset manuaalisesti, jotta pystyn valitsemaan, mitkä muutokset halutaan pitää. Huomaan näistä konfliktoivista muutoksista sen, että logiikka eräpäivän käsittelyssä on muuttunut aika paljon, joten joudun refaktoroimaan testejäni uudestaan.

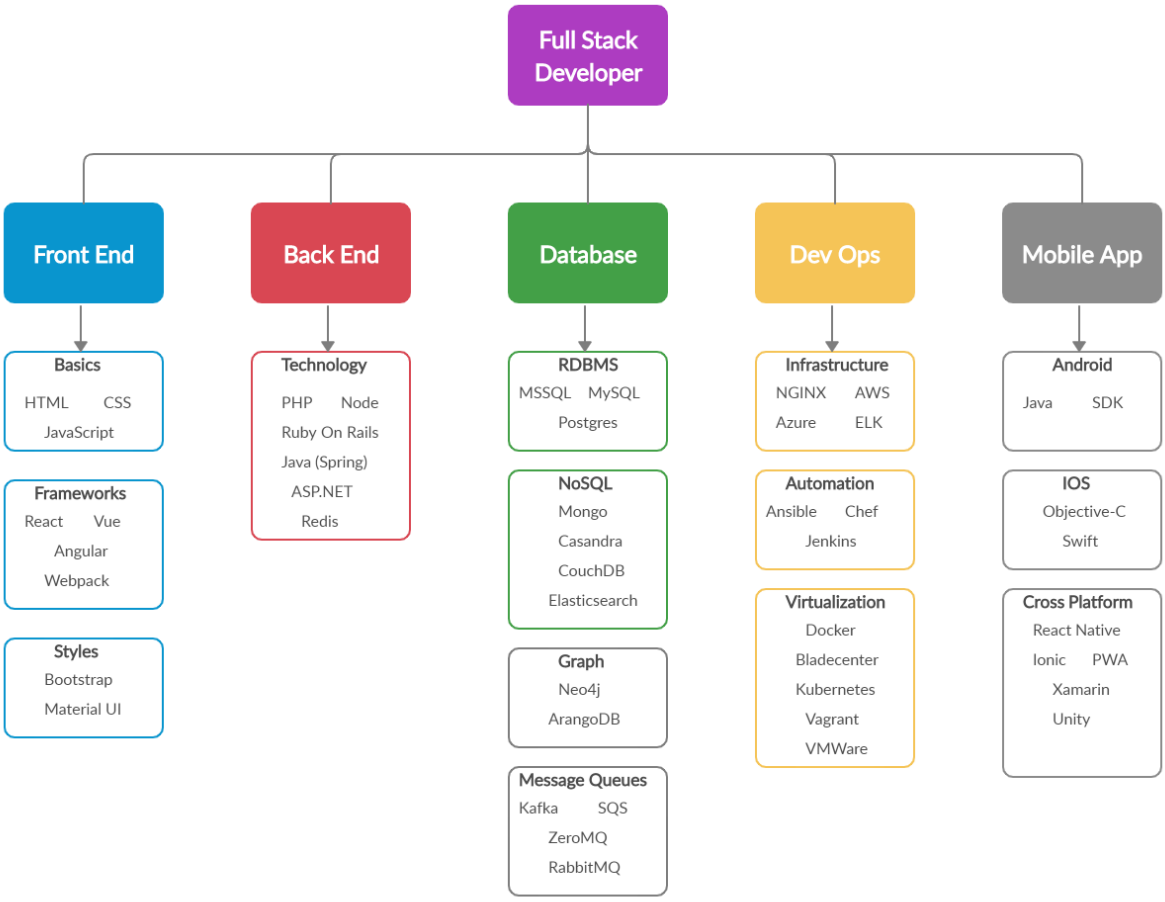
Päivän päätteeksi saan vahvistuspalvelun käyttöliittymän testikorjaukset valmiiksi, mutta en ehdi testaamaan niitä. En myöskään koko päivänä ehtinyt aloittamaan vahvistettujen maksujen yksikkötestien kirjoittamista, joten joudun tekemään niitä mahdollisesti viikonloppuna tai heti tiistaina aamusta, jotta voisimme esitellä demossa molempia ominaisuuksia.

Perjantai 10.4.2020

Pitkäperjantai.

Viikkoanalyysi

Viikon aikana oli monia palavereita, joissa olin mukana suunnittelemassa ja toteuttamassa ominaisuuksia. Työskentelen tittelillä full stack-kehittäjä, mikä yleensä tarkoittaa sitä, että työhöni kuuluu moniin eri sessioihin osallistuminen, sillä osaamistani ja tietämystäni tarvitaan päätettävissä asioissa. Full stack-kehittäjällä pitäisi yleensä olla taitotasoa ohjelmoinnin kaikista osa-alueista, jotta pystyy tarvittaessa jopa itsenäisesti tuottamaan asiakkaalle halutun ratkaisun (Kuvio 3).



Kuvio 3. Full stack-kehittäjän tuntemat osa-alueet (mukaillen Lovinus 2019).

Haluan itse jatkaa kehittymistä juuri tässä roolissa, sillä koen kyseisen roolin kaikista kiinnostavimmaksi. Minusta tuntuu, että keskittymällä pelkästään yhteen osa-alueeseen työpäivät kävisivät pian hieman yksitoikkoiseksi, enkä enää oppisi niin paljon mitä opin tällä

hetkellä. Full stack-kehittäjällä on tärkeää olla yksi osa-alue, mihin on syventynyt ja muut osa-alueet sellaisella tasolla, että niistä on jonkin verran kokemusta tai ainakin sen verran tietämystä, että pystyy oppimaan uusia teknologioita nopeasti näiltä alueilta (Morris 2019; Peham 2017).

Koen, että itselläni vahvin alue on tällä hetkellä frontend sekä devops, jota tukee myös AWS-sertifikaatti, jota varten opiskelen aktiivisesti. Tunnen myös hyvin backendissa Java ja Node.JS-kielet, joita pystyn käyttämään kohtuullisella varmuudella. Paljon on kuitenkin vielä opittavaa, jotta pystyisin hyvällä varmuudella sanoa, että pystyn toteuttamaan tarvittaessa itsenäisesti suuria kokonaisuuksia. Olen ostanut Udemysta kurssin nimeltä The Complete Junior to Senior Developer, jossa käydään läpi monia teknologioita ja käytänteitä, jotka ovat todella hyödyllisiä full stack -kehittäjälle. Siinä esitellään koodin tehokkuutta, koodianalyyseja, tietoturvaa ja eri ohjelmointikieliä. Löysin myös kirjan nimeltä *The Full Stack Developer: Your Essential Guide to the Everyday Skills Expected of Modern Full Stack Developer*, jonka on kirjoittanut Chris Northwood. Teoksessa ei käsitellä pelkästään teknisiä asioita, vaan myös projektin hallintaa ja käyttöliittymien sekä rajapintojen suunnittelumalleja. Nämä kaksi asiaa menevät ehdottomasti jonon kärkeen opiskeltaviksi asioiksi, kun olen saanut muut tehtävät pois alta.

Vahvistuspalvelun käyttöliittymää tehdessäni en ajatellut paljoakaan testejä, sillä ominaisuuden toteuttaminen oli niin kokeilevaa, että TDD-metodin tapainen ajattelu ei olisi ollut kovin helposti toteutettavissa. Minun olisi kuitenkin kannattanut suunnitella kirjoittamaani koodia tarkemmin, kuinka pystyn testaamaan sitä. Tämä aiheutti loppuen lopuksi aika paljon töitä jälkikäteen, jotta sain kaiken koodin katettua yksikkötesteillä.

The Pragmatic Programmer -kirjassa on erittäin hyvä esimerkki yksikkötestauksesta. Kirjassa esitellään koodia, johon on rakennettu logiikka käsittelemään rajatapauksia, virhekäsittelyä ja muita ominaisuuksia. Tämän jälkeen koodille pitäisi kirjottaa yksikkötestit, mikä on todella hankalaa, sillä pitää mahdollisesti mockata monia rajapintoja ja ympäristöjä, jotta funktiota saa mitenkään testattua. Miettimällä ensiksi, miten koodia voi testata, voi löytää loogisia yhteneväisyyksiä, minkä avulla koodista saa siistimpää heti aluksi, eikä sitä tarvitse välttämättä jälkeenpäin refaktoroida (Thomas & Hunt 2019, 260-261).

TDD-periaatetta on välillä hieman hankala toteuttaa frontendissä, sillä käyttöliittymässä logiikka saattaa olla hieman dynaamisempaa kuin backendissä, missä funktioilla saattaa olla selkeitä tehtäviä, eivätkä ne sisällä muuttuvia elementtejä. Aionkin testata yllä mainittua tapaa ajatella testejä koodia kirjoittaessa, mikä on perinteisen ”toteutus ensin” ja TDD-periaatteen välimaastossa ja miten saan hyödynnettyä tätä frontendissä tulevilla viikoilla.

3.9 Seurantaviikko 9: 13-17.4.2020

Maanantai 13.4.2020

Pääsiäismaanantai.

Tiistai 14.4.2020

Tänään pyrin keskittymään vahvistettujen maksujen käyttöliittymään. Iltapäivälle on varattu jälleen demotilaisuus, jossa pitäisi esitellä edellä mainittu käyttöliittymä sekä itse maksujen vahvistaminen. Maksujen vahvistaminen ei ole vielä valmis, sillä ehdin testata sen toimivuutta testiympäristössä, eikä se jostain syystä lataa testiympäristössä public kansiossa sijaitsevia kirjastoja.

Sain viime viikolla valmiiksi vahvistettujen maksujen lajittelulogiikan, minkä perusteella backendista palautuvat maksut lajitellaan omiin taulukoihin. Nyt minun pitäisi tehdä vielä yksikkötestejä tälle kokonaisuudelle sekä iltapäivällä suunnitella hieman, miten tallennan backendista palautuvan maksun ja vien tiedon siitä tälle käyttöliittymälle. Vaikka en tällä kertaa kirjoita koodia TDD-periaatteella, pyrin silti pitämään mielessäni Clean Code -kirjasta saamani opit. Teen testeistä laadullisesti mahdollisimman hyviä ja varmistan, että ne testaavat eri arvoja, joilla saan katettua kaiken kirjoittamani koodin.

Tämän jälkeen siirrymme demotilaisuuteen. Viime viikolla meillä oli hieman huono tuuri demon kanssa, sillä rajapinta oli päivittynyt juuri ennen demoa estäen suurimman osan esiteltävistä ominaisuuksista. Tämä demotilaisuus alkoi vähän samanlaisissa merkeissä, sillä rajapinta, joka palauttaa käyttäjän tilit, oli kaatunut. Onneksi kyseinen rajapinta lähti toimimaan pian ja pääsimme jatkamaan demoa. Esittelin itse tätä vahvistettujen maksujen käyttöliittymää, vaikka se ei ihan vielä toimi halutulla tavalla, mutta ulkoisesti sen pitäisi olla valmis. Asiakkailta tuli hyvää palautetta ja he toivoivat, että koko toiminnallisuus olisi valmis ensi viikon demossa.

Demon jälkeen ehdin hetken suunnittelemaan, kuinka toteutan vahvistetun maksun palautumisen backendista ja kuinka vien tiedon siitä käyttöliittymään. Komponentista pitää tehdä yleiskäyttöinen, sillä sitä pitää pystyä käyttämään sekä omassa siirrossa että normaaleja maksuja tehtäessä. Tiedon viemiseen tulen käyttämään React Contextia tilanhallinnassa, minkä avulla minun pitäisi saada asetettua palautuva maksu Reactin hookkiin, jonka sitten haen käyttöliittymässä.

Tämän jälkeen joudunkin lähteä valmistautumaan opinnäytetyöni seminaariin. Päivän aikana sain vietyä hyvin eteenpäin maksujen vahvistuksen käyttöliittymää siten, että huomenna minun pitäisi saada näytettyä vahvistettu maksu käyttöliittymässä. Viime viikkojen aikana opitut hyvien testien kirjoittamisen periaatteet olivat ahkerassa käytössä tänään ja kun olen saanut tämän kokonaisuuden valmiiksi, yritän reflektoida koko ominaisuuden rakentamista. Samalla arvioin, missä onnistuin ja missä voisin parantaa.

Keskiviikko 15.4.2020

Tänään tavoitteena on saada maksujen vahvistuksen käyttöliittymään näkymään vahvistettu siirto käyttäjän omien tilien välillä. Eilisen selvittelyn perusteella tämä saattaa vaatia pienimuotoista refaktorointia frontendin koodissa.

Lähden ensimmäisenä tekemään omaa Context-tilanhallintaa vahvistetulle maksulle. Tämän avulla saan lisättyä maksun Reactin hookkiin, kun se palautuu backendin vastauksesta. Contextin pitää olla myös yleiskäyttöinen, jotta voin käyttää sitä myös toisessa käyttöliittymässä, jossa käyttäjä voi vahvistaa muita maksuja. Contextia tehdessäni huomaan, että tyyppien nimeämisessä on virheellisyyksiä. Käytän tyyppiä *Payments*, jonka pitäisi olla tyypiltään lista erilaisia maksuja, mutta se on jostakin syystä tyypitetty olemaan vain lista ulkomaisia maksuja. Nimeän tämän tyypityksen uudestaan ja lisään uuden kuvaavamman tyypin pelkille ulkomaisille maksuille. Joudun myös korjaamaan tyypitykset muodostuneet virheet suhteellisen monessa tiedostossa.

Työn aikana saan myös kiireellisen pyynnön tiimimme Scrum Masterilta. Hän pyytää, että ajaisin kaikki automaattitestimme. Testiympäristömme on vaihtumassa uuteen ja nyt pitäisi testata, että legacy-järjestelmämme toimivat. Ajan nämä testit seuraten samalla niiden etenemistä, jotta näen mahdolliset virhetilanteet. Suurin osa testeistä menee läpi, mutta ulkomaanmaksuissa on ongelmia. Ajan testisetit vielä muutaman kerran läpi, jotta saan minimoitua mahdolliset false positive-tulokset.

Testien jälkeen raportoin tulokset Scrum Masterille. Hän pyytää tutkimaan, mikä mahdollisesti aiheuttaa nämä virhetilanteet ulkomaanmaksuissa. Koska testiympäristö on uusi, niin joudun konfiguroimaan SSH-yhteyden uudestaan, jotta pääsen kirjautumaan palvelimelle tutkimaan lokeja. Uuteen testiympäristöön vaaditaan kuitenkin uusia käyttöoikeuksia, joita minulla ei vielä ole, joten teen niistä tiketin IT-tukeen ja jään odottamaan vastausta.
Tämän jälkeen palaan takaisin aamupäivällä kesken jääneeseen työhön. Minun pitäisi vielä kutsua Contextia vahvistettujen maksujen käyttöliittymässä, jonka jälkeen backendis-

ta palautuvan maksun pitäisi näkyä käyttöliittymässä. Teen muutokset ja muutamat virheet lajittelulogiikassa ja huomaan, että maksu näkyy onnistuneesti käyttöliittymässä.

Päivän tavoite tuli täyteen, sillä sain aamulla tavoitteeksi asetetun tehtävän tehtyä, vaikkakin jouduin keskeyttämään tekemisen hetkeksi päivän aikana. Päivän aikana en törmännyt sen isompiin haasteisiin, mutta huomasin, kuinka tärkeää on kiinnittää huomiota muuttujien nimeämiseen, sillä liian geneeriset muuttujien nimet saattavat olla epäjohdonmukaisia ja vaikeita ymmärtää muille kehittäjille.

Torstai 16.4.2020

Tänään joudun refaktoroimaan eilen valmiiksi tullutta vahvistettujen maksujen käyttöliittymää, sillä huomaan, että käyttöliittymässä käytettävät komponentit ovat aika suuria. Niissä käytettävät avustusfunktiot voisi siirtää johonkin muualle, sillä tulen tarvitsemaan niitä myös myöhemmässä vaiheessa.

Lähden ensimmäisenä suunnittelemaan, miten toteutan tehtävän ja mitä koodia siirrän mihinkin tiedostoon ottaen huomioon samalla mahdolliset riippuvuudet. Lähtöhetkellä React komponentin pituus on lähes 200 riviä, mikä on aivan liian pitkä, sillä silloin sen luettavuus ja ymmärrettävyys heikkenee.

Teen ensimmäisenä oman utility-tiedoston, johon avustusfunktioita voidaan siirtää. Määrittelen ne myös exportiksi, jolloin niitä pystyy käyttämään missä tahansa sovelluksessa. Tämän jälkeen lähden refaktoroimaan itse näytettäviä elementtejä eli TSX-elementtejä, jotka ovat hyvin HTML-elementtien kaltaisia. Olen sekoittanut TSX:n sekaan ternary-operaattoreita, jotka tekevät koodista hieman vaikeammin ymmärrettävää. Koodi näytti aiemmin tältä (Kuva 5):

```
debtor={
    isOwnTransfer(payment) && !isNil(acknowledgementData.debtor)
        ? acknowledgementData.debtor.ownerName
        : payment.debtorName
}
```

Kuva 5. TSX-elementin parametri, jossa ternary-operaattori.

Kun refaktoroinnin jälkeen se näyttää tältä (Kuva 6):

```
debtor={getDebtorName(payment, acknowledgementData)}
```

Kuva 6. Sama parametri kuvailevalla funktiolla.

Tästä huomataan, että jälkimmäinen koodi näyttää siistimmältä. Selkeiden muuttujien nimien avulla lukija myös ymmärtää, mitä siinä luultavasti tapahtuu ilman, että katsoo funktion koodia.

Päivä menee pitkälti komponentin refaktoroinnissa, minkä jälkeen jouduin korjaamaan myös yksikkötestit, sillä ne menivät rikki asioiden muututtua. Refaktoroinnin tarve tuli siitä, että olin tehnyt työtä aika kokeilevalla asenteella yrittäen saada käyttöliittymää toimimaan halutulla tavalla. Tämä aiheutti aika paljon refaktoroitavaa koodia, sillä monia asioita oli jätetty optimoimatta tästä syystä. Onnistuin tehtävässä hyvin, sillä sain vähennettyä komponentin rivimäärän lähes 200 rivistä noin 40 riviin.

Perjantai 17.4.2020

Aamun ensimmäisenä tehtävänä esittelen Scrum Masterillemme eilen valmiiksi tullutta vahvistettujen maksujen käyttöliittymää käyttäjän oman siirron osalta, jotta saisin tietää, olisiko siinä mahdollisesti jotain parannettavaa. Hän toivoisi, että käyttöliittymässä näytettäisiin myös maksajan ja saajan nimi, sillä käyttäjä voi siirtää rahaa esimerkiksi kahden oman yrityksensä välillä.

Käymme läpi mitä tällainen vaatisi ja kerron hänelle, että helpointa olisi, jos nämä tiedot palautuisivat maksun mukana minkä saamme backendin vastauksesta. Tämä ei kuitenkaan ole vielä mahdollista, sillä backendissa kutsuttava rajapinta ei palauta näitä tietoja, kun käyttäjä tekee omaa siirtoa. Voimme tehdä pyynnön tiimille, joka on vastuussa rajapinnasta. Tavoitteena olisi, että he lisäisivät rajapinnasta palautuvaan dataan myös nämä tiedot, mutta sillä välin Scrum Masterimme suosittelee kehittelemään oman ratkaisun.

Koska kyse on omasta siirrosta, saan ajatuksen, että pystyn hakemaan tilin omistajan tiedot käyttöliittymään palautuvista tilitiedoista ja tällä tavoin lisäämään nämä tiedot saajan ja maksajan tietoihin. Lähden refaktoroimaan aika isosti Contextia, jota käytetään backendista palautuvan maksun tilan tallettamiseen, sillä sen pitää tallettaa nyt myös tilitiedot. Eilen valmistuneen refaktoroinnin myötä komponentti on nyt myös pilkottu hieman pienempiin osiin, joten joudun miettimään tarkkaan, missä kaikkialla tilitietoa pitää liikkua.

Olen saamassa työn lähes valmiiksi, kun saan viestin tiimimme pääsuunnittelijalta. Toinen tiimi tekikin heti pyytämämme muutoksen ja hän haluaisi tehdä nyt muutokset backendiin, jotta tiedot saadaan lähetettyä frontendiin. Koko päivän työni valuu siis muutamassa minuutissa hukkaan, koska en tiennyt, että toinen tiimi pystyy tekemään muutoksen heti.

Tästä päivästä opin sen, että kommunikaatio tiimin sisällä on erittäin tärkeää. Jos olisin heti saanut tiedon, että toinen tiimi voi toteuttaa halutun ominaisuuden heidän rajapintaansa, olisin voinut keskittyä johonkin muuhun tehtävään. Nyt hukkasin aikaani kehittelemällä uuden ratkaisun, jolla ei loppujen lopuksi tehty mitään.

Viikkoanalyysi

Tällä viikolla lähes joka päivä tein koodin refaktorointia. Se on välttätön osa koodausta, sillä vaikka kuinka suunnittelee ennen toteuttamista, niin vaatimukset ja tarpeet muuttuvat, jolloin toteutusta pitää muuttaa jollakin tapaa. Martin Fowler määrittelee refaktoroinnin seuraavasti: *"disciplined technique for restructuring an existing body of code, altering its internal structure without changing its external behavior"* (Thomas & Hunt 2019, 252).

Refaktorointia tulisi suorittaa silloin kun on oppinut jotakin uutta ja ymmärtää jonkin asian paremmin. Syitä voi olla esimerkiksi koodin toistuvuus, vanhentunut tieto, käyttötapaukset tai suorituskyky (Thomas & Hunt 2019, 252-253). Tällä viikolla kyse oli enemmänkin siitä, että olin tehnyt paljon koodia ilman, että olisin paljoa keskittynyt sen optimoimiseen. Sitä on kuitenkin hyvä suorittaa näin varhaisessa vaiheessa, sillä mitä myöhemmäksi optimoinnin jättää, sitä hankalampaa sitä on toteuttaa enää. Kyseinen komponentti saattaa olla laajentunut jo niin paljon myöhemmässä vaiheessa, että työtuntien määrä sen optimoimiseen moninkertaistuu.

Refaktorointi yleensä rikkoo automaattitestit, sillä asiat vaihtavat paikkojaan ja funktioiden sisäinen logiikka saattaa myös hieman muuttua. Itse olisin voinut ajatella testejä kirjoittaessani ja jopa korjannut testit ensin ennen kuin lähden refaktoroimaan koodia TDD-periaatteen mukaisesti. Tällöin testit olisivat ohjanneet työskentelyäni ja toimineet suuntaviivoina, kuinka refaktoroisin koodin, sillä testit olisivat menneet läpi siinä vaiheessa, kun työ olisi valmis.

Tällä viikolla huomasin myös, kuinka oleellista hyvä kommunikaatio on ohjelmistoprojektissa. Tiimissämme on tällä hetkellä 10 kehittäjää, joten muiden töistä ja tekemisistä on välillä hankalaa ottaa selvää varsinkin, kun kaikki työskentelevät nyt etänä. Ketterässä

tiimissä tämä määrä henkilöitä alkaa olemaan jo äärirajoilla, sillä kommunikaatiolinkit ovat kasvaneet eksponentiaalisesti. Jos olet tiimissä, jossa on sinun lisäksi vain toinen henkilö, niin kommunikaatiolinkkejä on vain yksi. Neljän hengen tiimissä luku on jo 6 ja kymmenen hengen tiimissä mahdollisia kommunikaatiolinkkejä on 45, joten tieto ei välttämättä aina välity kaikille (DevIQ).

Tiimimme pääsääntöiset kommunikaatiovälineet tällä hetkellä on Microsoftin Teams-alusta sekä sähköpostit. Teams-kanavaa ei tule välttämättä aina seurattua, kun tekee töitä, sillä se on häiritsevä tekijä eri kanavilta saapuvien viestien vuoksi. Tämä vaikeuttaa keskittymistä, jolloin tehokasta koodia on hankala kirjoittaa. Tutkimuksen mukaan kasvokkain keskustelu on 34 kertaa tehokkaampaa mitä sähköpostin välityksellä (Bauer 2018). Pystyn lähes allekirjoittamaan edellä mainitun väitteen, sillä toimistolla kaikkien ollessa paikalla oli huomattavasti helpompaa kävellä toisen luokse vaihtamaan muutama sana ongelmasta tai kysymyksestä.

Tästä herääkin idea, voisimmeko yrittää jotain kommunikaatiotapaa, joka olisi mahdollisimman lähellä toimistoympäristöä, ilman että olisimme fyysisesti lähekkäin. Mieleeni herää ainakin ajatus, että voisimme työpäivän alussa liittyä yhteiseen puheluun Teamsin kautta, jossa kenenkään ei tarvitse puhua mitään. Jos on asiaa jollekin henkilölle, niin on helppo kysyä häneltä puhelussa tätä asiaa tai mahdollisesti siirtyä yksityiseen puheluun, jos asiaa pitää käsitellä enemmän eikä haluta häiritä muita.

3.10 Seurantaviikko 10: 20-24.4.2020

Maanantai 20.4.2020

Testiympäristöämme päivitetään tänään koko päivän, joten en pysty oikeastaan edistämään mitään töitä projektin suhteen. Käytän siis tilaisuuden hyväksi ja harjoittelen AWS-sertifikaattia varten, sillä en viime viikolla ehtinyt käydä ollenkaan materiaaleja läpi sen osalta.

Koe on noin kahden viikon kuluttua ja olen käynyt kurssin kaikki osa-alueet läpi. Nyt jäljellä on vain kertaus ja harjoituskokeiden teko. Aloitan tekemällä harjoituskokeen, jotta näkisin, millä alueella minulla on eniten vaikeuksia. Saan tällä kertaa oikeita vastauksia 80 %, mikä alkaa olemaan suositusten rajalla, mikä harjoituskokeista pitäisi saada ennen kuin yrittää oikeaa koetta. Eniten ongelmia minulla oli VPC:tä käsittelevällä alueella, mitä opiskelin seurantaviikolla 5. Taustani on aika lailla pelkästään sovelluskehityksessä, joten verkkoinfrastruktuuri, IP-protokollat ja muut näihin liittyvät asiat eivät ole vielä niin selkeitä minulle.

Internetistä löytyy onneksi paljon oppaita, jotka käsittelevät VPC:tä. Kaikista luotettavimpia lähteitä on AWS:n omat oppaat, jotka selittävät erittäin tarkasti, miten palvelut toimivat ja miten niitä voi konfiguroida omaan tarpeeseen. Nämä oppaat kuitenkin olettavat, että tunnet aika hyvin AWS:n palveluverkoston, joten joudun käyttämään Googlea apuna, jotta voin tarvittaessa etsiä termejä, joita en vielä oikein tunne.

Löydän myös muutamia luentoja videon muodossa, mutta niiden kanssa pitää olla tarkkana, sillä ne saattavat olla jo muutaman vuoden vanhoja. AWS:n sertifikaattikokeet päivittyvät joka vuosi, joten luennoissa saattaa olla tietoa, mikä ei ole enää ajankohtaista.

Päivä menee pitkälti sertifikaatin osa-alueiden kertaamisessa ja varsinkin VPC:n opiskeluun. Löysin paljon uutta tietoa AWS:n dokumentaatiosta ja kurssin materiaaleista. Ne osaltaan myös vastasivat kysymyksiin, jotka minulla menivät väärin harjoituskokeessa. Tulevina päivinä minun pitää jatkaa muiden alueiden kertausta, jotta kaikki pysyy tuoreessa muistissa pian suoritettavassa sertifikaattikokeessa.

Tiistai 21.4.2020

Aamun Scrum Dailyssa ilmoittaudun vapaaehtoiseksi käymään kaikki koodikatselmoinnit läpi. Niitä on kertynyt jonoon jo yli 10 kappaletta ja muutamissa on ominaisuuksia, joita tarvitsen frontendissä, jotta voin edistää omia töitäni.

Aloitamme päivän grooming palaverilla, missä käymme läpi JIRAssa olevat tiketit sekä niiden statuksen. Kaikilla kehittäjillä on ollut nyt paljon tehtäviä työn alla, joten välillä on järkevää tarkistaa kaikkien töiden status, sillä tiketin käsittelijä on saattanut unohtaa siirtää tiketin JIRAssa oikeaan tilaan. Groomingissa yleensä priorisoidaan ja tarkennetaan joitakin tehtäviä, mutta koska sovelluksen julkaisupäivä on kuukauden päästä, niin mitään uusia tehtäviä ei enää tule. Kaikki tehtävät pitäisi olla prioriteettijärjestyksessä jo nyt.

Tänään en yritä löytää mitään uusia tapoja tehdä koodikatselmointeja, vaan yritän hyödyntää kaikkea aiemmilla viikoilla oppimaani. Koodikatselmointien ohella käytän Googlen kirjoittamaa opasta, kuinka koodikatselmointi tehdään sekä kirjoitan positiivisia kommentteja muutoksiin, jotka on mielestäni tehty erinomaisesti. Koodikatselmoinneissa on paljon lisäyksiä sekä muutoksia maksujen validointeihin. Näitä katselmoidessa pidän myös silmällä vaatimusmäärittelyä ja validaatioiden vuokaavioita, jotka on piirretty helpottamaan logiikan ymmärtämistä.

Katselmoidessani tiimimme kokeneimman kehittäjän koodia huomaan, kuinka hyvää se on laadultaan. Muuttujat ovat nimetty kuvaavasti, koodissa käytetään Javan uusimpia syntaksitapoja ja yksikkötestit on pilkottu pieniin osiin, jotka testaavat kattavasti koko toteutusta. Käytän näihin katselmointeihin enemmän aikaa mitä tarvitsisi, sillä yritän ymmärtää, miten koodista on saatu niin laadukasta ja miten olisin mahdollisesti itse sen tehnyt. Näin pyrin sisäistämään kaiken koodista, jotta pystyisin hyödyntämään opittuja asioita tulevaisuudessa.

Ehdin päivän aikana käymään läpi kaikki koodikatselmoinnit, jotta mitään ei jää huomiselle. Tuntuu, että opin päivän aikana paljon varsinkin Javasta, sillä seniorikehittäjiemme koodi oli todella selkeää ja ymmärrettävää. Java 8 ja uudempien versioiden mukana tulleet streamit ja ::-syntaksi ovat minulle vielä hieman vieraita, mutta tämän päivän jälkeen uskon, että pystyn oppimaan ne aika nopeasti.

Keskiviikko 22.4.2020

Alamme olla vahvistuspalvelun käyttöliittymän kanssa loppusuoralla. Kollega oli eilen illan päätteeksi saanut konfiguroitua vahvistuspalvelun backendia siten, että sinne voisi koittaa nyt lähettää vahvistettavat maksut ja katsoa, mitä rajapinta vastaa tähän. Tänään päivän ainoa palaveri on demotilaisuus, joten aikaa pitäisi olla riittävästi saada tehtävä valmiiksi.

Tehtävä on suhteellisen yksinkertainen. Minun pitää tehdä funktio, joka tallettaa muuttujaan valitut maksut, jonka jälkeen teen toisen funktion, joka lähettää ne backendille nappia painamalla. Tässä vaiheessa ohitamme kokonaan itse käyttäjän tekemän manuaalisen vahvistuksen, sillä haluamme testata miten backend ottaa nämä maksut vastaan ja mitä rajapinta vastaa, kun lähetämme tiedot sinne.

Tehdessäni toteutusta teen tietoisen päätöksen, että en keskity nyt optimoimaan koodia. Koko prosessi on hieman sekava vielä, enkä voi mitenkään suunnitella hyvää ja toimivaa ratkaisua tällä hetkellä. Tässä tapauksessa pitää mennä ikään kuin askel kerrallaan ja optimoida vasta sen jälkeen, kun koko prosessi menee onnistuneesti läpi.

Päivän aikana saan pyynnön, että päivän valitsijakomponentti pitäisi korjata ennen demoa, jotta voisimme tehdä eräpäivällisiä maksuja. Keskeytän nykyisen työtehtäväni ja lähden tutustumaan tähän ongelmaan. Käytössämme oleva komponentti on toisen tiimin tekemä ja se päivittyy jatkuvasti, joten aloitan päivittämällä sen viimeisimpään versioon. Kun laitan komponentin kiinni uuden maksun lomakkeeseen, huomaan, että tyylit menevät uudessa komponentissa täysin rikki. Näyttää muutenkin, että muutamassa käyttöliittymässä tyylit eivät lataudu oikein. Raportoin tämän tiimille ja kerron, että tämä tehtävä ei tule valmiiksi demoon mennessä, sillä tyylien rikkoutumisessa voi mennä tovi.

Päivän päätteeksi saan toimivan version valmiiksi funktiosta, mikä lähettää valitut maksut backendille. Emme pääse tätä vielä testaamaan, sillä tätä pitää testata testiympäristössä, koska lokaalisti vahvistuspalvelun rajapintaa ei pysty kutsumaan. Toivon mukaan huomenna ehtisimme testaamaan tätä kollegan kanssa, jotta saisimme nopeasti koko prosessin valmiiksi ja implementoitua koko vahvistuspalvelun kokonaisuudessaan käyttöliittymäämme.

Torstai 23.4.2020

Viime päivien aikana olen tehnyt aktiivisesti vahvistettujen maksujen käyttöliittymää. Tänään onkin tavoitteena saada lisättyä kuittikomponentti käyttöliittymään, jotta käyttäjä saa

tulostettua maksun tiedot kuitille. Testaamme tänään myös viime viikolla pohtimaani koko päivän kestävää Teams-puhelua, jossa tiimin jokainen jäsen on paikalla mikit vaimennettuna ja voi kysyä, jos tulee jotain mieleen.

Kehittäjä toisesta tiimistä on tehnyt meille tilitapahtumiin keskittyvän käyttöliittymän, jonne hän on lisännyt myös kuittikomponentin. Lähden ensimmäisenä tutkimaan, miten hän on tehnyt kyseisen toteutuksen, jotta pystyisin mahdollisesti käyttämään sitä hyödyksi. Huomaan, että hän on käyttänyt modaalia, joka aukeaa popup-ikkunan muotoisesti keskelle näyttöä. Ratkaisu on sinällään siisti ja toimii hyvin, mutta siinä on yksi suuri ongelma. Koska modaali ei ole itsessään oma sivu, käyttäjällä ei ole mitään mahdollisuutta tulostaa kuittia. Varmistan tämän menemällä tulostuksen esikatseluun ja kuten arvelinkin, tulostus ei toimi ollenkaan, sillä papereita tulostuisi seitsemän kappaletta. Jos kuitti olisi omassa ikkunassa, papereita tulostuisi vain yksi.

Lähden siis toteuttamaan omaa toteutusta ja yritän tehdä siitä mahdollisimman yleiskäyttöisen, jotta voin jälkeenpäin refaktoroida toisen kehittäjän toteutuksen hyödyntämään omaani. En ole aiemmin tehnyt mitään komponentteja, jotka avautuvat uuteen ikkunaan. Lähden ensimmäisenä tutkimaan, miten kyseisen tehtävän saisi suoritettua. Netistä haetun tiedon perusteella Javascriptissä on window.open()-niminen funktio, joka avaa uuden ikkunan. Nyt minun pitäisi vielä selvittää, miten saan React-komponentin parametreineen tuon ikkunan sisään, sillä kyseinen funktio ottaa argumenttina vain string-tyyppiä.

Työn ohessa Scrum Masterimme pyytää yhteisessä puhelussa kehittäjiltä apua bugin selvittämiseen. Ilmoittaudun vapaaehtoiseksi ja lähdemme selvittämään, mikä on vialla. Hän kertoo, että jostakin syystä valuuttatilit ovat disabloitu, kun yrittää tehdä omaa siirtoa. Kuulostaa vialta frontendissä tapahtuvassa logiikassa, joten lähden tutkimaan koodia sen osalta. Bugi on helppo selvittää, sillä löydän heti funktion, mikä tarkistaa, onko tili tyypiltään valuuttatili ja disabloi oman siirron tästä syystä. Poistan tämän ehdon ja varmistan, että bugi on korjaantunut kirjoittamalla muutokselle yksikkötestin.

Päivä tulee päätökseen, enkä ihan vielä saa valmiiksi kuitin lisäämistä vahvistettujen maksujen käyttöliittymään. Minun pitää huomenna jatkaa tutkimista, miten saisin React-komponentin renderöityä uudessa ikkunassa. Koko päivän kestävä Teams-puhelu oli aika onnistunut. Tiimiltä tuli pääsääntöisesti hyvää palautetta, että saa tarvittaessa kysyttyä kollegalta apua. Joitain tämä hieman häiritsi, sillä välillä saatettiin puhua asioista, jotka eivät koskettaneet kuin yhtä tai kahta henkilöä. Mutta tätä voi ehdottomasti lähteä parantamaan ja kokeilemaan tulevinakin päivinä, jos koronapandemian rajoituksia jatketaan.

Perjantai 24.4.2020

Tänään tavoitteena on viimeistellä eilen kesken jäänyt kuitin näyttäminen vahvistettujen maksujen käyttöliittymässä. Toivon mukaan saan tehtävän tänään valmiiksi, sillä tämän jälkeen olisi tarkoitus hypätä takaisin vahvistuspalvelun pariin ja saattaa se valmiiksi.

Tehtävä jäi minulta eilen siihen, että en saanut renderöityä React-komponenttia uuteen ikkunaan. Yritin tutkia netistä vinkkejä, miten saisin tehtävän suoritettua. Löysinkin muutamia oppaita, jotka tekivät niin, mutta niissä käytettiin vanhempaa React-versiota, missä on vielä käytössä lifecycle-metodit. Tämä tekee dokumentaation lukemisesta hieman haastavampaa, sillä joudun miettimään, miten vanhassa versiossa tehty toteutus kääntyy uuteen versioon.

Tutkiessani nettiä löydän myös ulkopuolisen kirjaston nimeltä React-new-window, joka tekee juuri sen, mitä nimessä sanotaan. Kyseisellä kirjastolla saa renderöityä mitä tahansa uuteen ikkunaan laittamalla sen kirjaston mukana tulleen *NewWindow*-elementin sisään. Asennan tämän kirjaston ja yritän tehdä testitoteutuksen sillä. Testi onnistuu ja simppeli React-komponentti renderöityy uuteen ikkunaan, joten voin lähteä toteuttamaan kuitin ja maksun tietojen näyttämistä.

Päivän päätteeksi saan kuitin valmiiksi. Se tulee vaatimaan vielä paljon CSS-tyylejä, jotta se näyttää siltä miltä halutaan, mutta sen työn voi jättää lähemmäksi julkaisua, sillä tärkeämpää on saada kaikki toiminnallisuudet valmiiksi.

Viikkoanalyysi

Vahvistuspalvelun käyttöliittymää tehdessäni päädyin käyttämään ulkopuolista kirjastoa. Projektissamme on käytössä jo todella monta kirjastoa, ja nämä alkavat kasvattamaan sovelluksen kokoa jo aika paljon. Tämä näkyy siinä, että koodin kääntymisessä kestää kauemmin ja CI/CD-putkessa menee kauemmin. Putkessa pitää aina asentaa kaikki kirjastot, mihin tällä hetkellä kuluu noin 5 minuuttia.

Kirjastojen hyötynä pidetään sitä, miksi keksiä pyörä uudestaan, kun joku on sen jo tehnyt. Tämä on sinällään totta, sillä osa kirjastoista ovat todella hyvin toteutettuja ja ne saattavat auttaa todella paljon jonkin sellaisen asian toteuttamisessa, missä itsellä vastaavan toteutuksen koodaamisessa menisi kauan. Välillä kannattaa kuitenkin kysyä itseltään, kannattaako ulkopuolista kirjastoa käyttää tässä tilanteessa. Esimerkkinä tästä voisi olla kirjasto, joka formatoi summan oikeaan muotoon maan perusteella. Kirjasto voi olla todel-

la helppokäyttöinen ja sisältää vain yksinkertaisen funktion. Esimerkiksi seuraavanlainen voisi olla mahdollinen: *formatCurrency("fi", 1000.50)*. Kyseinen funktio ottaa argumenttina maan ja summan, minkä mukaan summan pilkut laitetaan maakohtaisesti oikein. Tämä on helppo ja nopea ottaa käyttöön, mutta kirjaston mukana saattaa tulla todella paljon toiminnallisuutta, mitä et tarvitse. Tämä kasvattaa turhaan sovelluksen kokoa, sillä voisit tehdä itse vastaavanlaisen toteutuksen pelkällä Typescriptilla. Ulkopuolisten kirjastojen ongelmana on myös se, että ne ovat monesti riippuvaisia toisista kirjastoista. Kirjastoja ei myöskään välttämättä kehitetä aktiivisesti, jolloin ne ovat alttiita tietoturvauhkille tai uudet versiot ohjelmointikielistä eivät välttämättä enää tue niitä. Pahimmillaan kirjasto voidaan ottaa NPM:stä pois kokonaan, jolloin kaikki sovellukset, jotka käyttävät niitä eivät käänny ja hajoavat, kuten TechRepublic ja TheRegister -uutissivustot kertovat (Williams 2016; Sanders 2018)

Kehittäjän pitää siis olla tarkkana, milloin on hyvä tilaisuus käyttää ulkopuolista kirjastoa. Onko parempi käyttää ehkä vähän enemmän aikaa ja tehdä oma toteutus, joka toimii varmasti ja jota voi itse ylläpitää. Vai olisiko parempi käyttää ulkopuolista kirjastoa, joka tekee työn nopeasti, mutta jota ei voi itse hallita?

Viikon alussa sain AWS:n harjoituskokeesta ensimmäistä kertaa suositellun 80 % oikein. Olen hieman jäljessä aikataulustani, sillä toivoin saavani tällä viikolla jo 85 % oikein. Opiskeluaikataulusta on ollut todella paljon hyötyä, sillä olen voinut seurata tarkasti, mitä opiskelen milläkin viikolla ja miten olen onnistunut asetetuissa tavoitteissa. Koe on kahden viikon kuluttua ja tällä hetkellä uskon vahvasti, että pääsen kokeesta läpi. Tämän jälkeen onkin aika kääntää katse jo seuraavaan vaiheeseen.

Kun olen suorittanut sertifikaatin, uskon vahvasti, että minulla on sellaisen kehittäjän taitotaso, jota odotetaankin olevan noin kaksi vuotta työelämässä olleella. Tämän jälkeen lähden suunnittelemaan, miten voisin kehittyä senior-tasolle kehittäjänä ja teen samantyylisen opiskeluaikataulun tälle polulle, jotta voin seurata kehittymistäni myös tässä. Tämä tulee olemaan huomattavasti pidempi ajanjakso kuin sertifikaattia varten. Pyrin tekemään sille vuoden mittaisen aikataulun, jotta voisin vuoden päästä reflektoida, kuinka olen kehittynyt ja olisinko jo lähempänä senior-tasoa.

Olen ostanut Udemysta Andrew Neagoien kurssin *"The Complete Junior to Senior Developer"*. Hän on kirjoittanut artikkelin, mitä taitoja senior-kehittäjältä odotetaan. Hän tuo esiin artikkelissaan kuusi kohtaa, jotka ovat tärkeitä senior-kehittäjälle. Nämä ovat: tekninen taito, tiimityöskentely, asiakastaidot, kehittymiskyky, myynnin ja rekrytoinnin taidot sekä yhteisötaidot. Teknisistä taidoista esiin tuodaan ajattelutapa, miksi jotain teknologiaa pitäi-

si käyttää ja miksi tämä ongelma on olemassa. Tiimityöskentelyssä kehittäjän pitäisi pystyä työskentelemään missä tahansa tiimissä ja opastamaan mahdollisesti nuorempia kehittäjiä. Asiakastaidoiltaan senior-kehittäjän pitää pystyä ymmärtämään, mitä asiakas oikeasti haluaa ja tarjota ratkaisua näihin ongelmiin. Kehittymisen kannalta on tärkeää jatkaa edelleen oppimista ja ymmärtää miksi esimerkiksi jQuery on ollut aikanaan hyvä, mutta ei ehkä enää nykypäivänä. Myynnissä kehittäjät voivat toimia tukena antamassa arvioita, kuinka kauan projektin tekeminen mahdollisesti kestää ja mitä teknologioita sen tekemisessä voidaan hyödyntää. Viimeisenä taitona on yhteisötaidot ja kuinka kehittäjä voi tuottaa arvoa koko kehittäjäyhteisölle. Senior-kehittäjällä on yleensä sen verran kokemusta tietyistä asioista, että osaamista voi esitellä erilaisissa kehittäjien tilaisuuksissa presentaatioiden avulla tai vaikkapa itse tehdyn open source -kirjaston avulla (Neagoie 2017).

Nämä kaikki taidot tulen lisäämään myös yksittäisiksi kohdiksi omaan opiskeluaikatauluuni, jotta voin seurata myös näiden osa-alueiden kehittymistäni. Täten voin seurata, olenko jäänyt jälkeen esimerkiksi myynnin taidoissa ja toisaalta taas kehittynyt paljon enemmän asiakastaidoissa.

4 Pohdinta ja päätelmät

Olen työskennellyt koko työn seurannan ajan saman projektin ja sovelluksen kehittämisen parissa. Tämä tekee lähtötilanteen ja nykytilanteen vertailusta huomattavasti helpompaa, kun jakson ajalle suunnitellut omat työtehtävät ovat toteutuneet suunnitelmieni mukaisesti ja myös tiimikollegat sekä asiakasyrityksen sidosryhmät ovat pysyneet samoina. Oikeastaan vain lähtötilanteessa kuvattu osaamistasoni ja myös työskentelytapani ovat kehittyneet tänä aikana isoin harppauksin. Merkittävä muutos on tapahtunut myös fyysisessä työympäristössä, sillä keväällä 2020 puhjenneen maailmanlaajuisen koronapandemian vuoksi koko tiimimme siirtyi maan hallituksen suositusten mukaisesti etätyöskentelemään kotoa käsin. Fyysisestä etäisyydestä huolimatta tämä ei kuitenkaan suuresti muuttanut tiimimme työskentelyä, vaan työskentelymme jatkui keskeytyksettä kotoa käsin hyvien tietoliikenneyhteyksien ansiosta. Tätä tilannetta helpotti osaltaan se, että projektimme oli päässyt jo hyvään vauhtiin ja työtapamme vakiintuneet rajoitusten tullessa voimaan huhtikuussa 2020.

Lähtötilanteessa kerroin kehitettävässä sovelluksessa käytettävistä teknologioista, joita olivat Typescript, React, Java, Jenkins, AWS ja Robot Framework. Projektin alussa varsinkin Typescript oli minulle uusi teknologia, vaikka Javascriptilla olinkin jo tottunut kirjoittamaan koodia. Vaihtaminen dynaamisesta Javascriptistä staattiseen Typescriptiin tuntui aluksi todella haastavalta tehtävältä, sillä olin tottunut, että muuttujien ja funktioiden tyyppejä ei tarvitse miettiä. Erityisesti seurantaviikkojen alussa minulla oli kyseisen kielen kanssa suuria haasteita ja sitä lisäsivät vielä tiukat linter-säännöt. Pikkuhiljaa kuitenkin totuin edellä mainitun kielen syntaksiin, opin etsimään tietoa eri tyypeistä, opin myös hahmottamaan, kuinka voin rakentaa omia tyypityksiä ja miten pystyn konfiguroimaan linterin sääntöjä, jotta niitä voi tarpeen mukaan tiukentaa tai löysentää. Seurantajakson lopussa koen hallitsevani edellä mainitut teknologiat, ja osaan tarvittaessa opastaa muita niiden käytössä.

Seurantaviikkojen alussa olin asiakasprojektissa työskentelyn lisäksi aloittamassa AWS-sertifikaatin opiskelua, enkä tiennyt tuosta palvelusta vielä kovinkaan paljoa. Nyt 10 viikon jälkeen olen lähes valmis suorittamaan sertifikaattikokeen. Minulla on nyt selkeä ymmärrys AWS:n keskeisimmistä palveluista, joita voin jatkossa hyödyntää sekä omissa projekteissani että konkreettisesti työelämässä, kun mietin, mitä palveluja erityyppisten ongelmien ratkaisemiseksi voisi hyödyntää.

Tunnen kehittyneeni yleisesti myös full stack -kehittäjänä paljon. Vaikka tässä projektissa toteuttamani ominaisuudet olivat suurelta osin frontendissä tehtäviä töitä, olivat työtehtä-

väni kokonaisuudessaan silti todella monipuolisia ja vaativat laaja-alaista osaamista. Joka viikko katselmoin omien työtehtävieni ohessa muiden tiimiläisten koodimuutoksia ja tein niihin parannusehdotuksia. Löysin myös erittäin paljon hyödyllistä materiaalia lähdekirjallisuudesta, minkä avulla pystyin antamaan rakentavaa palautetta tiimiläisille sekä ehdottamaan, miten asioita voisi vielä optimoida. Koodikatselmointien avulla pystyin myös syventämään oppimaani niistä teknologioista, joiden parissa en työskennellyt päivittäin. Opin projektin aikana uusia käytänteitä ja hyviä tapoja myös tiimimme senior-kehittäjiltä. Pikkuhiljaa sain enemmän vastuuta myös tekemällä julkaisupaketteja legacy-järjestelmistä, jotka viedään tuotantoon sadoille tuhansille käyttäjille. Seurantajakson lopussa voin todeta hallitsevani testiautomaation niin vahvasti, että tiimini luottaa minulle koko osa-alueen täydellisen vastuun.

Lähtötilanteessa kuvailin vuorovaikutustaitojen merkitystä konsultin työssä. Omasta mielestäni olen kehittynyt paljon myös tämän osalta näiden seurantaviikkojen aikana. Jokaviikkoiset tapaamiset niin oman tiimin kesken kuin myös asiakasyrityksen sidosryhmien kanssa ovat kehittäneet erittäin paljon omia vuorovaikutustaitojani. Osaan ottaa sovelluskehityksessä huomioon niin asiakkaan tarpeita kuin erilaisten teknologioiden vaatimuksia ja esittää ne asiakkaalle ymmärrettävällä tavalla. Teknologioiden ymmärtäminen auttaa kuvailemaan niiden mahdollisuuksia ja rajoituksia myös asiakkaalle.

Oma luottamukseni omiin taitoihini on kasvanut vahvasti näiden viikkojen aikana. Yhteiset palaverit, sovelluksen arkkitehtuurin läpikäynti ja monet selvitystyöt ovat tuoneet minulle huomattavasti paljon enemmän itsevarmuutta esittää mahdollisia korjauksia vaativia asioita ja selvittää erilaisia esiin nousevia ongelmia. Pohtiessamme kehitystiimin kanssa sovelluksen ominaisuuksiin liittyviä vaatimuksia, pystyn aktiivisesti punnitsemaan erilaisia vaihtoehtoja ja tuomaan niitä esiin, kuinka halutut ominaisuudet voitaisiin toteuttaa.

Viikkojen aikana pääsin tutustumaan myös mob-programmingiin, jota en ollut aiemmin kokeillut. Huomasimme tämän olevan todella toimiva malli ja opin siitä itse henkilökohtaisesti paljon, sillä sessioiden aiheiksi valikoituivat kaikista haastavimmat tehtävät, joita olisi ollut hankala koodata yksin. Tätä tulen ehdottomasti käyttämään myös tulevaisuuden projekteissa.

Päiväkirjamuotoinen opinnäytetyö oli loistava tapa konkretisoida omaa tekemistä ja kehittymistä. Tein seurantajaksoni työpäivien aikana jo merkintöjä, mikä helpotti huomattavasti työtehtävien, esiin nousevien haasteiden sekä omien oppimiskokemusten kuvailua työpäivän jälkeen. Viikkoanalyysia tehdessäni keräsin yhteen viikon kannalta keskeisimmät työtehtävät ja vertailin niitä uutisartikkeleista, blogeista ja kirjallisuudesta hakemaani tietoon.

Näin pystyin analysoimaan tehtäviäni eri näkökulmista ja löytämään uusiakin tapoja kehittää osaamistani. Tämä näkyi myös seurantaviikkojen edetessä, kun otin käyttöön viikkoanalyysissäni esiin nostettuja uusia tapoja ja kokeilin implementoida niitä seuraavan viikon työskentelyyni. Ilman tätä opinnäytetyötä en olisi tehnyt näin, mutta opittuani työskentelemään tällä tavalla, uskon, että tulen jatkamaan samalla linjalla. En ehkä välttämättä yhtä tarkasti päiväkirjamaisesti kirjaamalla, mutta kuitenkin kirjaamalla viikon keskeisimmät teemat ja yrittämällä löytää niihin uusia näkökulmia alan kirjallisuudesta. Näihin asioihin on sitten myös helpompi myöhemmin palata, kun ne ovat kirjattuna muistiin.

Opinnäytetyöskentelyn aikana huomasin, että yritin keskittyä hyvin moneen ohjelmoinnin osa-alueeseen samaan aikaan. Huomasin kuitenkin, että tämä oikeastaan hidastaa kehittymistä, sillä jatkuva fokuksen vaihtaminen asiasta toiseen vaikeuttaa asioiden syvällistä ymmärtämistä. Yritin samaan aikaan opiskella niin Typescriptia, CSS:ää, Javaa kuin valmistautua AWS-sertifikaattiin. Lopulta huomasin, että aikani ei yksinkertaisesti riitä, joten on parempi keskittää fokusta ja edetä asia kerrallaan. Full stack -kehittäjän pitää hallita laaja-alaisesti eri osa-alueilta teknologioita ja taitoja, mutta tärkeämpää on osata tietyt asiat todella hyvin, minkä jälkeen osaamista voi laajentaa vähitellen sitten muihin alueisiin.

Oppimiskokemukseni laajeni myös työajan pienten taukojen hyödyntämisen osalta. Huomasin nimittäin, kuinka lyhyet tauot voi käyttää työpäivän aikana hyödyksi. Esimerkiksi sen odotteluajan, että testit pyörähtävät tai koodimuutokset menevät CI/CD-putkesta läpi, pystyi hyödyntämään ja käyttämään asioiden opiskeluun. Tuona aikana ehdin esimerkiksi lukemaan ajankohtaisen blogin tai artikkelin tai vaikkapa yhden kappaleen Thomaksen ja Huntin kirjasta The Pragmatic Programmer (2019). Kaikesta oppi jotain uutta ja jota pystyi myös mahdollisesti käyttämään heti apuna työtehtävissä.

Oman työn analysoiminen ei ole kehittänyt minua pelkästään ohjelmistokehittäjänä. Päädyin tälle alalle alun perin sen takia, että teknologia on kiinnostanut minua aina todella paljon. Työn analysoinnin kautta olen kuitenkin löytänyt paljon muitakin näkökulmia työhöni kuin pelkän teknisen näkökulman. Kirjallisuudesta ja ajankohtaisista artikkeleista saadun tiedon avulla on ohjelmistokehittäjältä vaadittavien taitojen lista laajentunut koskemaan paljon muutakin kuin vain tuon teknisen osaamisen. Ohjelmistokehittäjän työssä vaaditaan hyvin paljon myös aivan muunlaisia taitoja. Ohjelmistokehittäjän työssä tarvitaan esimerkiksi tiimityöskentelytaitoja, vuorovaikutustaitoja yleensä, kykyä ymmärtää erilaisia asiakkaiden tarpeita ja taitoa suunnitella isojakin kokonaisuuksia. Aikaisemmin ajattelin, että kehittäjä toteuttaa pelkästään mielenkiintoisia uusia teknisiä ratkaisuja eikä toimenkuvaan liity juuri muuta.

Aion huolehtia oman ammattitaitoni kehittämisestä myös jatkossa. Tämän työn aikana olen huomannut, kuinka paljon muuta konsultin ja ohjelmistokehittäjän työhön liittyy ja mitä kaikkia taitoja tarvitaan, jotta pystyy ylläpitämään omaa ammattitaitoa ja kehittymistä. Kehittäjä ei tuota arvoa yritykselle välttämättä pelkästään laskuttamalla asiakasprojektin suorituksia, vaan on tärkeä henkilö myös myynnin ja rekrytoinnin tukena. Voin auttaa esimerkiksi hyödyntämällä omaa ammattitaitoani arvioimalla, mitä teknologisia ratkaisuja projektissa voitaisiin käyttää tai kuinka kauan se mahdollisesti vaatii aikaa. Rekrytointia pystyn auttamaan arvioimalla esimerkiksi rekrytoitavan henkilön osaamista ja kokemusta. Nämä ovat vain osa niistä näkökulmista, jotka tämän työn aikana ovat avautuneet minulle ja joissa haluan ehdottomasti jatkaa kehittymistä myös tulevaisuudessa.

Lähteet

Agile Alliance. Backlog Refinement. Luettavissa: https://www.agilealliance.org/glossary/backlog-grooming Luettu: 27.4.2020

Agile Alliance. Mob-programming. Luettavissa: https://www.agilealliance.org/glossary/mob-programming Luettu: 4.4.2020

Agile Alliance. TDD. Luettavissa: https://www.agilealliance.org/glossary/tdd/ Luettu: 27.4.2020

Atlassian. Confluence. Luettavissa: https://www.atlassian.com/fi/software/confluence Luettu: 7.4.2020

Atlassian. DevOps: Breaking the Development-Operations barrier. Luettavissa: https://www.atlassian.com/devops#history-of-devops Luettu: 27.4.2020

Atlassian. Jira Software. Luettavissa: https://www.atlassian.com/software/jira/guides/getting-started/overview Luettu: 27.4.2020

Avoin rajapinta. 2014. Avoimen rajapinnan määritelmä. Luettavissa: http://avoinrajapinta.fi/ Luettu: 27.4.2020

AWS. Luettavissa: https://aws.amazon.com/what-is-aws/ Luettu: 7.4.2020

Bauer, T. 2018. Face-to-face communication is 34x more effective than email. Luettavissa: https://medium.com/@tedbauer2003/face-to-face-communication-is-34x-more-effective-than-email-5bbdd3da2240 Luettu: 19.4.2020

CBT Nuggets. 2019. How to Study for Your AWS Certified Solutions Architect – Associate Exam in 10 Weeks. Luettavissa: https://www.cbtnuggets.com/blog/career/career-progression/how-to-sit-for-your-aws-certified-solutions-architect-associate-exam-in-10-weeks Luettu: 27.3.2020

Cisco. What Is a VPN? - Virtual Private Network. Luettavissa: https://www.cisco.com/c/en/us/products/security/vpn-endpoint-security-clients/what-is-vpn.html Luettu: 27.4.2020

Dabhadkar, K. 2019. Ace Your First AWS Certification in 10 Days. Luettavissa: https://towardsdatascience.com/ace-your-first-aws-certification-in-10-days-f6b2b7cbea34 Luettu: 27.3.2020

DevIQ. Programming is communicating. Luettavissa: https://deviq.com/communication/ Luettu: 19.4.2020

Emery, D. 2009. Writing Maintainable Automated Acceptance Tests. Luettavissa: http://dhemery.com/pdf/writing_maintainable_automated_acceptance_tests.pdf Luettu: 10.3.2020

Fowler, M. 2019. Technical Debt. Luettavissa: https://martinfowler.com/bliki/TechnicalDebt.html Luettu: 27.4.2020

Frontend Masters. What Is a Front-End Developer? Luettavissa: https://frontendmasters.com/books/front-end-handbook/2018/what-is-a-FD.html Luettu: 27.4.2020

Github. Luettavissa: https://help.github.com/en/github/collaborating-with-issues-and-pull-requests/about-pull-requests Luettu: 27.4.2020

Google. 2020. How to do a code review. Luettavissa: https://github.com/google/eng-practices/blob/master/review/reviewer/index.md Luettu: 24.3.2020

Guru99. What is Backend Developer? Skills to become a Web Developer. Luettavissa: https://www.guru99.com/what-is-backend-developer.html Luettu: 7.4.2020

Habib, S. 2018. Learn CSS Grid and Flexbox fundamentals before 2019 ends! Luettavissa: https://medium.com/@shuvohabib/learn-css-grid-and-flexbox-fundamentals-before-2019-hits-the-ground-765010c6e21f Luettu: 26.2.2020

Hanslik, M. 2019. 8 Best Practices for Future-Proofing Your Typescript Code. Luettavissa: https://medium.com/better-programming/8-best-practices-for-future-proofing-your-typescript-code-2600fb7d8063 Luettu: 27.2.2020

Jenkins. Luettavissa: https://www.jenkins.io/ Luettu: 27.4.2020

jQuery. Luettavissa: https://jquery.com/ Luettu: 27.4.2020

Li, C. 2018. Why TDD is Bad (and How to Improve Your Process). Luettavissa: https://medium.com/@charleeli/why-tdd-is-bad-and-how-to-improve-your-process-d4b867274255 Luettu: 4.4.2020

Lopian, E. 2018. Defining Legacy Code. Luettavissa: https://dzone.com/articles/defining-legacy-code Luettu: 27.4.2020

Lotanna, N. 2020. The future of CSS features in 2020. Luettavissa: https://blog.logrocket.com/the-future-of-css-features-in-2020/ Luettu: 26.2.2020

Lovinus, A. 2019. What Hiring Managers look for in a Full Stack Developer. Luettavissa: https://www.cybercoders.com/insights/what-hiring-managers-look-for-in-a-full-stack-developer/ Luettu: 15.4.2020

Martin, R.C. 2009. Clean Code. A Handbook of Agile Software Craftmanship.

Martin R.C. 2011. The Clean Coder. A Code of Conduct for Professional Programmers.

Microsoft. 2010. Stubs and Mocks. Luettavissa: https://docs.microsoft.com/en-us/previous-versions/msp-n-p/ff798400(v=pandp.10) Luettu: 27.4.2020

Morris, S. 2019. Full Stack, Front End, Back End—What Does It All Mean? Luettavissa: https://skillcrush.com/blog/front-end-back-end-full-stack/#full Luettu: 15.4.2020

Mozilla. 2019. What is CSS?. Luettavissa: https://developer.mozilla.org/en-US/docs/Learn/CSS/First_steps/What_is_CSS Luettu: 27.4.2020

NASDAX OMX NORDIC. 2012. Siili Solutions. Luettavissa: http://www.nasdaqomxnordic.com/news/listings/firstnorth/2012/siilisolutions. Luettu: 9.2.2020

Neagoie, A. 2017. The Developer's Edge: How To Become A Senior Developer. Luettavissa: https://medium.com/zerotomastery/developers-edge-how-to-become-a-senior-developer-f1ec1738cf45 Luettu: 26.4.2020

Oracle. Java™ Programming Language. Luettavissa: https://docs.oracle.com/javase/8/docs/technotes/guides/language/index.html Luettu: 27.4.2020

Palantir. 2020. TSLint core rules. Luettavissa: https://palantir.github.io/tslint/rules/ Luettu: 26.3.2020

Peck, N. 2019. Being a Programmer is Not About Writing Code. Luettavissa: https://medium.com/the-heart-of-programming/being-a-programmer-is-not-about-writing-code-84aa54cb1c61 Luettu: 24.3.2020

Peham, T. 2017. 7 Tips On How to Become a Full Stack Developer! Luettavissa: https://usersnap.com/blog/how-to-become-a-full-stack-developer/ Luettu: 15.4.2020

ProductPlan. Agile Manifesto. Luettavissa: https://www.productplan.com/glossary/agile-manifesto/ Luettu: 7.4.2020

Postman. Luettavissa: https://www.postman.com/ Luettu: 27.4.2020

React. Luettavissa: https://reactjs.org/ Luettu: 27.4.2020

Redux. Luettavissa: https://redux.js.org/introduction/getting-started Luettu: 27.4.2020

Robotframework. 2019. How to write good test cases using Robot Framework. Luettavissa:https://github.com/robotframework/HowToWriteGoodTestCases/blob/master/HowToWriteGoodTestCases.rst Luettu: 10.3.2020

Robot Framework. Luettavissa: https://robotframework.org/ Luettu: 27.4.2020

Ryan. 2016. Does Test Driven Development (TDD) really matter? Luettavissa: http://www.digigene.com/general-software-development/tdd-skills-importance/ Luettu: 4.4.2020

Sanders, J. 2018. Why it's finally time for developers to address the chaos of Node.js and NPM. Luettavissa: https://www.techrepublic.com/article/why-its-finally-time-for-developers-to-address-the-chaos-of-node-js-and-npm/ Luettu: 26.4.2020

Scrum. What is a Daily Scrum? Luettavissa: https://www.scrum.org/resources/what-is-a-daily-scrum Luettu: 27.4.2020

Scrum. What is Scrum? Luettavissa: https://www.scrum.org/resources/what-is-scrum Luettu: 27.4.2020

Sharma, A. 2018. Why You Should Use Typescript for Developing Web Applications. Luettavissa: https://dzone.com/articles/what-is-typescript-and-why-use-it. Luettu: 15.2.2020

Siili Solutions Oyj. 2018. Siili Solutions Oyj 2018 Vuosikertomus. Luettavissa: https://campaign.siili.com/hubfs/Siili%20Solutions%20Oyj%202018%20Vuosikertomus.pdf Luettu: 9.2.2020

Techopedia. 2017. Luettavissa: https://www.techopedia.com/definition/16373/debugging Luettu: 27.4.2020

Thomas, D & Hunt, A. 2019. The Pragmatic Programmer: your journey to mastery, 20th Anniversary Edition. O'Reilly Media, Inc.

Typescript Handbook. Luettavissa: https://www.typescriptlang.org/docs/handbook/functions.html Luettu: 10.3.2020

Typescript. Luettavissa: https://www.typescriptlang.org/ Luettu: 27.4.2020

Vashishtha, S. 2015. Traditional Testing will be Dead Soon! Luettavissa: http://www.agilebuddha.com/agile/traditional-testing-will-be-dead-soon/ Luettu: 12.3.2020

Westberg, H. 2018. What is a Full Stack Developer? Luettavissa: https://codeup.com/what-is-a-full-stack-developer/ Luettu: 27.4.2020

Williams, C. 2016. How one developer just broke Node, Babel and thousands of projects in 11 lines of JavaScript. Luettavissa: https://www.theregister.co.uk/2016/03/23/npm_left_pad_chaos/ Luettu: 26.4.2020

Yuruk, H. 2019. Fundamentals of a Good Developer Mindset. Luettavissa: https://huseyinpolatyuruk.com/fundamentals-of-a-good-developer-mindset/ Luettu: 8.3.2020

Liitteet

Liite 1. Ammattisanasto ja lyhenteet

Agile Manifesto: lyhyt dokumentti, joka sisältää ketterän ohjelmistokehityksen 4 arvoa ja 12 periaatetta (ProductPlan).

API: Rajapinta, jonka avulla muut sovellukset voivat tehdä pyyntöjä ja vaihtaa tietoja (Avoin rajapinta 2014).

AWS: Amazon Web Services pilvipalvelualusta (AWS).

Backend development: Sovelluksen palvelinpuoleen keskittyvä kehitys (Guru99).

Confluence: Yritysten ja organisaatioiden käyttämä yhteistyö- ja dokumentaatiopalvelu (Atlassian).

CSS: Cascading Style Sheets on web-sivustojen tyylittelyyn käytetty merkintäkieli (Mozilla).

Daily: maksimissaan 15-minuuttia kestävä palaveri, joka pidetään yleensä jokaisen työpäivän alussa. Dailyssa käydään läpi, mitä jokainen tekee seuraavan 24 tunnin aikana ja mitä on saanut valmiiksi edellisen Dailyn jälkeen (Scrum).

Debuggaus: Debuggaus on rutiininomainen prosessi, jonka tarkoituksena on poistaa virheet, viat tai epätavalliset toiminnallisuudet ohjelmasta (Techopedia).

Devops: Sarja käytäntöjä, joiden avulla automatisoidaan ja integroidaan prosesseja ohjelmistokehityksen tiimien välillä, jotta voidaan rakentaa, testata ja julkaista sovelluksia nopeammin ja luotettavammin (Atlassian).

Frontend development: myös tunnettu nimellä web development, keskittyy sovellusten ja verkkosivujen käyttäjäpuolen kehitykseen kuten käyttöliittymiin ja ulkoasuun (Frontend Masters).

Full stack development: Full stack kehitys tarkoittaa sovelluksen kaikkien osien kehittämistä, kuten frontendin, backendin sekä tietokannan (Westberg 2018).

Grooming: Scrum-seremonia, minkä tarkoituksena on käydä backlogilla olevia töitä läpi ja priorisoida ne (Agile Alliance).

Java: Java ohjelmointikieli on yleiskäyttöinen, voimakkaasti tyypitetty, luokkaperusteinen olio kieli (Oracle).

Jenkins: Jenkins on ilmainen avoimeen lähdekoodiin perustuva automaatiopalvelin, mikä automatisoi ohjelmistokehityksen osia, mitkä liittyvät rakentamiseen, testaamiseen ja käyttöönottoon (Jenkins).

Jira: Jira on tiimien työnhallintakalu, mihin voidaan kirjata tehtäviä ja tikettejä, joiden avulla voidaan seurata yksittäisten töiden etenemistä (Atlassian).

jQuery: Monipuolinen Javascript-kirjasto, mikä laajentaa Javascriptin toiminnallisuutta (jQuery).

Legacy: Legacy on lähdekoodia, joka liittyy järjestelmiin, jota ei enää tueta tai kehitetä aktiivisesti (Lopian 2018).

Mock, mockaus: Olio-ohjelmoinnissa käytettäviä simuloituja objekteja, jotka matkivat todellisten toteutusten käyttäytymistä hallitulla tavalla, useimmiten osana ohjelmistotestausta (Microsoft).

Postman: Rajapintakehityksessä käytettävä työkalu, minkä avulla rajapintaa voidaan kutsua (Postman).

Pull request: Pull request on versionhallintaan tehtävä pyyntö, jonka avulla voit kertoa muille koodimuutoksista ja tarkastella niitä (Github).

React: Javascript-kirjasto, minkä avulla voidaan rakentaa käyttöliittymiä (React).

Redux: Redux on avoimen lähdekoodin Javascript-kirjasto, minkä avulla voidaan hallita sovelluksen tilaa (Redux).

Robot Framework: Yleinen avoimen lähdekoodin automaatiokehys, minkä avulla voidaan rakentaa testiautomaatiota tai robottiprosesseja (Robot Framework).

Scrum: Ketterä prosessi, jonka avulla ihmiset voivat käsitellä monimutkaisia ongelmia samalla tuottaen mahdollisimman korkean arvon tuotteita (Scrum).
Spring: Java-alustan sovelluskehys, joka tarjoaa kattavan ohjelmointi- ja konfigurointimallin nykyaikaisille Java-pohjaisille yrityssovelluksille.

Tekninen velka: Ohjelmistokehityksen käsite, joka heijastaa lisämuutosten implisiittisiä kustannuksia, jotka aiheutuvat siitä, että teknisiä asioita ei ole optimoitu (Fowler 2019).

TDD: Test-Driven-Development, eli testilähtöinen kehitys mikä viittaa ohjelmointityyliin, jossa kirjoitetaan ensin yksikkötestit, minkä jälkeen tehdään toteutus (Agile Alliance).

Typescript: Typescript on Microsoftin kehittämä ja ylläpitämä avoimen lähdekoodin ohjelmointikieli. Se on Javascriptin yläjoukko, joka lisää kielelle staattisen tyypittämisen (Typescript).

VPN: Virtuaalinen yksityinen verkko, jonka avulla käyttäjä voi luoda salatun yhteyden laajentamalla julkisen verkon yksityiseksi (Cisco).