OULUN AMMATTIKORKEAKOULU

Arto Halonen

PÄIVÄKIRJAMUOTOINEN OPINNÄYTETYÖ: ENSIASKELEET FULL STACK -OHJELMISTOKEHITTÄJÄNÄ

PÄIVÄKIRJAMUOTOINEN OPINNÄYTETYÖ: ENSIASKELEET FULL STACK -OHJELMISTOKEHITTÄJÄNÄ

Arto Halonen
Opinnäytetyö
Kevät 2021
Tietojenkäsittelyn tutkinto-ohjelma
Oulun ammattikorkeakoulu

TIIVISTELMÄ

Oulun ammattikorkeakoulu
Tietojenkäsittelyn tutkinto-ohjelma

Tekijä(t): Arto Halonen
Opinnäytetyön nimi: Päiväkirjamuotoinen opinnäytetyö: Ensiaskeleet Full Stack -ohjelmistokehittäjänä
Työn ohjaaja(t): Ritva Virkkala
Työn valmistumislukukausi ja -vuosi: Kesä 2021 Sivumäärä: 61

Tämä päiväkirjamuotoinen opinnäytetyö kuvaa kymmenen viikon ajan uraansa aloittelevan Full Stack -ohjelmistokehittäjän arkea ja yleisimpiä työtehtäviä. Päiväkirjaa täytetään päivittäin ja viikon lopuksi viikon pääaiheesta kirjoitetaan tarkempi viikkoanalyysi. Työtehtävät tänä aikana liittyvät pääasiassa olemassa olevan sovelluksen bugikorjauksiin, muutostöihin ja refaktorointiin.

Opinnäytetyön tavoitteena on auttaa kirjoittajaansa kehittymään parhaalla mahdollisella tavalla roolissaan, sekä antaa esimakua tuleville uraansa aloitteleville ohjelmistokehittäjille siitä, millaista työ todellisuudessa on. Viikkoanalyyseissa pyritään pureutumaan viikon haastavimpiin tehtäviin tutkien. Tämä tarjoaa oppimispaikkoja niin kirjoittajalle kuin lukijallekin.

Ohjelmointikielien viralliset ajantasaiset dokumentaatiot, ohjelmointiaiheiset kirjat sekä laadukkaat blogikirjoitukset toimivat pääasiallisina tietoperustoina tässä työssä. Työtä tehtäessä tullaan myös käyttämään ohjelmistoaiheisia keskustelusivustoja apuna, ne tarjoavat erinomaista tukea ongelmatilanteissa.

Pohdintaosiossa tarkastellaan kirjoittajan kehitystä kymmenen viikon aikana sekä mietitään keinoja tulevaan itsensä kehittämiseen. Lopputuloksena saatiin aikaiseksi varsin kattava paketti teoriapohjaisia viikkoanalyyseja ja päiväkirjan muodossa läpileikkaus normaaliin Junior-kehittäjän arkeen.

Asiasanat: Ohjelmointi, Full-Stack, Angular, NestJS, Git, PostgreSQL

ABSTRACT

Oulu University of Applied Sciences
Degree Programme in Business Information Systems

Author(s): Arto Halonen
Title of thesis: A diary-based thesis: First steps as a Full-Stack -developer
Supervisor(s): Ritva Virkkala
Term and year when the thesis was submitted: Summer 2021.
Number of pages: 61

This thesis follows the normal workdays of a Junior software developer. It is written like a diary for 10 weeks, and in the end of every week there will be a deeper analysis of the main subject of the week. Normal daily tasks involve refactoring, debugging and modification of the software that the team develops.

The purpose of this thesis was to show how it is like starting your career in software development, and also provide the writer the best tools for self-development. Weekly analyses will also offer learning points for both, the reader and the writer.

The result was a comprehensive package of theoretical weekly analyses and a realistic picture what a Junior developer faces during the beginning of their career.

Keywords: Full Stack -development, Angular, NestJS, Git, PostgreSQL

SISÄLLYS

1 JOHDANTO

Opinnäytetyöni tavoitteena on kuvata tavanomaisia työtehtäviä ohjelmistoalalla, sekä tukea omaa kehittymistäni Full Stack -ohjelmistokehittäjänä. Päiväkirja on kirjoitettu aikavälillä 22.3.2021 – 30.5.2021, eli se on kestoltaan 10 viikkoa. Päiväkohtaisesti kerron omista työtehtävistäni ja jokaisen viikon päätteeksi valitsen yhden viikon aiheista viikkoanalyysini kohteeksi. Analyysissa pyrin pureutumaan kyseessä olevaan asiaan tarkemmin ja tutkivalla otteella.

Pyrin selittämään asiat mahdollisimman helppolukuisesti, mutta todella laajasta ammattikielestä johtuen niin sanotulta IT-jargonilta ei tulla täysin välttymään. Käsitteistössä avaan tärkeimpiä lyhenteitä, sekä tuota edellä mainittua ammattikieltä.

Yritys, jossa työskentelen, on perustettu vuonna 1995 ja on vuosien saatossa yritysostoin, sekä rekrytoinnein kasvanut yli 600 henkilöä työllistäväksi konserniksi. Yritys kehittää sekä omia ohjelmistotuotteitaan että asiakkaan tilaamia räätälöityjä ratkaisuja. Yrityksellä on asiakkaita yli kolmessakymmenessä eri maassa ja sen pääasiallinen toimiala on teollisuuden digitalisointi.

2 KÄSITTEISTÖ

Angular = Googlen kehittämä TypeScript-pohjainen ohjelmistokehys.

API = Application Programming Interface eli ohjelmointirajapinta. Sen avulla sovellukset voivat kommunikoida toisten sovellusten kanssa.

Back End = Sovelluksen osa, joka on vastuussa muun muassa tietokannan kanssa kommunikoimisesta ja tiedon hakemisesta ja muokkaamisesta.

Bugi = Sovelluksen koodissa oleva virhe, joka aiheuttaa epätoivottavaa käyttäytymistä sovellusta käytettäessä.

CI/CD = Continuos integration/Continuous deployment. Nykyaikainen ohjelmistotuotannon malli, jossa pyritään automatisoimaan kaikki mahdollinen.

Debuggaus = Virheenjäljitystä, jossa erinäisillä tekniikoilla yritetään löytää bugi koodista.

Decorator = Sillä voidaan lisätä yksittäiseen olioon toiminnallisuuksia, joita samasta luokasta tehdyillä muilla olioilla ei ole, häiritsemättä noiden muiden olioiden toimintaa.

Docker = Teknologia, jossa ohjelmistot kehitetään, julkaistaan ja ajetaan virtuaalisten konttien sisällä.

DTO = Data Transfer Object on olio, joka kuljettaa dataa prosessien välillä, vähentäen funktiokutsujen määrää.

Enum = Luokka, joka sisältää ryhmän vakioita, eli muuttumattomia muuttujia.

Front End = Sovelluksen osa, joka näytetään käyttäjälle selaimessa.

Full Stack -ohjelmoija = Ohjelmoija, joka osaa ohjelmoida sovelluksen molempia puolia.

GitLab = Verkkopohjainen versionhallintajärjestelmä.

GraphQL = Facebookin luoma datan kysely- ja manipulointikieli.

HTTP = Protokolla asiakkaan ja palvelimen väliselle liikenteelle.

Integraatio = Kahden erillisen järjestelmän toimintojen yhdistäminen rajapinnan välityksellä.

Interface = Rajapinta, joka määrittää sovelluksen osien rakenteen.

JavaScript = Ohjelmointikieli, jonka avulla saadaan dynaamiset ominaisuudet internet-sivuilla toimimaan. Maailman yleisin ohjelmointikieli.

JWT = JSON Web Token. Autentikointiin käytettävä menetelmä, joka perustuu JSON-formaattiin.

Kanban = Tapa hallita ja parantaa tuotantoa. Kuuluu niin sanottuihin ketteriin menetelmiin.

Luokka = Ohje/pohjapiirros, jonka perusteella Olio luodaan.

Master-haara = Versionhallinnan päähaara, haara, josta julkaisut tehdään.

NestJS = NodeJS-pohjainen ohjelmistokehys. Tuo Nodeen muun muassa tuen TypeScriptille.

NodeJS = JavaScript-koodia palvelimella suorittava ympäristö.

Olio = Luokan ilmentymä, olio-ohjelmoinnin perusyksikkö, joka useasti vastaa jotain reaalimaailman asiaa.

Olio-ohjelmointi = Ohjelmointityyli, joka perustuu olioiden väliseen yhteistoimintaan.

ORM = Object-relational mapping, Ohjelmointitekniikka, jonka avulla voidaan kirjoittaa tietokantakyselyitä halutulla ohjelmointikielellä.

Payload = HTTP-pyynnön mukana välitettävä tieto.

Postman = Ohjelmisto rajapintojen testaamiseen ja dokumentointiin.

Repositorio = Versionhallinnan tietosäiliö, jossa sovelluksen koodit ovat tallennettuna.

Scrum Master = Ketterän kehitystiimin johtaja, jonka tehtävä on mahdollistaa kehittäjien keskittymisen pelkästään omaan työhönsä.

Slack = Pikaviestintäsovellus, joka on eritoten suunniteltu organisaatioiden sisäiseen viestintään.

SQL = Kyselykieli, jolla relaatiotietokantaa voidaan lukea ja muokata.

TypeScript = Microsoftin luoma tyypitetty versio JavaScriptistä.

Virtuaalikone = Tietokoneella ohjelmallisesti luotu tietokonesimulaatio, jonka sisällä voidaan ajaa ohjelmia kuten oikealla tietokoneella.

3 NYKYTILANNE

3.1 Nykyisen työn analyysi

Erilaisia työtehtäviä päivän aikana:

- Toimintatapojen suunnittelu
- Versionhallintaan liittyvät työt
- Palaverit
- Ohjelmointi
- Testaus- sekä tuotantoympäristön päivitykset
- Opiskelu

Tyypillisesti työpäiväni ovat rakenteellisesti hyvinkin samanlaisia. Joka päivä samaan aikaan pidämme tiimin kanssa viidentoista minuutin mittaisen palaverin, jossa kerromme mitä kukin meistä on tehnyt ja aikoo tehdä seuraavaksi. Aamut alkavat sähköpostien lukemisella ja sen jälkeen käyn yleensä versiohallinnassa katselmoimassa työparini viimeisimmät koodit ja joko hyväksyn hänen yhdistämispyyntönsä, tai jätän korjausehdotuksen kommentin muodossa.

Tuote, jota kehitämme, on jo asiakaskäytössä, joten ohjelmointiin liittyvät työtehtävämme ovat pääasiassa havaittujen vikojen korjaamista, refaktorointia ja uusien ominaisuuksien lisäämistä. Olemme viikoittain yhteydessä asiakkaan edustajan kanssa. Hänen kertomistaan muutoksista tehdään niin kutsuttuja tikettejä Kanban-taululle, josta me kehittäjät ne sitten poimimme työn alle ja työn valmistuessa siirrämme katselmointisarakkeeseen.

Työssä tarvitaan monenlaista osaamista. Ohjelmointitaito on luonnollisesti tarpeellinen, mutta sitäkin tärkeämpää on kyky oppia jatkuvasti uutta. Ohjelmointikielet, kehitysympäristöt, kirjastot ja kehykset kehittyvät jatkuvasti ja on pysyttävä ajan tasalla. Ensimmäisten työviikkojen informaatiotulva voi yllättää.

Ennen töiden alkua omasin hyvät perustaidot muutamista ohjelmointikielistä, kehyksistä ja kirjastoista. Olin tehnyt kouluprojektien lisäksi omia projekteja, sekä oman yritykseni kautta muutamia maksullisia projekteja, joten omasin jo jonkinlaisen käsityksen projektityöskentelystä.

Olen oppinut työssäni jo näin lyhyessä ajassa kosolti uusia asioita. Muun muassa versionhallinnasta, virtuaalikoneista, ohjelmistokonttiteknologioista ja CI/CD-putkista. Nämä ovatkin asioita, jotka ovat nykypäivän ohjelmistokehittäjälle arkipäivää, mutta niihin on vaikea saada kosketuspintaa opinnoissa tai pienissä projekteissa

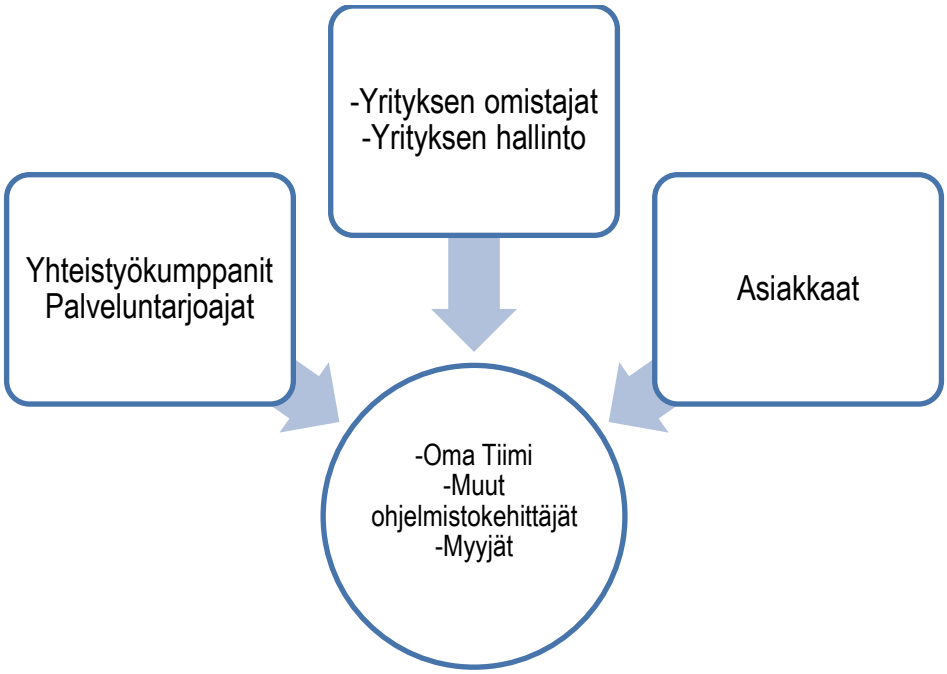
Koen olevani jo taitava suoriutuja työssäni. Teen ison osan minulle annetuista tehtävistä täysin itsenäisesti tai hyvin pienin neuvoin. Olen silti pitänyt avunpyynnin kynnyksen matalana ja uskon tämän auttavan kehittymiseeni edelleen. Tiimini jäsenet ovat todellisia ammattilaisia ja he osaavat selittää vaikeitakin konsepteja auki helposti ymmärrettävästi.

Jatkossa aion panostaa eniten käytössämme olevien teknologioiden, sekä itse projektin arkkitehtuurin opetteluun. Koodipohja on suurelta osin aiempien työntekijöiden aikaansaannosta ja koska ohjelmointi on eräänlaista käsityötä, vaatii paljon opettelua ja koodin läpikäyntiä, jotta täysin ymmärtää toisen ihmisen käsialaa. Tässä vaiheessa minulla kuluu vielä paljon aikaa löytää muokattava kohta koodista ja pidän tätä yhtenä suurimmista kehityskohteistani.

Ohjelmoijana kehittyminen on jatkuva oppimismatka, joten aion myös jatkaa ihan perusasioiden opettelua. Kukaan ei liene liian hyvä esimerkiksi algoritmeissa tai datarakenteissa. Eri ohjelmointikielien hyvät toimintatavat kiinnostavat myös. Haluan myös syventää ymmärrystäni ketterien kehitysmenetelmien suhteen ja alustavasti olenkin jo ilmoittautunut Scrum Master -koulutukseen.

3.2 Työpaikkani sidosryhmät

Oma tiimini koostuu projektipäällikön lisäksi kuudesta ohjelmistokehittäjästä. Muita sisäisiin sidosryhmiin kuuluvia ovat toisten tiimien ohjelmistokehittäjät, myyjät ja yrityksen hallinto sekä omistajat. Ulkoisiin sidosryhmiin kuuluvat asiakkaat ja palveluntarjoajat.



KUVIO 1. *Työpaikan sidosryhmät-kuvio.*

3.3 Vuorovaikutustaidot työpaikalla

Vallitsevasta koronatilanteesta johtuen suuri osa vuorovaikutuksesta tapahtuu etäyhteyksin. Yrityksemme työntekijöistä arviolta noin 90 prosenttia tekee työtään täysin etänä tällä hetkellä. Päivittäiset palaverit hoidetaan tavallisesti Google Meet -palvelun kautta. Asiakkaan kanssa viestintä tapahtuu pääasiassa puhelimitse sekä sähköpostilla. Yrityksellä on myös erittäin aktiivinen Slack-yhteisö kymmenine eri kanavineen. Työparini kanssa viestimme pääasiassa Slack -puheluin sekä viestein. Jätämme toisillemme myös viestejä versionhallinnan kautta.

Yrityksemme järjestää myös paljon tapahtumia sekä jokaisella tiimillä on oikeus järjestää tapahtumia keskenäänkin. Tapahtumien ideat tulee hyväksyttää henkilöstöhallinnossa, jonka jälkeen työnantaja huolehtii tapahtumasta aiheutuvista kuluista. Yritys kannustaa työntekijöitä liikkumaan, ja useasti tapahtumat ovatkin urheiluaiheisia.

Lähiesimieheni työskentelee myös pääasiassa etänä, mutta järjestämme hänen kanssaan tasaisin väliajoin tapaamisia, joissa käydään läpi kehittymistäni ja omia toiveitani urani suuntaan liittyen. Projektiini liittyvät asiat käyn läpi yleensä projektipäällikön kanssa.

4 PÄIVÄKIRJA

4.1 VIIKKO 12 – Refaktoroinnin tärkeys

Maanantai 22.3.2021

Edellisen viikon viimeisimpänä tehtävänä valmistelimme kaiken valmiiksi maanantaista tuotantoympäristön päivitystä varten. Pyrimme julkaisemaan uuden päivityksen kerran viikossa. Näin pidetään päivitykset mahdollisimman pieninä ja vähän rikkovina. Päivän tavoitteena olikin saada päivitys ajettua onnistuneesti ja testata uusien toimintojen toimivuus käytännössä.

Aloitimme myös muutama viikko sitten refaktoroimaan sovelluksen Front End -puolen koodipohjaa yhtenäiseksi, sekä hyvien käytänteiden mukaiseksi. Sovellus, jota on kehitetty useamman vuoden ajan, on osiltaan aina aikansa vanki. Käytänteiden ja ohjelmointikielien päivittyessä osa koodista jää vanhanaikaiseksi. On hyvä pitää sovellus ajantasaisena ja yhtenäisenä, jotta tulevaisuudessa muillakin ohjelmoijilla on helppo ymmärtää sitä. Päivän aikana saimme viimeisetkin tiedostot refaktoroitua.

Urani lyhyestä kestosta johtuen, en ole vielä ehtinyt olla päivityksissä mukana kovinkaan useasti. Koska viimeisimpien viikkojen ajan suurin osa päivitykseen liittyvistä töistä on asetettu minun kontolleni, huomaan osaamiseni karttuneen kuin varkain rutiinin myötä. Päivän muihin oivalluksiin kuuluu ehdottomasti refaktoroinnin yhteydessä tekemieni niin kutsuttujen barrel-tiedostojen kätevyys. Import-lausekkeiden niputus tuo niiden ansiosta mukavasti selkeyttä komponentteihin.

Tiistai 23.3.2021

Edellispäivän palaverissa totesimme sovelluksen kahdesta näkymästä puuttuvan sulje/takaisinpainikkeet. Nämä ovat muuten pieniä asioita, mutta käyttökokemuksen kannalta hyvin merkittäviä. Sovittiin, että minä otan ne työn alle ensi tilassa. Päivän tavoitteiksi tuli sen lisäksi vielä sovelluksen prosessikaavioon tutustumista, sekä sovelluksessa olevan SMS-automaation logiikan tutkimista tulevaa muutostyötä varten.

Ohjelmoinnin parissa ei tänä päivänä mennyt juurikaan aikaa. Iso osa päivästä meni tutustuessa sovelluksen prosessikaavioon. Sain jo aika hyvin selville päivän aikana, miten lähden toteuttamaan tulevaa muutostyötä. Tunnen myös, että selvästi ymmärrän sovelluksen rakennetta ja logiikkaa paremmin, kun sain päivän siihen rauhassa tutustua. Päivän ohjelmointitehtävän yksinkertaisuudesta huolimatta opin sitä tehdessä uuden tavan navigoida eri sivulle painiketta painettaessa.

Keskiviikko 24.3.2021

Kehittämämme sovellus on käyttäjän näkökulmasta hyvin pitkälle automatisoitu. Käyttäjän suuntaan tapahtuva viestintä on hoidettu SMS- eli tekstiviestipohjaisesti. Käyttäjän vahvistaessa tilaus sovelluksessa, hänelle näytetään kiitossivu. Asiakkaamme mielestä olisi hyvä, jos tässä vaiheessa asiakkaalle lähetettäisiin vielä vahvistus tilauksesta tekstiviestillä. Tekstiviestin sisältö täytyisi olla myös muokattavissa. Edellispäivän tutkimukseni pohjalta päivän tavoitteeksi muodostui saada asiakkaan haluama muutos toteutettua.

Kyseinen toiminnallisuus vaatii muutoksia niin koodin Front End kuin Back End -puoleen. Tein ensiksi valmiiksi näkyvimmän, eli selaimen näkymään vaikuttavan Front End -koodin muutokset. Tähän riitti uuden Input-kentän tekeminen, mutta koska kyseiset kentät luodaan dynaamisesti tietokannasta saatavan sisällön mukaan ei kyseessä ollut ihan perinteinen lomakkeen syötekentän lisääminen. Back End -muutokset vaativat huomattavasti enemmän miettimistä, puoliksi sattumalta huomasin lopuksi, että voinkin käyttää jo olemassa olevaa palvelua hyväksi. Luotuani tietokantaa muokkaavat toiminnot sain yhdellä koodirivillä varsinaisen tekstiviestin lähettämistoiminnon käyttööni. Päivän suurimpana oivalluksena pidänkin sitä, että asiat monesti ovat paljon yksinkertaisempia, kuin miltä ne aluksi näyttävät. Lukuisien eri testien jälkeen sain ilokseni todeta päässeeni päivän tavoitteeseen ja tein muutoksistani versionhallinnassa yhdistämispyynnöt.

Torstai 25.3.2021

Sovelluksestamme on kaksi versiota, toinen on asiakkaan käytössä oleva tuotantoympäristö ja toinen vain meidän kehittäjien sisäisessä käytössä oleva testiympäristö. Kaikki päivitykset tehdään ensin testiympäristöön ja vasta sen jälkeen tuotantoon. Näin muun muassa bugit tai toimimattomuudet saadaan suurelta osin kiinni ennen niiden tuotantoon joutumista. Olemme sopineet testiympäristön päivityspäiväksi torstain ja kuten aiemmin mainitsin, tuotantoympäristö päivitetään

maanantaisin. Testiympäristön päivitys on melkein täysin automatisoitu projekti. Kun tehdyn muutoksen sisältävä haara yhdistetään Master-haaraan Gitlab-versionhallinnassa, tekee Gitlabin CI/CD-putki Master-haaran uudesta versiosta Docker-imagen ja julkaisee sen Google Cloud Platformin konttirekisteriin. Kun halutaan saada uusi versio päivitettyä testiympäristöön, täytyy versionhallinnasta tehdä Master-haarasta julkaisu. Tämä tapahtuu yhtä painiketta painamalla.

Testiympäristön päivityksen jälkeen testasimme uusia toiminnallisuuksia ja totesimme muutokset toimiviksi. Torstain agendaan kuului myös kaksi tuntia kestävä palaveri asiakkaan kanssa. Palaverin aikana käytiin läpi projektiin liittyviä asioita, muun muassa tulevia muutoksia ja jo Kanban-taululla olevia tikettejä, joista puuttui vielä täsmennyksiä. Iltapäivällä kävin lähinnä läpi työparini tekemiä yhdistämispyyntöjä versionhallinnassa.

Varsinaisesti päivän aikana ei tullut uusia asioita, mutta tätä tekstiä kirjoittaessani jouduin käymään läpi testiympäristön päivittämiseen liittyvää automatiikkaa. Olen kyllä tähänkin asti saanut päivityksen ajettua, mutta syvällistä tietoa siitä, mitä pinnan alla tapahtuu, minulla ei ole aiemmin ollut.

Perjantai 26.3.2021

Perjantaille olen ottanut perinteeksi aloittaa päivän päivittämällä Front End -puolen sovelluksen riippuvuudet. Front End on tehty käyttäen JavaScriptin ohjelmistokehystä nimeltään Angular, joka koostuu useista eri paketeista. Näihin paketteihin tulee viikoittain useita pieniä päivityksiä ja on hyvien käytäntöjen mukaista pitää nämä riippuvuudet ajan tasalla. Koska kehitämme sovellusta Docker-kontissa virtuaalikoneella, täytyy päivityskomennotkin antaa kontin sisällä.

Riippuvuuksien päivityksen jälkeen aloin katselmoimaan työparini versionhallintaan jättämää yhdistämispyyntöä. Hän oli refaktoroinut sovelluksessa käytettävät enumit omiin tiedostoihinsa, ja tämä taas oli johtanut useisiin import muutoksiin muissa tiedostoissa. Enumien käyttö ohjelmoinnissa ei ole minulle täysin selvää, joten tässä olikin hyvä syy samalla alkaa tutustumaan niiden käyttöön ja mitä etuja ne tarjoavat. Päivän päätteeksi voisikin sanoa, että sisäistin konseptin huomattavasti paremmin, joten perjantaikin oli oppimisen suhteen hyvä päivä.

Viikkoanalyysi

Tähänastisen urani aikana olen oppinut päivittäin jotakin uutta. Tämä viikko ei tuonut poikkeusta sääntöön. Viikosta jäi erityisesti mieleen SMS-palvelun muutostyö, jonka tiimoilta tein paljon tutkimustyötä järjestelmään ja sen prosesseihin. Syvä ymmärtäminen sovelluksen toiminnoista on erittäin tärkeää kehittäjän roolissa ja ottaa oman aikansa omaksua toisten ihmisten tekemät järjestelmät. Viimeisten viikkojen aikana tapahtunut koodipohjan refaktorointi tuli näiltä osin päätökseensä tällä viikolla. Nyt sen tuomat hyödyt kantoivat selvästi hedelmää, kun uusia ominaisuuksia tehdessäni huomasin löytäväni oikeat muokattavat komponentit nopeammin. Tästä innoittuneena päätinkin pyhittää päiväkirjan ensimmäisen viikon viikkoanalyysin tutkimalla refaktoroinnin konseptia tarkemmin.

Ymmärtääkseen refaktoroinnin ja sen tarpeeseen johtavat syyt paremmin, on syytä tutustua teknisen velan käsitteeseen. Tekninen velka on luonnoltaan helposti kertyvää ja siihen on useita syitä. Valitessaan hyvien käytänteiden mukaisen hyvin suunnitellun ratkaisun sijasta pikaisen korjauksen ohjelmoija ottaa teknistä velkaa. Kuten oikea velka, myös tekninen velka on hyväksyttävää, jos sen maksaa takaisin ajallaan. Jos ohjelmoija jättää lainan maksut kokonaan suorittamatta, ajan saatossa tekninen velka korkoinen kasvaa niin suureksi, että sen takaisin maksaminen on kallista, kun joudutaan käyttämään aikaa korjaamiseen. (Sharma, Suryanarayana & Samarthayam 2014, 2.)

Refaktorointi on siis tuon velan takaisin maksamista. Aina refaktoroinnin syyksi ei tarvita teknistä velkaa, mutta se on hyvin yleinen syy ja toimii erinomaisesti esimerkkinä. Käytännön tasolla refaktorointi on järjestelmällinen tekniikka olemassa olevan koodipohjan sisäisen rakenteen muuttamiseksi, vaikuttamatta kuitenkaan suoraan sovelluksen toimintaan (Fowler 2000).

Tavanomaisia refaktoroinnin kohteita ovat esimerkiksi funktioiden nimet. On hyvien tapojen mukaista nimetä funktiot niin, että ne kertovat itse tehtävänsä. Jos ajatellaan, että luomme funktion, joka laskee yhteen käyttäjältä saamansa kaksi lukua, on huomattavasti kuvaavampaa nimetä funktio tyyliin: function sumOfInputs(), kuin function ab().

4.2 VIIKKO 13 – Käyttäjienhallinta

Maanantai 29.3.2021

Viikko alkoi totuttuun tapaan tuotantoympäristön päivityksellä. Tämänpäiväisessä päivityksessä tuotantoon meni muun muassa viime viikolla tekemäni tekstiviestipalvelun muutos. Edelleenkin tuntuu hieman ihmeelliseltä, että minun tekemiäni ominaisuuksia käyttää iso joukko ihmisiä. Asiakaskin oli päivitykseen tyytyväinen.

Aamu jatkui pitkällä palaverilla, jossa asiakas kertoi seuraavasta haluamastaan ominaisuudesta. Kyseinen ominaisuus vaatii integraatioita meidän ja kolmen muun palvelun välillä vuorovaikutteisesti, joten kehittämisen täytyy tapahtua yhteistyössä noiden palveluiden ohjelmistokehittäjien kanssa. Tehtävä kuulostaa erittäin mielenkiintoiselta, mutta vielä tässä vaiheessa ilmaan jää liikaa kysymyksiä riippumaan. Varaamme keskiviikolle ajan sisäiseen palaveriin aiheesta.

Valitsin loppupäivän työkseni sovelluksen muuttamisen siltä osin, että myös admin-tasoiset käyttäjät voisivat tästedes luoda, muokata ja poistaa käyttäjiä. Sovelluksessa käyttäjät jaetaan kahteen eri rooliin. Toiset ovat advisoreita, jotka pääsevät vain sovelluksen internal-näkymään käsittelemään ja muokkaamaan tilauksia, toiset taas ovat admineja, jotka pääsevät vain admin-näkymään muokkaamaan sovelluksen asetuksia. Tämän lisäksi käyttäjä voi olla näiden roolien yhdistelmä advisor & admin, jolloin hän pääsee molempiin näkymiin. Näiden käyttäjäroolien lisäksi meidän kehittäjien käytössä on niin kutsuttu superuser-rooli, joka pääsee kaikkialle. Tähän asti käyttäjien hallinta on ollut vain superuser käyttäjällä ja tästä syystä meidän aikaamme on tuhrautunut siihen, vaikka asiakas voisi aivan hyvin tehdä tämän itse. Joudun muuttamaan koodia aika paljon ja muutoksesta johtuen sovelluksen yksikkötestit eivät mene enää läpi. Jos viime viikolla opin, että isolta vaikuttava työ voikin olla helppo ja yksinkertainen niin olkoon tämän päivän oppi, että tämä toimii myös toisin päin.

Tiistai 30.3.2021

Päivä alkoi edellispäivänä kesken jääneen muokkaustyön viimeistelyllä. Testattuani, että tekemäni muutokset toimivat, kirjoitin niistä lyhyen selityksen muutoslokitiedostoon. Loki on hyvä tapa pitää

muistissa mitä uutta seuraavassa päivityksessä on tulossa. Syvennyttyäni käyttäjänhallintaan edellisessä tehtävässä, päätän jatkaa samalla tiellä. Asiakas on toivonut kahta muutakin muutosta, jotka liittyvät käyttäjiin ja roolituksiin.

Aloitan helpommalla tehtävällä. Tällä hetkellä sovelluksen admin-näkymän asetuksissa on kaikille näkyvillä API-asetukset välilehti. Kyseisellä välilehdellä on kaikkien sovelluksen käyttämien rajapintojen asetukset, kuten muun muassa käyttäjätunnukset ja rajapintojen osoitteet. Nämä tiedot ovat luonteeltaan sellaisia, että niihin ei normaalilla admin-käyttäjällä ole tarvetta päästä ja niitä muuttamalla sovelluksen voi nopeasti saada toimimattomaksi. Olemassa olevan toiminnallisuuden vuoksi tämän välilehden piilotus vaati vain yhden ehtodirektiivin lisäämisen HTML-koodiin. Tästedes näihin asetuksiin pääsee käsiksi ainoastaan superuser-käyttäjät.

Toinen asiakkaan toive oli saada internal-näkymän tilauksen muokkaaminen mahdolliseksi vain käyttäjille, joilla on sekä advisor että admin oikeudet. Tein käyttäjien tietoja tarkkailevaan autentikointipalveluun uuden metodin, jolla voidaan tarkistaa, onko käyttäjän roolituksissa molemmat oikeudet. Sitten lisäsin painikkeelle taas ehtodirektiivin, joka näyttää painikkeen ainoastaan, jos tuo metodi palauttaa TRUE-arvon. Tämän lisäksi estin muokkausnäkymään pääsyn osoiteriviä käyttämällä tekemällä urlia tarkkailevan Guardin. Viikon ensimmäiset päivät ovat olleet oikea tutkimusmatka käyttäjänhallinnan maailmaan. Tämän lisäksi tutuksi on tulleet Angularin oikeat käytänteet sen toteutuksessa.

Keskiviikko 31.3.2021

Maanantaina sovittu sisäinen palaveri aloitettiin suoraan päivittäisen palaverimme jälkeen. Saimme huomattavasti selkeytettyä tulevaa tehtävää asiakkaan piirtämän prosessikuvauksen ansiosta. Pystyimme päättelemään tarvittavat rajapinnat, joita meidän sovelluksessamme tulee olla ja minkälaisia pyyntöjä voimme tehdä toisten palveluiden rajapintoihin. Asiakas oli viestinyt, että tämä työ siirtynee ainakin kuukaudella yhden palvelutoimittajan kiireistä johtuen, mutta pystyimme jo aloittamaan valmisteluja.

Back end -koodipohjaa tarkemmin silmäiltyämme tulimme siihen tulokseen, että ennen kuin tätä muutosta aletaan tehdä, meidän tulee refaktoroida senkin rakenne yhtäläiseksi. Koodista löytyi muun muassa kosolti metodeja, joita ei käytetty missään. Päivän päätteeksi aloin korjaamaan työparini löytämiä vikoja tiistaina tekemästäni käyttäjänhallintaan liittyvästä muutostyöstä.

Torstai 1.4.2021

Torstaina tein työt etänä kotoa. Saamme itse valita teemmekö työt toimistolta vai kotoa käsin. Vallitsevasta koronatilanteesta huolimatta, olen itse kulkenut pääasiassa toimistolla. Itse kuulun niihin ihmisiin, jotka haluavat erottaa työpaikan ja kodin selkeästi toisistaan. Pääsiäisviikosta johtuen perjantai on vapaa, joten tein ensi viikon tuotannon päivityksen valmistelut jo tänään.

Ensin varmistin, että kehitysympäristössäni on Master-haaran tuorein versio ja tein haarasta uuden tagin, jonka lähetin versionhallintaan. Versionhallinta tekee tämän jälkeen haarasta automaattisesti uudet Docker-imaget, jotka se julkaisee konttirekisteriin nimenään tagille antamani semanttisen versioinnin mukainen versionumero. Seuraavaksi päivitin kyseisen versionumeron Deployment repositorion Docker-konfiguraatiotiedostoihin, tein siitä uuden haaran ja lopulta yhdistämispyynnön Production-haaraan. Ensi viikolla työparini tehtäväksi jää katselmoida yhdistämispyyntöni ja hyväksynnän jälkeen tuotantoympäristöön siirtyy sovelluksen uusin versio.

Loppupäivä meni Back End -koodia refaktoroidessa. Kehitysympäristöni asetuksissa on automatiikka, joka muokkaa kaikki importoinnit absoluuttisiksi, jos siirrän tiedoston eri kansioon. Tämä aiheutti päänvaivaa, koska kaikki testit ovat rakennettu käyttämään relatiivisia importointeja. Muutaman kansion läpikäytyäni huomasin, että kymmenet yksikkötestit eivät enää menneet läpi. Aikani yritin korjailla tekojani, kunnes huomasin olevan nopeampaa vain aloittaa alusta. Tämän päivän oppima olkoonkin siis se, että toisinaan kannattaa ennemmin aloittaa puhtaalta pöydältä, kuin yrittää sammutella pieniä paloja niiden ilmestyttyä.

Perjantai 2.4.2021

Pitkäperjantai, vapaapäivä.

Viikkoanalyysi

Viikon aikana tutustuin entistä syvemmin sovelluksen rakenteeseen sekä eritoten käyttäjänhallintaan. Se onkin hyvin tärkeä aihe, aika harvassa ovat nykyaikana sellaiset sovellukset, joissa ei olisi käyttäjänhallintaa jossakin muodossa. Erilaiset käyttäjäroolit ja niiden mukaan näytettävä sisältö auttaa myös pitämään koodirakenteen järkevänä.

Harmaita hiuksia saivat aikaan kehitysympäristön automatiikan aiheuttamat virheet. Hyvälaatuinen kehitysympäristö on yksi tärkeimpiä ohjelmistokehittäjän työkaluja ja sen useiden eri ominaisuuksien hyötykäyttäminen on taito, johon kannattaa panostaa. Tälläkin kertaa automatiikan aiheuttamat virheet olivat lopulta käyttäjälähtöisiä. Kehittäjä kun on itse vastuussa asetuksista.

Toisen viikon viikkoanalyysin tutkimusaiheeksi valikoitui luonnollisesti käyttäjänhallinta. Mitä käyttäjänhallinta oikein sitten on? Sen avulla voidaan hallita käyttäjien pääsyä vain heille kuuluviin resursseihin. Esimerkiksi markkinointipuolen työntekijä tuskin tarvitsee pääsyä taloushallinnan resursseihin, tai toisinpäin. Käyttäjälle itselleen hallinta on usein näkymätöntä, mutta sen tulos ei. Käyttäjät saavat näin turvallisen ja kitkattoman yhteyden omiin resursseihinsa tehdäkseen työnsä mahdollisimman hyvin. (Blanton 2021.)

Pääasialliset oikeudenhallinnan kategoriat ovat harkinnanvarainen, pakollinen sekä roolipohjainen.

- **Harkinnanvarainen oikeushallinta** perustuu järjestelmän omistajan päätöksiin. Omistaja voi evätä tai lisätä oikeuksia sen mukaisesti, kuinka paljon hän luottaa kyseiseen käyttäjään. Normaalisti käyttäjä saa järjestelmän täydet oikeudet ja tämä onkin harkinnanvaraisuuden huono puoli väärinkäytösten riskin takia.(Danturthi 2020, luku 5.)
- **Pakollisessa oikeushallinnassa** käyttäjälle annetaan tietyn tasoinen lupaluokitus ja vastaavasti resursseille annetaan noita luokituksia vastaavat lokeroinnit. Kukin käyttäjä saa käyttöönsä vain oman turvaluokituksensa mukaiset resurssit. Yleisesti käytössä silloin kun resursseissa on paljon salassa pidettävää tietoa.(Danturthi 2020, luku 5.
- **Roolipohjaista oikeudenhallintaa** käytetään ryhmittelemällä käyttäjät tiettyihin rooleihin, jotka liittyvät heidän omiin ammatteihinsa. Esimerkiksi pankkivirkailijan tulee päästä vain tiettyihin järjestelmän osiin ja perustason ohjelmoijalle ei välttämättä anneta oikeuksia julkaista koodia tuotantoympäristöön.(Danturthi 2020, luku 5.)

Oikeudenhallinnan päätarkoitus on turvata tietoturvan kolmijoukko: saatavuus, luottamuksellisuus sekä eheys (Danturthi 2020, luku 5).

4.3 VIIKKO 14 – Versionhallinta

Maanantai 5.4.2021

Toinen pääsiäispäivä, vapaapäivä.

Tiistai 6.4.2021

Pitkän pääsiäisviikonlopun jälkeen työpäivä alkoi totuttuun tapaan sovelluksen päivityksellä. Työparini ilmoitti aamupalaverissa löytäneensä edellisviikolla tekemästäni käyttäjienhallintamuutoksesta bugin. Hänen selaimensa näytti virheviestiä milteipä joka toiminnolla mitä hän yritti suorittaa. Luonnollisesti olin itse testannut toiminnallisuuden, joten vika vaikutti oudolta. Latasin versionhallinnasta tuon muutoksen sisältävän haaran ja aloin koestamaan tuotostani uudelleen.

Omalla koneellani vika toistui ja hyvin lyhyen tutkiskelun jälkeen sen syykin selvisi. Edellisviikolla olin tehnyt muutoksia samaan tiedostoon useammassa eri versiohaarassa, ja jokin noista haaroista oli yhdistetty jo Master-haaraan. Tämä oli saanut aikaan konfliktin, jonka olin luullut korjaavani, mutta tosiasiassa samalla olinkin tullut poistaneeksi tiedostosta tässä haarassa tekemäni muutokset. Tästä syystä tässä haarassa oli jäljellä vain HTML-muutokset, joiden myötä käyttöliittymä muuttui ja siihen tuli uusia painikkeita, mutta noilla painikkeilla ei ollut enää suoritettavia funktioita painettaessa. Korjaus oli hyvin suoraviivainen tapahtuma, täytyi vain luoda uudelleen jo kerran tekemäni funktiot. Päivän opetus oli tarkkaavaisuus versionhallinnan käytössä.

Keskiviikko 7.4.2021

Keskiviikolle oli jälleen sovittu palaveri johon asiakaskin olisi osallistumassa. Nykykäytännöistä poiketen palaveri pidettiin lähipalaverina toimistollamme. Jo aiemmin esillä ollut useita integraatioita vaativa ominaisuus oli jälleen palaverin isoimpana aiheena. Eräältä palveluntoimittajalta oli saatu jo tietoa, etteivät he ehdi toteuttaa tarvittavia muutoksia rajapintaansa lähiaikoina. Tämä vaikeuttaa olennaisesti meidän järjestelmässämme käyttäjän yksilöimistä. Rajapinta ei toimita minkäänlaista käyttäjää yksilöivää tunnistetta, joten yksilöinti olisi tehtävä vertailemalla useita perusparametreja. Tämä ei luonnollisestikaan ole hyvä tapa koska on mahdollista, että joillain käyttäjillä voi olla samat tunnisteet. Päätämme toteuttaa sovellukseen ensin historialokin, jota voimme myös käyttää apuna yksilöinnissä. Itse muutoksen toteutus siis siirtyy jälleen.

Eilisen korjaustyön tiimoilta havaitsimme työparini kanssa sovelluksesta uuden refaktorointia vaativan kohteen. Käyttäjän hallintaan liittyvä roolien tallentaminen on tehty hyvin monimutkaisella tavalla ja se käyttää tiliasetusten muuttamiseen tarkoitettua palvelukomponenttia, vaikka käyttäjienhallinnalla on omakin palvelunsa. Otan tehtävän työn alle ajatellen sen olevan näppärä pikku homma. Voi kuinka väärässä olinkaan.

Saadakseni muutoksen tehtyä, minun pitää muuttaa useita eri tietokannasta dataa hakevia funktioita, tehdä Back end -koodiin muutoksia ja lopulta parannella sovellusta käyttöliittymänkin osalta. Loppupäivä ei työhön riitä, joten sen tekeminen jatkuu torstaina.

Torstai 8.4.2021

Sain edellispäivänä käyttöliittymää koskevat muutokset tehtyä, joten tämän päivän agenda on Back end -koodin muutokset. Palvelinpuolen teknologiana sovelluksessamme käytetään NestJS:iä. Se on laajasti käytetyn NodeJS:n ohjelmointikehys, jonka avulla saadaan Nodeen TypeScript tuki, sekä Angularin kaltainen arkkitehtuuri. Helpottaa paljon kehitystyötä, kun sovelluksen molempien osien koodit ovat arkkitehtuuriltaan samankaltaisia.

Back end -koodi huolehtii muun muassa tietokantaan tehtävistä hauista ja muutoksista. Perinteiset relaatiotietokannat käyttävät SQL-kieltä, joka taas poikkeaa merkittävästi yleisistä ohjelmointikielistä. On mahdollista tehdä palvelimen koodi sekoittaen SQL-syntaksia ohjelmointikielen kanssa. Tämä ei kuitenkaan ole hyväksi koodin selkeydelle, joten tähän tarkoitukseen on kehitetty Object Relational Mapping(ORM), jonka avulla voidaan tehdä tietokantakyselyt käyttäen haluttua ohjelmointikieltä. Meidän sovelluksessamme käytössä oleva ORM on nimeltään TypeORM. Se on TypeScriptiä käyttäen tehty, joten se tarjoaa hyvän tuen TypeScript-kielellä tehdyille sovelluksille.

ORM:n käyttö ei ollut itselleni entuudestaan tuttua, mutta huomaan kyllä sen edut käytössä. Refaktorointitehtävässäni jouduin tekemään useampiakin muutoksia tietokantakyselyihin ja sainkin näin perehtyä enemmänkin TypeORM:n tekniikoihin. Päivän päätteeksi sain tehtävän valmiiksi ja tein siitä versionhallintaan yhdistämispyynnön. Back end -ohjelmointi ei ole itselleni se tutuin alue, joten aina kun siellä operoin, opin uusia asioita. Niin myös tänäänkin.

Perjantai 9.4.2021

Perjantain tavoitteena oli totuttuun tapaan saada seuraavan viikon päivitykset valmisteltua. Tein uudet versiotagit ja puskin ne versionhallintaan. Tein myös yhdistämispyynnön Production-haaraan valmiiksi. Tämän jälkeen jatkoin päivittämällä Angularin riippuvuudet. Päänvaivaa aiheutti TypeScript-kielen päivittyminen versioon 4.2.3. Angular tukee TypeScriptiä ainoastaan versioon 4.1.5 asti. Konsoli ei tulostanut mitään järkikielistä virhettä, että päivityksen epäonnistuminen johtui tästä, joten aikaa kului internetin keskustelupalstoilta syitä etsiessä.

Loppupäivä kului pitkälti työparini yhdistämispyyntöjen katselmoinnissa, hän oli saanut viikon aikana useita Back end -puolen kansioita refaktoroitua. Jo aiemmin mainitsemani yhdistämiskonfliktit tosin haittasivat tässäkin tapauksessa työtä, koska useat refaktoroinnit muuttivat vuoron perään samoja tiedostoja. Näin ollen, kun sain yhden haaran yhdistettyä Master-haaraan, seuraavan yhdistäminen ei ollut enää mahdollista konfliktien vuoksi. Voisipa sanoa konfliktien ratkomisen olevan opettavaista. Hiljalleen alan nimittäin huomata jo refaktorointia tehdessä, että mikä muutos tulee todennäköisesti aiheuttamaan konfliktin.

Viikkoanalyysi

Ohjelmointitehtäväni alkavat selvästikin laajenemaan isommiksi ja oppimiskäyrä viikon aikana on hyvin jyrkkä. Olenkin huomannut, että hyvä lepo, sekä rentoutuminen aivan muita asioita tehden ovat todella oleellisia asioita uran tässä vaiheessa. Aivojen kapasiteetin saa helposti käytettyä loppuun, jos ei niiden anna palautua. Tämän lisäksi viikon oppimat liittyivät Back end -ohjelmointiin. Varmuus omaan tekemiseen kasvaa kohisten, kun huomaa voivansa soveltaa oppimiaan asioita muuallakin.

Viikon ongelmat liittyivät pitkälti versionhallinnassa tuleviin konflikteihin. Nekin tosin tarjoavat aina oppimispaikan tullessaan, joten ei niitä voi pelkkänä ongelmana pitää. Valitsin viikon analyysin aiheeksi versionhallinnan, vaikka se voisi kyllä hyvinkin olla aiheena joka viikko. Ohjelmointiympäristön lisäksi ohjelmoija todennäköisesti viettää seuraavaksi eniten aikaa versionhallinnan parissa. Yrityksemme, niin kuin todella monet muutkin ohjelmistoalan yritykset käyttävät versionhallintaan Git:iä.

Versionhallinta on nimensä mukaisesti ohjelmiston eri versioiden hallintaa. Siellä pidetään muistissa kaikki ohjelmiston kehityksen eri vaiheet ja sen avulla on helppo palata takaisin vanhempaan

versioon, jos esimerkiksi uudesta versiosta löytyy todella vakava bugi. Nykyaikaiset versionhallintaan keskittyvät sivustot tarjoavat myös kosolti muita palveluja ja niiden kautta onkin kätevää automatisoida koko ohjelmiston jatkuva julkaisu.

Versionhallinnan muotoja ovat muun muassa paikallinen-, keskitetty- ja hajautettu versionhallinta.

- **Paikallinen versionhallinta** on yksinkertaisimmillaan tiedostojen kopioimista eri kansioihin aikajärjestyksessä. Tämä on hyvin yleinen tapa, koska se on niin helppoa, mutta se on myös erittäin virheherkkää. On helppo unohtaa missä kansiossa on menossa ja vahingossa kirjoittaa aikaisempien versioiden päälle. On myös olemassa yksinkertaisia paikallisia tietokantamallisia versionhallintajärjestelmiä.(Chacon & Straub 2021, luku 1.1.)
- **Keskitetty versionhallinta** mahdollistaa eri järjestelmiä käyttävien kehittäjien yhteistyön samalla serverillä, joka sisältää kaikki ohjelmiston versiointitiedostot. Tämän mallin huonona puolena voidaan pitää sen suuria vaikutuksia vikatilanteissa. Serverin kaatuessa muutamaksi tunniksi, kukaan kehittäjistä ei voi tallentaa muutoksiaan tai nähdä muiden muutoksia. Projektin historian pitäminen yhdessä paikassa mahdollistaa koko projektin kerralla menettämisen.(Chacon & Straub 2021, luku 1.1.)
- **Hajautetussa versionhallinnassa** käyttäjät eivät ota itselleen vain muokkaamiansa tiedostoja, vaan he peilaavat koko repositorion historioineen itselleen. Näin ollen, jos jokin yksittäinen serveri kaatuu ja järjestelmät olivat yhteydessä juuri sen serverin kautta, voidaan käyttäjien repositorio kopioida takaisin serverille.(Chacon & Straub 2021, luku 1.1.)

Git on tyypiltään hajautettu versionhallintajärjestelmä. Sen loi vuonna 2005 suomalainen Linus Torvalds ja siitä on sittemmin muotoutunut versionhallinnan de facto -standardi. Sitä käyttää sekä useimmat kaupalliset ohjelmistoprojektit että kaikenkokoiset avoimen lähdekoodin projektit. Git toimii useimmissa käyttöjärjestelmissä ja kehitysympäristöissä.(Atlassian 2021.)

4.4 VIIKKO 15 – Olio-ohjelmoinnin perusteet

Maanantai 12.4.2021

Aloitin viikon refaktoroimalla Back end -koodissa superuser-komponentin omaavan kansiorakenteen. Ajattelin, että se ei kosketa niin monia muita komponentteja, joten pääsisin sen kanssa aika vähillä konflikteilla. Ilokseni huomasin olevani oikeassa, ainoat konfliktit tulivat muutoslokitiedostosta. Yhdistin Master-haaran omaan muutoshaaraani, ja sain näin ajantasaisen muutoslokitiedoston itselleni ja sen jälkeen muutos olikin valmis versionhallintaan.

Työparini tiedusteli minulta iltapäivällä, josko huomaan tekemässäni käyttäjänhallintamuutoksessa mitään sellaista, jolla käyttäjä voisi aiheuttaa ei-haluttuja asioita. Hetken aikaa tutkittuani havaitsin, että olin jättänyt myös kirjautuneena olevalle käyttäjälle mahdollisuuden poistaa itse itsensä. Tämä ei luonnollisestikaan ole oikea tapa toimia, ainakaan tässä sovelluksessa. Jälleen kerran huomaan, kuinka opittavaa riittää. Tällaiset käyttöliittymiin liittyvät sudenkuopat ovat päivänselviä senioritasoiselle ohjelmoijalle. Sovimme, että korjaan tämän virheen loppuviikosta.

Tiistai 13.4.2021

Käyttäjienhallinnan muutoksista johtuen superuser-roolitetut käyttäjät menettivät mahdollisuuden poistaa ja muokata käyttäjiä. Tämä huomattiin toki jo kesken muutostyön ja päätettiin saattaa sen hetkinen työ valmiiksi ja palata tähän ongelmaan myöhemmin. Päätin nyt olevan hyvä hetki tälle muutostyölle. Käyttäjienhallinta nojaa vahvasti käyttäjän omistavan tilin yksilöivään tunnukseen, mutta superuser:n tili ei varsinaisesti omista mitään käyttäjiä. Tämä aiheuttaa hieman päänvaivaa koska sovellus on rakennettu niin, että palvelinpuolella JWT-tokenista parsitaan käyttäjän tilitiedot, joilla saadaan oikeuksia eri tapahtumiin. Halusin ehdottomasti, että tästedes sovellus käyttää samoja kontrollereita tapahtuipa käyttäjiin liittyvät muutokset mistä komponentista tahansa. Jotta tämä onnistuisi, täytyy superuser:n ollessa kirjautuneena välittää halutun tilin yksilöivä id payloadin mukana.

Angular on rakennettu käyttäen TypeScript-kieltä, joten se on myös kehityksessä lähes pakollinen valinta. TypeScript tuo JavaScriptiin verrattuna mukanaan muun muassa vahvan tyypityksen. Kaikki data tulisi pyrkiä tyypittämään, sekä olioille tehdä niiden rakenteen määrittävät luokat tai niin kutsutut Interfacet. Sovelluksessamme käyttäjä on olio, jota käytetään useissa eri yhteyksissä ja

sille on siis määrätty tietty rakenne luokkansa mukaisesti. Jotta saan ylimääräisen tietueen payloadin mukana, tulee minun joko muokata luokan rakennetta tai tehdä esimerkiksi uusi vain tätä tarkoitusta varten oleva interface, joka ottaa käyttöönsä käyttäjäolion ominaisuudet, mutta lisää siihen myös omansa. Luokkarakenteen muokkaaminen toisi mukanaan lieveilmiöitä ja isomman muokkaustyön, joten päädyn jälkimmäiseen vaihtoehtoon.

Oma historiani on suurimmalta osin JavaScriptin parista, eikä se ole puhdas luokkapohjainen olio-ohjelmointikieli. Luonnollisesti olio-ohjelmoinnin perusteet luokkineen ovat minulle tuttuja käsitteinä, mutta niiden kanssa toimiminen on jäänyt vähemmälle JavaScriptin luonteesta johtuen. TypeScriptin myötä olen päässyt tutustumaan näihin konsepteihin paremmin ja pidänkin TypeScriptiä JavaScriptin paranneltuna versiona.

Keskiviikko 14.4.2021

Päivä alkoi versiohallinnassa työparini yhdistämispyyntöjä katselmoiden. On harmillista, että miltei aina, kun yhdistää muokatun haaran Master-haaraan se saa aikaiseksi konflikteja muihin yhdistämispyyntöihin. Olemme sopineet työparini kanssa, että kumpikaan ei hyväksy omia yhdistämispyyntöjään, mikäli ei ole ihan pakko. Tästä johtuen yhdistämispyynnöt tahtovat jäädä roikkumaan, kun aina pitää jäädä odottamaan, että toinen on korjannut konfliktit omasta muutoksestaan, ennen kuin voi yhdistää.

Tein myös eiliseen työhön liittyen Back end -koodiin tarvittavat muutokset. Muutoksen yhteydessä, tuli tarpeelliseksi saada kutsun yhteydessä tieto siitä, onko kutsun lähettänyt käyttäjä superuser. Ajattelin, että tätä tarkistusta voisi tarvita muissakin kontrollereissa, joten päätin tehdä niin sanotun Decoratorin, joka on aina tietoinen tästä asiasta. Decorator on kaikkien kontrollereiden saatavissa ja sitä voi kutsua tarvittaessa.

Torstai 15.4.2021

Viikon työt alkoivat olla sillä mallilla, että päätin alkaa korjaamaan maanantaina huomaamaamme käyttäjäkokemukseen liittyvää virhettä, jossa käyttäjä pystyy poistamaan itse itsensä. Korjauksen voisi tehdä pelkästään selainpuolella pyörivään koodiin, mutta on aina turvallisempaa, kun estää tällaiset toiminnot myös Back End -koodissa.

Selainpuolella toteutan korjauksen käyttämällä painikkeen luonnin yhteydessä ehtodirektiiviä, joka tarkistaa onko käyttäjän identifioiva id-tunnus sama, kuin kyseisellä hetkellä sisään kirjautuneella käyttäjällä. Mikäli on, painiketta ei tuoda näkyviin. Tämä poistaa tehokkaasti poistamisen mahdollisuuden, mutta tietämällä oikean URL-osoitteen ja käyttäjän id-tunnuksen, voi mahdollisesti saada poistopyynnön lähtemään.

Kyseinen ominaisuus estetään palvelinpuolen koodissa. Jo tällä hetkellä poistamiseen liittyy ehtoja, esimerkiksi poistettavan käyttäjän tulee olla saman tilin käyttäjä, kuin poistajakin. Lisään vain ehtojen joukkoon vielä kohdan, joka tarkistaa, ettei poistettava käyttäjä ole sama, kuin jo kirjautunut käyttäjä.

Perjantai 16.4.2021

Yritys, jossa työskentelen, on aika nopeastikin kasvanut isoksi konserniksi ja sen rakenne on aika hajanainen. Toimialoja on monia ja eri toimialojen edustajia erinäinen määrä eri toimistoilla ympäri Suomen, joten sekaannuksilta ei voi välttyä. Perjantai olikin pyhitetty organisaatiorakenteen esittelyyn, sekä muihin yrityksen sisäisiin koulutuksiin.

Perinteisten perjantaiaamuisten päivitystoimien lisäksi en juurikaan tietokoneeseen loppupäivänä koskenut. Ihan kiva välillä tutustua enemmän yritykseen ja sen ihmisiin paremmin. Päivä selkeyttikin mukavasti omaakin paikkaani yrityksessä. Siinäkin kun oli ollut omat epäselvyytensä.

Viikkoanalyysi

Viikot alkavat tässä vaiheessa jo toisinaan toistaa itsejään. Tällä viikolla keskityttiinkin pääasiassa samoihin tehtäviin, joita oli tehty jo aiemmilla viikoilla. Pidän toistoa tässä vaiheessa uraani kuitenkin tärkeänä, kaikki tärkeimmät periaatteet iskostuvat sen avulla syvälle muistiin, josta ne ovat tarvittaessa helppo hakea käyttöön. Edelleenkin kuitenkin tuli opittua uusia asioita ja ennen kaikkea kerrattua jo aiemmin opittua. Käyttäjäkokemukseen liittyvät ongelmat ovat hyvin tärkeitä sisäistää, koska sovelluksia tehdään pääasiassa käyttäjille, jotka eivät itse ole ohjelmoijia. Ei voida siis olettaa, että käyttäjä käyttäytyisi samalla tavalla kuin itse ajattelee, vaan on estettävä kaikki huomaamansa virhemahdollisuudet. Viikolta parhaiten mieleen jäivät olio-ohjelmointiin liittyvät luokkamäärittelyt ja rajapinnat. Tässäpä siis oiva syy perehtyä tähän itseänikin kiinnostavaan aiheeseen tarkemmin.

Olio-ohjelmoinnin peruskomponentit ovat luokka ja olio. Luokka on eräänlainen pohjapiirros tai lista ohjeita, joilla rakennetaan tietynlainen olio. Se kuvastaa yleensä jotakin tosielämän entiteettiä ja määrittää miten olio käyttäytyy ja mitä se sisältää. (Guru99 2021.)

Olio on luokan ilmentymä. Se on itsenäinen komponentti, joka sisältää metodeja ja ominaisuuksia tehdäkseen tietynlaisesta datasta hyödyllistä. Kun lähetät oliolle viestin, pyydät oliota kutsumaan tai suorittamaan jotakin sen metodeista, niin kuin luokka määrittelee. (Guru99 2021.)

Jotta ohjelmointikieli olisi oikea olio-ohjelmointikieli, sen täytyy käyttää neljää olio-ohjelmoinnin perusperiaatetta. Nämä ovat Kapselointi, Abstraktio, Perintä ja Polymorfismi. Niitä kutsutaan myös olio-ohjelmoinnin neljäksi pilariksi.(Chandel 2018.)

Kapselointi tapahtuu, kun jokainen olio ylläpitää oman yksityisen tilansa luokan sisällä. Muut oliot eivät voi suoraan päästä käsiksi tähän tilaan, sen sijaan ne voivat kutsua luokan julkisia funktioita. Olio ylläpitää tilaansa näillä funktioilla, eikä muut luokat voi muuttaa sitä, ellei sitä erikseen sallita. (Banda 2020.)

Abstraktio on kapseloinnin jatke. Se on prosessi, jossa isosta datamäärästä näytetään oliolle ainoastaan tarvittavat asiat. Jos esimerkiksi vastaanotat käyttäjistä paljon eri informaatiota, mutta et tarvitse siitä kuin nimen, tuon nimen hakeminen on abstraktiota. (Banda 2020.)

Perintä tarkoittaa olion kykyä saada käyttöönsä toisen olion kaikki tai jotkin ominaisuudet. Perimisen suurimpia hyötyjä on uudelleenkäytettävyys, voit käyttää jo olemassa olevan luokan kenttiä ja metodeja hyväksesi. (Banda 2020.)

Polymorfismi antaa mahdollisuuden käyttää luokkaa samalla tavalla kuin sen hierarkiassa ylempääkin luokkaa, jottei tyypitysten kanssa tule sekaannuksia. Jokaisella alaluokalla on omat funktiot ja metodit juuri sellaisena kuin luokka ne tarvitsee. Esimerkiksi luokalla nisäkäs voi olla metodi nimeltään nisäkkäänÄäni(). Nisäkkään Alaluokkina voisi olla koira, valas, elefantti ja hevonen. Kaikilla noilla alaluokilla olisi oma toteutuksensa nisäkkäänÄäni() funktiolle. (Banda 2020.)

4.5 VIIKKO 16 – Debuggaus

Maanantai 19.4.2021

Asiakas oli löytänyt edellisviikon päivityksen jälkeen testiympäristöstä vakavan bugin. Juuri käyttäjän valitessa maksutapaa, sovellus jämähtää paikalleen eikä anna asiasta mitään virheilmoitusta. Tämä olikin erittäin hyvä esimerkki, miksi päivitykset tulee tehdä ensin testiympäristöön. Vaikkakin on lähes mahdotonta testata kaikki mahdolliset skenaariot, joissa bugeja voi ilmetä.

Työparini joutui hätäavuksi toiseen projektiin, joten päädyin selvittelemään bugia itsekseni. Debuggaus, eli virheenjäljitys koodista voi olla haastavaa. Virheet voivat toistua esimerkiksi vain tietyssä tilassa ja sovellus voi toimia toisella selaimella eri tavalla kuin toisella. Debuggauksesta tulee vielä entistä haastavampaa, mikäli koodin on alun perin kirjoittanut toinen ohjelmoija.

Googlen tekemässä Chrome-selaimessa on sisäänrakennettu ja erittäin käytännöllinen DevTools, jolla pystyy hyvin tarkastelemaan muun muassa eri pyyntöjä, joita selain suorittaa sovelluksen käynnissä ollessa. Monissa tapauksissa myös DevTools-konsoliin tulostuu virheilmoituksia, ainakin selkeimmistä virheistä joihin selain törmää. Kehittäjät yleensä, niin kuin minäkin tässä tapauksessa, aloittavat virheenjäljityksen juuri konsolista. Seurasin asiakkaan kertomusta ja melko pian sain itsekin virheen toistumaan ja konsolissa näkyikin punainen virhe heti kun valitsin jommankumman maksutavoista. Tämän myötä pääsin itse bugin jäljille ja löysinpä siinä samalla toisen vielä huomaamattona pysyneen virheen. Alkuperäinen virhe osoittautui minun aikaansaannoksekseni. Refaktoroidessani maksutoimituksiin liittyvää komponenttia, olin poistanut luokan alusta erään muuttujan alustuksen ja aikeeni oli siirtää kyseinen alustus luokan konstruktoriin, mutta olin lopulta unohtanut tehdä sen. Aikaa virheen löytämiseen ja korjaamiseen kului pari tuntia, joten on selvää, että bugit voivat tulla kalliiksi. Loppupäivä kului tutuissa refaktorointi puuhissa, tällä kertaa vuorossa oli maksamiseen liittyvien komponenttien Back End -koodipohja. Tapahtumien johdosta pyrin olemaan entistäkin tarkkaavaisempi tekemisissäni.

Tiistai 20.4.2021

Aloitimme aamun päivittämällä testiympäristöön bugittoman version koodista. Asiakas pyysi meitä testaamaan ominaisuuksia kanssaan, jotta mahdollisimman monet virheet saataisiin kiinni jo tässä vaiheessa. Mitään vakavaa ja särkevää bugia emme löytäneet parin tunnin testaamisella. Asiakas

aikoi kuitenkin testailla sovellusta vielä yhteistyökumppanienkin kanssa, joten tuotannon päivitys siirtyy edelleen seuraavalle päivälle.

Samalla saimme kuulla, aiemmasta tiedosta poiketen, että yhteistyökumppanimme saa sittenkin tarvittavat muutokset rajapintaansa tehtyä jo ennen kesää. Saimme jopa tulevien rajapintojen kuvaukset etukäteen, jotta voimme aloittaa työn valmistelut. Rajapintoihin liittyvät muutokset ja niiden suunnittelu on minulle vielä suhteellisen uutta asiaa, joten pyydän työpariani avaamaan omaa ajatusprosessiaan varsinkin suunnitteluvaiheesta. Päätimme aloittaa työn torstaina, kun olemme saaneet kyseistä rajapintakomponenttia koskevat refaktoroinnit tehtyä ja katselmoitua

Keskiviikko 21.4.2021

Edellisiltana asiakas oli kuitannut sähköpostilla testausten olleen onnistuneita, joten voimme vihdoinkin päivittää tuotantoympäristöönkin uuden version. Olen edelleenkin ollut vastuussa jokaisesta päivityksestä, joten työ alkaa tosiaan menemään rutiinilla.

Saimme alkuviikosta uuden työntekijän tiimiimme. Hän on muuttanut Suomeen Intiasta noin kolme vuotta sitten ja on alan todellinen ammattilainen. Aluksi tuntui oudolta muuttaa toimistolla käytetty puhekieli englanniksi, mutta siihen tottui onneksi melko pian. On hienoa huomata yrityksemme houkuttelevan myös kokeneita ammattilaisia. Mielestäni on hyvä merkki, jos yrityksen henkilöstöllä on erilaisia taustoja. Me juniorit saamme senioreilta arvokkaita neuvoja ja toivottavasti vaihtokauppana tuomme heille tuoreempaa näkemystä joidenkin fakkiutuneiden toimintatapojen tilalle.

Loppupäivästä lähdimme tiimin kesken Letonniemen luonnonsuojelualueelle makkaran paistoon ja tutustumaan paremmin. Aivan kaikki eivät paikalle päässeet, mutta oli todella mukava tavata oikeasti työkavereita, jotka olivat tähän asti tulleet tutuiksi vain etäyhteyksien välityksellä.

Torstai 22.4.2021

Yhteistyökumppanimme tuottamasta palvelusta tulee tietyissä tapauksissa välittää tietoa meidän sovellukseemme. Heidän sovelluksensa saa tiedon toisesta palvelusta, jonka se välittää meidän sovellukseen rajapintamme kautta tiettyjen ehtojen täyttyessä. Kun meidän sovelluksemme saa tämän tiedon, suorittaa se kyselyn kolmanteen palveluun, josta saa lisää tietoa tapahtumasta, ja lopulta välittää tämän tiedon jossain muodossa vielä kolmelle eri palvelulle. Kun prosessi on jossain

vaiheessa valmis, lähtee siitä tieto meidän sovelluksestamme yhteistyökumppanimme sovellukselle.

Tämänpäiväisessä rajapintamuutoksessa on tarkoitus tehdä niin sanottu ”Mock”-rajapinta, jota yhteistyökumppanimme voi käyttää oman sovelluksensa testaamisessa. Rajapinta palauttaa aluksi vain kovakoodattuja arvoja, ja on siis puhtaasti vain testauskäyttöön. Päädyimme suorittamaan muutokset pariohjelmointina, jossa minä en kuitenkaan ohjelmoinut vaan vain katselin mitä työparini teki. Hän myös selosti koko ajan mitä ja miksi on tekemässä. Tämä oli todella tärkeä hetki oppimiseni kannalta. On todella hienoa, kun on osaava työpari, joka vielä kaiken lisäksi osaa opettaa ja haluaa jakaa tietoaan meille nuoremmille ohjelmoijille.

Saatuamme muutokset tehtyä, pidimme nopean palaverin asiakkaamme ja yhteistyökumppanimme kanssa, jossa sovittiin yhteisistä pelisäännöistä ja siitä, missä muodossa datan tulee olla, kun se lähetetään meille. Päivä oli oppimisen kannalta erittäin arvokas. Erilaiset integraatiot palveluiden välillä ovat erittäin tärkeä asia osata, ja pyrinkin sisäistämään niin paljon aiheesta kuin oli mahdollista.

Perjantai 23.4.2021

Eilinen rajapintamuutos ehdittiin saada mukaan testiympäristön päivitykseen, joten tämänpäiväisessä versiotagien luonnissa se saadaan mukaan myös ensi viikon tuotannon päivityksen. Asiakas oli tästä erittäin tyytyväinen, koska tämä kyseinen integraatio on keskeisessä asemassa hänen asiakashankinnassaan ja kaikki edistysaskeleet ovat siksi hänelle tärkeitä.

Viikon lopuksi päivitettiin myös muutosloki vastaamaan tilannetta. Versiotagien luonnin jälkeen on taas mahdollista tehdä yhdistyksiä Master-haaraan, joten otimme tehtäväksemme käydä niin paljon toistemme tekemiä yhdistämispyyntöjä läpi kuin ehdimme. Osa refaktoroinneista oli erittäin massiivisia, joten niiden katselmoinnissakin menee aikaa. Eräs työparini tekemä yhdistämispyyntö sisälsi muutoksia miltei kahdessasadassa eri tiedostossa. Vaatii todellista tarkkaavaisuutta käydä ajatuksen kanssa kaikki muuttuneet rivit läpi.

Viikkoanalyysi

Viikko oli todella vaihteleva ja sisälsi vian etsintää ja korjaamista, rajapintamuutoksia sekä jo kovin tutuksi käynyttä refaktorointia. Vaikkei viat sinällään ole mitenkään positiivisia asioita, tämän vian etsintä oli mielestäni mukava ja opettavainen tehtävä. Sopivasti olin vielä itse sen alun perin aiheuttanut. Refaktoroinnin tärkeydestä huolimatta se voi välillä käydä vähän puuduttavaksi, joten on välillä kiva tehdä muutakin. Rajapintamuutoksen aloittelu oli itselleni hyvä esimerkki siitä, kuinka ihminen mielessään paisuttelee asioita monimutkaisemmaksi, kuin ne lopulta oikeasti ovatkaan.

Päätin jo alkuviikosta bugia koodista etsiessäni, että tämän viikon viikkoanalyysin aiheena tulee olemaan Debuggaus eli virheenjäljitys. Aiheena se on takuuvarmasti tuttu jokaiselle ohjelmoijalle, ja kaikista ohjelmoinnin tärkeistä aiheista sellainen, johon olen itse vähiten perehtynyt. Olen kyllä monesti joutunut vikoja etsimään, mutta en ole aktiivisesti tutkinut keinoja, joilla siitä voisi tehdä helpompaa itselleen.

Ohjelmistoja testataan, päivitetään ja huolletaan kehitysprosessin aikana. Tavallisesti ohjelmistot sisältävät virheitä, jotka korjataan tavatessa. Debuggaus on prosessi, jossa vika korjataan ohjelmistosta. Siihen liittyvät vian tunnistaminen, analysointi sekä sen poistaminen. Prosessi alkaa, kun ohjelmisto ei käyttäydy halutulla tavalla ja päättyy, kun vika on saatu korjattua ja ohjelmiston testaus on onnistunut. (Debedureka 2021.)

On olemassa lukuisia eri tyylejä debuggaamiseen. Eri ohjelmointikielien välillä voi olla suuriakin eroja vianhakuun käytettävissä apuvälineissä ja metodeissa. Seuraavaksi käyn läpi muutamia eri tekniikoita, jotka eivät ole teknologiakohtaisia.

Jäljestämiseen perustuva vianhaku. Perinteinen ja yleisimmin käytetty vianhaku-tekniikka, joka on käytössä myös useimmissa moderneissa Debuggaus-työkaluissa. Perustuu pysäytyspiste-konseptiin. Pysäytyspiste on merkintä koodissa, jossa ohjelman suoritus pysäytetään, jotta voidaan tutkia ohjelman tilaa tarkemmin rivi kerrallaan. (Metwalli 2020.)

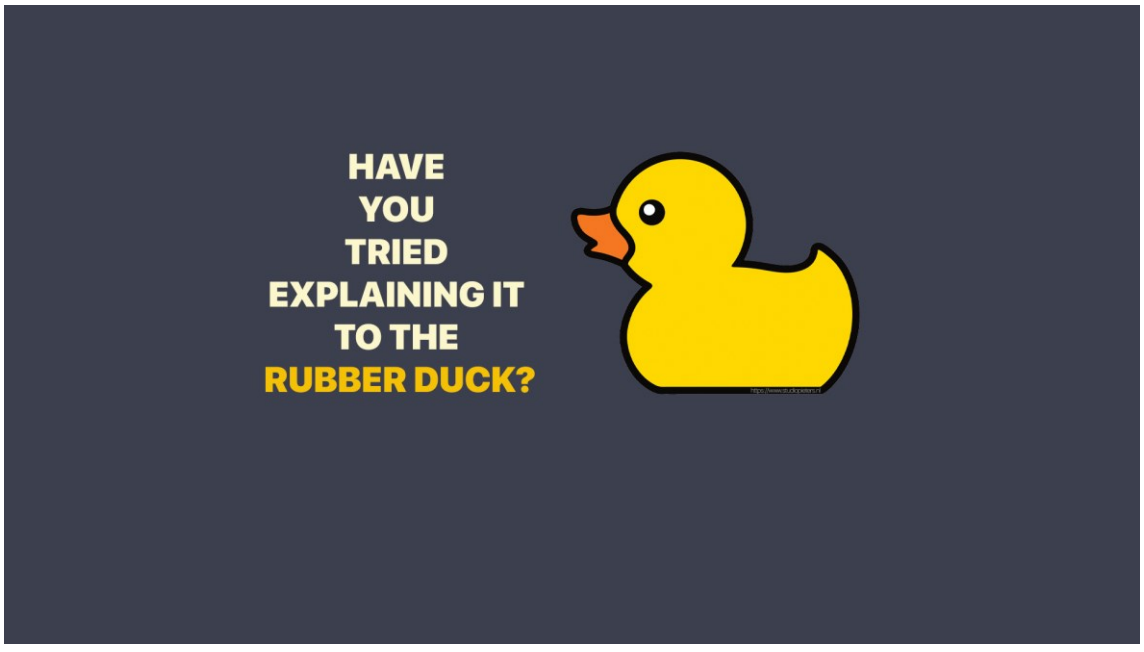
Spektri-pohjainen vianhaku. Prosessi tapahtuu monitoroimalla lausekkeita, jotka kuuluvat johonkin tiettyyn suorituspuuhun. Tämä saavutetaan käyttämällä ohjelman spektriä ohjelman aktiivisen osion tunnistamiseen. (Metwalli 2020.)

Delta vianhaku. Prosessissa minimoidaan automatisoidut testitapaukset. Otetaan vain ne testitapaukset, jotka voivat aiheuttaa virheitä ja tehdään niistä lista. Tuosta listasta valitaan tapaukset, joilla on korkea mahdollisuus virheen tuottamiseen. Useat pienet testitapaukset johtavat samaan virheeseen ja auttavat näin ollen kehittäjää löytämään sen. (Metwalli 2020.)

Itse käytän eniten vianhakuun selaimen konsolia. Se on JavaScript-kehittäjille erittäin yleinen testialusta. Usein myös auttaa paljon, kun kertoo ääneen mitä ohjelman kunkin rivin kuuluisi tehdä, ja mitä sen sijaan tapahtuu. Mahdollisuuksien mukaan tämän voi tehdä toisen ohjelmoijan kanssa. Kun yrittää selittää ohjelman toimintaa mahdollisimman tarkasti, yleensä tulee huomanneeksi mistä vika johtuu. Tähän perustuu myös alalla ilmiöksi noussut, oma suosikkini, kumiankka-metodi.

Kumiankka-metodi:

1. Hanki itsellesi kumiankka, keinolla millä hyvänsä.
2. Aseta kumiankka pöydälle ja ilmoita sille, että aiot käydä sen kanssa koodiasi läpi.
3. Selitä kumiankalle, mitä koodisi tulisi tehdä, ja lopulta selitä sen toiminta rivi riviltä.
4. Jossakin kohtaa, kun kerrot kumiankalle mitä seuraavaksi tapahtuu, huomaat ettei haluamaasi asiaa tapahdukaan ja olet löytänyt virheen. Kumiankka tuijottaa sinua tyytyväisenä ja iloisena siitä, että sai auttaa sinua. (Rubber duck debugging 2008.)



KUVIO 2. Rubber duck debugging (Achim Pieters 2021).

4.6 VIIKKO 17 – Rest API-rajapinnat

Maanantai 26.4.2021

Tutut päivitystoimenpiteet, sekä niihin liittyvät testaamiset käynnistivät vappuviikon. Päivä kului hyvin pitkälti edellisviikolla tekemieni muutosten korjausten parissa, työparini oli huomannut niissä joitakin lähinnä muotoiluun liittyviä asioita, jotka hänen mielestään tulisi tehdä toisella tavalla.

Löysin myös pari kansiota Back end -koodistamme, joita ei oltu vielä refaktoroitu. Kansiot sisälsivät suhteellisen vähän tiedostoja, joten refaktorointi tapahtui nopeasti. Käytin päivästä myös osan tutoriaalien parissa. Angularissa paljon käytettävät Observablet ovat kyllä minulle tuttuja, mutta niiden tehokas käyttö vaatii vielä petraamista.

Tiistai 27.4.2021

Tiistai oli miltei täydellinen maanantain toisinto. Ainoa erottava tekijä oli pitkähkö asiakaspalaveri. Viime viikolla aloitetut rajapintamuutokset saavat tällä viikolla jatkoa ja niiden sovittelussa riittää vielä tekemistä. Kuulemma useita järjestelmiä yhdistävät integraatiot ovat yleensäkin aika hitaasti alkavia. Standardeista huolimatta kaikilla on jossain määrin oma tapansa tehdä asioita.

Aloitan itse rajapintamuutosten seuraavan vaiheen huomenna ja vietinkin loppupäivän pitkälti aihetta sivuavien tutoriaalien parissa. Viimeistelin myös viimeiset työparini tekemät korjausehdotukset.

Keskiviikko 28.4.2021

Eilisessä palaverissa sovittiin, että minä tekisin loput rajapintamuutokseen tarvittavat työt itsenäisesti, mutta työparini auttaisi tarvittaessa. Tämän päivän tavoitteena on, että saadaan yhteistyökumppanillemme rajapinta, johon hän voi lähettää tietynmuotoisen http POST-pyynnön ja rajapintamme vastaa http-statuksella 201 (Created). Vielä sen ei siis varsinaisesti tarvitse käsitellä dataa sen kummemmin.

Rajapintamme, niin kuin monet muutkin rajapinnat, toimii niin, että tiettyyn ennalta määritettyyn URLiin lähetetään http-pyyntö, jonka vastaanottaa sovelluksemme kontrolleri. Kontrollerilla on eri

funktiot eri http-pyynnöille, joita se kutsuu automaattisesti oikean muotoisen pyynnön vastaanottaessaan. Tuosta funktiosta yleensä kutsutaan jonkun palvelukomponentin funktiota, joka joko suorittaa tietokantakyselyjä tai ottaa yhteyden muihin palveluihin.

Jotta rajapinta saadaan toimimaan halutulla tavalla, tulee kontrollerit määrittää vastaanottamaan dataa, jolla on tietty rakenne. Näin rakenteen poiketessa voidaan heti lähettää virheilmoitus. Tähän tarkoitukseen useissa ohjelmointikielissä on käytössä DTO (Data Transfer Object). DTO on tavallaan kuin luokka, joka määrää tulevan datan rakenteen. Sitä käytetään tosin myös muokkaamaan kontrollerin palauttamaa dataa. Kontrollerilla on pääsy paljon isompaan määrään dataa, kuin monesti on tarpeen, joten DTO:n avulla voidaan määritellä vain tarvittava määrä palautettavaa dataa. Tein molemmille kontrollereille omat DTO:t, sekä validointiputken, joka validoi datan ja on vastuussa mahdollisista virheviesteistä. Tämän muutoksen jälkeen yhteistyökumppanimme voi alkaa jo testaamaan rajapintamme toimintaa oman sovelluksensa kanssa. Vielä rajapinta ei palauta mitään, eikä varsinaisesti edes suorita mitään, mutta se antaa sen kutsujalle tietyt viestit, jos se saa oikean muotoista dataa.

Päivän aikana opin monia uusia asioita käytännön tasolla, sekä sisäistin paremmin jo aiemmin oppimiani asioita. Muun muassa DTO oli minulle tuttu konsepti ennestäänkin, mutta sellaisen määrittäminen oli sangen avartava kokemus.

Torstai 29.4.2021

Päivän tavoitteena oli aloittaa varsinainen muutostyö ja saada vähintään toinen endpoint palauttamaan jotakin siihen lähetettyyn pyyntöön liittyen. Aloitin muokkaamalla kontrollerit oikeanlaiseksi, jotta ne voivat kutsua oikeita palvelukomponentin funktioita, oikeilla parametreilla. Seuraavaksi vuorossa oli varsinaisen business-logiikan luominen. Rajapinnan toiminnan testaaminen on erittäin oleellista myös luontivaiheessa. Itse käytän rajapinnan testaamisessa hyvin yleistä Postman-sovellusta.

Haasteeksi muodostui oikean testidatan saaminen. Kuten jo aiemmin selitin, kyseessä on useamman palvelun integraatio, eikä käyttäjillämme ole juurikaan käytössä tämän palveluntoimittajan palvelua, joten testaaminen täytyi suorittaa vanhan muotoisella datalla. Tämä voi aiheuttaa ongelmia sitten kun siirrytään tuotantoon, mutta se täytyy vain tiedostaa ja olla valmiina tekemään muutoksia.

Koska testidata oli väärän mallista, en saanut sitä päivittymään meidän järjestelmäämme automaattisesti. Ainoa mahdollisuus oli siis lisäillä itse kaikki tarvittavat tiedot tietokantaan ja sitten testata mitä tapahtuu. Päivän päätteeksi sain kuin sainkin rajapinnan POST-endpointin palauttamaan minulle oikean työtilauksen. Vielä on kuitenkin paljon tehtävää jäljellä.

Perjantai 30.4.2021

Rajapinnasta tulee kaksi eri versiota, V1 ja V2. Tämä on yleinen rajapintaversioiden nimeämistapa. Alkuvaiheessa käytössämme on ainoastaan V1, koska yhteistyökumppanimme ei vielä tuota meille palvelua, josta saisimme tarvittavaa tietoa V2 rajapinnalle. Pyynnön tullessa rajapintaamme, sen mukana tulee muutaman muun tietueen lisäksi asiakasnumero. Tuota asiakasnumeroa käytämme, kun haemme olemassa olevia työtilauksia tietokannasta. Ongelmana tässä on se, että samalla asiakasnumerolla voi olla useampia työtilauksia, joten joudumme asiakasnumeron lisäksi tekemään vertauksia työtilauksen hintaan ja päivämäärään. Oikean työtilauksen löydyttyä tilauksen tiedot päivitetään yhteistyökumppanimme palvelusta.

Vappuaatosta valtaosan käytin tuon päivitystapahtuman implementointiin. Yrittäessäni luoda oikeaa hakuosoitetta palauttavaa funktiota, huomasin olemassa olevien funktioiden toimivan jo samankaltaisesti. Lähdin seuraamaan tuota toimintaketjua ja huomasin että sovelluksessamme on käytännössä jo valmis toteutus tälle täysin samalle tapahtumalle. Innoissani tietysti kerroin havainnostani työparilleni, jota olin pommittanut päivän mittaan kymmenillä kysymyksillä. Hänen vastauksensa: ”Olenkin tässä jo pari tuntia odotellut, että milloinkahan oikein huomaat” myötä olikin hyvä hetki painaa tietokoneen luukku kiinni ja unohtaa ohjelmointiin liittyvät asiat viikonlopuksi.

Viikkoanalyysi

Viikko oli todella haastava. Olin ajatellut isojen rajapintamuutosten olevan vielä aivan liian vaativia minun tehtäväksi, mutta helpottaa paljon tilannetta, kun voi kysyä työparilta neuvoja. Yritän parhaani mukaan sisäistää oppimiani asioita. Sain myös viikon aikana positiivista palautetta esimieheltäni. Motivoi kummasti yrittämään parhaansa, kun tähänastisetkin ponnistelut huomioidaan.

Suurimmalta osin viikko vierähti siis rajapintojen parissa. Lieneekin siis enemmän kuin sopivaa käydä hieman aihetta läpi tarkemmin viikkoanalyysin muodossa. Erilaiset integraatiot ja rajapinnat muutenkin ovat kuitenkin yksi tärkeimmistä nykypäiväisen webkehityksen kulmakivistä.

Esimerkiksi, kun käytät puhelimellasi jotain sovellusta, sovellus lähettää tietoa palvelimelle internet-yhteyden välityksellä. Palvelin vastaanottaa tuon tiedon, tulkitsee sen ja suorittaa määritetyt tehtävät ja palauttaa tiedon takaisin puhelimeesi. Puhelimesi sovellus näyttää sinulle haluamasi informaation luettavassa muodossa. Kaikki tämä tapahtuu API:n, eli rajapinnan avulla. (Mulesoft 2021.)

GraphQL-querykielen suosiosta huolimatta edelleen de facto -standardi rajapinnoista puhuttaessa on REST API. Iso osa tämän päivän rajapinnoista vähintäänkin mukailee RESTin sääntöjä, joskin täysin RESTin mukaiset rajapinnat ovat edelleen ani harvassa.

REST, eli Representational State Transfer on Roy Fieldingin kehittämä arkkitehtuurinen tyyli, jonka hän esitteli omassa väitöskirjassaan vuonna 2000. Jotta palvelua voidaan kutsua REST-palveluksi, tulee se toteuttaa kuuden ohjaavan rajoitteen mukaisesti.

REST-APIn rajoitteet:

- **Asiakas-palvelin.** RESTin ensimmäinen rajoite koskee asioiden eriyttämistä toisistaan, eriyttämällä käyttäjäliittymään liittyvät tehtävät tietokantaa muokkaavista tehtävistä, parannetaan käyttäjäliittymän siirrettävyyttä eri alustoille sekä skaalautumista yksinkertaistamalla palvelinkomponentteja. (Fielding 2000, 78.)

- **Tilattomuus.** Asiakas-palvelin välisen kommunikaation tulee olla luonnoltaan tilatonta. Jokaisen palvelimelle tulevan pyynnön tulee sisältää kaikki tarvittava tieto pyynnön ymmärtämiseksi, eikä pyyntö voi hyödyntää mitään palvelimelle tallennettua lisätietoa. (Fielding 2000, 78–79.)

- **Välimuisti.** Verkkoliikenteen optimoimiseksi otetaan käyttöön välimuisti. Pyynnön vastauksena tuleva datan tulee olla selkeästi merkitty joko välimuistitettavaksi tai ei. Mikäli vastaus voidaan välimuistittaa, annetaan asiakkaan välimuistille lupa uudelleen käyttää vastauksen dataa myöhemmin. (Fielding 2000, 79–81.)

- **Yhdenmukainen rajapinta.** Keskeisin ominaisuus, joka erottaa REST-arkkitehtuurin muista verkkopohjaisista tyyleistä, on sen painotus yhdenmukaisiin komponenttien välisiin rajapintoihin. Tämä yksinkertaistaa järjestelmäarkkitehtuuria ja komponenttien interaktiivisuuden näkyvyys paranee. (Fielding 2000, 81–82.)

- **Kerroksittainen järjestelmä.** Mahdollistaa arkkitehtuurin luomisen hierarkkisista kerroksista rajoittamalla komponentin käyttäytymistä siten, ettei mikään komponentti näe sen kerroksen taakse minkä kanssa se on kanssakäymisessä. Rajoittamalla järjestelmän tieto yhteen kerrokseen asetetaan raja koko järjestelmän monimutkaisuudelle. (Fielding 2000, 82–84.)

- **Code-On-Demand.** Ainoa vapaaehtoinen REST-arkkitehtuurin rajoite. Asiakkaalle annetaan mahdollisuus laajentaa sen toiminnallisuutta lataamalla ja suorittamalla koodia skripteinä. Näin yksinkertaistetaan asiakasjärjestelmää vähentämällä etukäteen toteutettuja toiminnallisuuksia. (Fielding 2000, 84–85.)

4.7 VIIKKO 18 – Semanttinen versiointi

Maanantai 3.5.2021

Koska edellisviikolla teimme työparini kanssa molemmat isoja muutoksia, ei maanantaille tullut uusia päivityksiä. Oma tehtäväni yhteistyökumppanillemme tarjottavan rajapinnan parissa oli vielä kesken, siitäkin huolimatta, että perjantaina huomasin voivani käyttää olemassa olevia palveluja tehtäväni logiikan teossa.

Tälle päivälle asetin tavoitteeksi saada toimintaketjun toimimaan oikein kutsuessani noita olemassa olevia funktiota. Jonkin verran niissäkin oli muokattavaa, koska niiden teosta on jo sen verran aikaa, että niille dataa toimittavat rajapinnat ovat ehtineet muuttua. Puutteellisen rajapintakuvauksen takia päivän tavoite ei täyttynyt. On yllättävän aikaa vievää puuhaa yrittää selvittää, miksei toisten tekemä koodi toimi.

Tiistai 4.5.2021

Aiemmilla viikoilla mainitsemani interface oli jälleen syypää eilisiin ongelmiin. Ajan saatossa rajapinnan tuottaja oli päättänyt vaihtaa joidenkin kenttien tyypitystä ja nimiä. Meidän järjestelmäämme tehty interface taas oli edelleen odottamassa dataa samassa muodossa, kuin se aiemmin on tullut,

joten ongelmiahan siitä tulee. Korjausten jälkeen kaikki alkoi toimimaan moitteettomasti ja sain datan haettua omalle palvelimellemme. Seuraavana päätin toteuttaa toiminnon, joka tallettaa yksilöivän tunnisteen tietokantaan siinä vaiheessa, kun yhteistyökumppanimme tekee kutsunsa rajapintaamme. Tämä vaatii pieniä muutoksia tietokannan rakenteeseen, sekä muutaman uuden funktion prosesseja käsittelevään palvelukomponenttiin.

TypeORM ei ole minulle tuttu, joten joudun jonkin verran hakemaan tietoa dokumentaatioista ja keskustelupalstoilta. Käy ilmi, ettei TypeORMin migraatiot generoidu automaattisesti oikein, vaan niitä tulee muokata ennen migraatioiden suorittamista. Lopulta aikaa tuhrautui niin paljon, että varsinaisen toiminnallisuuden koodaaminen siirtyy seuraavalle päivälle.

Keskiviikko 5.5.2021

Tietokantamuutosten valmistuttua, aloin toteuttamaan varsinaista toiminnallisuutta. Työ onnistui lopulta aika helposti, tein omat metodit sekä tunnisteen lisäämiselle että poistamiselle. Yhteistyökumppanimme tarvitsee myös mahdollisuuden lähettää DELETE-pyyntö rajapintaamme, jos kyseessä oleva maksutapahtuma onkin maksettu jo jossain muussa yhteydessä. Nimestään huolimatta poisto ei varsinaisesti poista mitään järjestelmästämme, vaan asettaa tietokannan kentän tyhjäksi, sekä muuttaa prosessin tilaksi ”maksettu muualla”.

Saatuani testattua tekemieni muutosten toimivuuden, seuraavana oli vuorossa palvelimelta saadusta datasta oikean tilauksen etsiminen ja sen muokkaaminen. Haasteita aiheutti jälleen kerran TypeORM. Tietokannasta etsimiselle on useita funktioita, ja funktioiden parametreiksi voidaan antaa erilaisia hakuehtoja, joiden perusteella TypeORM sitten palauttaa haetun tietueen. Koska kyseessä on relaatiotietokanta, jossa on useita eri tauluja, joiden välillä on suhteita, tulisi pystyä hakuja suorittamaan noiden suhteiden mukaisesti. TypeORM ei kuitenkaan suoraan tarjoa tätä mahdollisuutta, vaan sen ”where” hakuehto toimii ainoastaan haettavan päätaulun sisällä, eikä sillä voida hakea tietoa relaation mukaan. Loppupäivä menikin sitten tämän ongelman parissa.

Torstai 6.5.2021

Päivän tavoitteena oli saada alustava yhdistämispyyntö työparini katselmoitavaksi. Alustava, koska missään nimessä työ ei tule valmiiksi tämän päivän aikana, mutta on kuitenkin hyvä ottaa väliaikatarkistus, jossa työparini kokeneempana kehittäjänä käy läpi valitsemiani ratkaisuja.

Tarkoitus olisi saada V1-rajapinta siis sille mallille, että työparini voi testata sen toimintaa testidatalla. Lisäsin logiikkaani ehtolausekkeita, jotka palauttavat erilaisia virheitä. Esimerkiksi jos yhteistyökumppanimme tekee pyynnön, jonka mukana tulevaa asiakasnumeroa ei löydy tietokannastamme, annetaan virheenä http-status 404 ja virheviesti ”Asiakasta ei löydy”. Tämän lisäksi tein lopulliset datan muokkaukseen liittyvät toiminnallisuudet ja siirsin haaran versionhallintaan.

Perjantai 7.5.2021

Koska tekemäni työ oli nyt työparini tarkasteltavana, oli minulla aikaa paneutua normaaleihin viikoittaisiin toimiin, jotka olivat jääneet laiminlyödyksi. Aloitin päivittämällä riippuvuudet, joka osoittautuikin lopulta paljon isommaksi työksi kuin ajattelin. Erääseen meidänkin sovelluksessamme isossa osassa olevaan pakettiin oli tullut niin sanottu Major-päivitys, eli päivitys, jonka yhteensopivuutta ei voida taata taaksepäin.

Otti huomattavan määrän aikaani tutkia mihin tuon päivityksen vaikutus ylettyy sovelluksessamme. Muutaman tunnin selvitystyön jälkeen tulimme työparini kanssa siihen tulokseen, ettemme voi päivittää tätä pakettia vielä. Se aiheuttaa liikaa rikkovia muutoksia, joiden korjaamiseen meillä ei juuri nyt ole aikaa. Päivittelin muut paketit ja loppupäivän kulutin työparini lähettämien korjauskehotuksien parissa.

Viikkoanalyysi

Viikko meni todella nopeasti. Tuntuu hieman hölmöltä huomata, kuinka pienten asioiden kanssa tässä ammatissa voi tuhrautua pitkiäkin aikoja. Sain työpariltani hyvää palautetta työni jäljestä ja se todellakin valoi uskoa toisinaan huijarisyndroomasta kärsivään mieleeni. Haasteita aiheutti ennen kaikkea tietokannan ja logiikan välillä toimiva TypeORM.

Opin edelleenkin paljon uusia asioita ja etenkin uusia tapoja, jotka nopeuttavat työtäni. Viikon haastavimpana tapauksena pidin, ehkä hieman yllättäen, päivityksen estänyttä pakettia. On sinänsä nerokasta, että semanttisen versioinnin ansioista rikkovat muutokset voidaan huomata nopeasti, ennen kuin päivitys tehdään. Asiaa tutkiessani päätin viikkoanalyysissa käyttää aiheen pureskeluun hieman enemmän aikaani.

Suomen kielessä sana: ”päivitys” tarkoittaa kahta toisaalta hyvinkin eriä asiaa. Englanniksi tarkoittamani termit olisivat: ”Update” sekä ”Upgrade”. Jälkimmäinen suomentuu sanaksi: ”parannus”, jota ei kyllä ainakaan ohjelmistoalalla käytetä. Parannuksesta kuitenkin yleensä on kysymys, mikäli päivitys on sellainen, jossa tulee uusia ominaisuuksia.

Semanttinen versiointi perustuu kolmeen eri tasoiseen päivitykseen. ”Major” on kyseessä silloin, kun tehdään taaksepäin yhteensopimattomia niin sanottuja rikkovia muutoksia rajapintaan. ”Minor” päivityksessä lisätään toiminnallisuuksia, mutta säilytetään taaksepäin yhteensopivuus. ”PATCH” on käytössä, kun tehdään vikakorjauksia, jotka eivät vaikuta yhteensopivuuteen. (Preston-Werner 2011.)

Järjestelmissä, joissa on käytössä useita riippuvuuksia, uusien versioiden julkaisemisesta voi tulla hankalaa. Mikäli riippuvuusmäärittelyt ovat turhan tiukat, on uhkana versiolukko(uuden version voi julkaista vain, kun on päivittänyt kaikki riippuvuudet). Mikäli taas määrittelyt ovat turhan löysät, se johtaa versioiden umpimähkäisyyteen. Silloin, kun jompikumpi näistä estää turvallisen sovelluksen päivittämisen puhutaan tilasta nimeltään riippuvuushelvetti. (Preston-Werner 2011.)

Semanttisen versioinnin määrittely

1. Ohjelmiston, joka käyttää semanttista versiointia, tulee määrittää julkinen rajapinta. Tämä rajapinta voidaan määritellä koodissa, tai se voi esiintyä pelkästään dokumentoinnissa.
2. Normaalin versiointinumeron täytyy olla muotoa X.Y.Z. X vastaa Major-versionumeroa, Y vastaa Minor-versionumeroa ja Z on Patch-versionumero. Jokaisen elementin tulee kasvaa numeerisesti yhdellä. Esimerkiksi versio 1.9.0 saadessaan Minor-päivityksen muuttuu versionumeroon 1.10.0.
3. Aina, kun Major-versionumeron arvo nousee, nollataan Minor ja Patch. Aina, kun Minor-versionumeron arvo nousee, nollataan Patch.
4. Esijulkaisun versionumero voidaan merkitä liittämällä satunnainen merkkijono välittömästi Patch-luvun jälkeen viivalla erotettuna. Merkkijonon tulee koostua aakkosnumeerisista numeroista viivan lisäksi.
5. Version julkaisun jälkeen version sisältöä ei tule muokata. Kaikki muokkaukset tulee tehdä uusina julkaisuina.
6. Major-versionumeron arvo nolla on tarkoitettu alkukehitykselle. Mikä tahansa voi muuttua milloin vain, eikä julkista rajapintaa voida pitää vakaana.

7. Versio 1.0.0 määrittää julkisen rajapinnan. Se miten versionumeroa tämän jälkeen kasvatetaan, on riippuvainen julkisen rajapinnan muutoksista.
8. Patch-versionumeroa tulee kasvattaa ainoastaan, mikäli tehdään vikakorjauksia, jotka ovat taaksepäin yhteensopivia. Viankorjaus määritellään sisäisenä muutoksena, joka korjaa sovelluksen epätoivottavaa käyttäytymistä.
9. Minor-versionumeroa tulee kasvattaa, mikäli julkiseen rajapintaan tuodaan uusi ominaisuus, joka on taaksepäin yhteensopiva. Sitä voidaan kasvattaa, jos uusi toiminnallisuus tai parannus koskee vain sisäistä toiminnallisuutta. Se voi sisältää Patch-tason muutoksia. Patch-versionumero tulee asettaa nollaan, kun Minor-versionumeroa kasvatetaan.
10. Major-versionumeroa kasvatetaan, mikäli julkiseen rajapintaan tulee taaksepäin yhteensopimattomia muutoksia. Se voi sisältää Minor- sekä Patch-tason muutoksia. Minor- sekä Patch-versionumerot tulee asettaa nollaan, kun Major-versionumeroa kasvatetaan. (Preston-Werner 2011.)

4.8 VIIKKO 19– Asynkroninen JavaScript

Maanantai 10.5.2021

Aamulla sähköposteja lukiessani huomasin, että työparini oli löytänyt edellisviikolla tekemästäni rajapintamuutoksesta jo reilusti korjailtavaa. Lähempi tarkastelu versionhallinnassa osoitti, että kyseessä on enimmäkseen koodin muotoiluun liittyviä korjauksia. Työparini onkin kertonut tarkastelevansa koodiani erityisen tarkalla silmällä myös muotoilun kannalta, jotta vääriä toimintatapoja ei pääse muodostumaan. On omastakin mielestäni paljon helpompaa puuttua tällaisiin asioihin tässä vaiheessa, kuin yrittää korjata toimintatapojaan vuosien jälkeen.

Saatuani muotoiluvirheet korjattua aloin toteuttamaan rajapinnan V2-versiota. Sen pitäisi olla huomattavasti suoraviivaisempi, koska siinä tiedon hakemiseen voidaan aina käyttää yksilöivää työtilausnumeroa. Olen toisaalta onnellinen kaikista edellisviikoilla kohtaamistani haasteista, koska niiden myötä tällaiset tehtävät ovat suorastaan helppoja. Sain tehtävän lyhyessä ajassa jo lähes valmiiksi. Seuraavalle päivälle jäi lähinnä koodin siistiminen ja kattavampi testaaminen.

Tiistai 11.5.2021

Päivän tavoitteena oli saada rajapinnan V2-version koodit siistittyä ja toiminta testattua. Siistimisellä tarkoitan tässä yhteydessä koodin läpikäymistä muotoiluvirheiden varalta. Näitä ovat muun muassa sisennysvirheet, rivin päättävän puolipisteen puuttuminen, turhat lokituskomennot, muuttujien ja funktioiden epäselvät nimet ja muut koodiin kuulumattomat asiat.

Testaamisen yhteydessä huomasin ongelmia, jotka selvästi aiheutuivat eri palvelukomponenttien asynkronisista funktiokutsuista. Asynkronisen JavaScriptin ymmärtäminen on yksi omista puutteistani, johon minun tulee jatkossa perehtyä selvästi enemmän. Puutteistani johtuen vian selvittämiseen kului tolkuton määrä aikaa. Sain kuitenkin lopulta vian korjattua ja siirrettyä muutokseni versionhallintaan.

Keskiviikko 12.5.2021

Yksi kohta rajapintatehtävässäni oli jäänyt vaivaamaan minua. Prosessikuvauksen mukaan meidän pitäisi toimittaa maksunvälittäjälle yksilöllinen maksuviite, joka meidän tulisi saada yhteistyökumppanimme lähettämän pyynnön mukana. Kuitenkin viime aikoina kaikki palaverit ja sähköpostit ovat keskittyneet asiakasnumeron ja työtilausnumeron muotoseikkoihin, eikä minulla ollut täyttä selkeyttä missä yhteydessä ja muodossa tuo maksuviitenumero pyynnön mukana tulisi. Päätinkin ottaa suoraan yhteyttä yhteistyökumppanimme kehittäjiin asian tiimoilta.

Heiltä saamani vastaus oli karmeaa luettavaa. Pahoiteltuaan sitä, että he olivat virheellisesti puhuneet rajapintaversiosta riippuen joko asiakas- tai työtilausnumerosta, he kertoivat välittävänsä meille pyynnön mukana ainoastaan tuon viitenumeron eikä heillä ole edes pääsyä asiakasnumeroon. Tämä tarkoittaa käytännössä sitä, että työtilauksen yksilöinti muuttuu lähes mahdottomaksi, eikä parin viimeisen viikon aikana tekemilläni rajapinnan logiikoilla ole mitään käyttöä. Maksuviitettä, joka pyynnön mukana tulee, ei voi vielä siinä vaiheessa löytyä tilauksesta meidän tietokannassamme, joten sillä ei voida yksilöintiä suorittaa.

Olin lisännyt sähköpostiviestieni kopiokenttään työparini, projektipäällikkömme sekä asiakkaamme ja pidimmekin pikaisella tahdilla hätäpalaverin aiheen tiimoilta. Päätimme laittaa tämän rajapintamuutoksen jäihin siksi aikaa, kunnes asia selkeytyisi. Koko päivä olikin kulunut jo asiaa selvitettäessä, joten päätin valita seuraavan tehtäväni vasta huomenna.

Torstai 13.5.2021

Helatorstai, vapaapäivä.

Perjantai 14.5.2021

Sovelluksemme käyttäjät jakautuvat kahden eri toiminnanohjausjärjestelmän välillä. Olemassa olevat integraatiot noiden ja meidän järjestelmämme välillä ovatkin ajan saatossa todettu luotettaviksi ja uusien toimintojen luominen niiden välillä on suhteellisen suoraviivaista. Kyseisellä alalla toimii kuitenkin myös yrityksiä, joilla on käytössään kolmas suurehko toiminnanohjausjärjestelmä, johon meidän järjestelmästämme ei vielä ole toiminnassa olevaa yhteyttä. Asiakkaamme pyynnöstä sovelluksemme aiemmat kehittäjät ovat luoneet valmiudet mahdolliseen tulevaan integraatioon, mutta heillä ei ole ollut käytössään tarkkoja rajapintakuvauksia, joten toteutus on suurilta osin kesken tai tehty sen varassa mitä kyseessä olevan toimittajan verkkosivuilta on saatu irti.

Asiakkaamme ilmoittikin meille nyt saaneensa myytyä sovelluksemme uudelle käyttäjälle, jolla on käytössään tuo toiminnanohjausjärjestelmä ja hän oli saanut jo toimittajalta tunnukset ja rajapintakuvaukset integraation kuntoon saattamiseksi. Päätinkin aloittaa tutkimalla olemassa olevaa toteutusta ja vertailemalla sitä saatuun kuvaukseen. Melko pian sain huomata, että joudun tekemään monia osia kokonaan uusiksi, mutta isolta osin toteutus on toimiva. Useat sovelluksemme luokat odottivat datan tulevan eri muotoisena, joten jouduin muokkaamaan datarakenteet vastaamaan palvelimelta tulevaa dataa ja käyttämiemme datatulkkien kääntämään saatu data siihen muotoon missä meidän järjestelmämme sitä käyttää. Pikaisten testien perusteella muutokset onnistuivat helposti.

Päivän tavoitteena oli saada oikea asiakkaan data siirtymään järjestelmien välillä. On todella tärkeää testata järjestelmien toimivuus oikealla käytössä olevalla datalla. Toiminnanohjausjärjestelmän rajapinta on toteutettu hyvin ja sen suunnittelussa on selvästi käytetty REST-arkkitehtuuria. Ensimmäisenä meidän tulee suorittaa autentikointipyyntö, jossa välitämme heidän rajapintaansa käyttäjätiedot. Vastaukseksi oikeanmuotoiseen pyyntöön saamme autentikointiavaimen, jonka sitten liitämme kaikkiin rajapintaan tekemiimme pyyntöihin.

Testailin aluksi rajapinnan toimintaa Postman-sovelluksella ja saatuani yhteyden toimimaan ja pyyntöjen tuottavan oikeaa dataa, olikin helppo kopioida tekemäni pyynnöt koodiimme. Melko pian

sainkin sovelluksemme vastaanottamaan tietoja rajapinnasta. Jouduin vielä jonkin verran muokkaamaan tietokantaehtojamme, jotta kaikki tilaukset saatiin talteen.

Viikkoanalyysi

Vaikka parin edellisviikon aikana tekemäni isohko rajapintatyö osoittautui tässä vaiheessa turhaksi, en voi sanoa sen olleen sitä oman kehittymiseni kannalta. Huomaan osaamiseni Back End -kehityksessä kasvaneen roimasti ja ymmärrän viikko viikolta paremmin, miksi asioita tehdään niin kuin niitä tehdään. Teknisten asioiden lisäksi tämä viikko opetti myös ihmisten välisen kommunikoinnin tärkeydestä. Aionkin entistä herkemmin kysyä asioista, jotka minua häiritsevät.

Viikon aikana oli niin useita haasteellisia tilanteita, että oli jo hieman vaikeaa päättää minkä niistä valitsen viikkoanalyysin aiheeksi. Päätin kuitenkin valita Asynkronisen JavaScriptin, koska sen kanssa taistellessa kului viikolla ylivoimaisesti eniten aikaa.

Synkronisissa operaatioissa tehtäviä ajetaan yksi kerrallaan, ja vasta kun tehtävä tulee valmiiksi, voidaan siirtyä seuraavaan. Toisin sanoen, joudut odottamaan tehtävän valmistumista päästäksesi seuraavaan tehtävään. Asynkronisissa operaatioissa voit siirtyä seuraavaan tehtävään jo ennen kuin edellinen valmistuu. Asynkronisella ohjelmoinnilla voidaan siis käsitellä useita pyyntöjä samanaikaisesti, ja näin ollen suorittaa enemmän tehtäviä lyhyemmässä ajassa. (Costa 2021.)

JavaScript on yksisäikeinen kieli. Tämä tarkoittaa sitä, että se ei lähtökohtaisesti voi suoriutua kuin yhdestä käskystä kerrallaan. Tämä tuo ongelmia muun muassa siinä vaiheessa, kun joudutaan esimerkiksi tekemään kallis HTTP-pyyntö ulkoiseen rajapintaan. Rajapinnasta tulevaa dataa voidaan joutua odottamaan kauankin, ja lähtökohtaisesti JavaScript keskeyttää suorittamisen, kunnes saa rajapinnalta vastauksen ja tehtävässä voidaan edetä. Pahimmillaan tämä voi lamauttaa selaimen pitkäksi ajaksi. Tästä syystä JavaScriptiin on luotu keinoja kirjoittaa sitä asynkronisesti.

Pohjimmiltaan asynkroninen ohjelmointi JavaScriptissä on toteutettu takaisinkutsufunktioilla. Takaisinkutsufunktio on funktio, jonka kirjoitat ja välität parametrina toiselle funktiolle. Tuo toinen funktio sitten kutsuu funktiotasi, kun jokin ehto täyttyy, tai jokin asynkroninen tapahtuma ilmenee. Takaisinkutsufunktion kutsuminen ilmoittaa tuosta tapahtumasta tai ehdosta, ja toisinaan kutsu pitää sisällään argumentteja, jotka toimittavat lisätietoja. (Flanagan 2020, Luku 13.1.)

Takaisinkutsufunktioiden jälkeen seuraava kehitysaskel olivat lupaukset (Promises). Lupaukset suunniteltiin helpottamaan asynkronista ohjelmointia JavaScriptillä. Takaisinkutsufunktioiden yhtenä suurimpana ongelmana pidetään joutumista niin sanottuun takaisinkutsuhelvettiin, jossa useita takaisinkutsufunktioita on sisäkkäin, ja näin ollen koodia joudutaan sisentämään niin paljon, että se muuttuu vaikealukuiseksi. Lupaukset mahdollistavat tällaisen sisäkkäisen takaisinkutsunnan kirjoittamisen lineaarisesti ketjutetuilla lupauksilla. Tämä tekee koodista helppolukuisemman ja helpommin ymmärrettävän. Lupaus on olio, joka kuvaa asynkronisen tapahtuman tulosta. Tuo tulos voi olla joko valmis tai kesken, ja Promise-API on tarkoituksella salamyhkäinen asian suhteen. Ei ole mitään keinoa saada lupauksen arvoa synkronisesti. Voit ainoastaan pyytää lupausta kutsumaan takaisinkutsufunktiota, kun sen arvo on valmis. (Flanagan 2020, Luku 13.2.)

JavaScriptin versio ES2017 esitteli kaksi uutta avainsanaa, jotka kuvaavat paradigman muutosta asynkronisessa JavaScript-ohjelmoinnissa. Nuo avainsanat, async ja await, yksinkertaistavat lupausten käyttämistä dramaattisesti, ja mahdollistavat lupauksiin perustuvan asynkronisen koodin, joka näyttää synkroniselta ja keskeyttää suorituksen odottaessaan esimerkiksi verkkopyyntöjä tai muita asynkronisia tapahtumia. (Flanagan 2020, Luku 13.3.)

```
        // Takaisinkutsu(Callback)
getProcess(function (process) {
    getUser(process, function (user) {
        getAccount(user, function (acc) {
            getWorkOrder(acc, function (workorder) {
                sendStatistics(workorder, function (e) {
                    console.log("Something")
                });
            });
        });
    });
});
```

```
    // Async / Await
await getAll([
    getProcess,
    getUser,
    getAccount,
    getWorkOrder,
    sendStatistics,
], () => console.log("Completed")
).catch((error) => consolerror.error(error));
```

```
    // Lupaus(Promises)
getProcess()
    .then(getUser)
    .then(getAccount)
    .then(getWorkOrder)
    .then(sendStatistics)
    .then((success) => console.log(success))
    .catch((erro) => console.error(error))
```

KUVIO 3. Esimerkit asynkronisista funktiokutsuista JavaScriptissä

4.9 VIIKKO 20– Ketterä kehitys

Maanantai 17.5.2021

Viikko alkoi jatkamalla edellisviikon integraatiota. Tavoitteena olisi saada muutoksesta yhdistämispyyntö työparini katseltavaksi. Aamupalaverissa kävi ilmi, että työparini alkaa hiljalleen siirtyä muihin projekteihin, joten tämän sovelluksen kehittäminen jää enenevissä määrin minun vastuulleni. Saan toki apua muulta tiimiltä ja työparinikin on luvannut auttaa aina tarvittaessa.

Muutaman virhetilanteen selviteltyäni, sovelluksen muutokset olivat valmiita ja testauskin osoittautui onnistuneeksi, joten siirsin muutokseni versionhallintaan katseltavaksi. Loppupäiväksi meillä olikin tiedossa koulutusta, joten en ottanut enää uusia tehtäviä työn alle.

Tiistai 18.5.2021

Jälleen työparini oli löytänyt edellispäivän työstäni muotoiluvirheitä. Lähinnä ylimääräisiä rivinvaihtoja ja muutamia puuttuvia puolipisteitä. Yritin kyllä tehdä erittäin tarkkaa työtä, mutta on vaikea huomata kaikkea. Päivän tavoite noiden virheiden korjaamisen lisäksi oli katselmoida työparini tekemä iso lisäominaisuus. Kyseessä on sovelluksen historian lokitus, jota asiakkaamme on pyytänyt jo hetken aikaa. Se on kuitenkin sen verran massiivinen lisäominaisuus, että ennen refaktorointien valmistumista emme voineet sitä aloittaa.

Saadakseen toiminnon helposti lisättyä, työparini otti sovelluksessamme käyttöön NgRx-lisäosan, joka tuo Redux-tyylisen tilanhallinnan myös Angulariin. NgRx ei ole itselleni entuudestaan tuttu, joten katselmoinnin lisäksi pyrin kyselemään työpariltani mahdollisimman paljon asioita hänen koodistaan ymmärtääkseni sen paremmin. Löysin koodista myös muutaman muotoiluvirheen, jotka ilmoitin hänelle korjausehdotuksen muodossa. Hänen korjattuaan ne, hyväksyin yhdistämispyynnön. Työparini kanssa tulimme siihen lopputulokseen, että ominaisuuden olisi parempi olla oletuksena pois päältä. Sovimme, että alan toteuttamaan muutosta huomenna.

Keskiviikko 19.5.2021

Aloitin edellispäivänä sovitun muutostyön teon Front End -puolelta. Lisäsin Admin-käyttäjän asetuksiin yksinkertaisen valintaruudun, jonka valitsemalla käyttäjä saa historialokituksen päälle. Muutos oli suhteellisen suoraviivainen, komponenttitasolla se vaati vain yhden uuden boolean-muuttujan, jonka lisäsin käyttäjän asetuksia ylläpitävään luokkaan, sekä lomakkeelle, jolla asetuksia muutetaan. Back End -koodiin lisäsin myös samannimisen muuttujan, joka tallennetaan tietokantaan käyttäjän asetukset sisältävään tauluun.

Tässä vaiheessa muutos toimi jo niin, että muuttujan arvo tietokannassa muuttui sen mukaisesti missä asennossa valintaruutu oli. Tein lisäksi funktion, jonka avulla voidaan tarkistaa tilin asetuksista, mikäli historian lokitus on käytössä. Lisäsin tämän tarkistuksen tapahtumaan ennen historian lokitusta käynnistävää funktiota, ja muutos olikin testauksia vaille valmis. Sain muutoksen valmiiksi ennen päivän päättymistä ja ilmoitin työparilleni sen olevan katselmointia vailla.

Torstai 20.5.2021

Pidimme aamulla asiakkaan kanssa pitkäksi venyneen palaverin, jossa käsiteltiin aiemmin tekemääni rajapintamuutosta, joka osoittautui turhaksi. Asiakas oli saanut varmistuksen, ettei meidän ole mahdollista saada yhteistyökumppaniltamme tilausta yksilöivää tietoa, joten yksilöinti tulisi toteuttaa muilla keinoin. Tämä toteutus kuitenkin siirtyy tässä vaiheessa myöhäisemmäksi.

Sovelluksestamme tehdään erääseen toiseen palveluun varauksia tilauksen yhteydessä. Asiakas oli huomannut, että kun tuota varauksen aikaa muokataan, se päivittyy meidän järjestelmäämme vasta viiden minuutin jälkeen. Tämä on käyttäjän kannalta ikävää, koska hänelle ei edes ilmoiteta päivityksen kestävän hetken, joten käyttäjä tuppaa tekemään muokkauksen uudelleen.

Sovimme minun ottavan tämän asian tutkittavakseni, ja sen parissa sainkin päivän kulumaan. Pääsin jo vian jäljille, mutta ihan selvää ei vielä ole kuinka tulen asian ratkaisemaan. Sovimme työparini kanssa ottavamme huomenna yhteisen tuumaushetken asian tiimoilta.

Perjantai 21.5.2021

Työparini joutui taas muihin töihin perjantaiksi, joten siirsimme tuumaushetken maanantaille. Sovinkin projektipäällikkömme kanssa, että hoitaisin perinteiset päivitystoimet ja pitäisin sen jälkeen

niin sanotun Lab-päivän. Yrityksemme haluaa tukea työntekijöiden kehittymistä, ja joka kuukausi on mahdollista käyttää yksi päivä uuden opetteluun, mikäli projektin tilanne niin sallii.

Angularista on julkaistu uusi Major-versio, joten on hyvä selvittää millaisia muutoksia se tulee vaatimaan. Päätinkin kuluttaa päivän Angularin uusia ominaisuuksia tutkien. Samassa tuli sivuttua myös vanhempia asioita, jotka ovat olleet hieman epäselviä ja koinkin muutaman valaistumisen sen ansiosta.

Viikkoanalyysi

Kuultuani, että työparini siirtyy hiljalleen pois projektistamme, ensimmäinen reaktio oli luonnollisesti järkytys. Kuitenkin asiaa hiljan tuumattuani huomasin tekeväni työtäni itsenäisesti suurimmilta osin jo nyt. Päätinkin nähdä tämän enemmän mahdollisuutena todellakin näyttää kynteni. Uskon myös, että tämä on jonkin asteinen luottamuksen osoitus. Tuskin projektia minun käsiini uskottaisiin, jos en olisi osoittanut pystyväni sitä hoitamaan.

Viikon haastavimmat hetket koin lähinnä katselmoidessani työparini tekemää lokituksen mahdollistavaa ominaisuutta. Tuntui hieman hölmöltä etsiä siitä virheitä, kun en juurikaan ymmärtänyt sen toimintaperiaatteitakaan. Onneksi työparini samalla tutustutti minua NgRx-teknologian perusperiaatteisiin. Samalla koinkin myös viikon opettavimmat hetket.

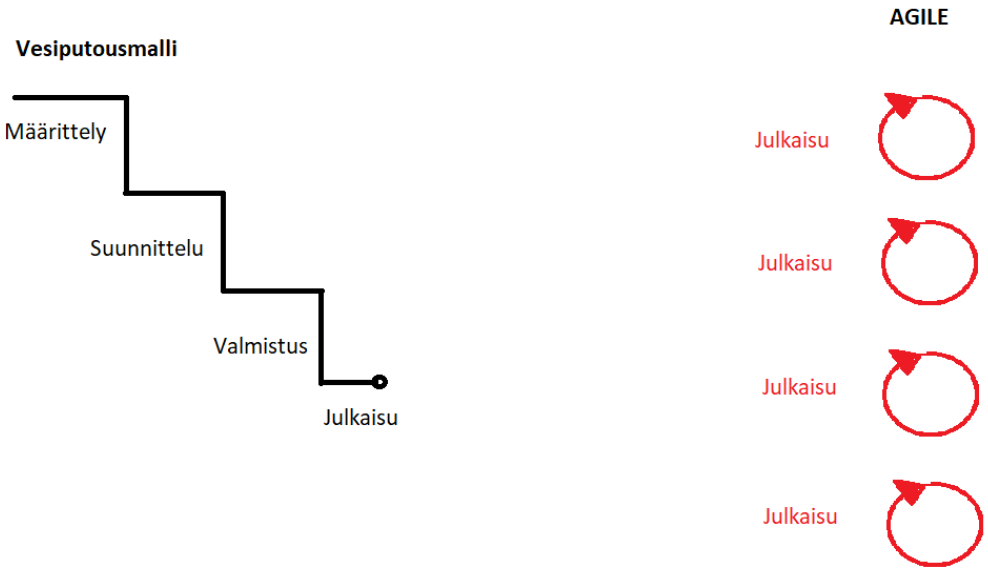
Maanantainen koulutuksemme sivusi ketteriä kehitysmenetelmiä ja mietinkin, että voisinpa tutkia asiaa hieman enemmänkin. Se on yksi niin sanotuista kuumista aiheista ohjelmistokehityksen ympärillä. Tutkimusmielessä kävin läpi avoimia ohjelmistokehittäjän työpaikkoja, eikä kovin montaa ilmoitusta löytynyt, jossa ei olisi jossain määrin odotettu hakijalta ymmärrystä ketteristä menetelmistä.

Ketterän kehityksen voi sanoa saaneen alkunsa ketterän ohjelmistokehityksen julistuksesta. Julistuksen antoivat seitsemäntoista ohjelmistokehittäjää, jotka kokoontuivat vuonna 2001 keskustellakseen ohjelmistokehityksen silloisesta nykytilasta. Heidän mielestään silloinen dokumentointivoittoinen ja raskas ohjelmistokehitysprosessi kaipasi muutosta. (Highsmith 2001.)

Ketteriä menetelmiä verrataan usein perinteiseen vesiputousmalliin. Vesiputousmallissa, jokaisen tuotteen tai projektin kehityksen vaiheen toteuttaa eri tiimi, jolla on oma hyvin yksityiskohtainen

taito. Esimerkiksi tuotteen alkuperäisen suunnitelman voi tehdä tiimi, joka koostuu kyseisen toimialan eksperteistä. He sitten antavat tuon suunnitelman toiselle tiimille, joka on vastuussa tuotteen suunnittelusta. Tuo suunnitelma taas sitten toimitetaan tiimille, joka on vastuussa tuotteen valmistamisesta. Yleensä kuluu kuukausia, tai jopa vuosia, ennen kuin mitään saadaan valmiiksi, mutta ainakin periaatteessa valmiin tuotteen pitäisi olla vaatimustenmukainen. (LeMay 2018, luku 1.)

Ketterä lähestymistapa taas perustuu monialaiseen tiimiin, joka julkaisee pienempiä valmiita tuotoksia lyhyemmässä ajassa. Monialaisuus yleensä tarkoittaa tiimiä, joka koostuu kaikkien niiden alojen ammattilaisista, joita tarvitaan projektin loppuun saattamiseksi. Tiimi työskentelee yhdessä saadakseen valmiita pienempiä osia rajallisen ja säännönmukaisen aikataulun sisällä. Jokaisen aikataulutetun osion tuotos julkaistaan sille tarkoitetulle yleisölle, jonka palautteen perusteella ohjataan ja priorisoidaan tulevien aikataulujen tuotoksia. Näin asiakkaalle voidaan tuottaa arvoa nopeasti, mutta valmis tuote on yleensä muuttunut alkuperäisestä suunnitelmasta. (LeMay 2018, luku 1.)



KUVIO 4. Perinteisen vesiputousmallin vertaaminen Agileen

Yleisimmät ketterän kehityksen metodologiat:

Scrum: Epäilemättä kaikista käytetyin ketterän kehityksen metodologia. Scrum rakentuu sykleistä tai kehityksen tasoista, joita kutsutaan sprinteiksi ja joiden aikana pyritään maksimoimaan ohjelmistotuotteen kehitysaika. Sitä käytetään yleensä projektin hallintaan sekä kehitykseen, mutta sitä voidaan myös käyttää liiketoimintakontekstissa. Joka päivä järjestetään noin viisitoista minuuttia kestävä kokous, jonka tarkoitus on synkronoida tehtävät ja löytää parhaat tavat työpäivän suunnitteluun. (Lamelas 2018.)

Kanban: Kanban tulee Japanista, ja sen tarkoitus on liittää käsite ”Just-in-Time” käytäntöön. Metodi koostuu Kanban-pöydästä, joka on jaettu kolumneihin, jotka kuvaavat jokaista ohjelmistotuotannon flowta. Kun kehitys kehittyy, pöydällä sijaitseva tieto muuttuu, ja uusille tehtäville avataan uusi Kanban-kortti. Kanban-metodi vaatii kommunikaatiota ja läpinäkyvyyttä, jotta tiimin jäsenet tietävät tarkkaan missä vaiheessa kehitys on. (Lamelas 2018.)

Extreme Programming: Korostaa arvoja kuten kommunikointi, yksinkertaisuus, palaute, rohkeus ja kunnioitus. Priorisoi asiakkaan tyytyväisyyden kaiken yli. Metodologia kehottaa hyväksymään asiakkaan haluamat muutokset, vaikka ne tulisivat kehityssyklin loppuvaiheilla. Tiimityö on erittäin tärkeää, koska ongelmat ratkaistaan koko esimiehistä, kehittäjistä ja asiakkaista koostuvan tiimin kesken. Ohjelmistoa testataan ensimmäisestä päivästä alkaen, keräten palautetta kehityksen parantamiseksi. (Lamelas 2018.)

4.10 VIIKKO 21– Algoritmien tehokkuus

Maanantai 24.5.2021

Kävimme aamulla työparini kanssa läpi löytämääni vikaa, ja hänellä olikin heti ehdotus, kuinka vika tulisi korjata. Hän oli myös löytänyt edellisviikolla tekemästäni työstä bugin ja otinkin päivän tavoitteeksi saada molemmat viat korjattua. Emme ole tehneet uutta päivitystä tuotantoon muutamaan viikkoon, joten halusin nämä molemmat muutokset mukaan uuteen päivitykseen, joka tehdään tällä viikolla.

Jouduin hieman venyttämään päivää pitemmäksi, mutta sain kuin sainkin molemmat viat korjattua. Jälleen kerran molemmat viat olivat varsin opettavaisia enkä usko samaa virhettä enää tekeväni. Erittäin hyvä päivä siis oppimisen kannalta.

Tiistai 25.5.2021

Asiakkaamme oli lähettänyt edellisiltana sähköpostin, jossa pyysi avaamaan tilin uudelle käyttäjälle. Uuden tilin luominen on sinänsä nopea homma, mutta se vaatii muutamalta yhteistyökumppanilta API-avaimen saannin, jotta sovellus saadaan toimintaan. Aamupäivä menikin pitkälti tilin luomisessa ja API-avaimia odottaessa.

Itseäni oli jo pitkään häirinnyt eräs tietokantaan haun tekevä funktio, jonka olin tehnyt muutama viikko sitten. Haku perustuu useisiin eri ORM-metodien ketjutuksiin ja on suorituskyvyltään luvattoman surkea. Kyseinen haku voitaisiin kirjoittaa SQL-kielellä huomattavasti helpommalla ja tehokkaampana. Päätinkin alkaa verestelemään muistiani puhtaiden SQL-hakujen osalta. Päädyin lopulta käyttämään TypeORMin omaa Querybuilderia, jolla voi helposti rakentaa haluamansa SQL-lausekkeen. Tämä oli refaktorointia parhaimmillaan. Funktion toiminnallisuus pysyi täysin samana, mutta suorituskyky kasvoi rutkasti. Suorituskykyongelma olisi tulevaisuudessa vielä kertaantunut sovelluksen käyttäjien määrän kasvaessa.

Keskiviikko 26.5.2021

Käyttäjälle näkyvän historialokituksen lisäksi sovelluksemme Back End -koodi tuottaa lokia useista päätoiminnoistaan. Tämä loki on luettavissa Kubernetes-konttirekisterin lokitiedoissa. Tiedot, jotka lokiin päätyvät, ovat meidän itsemme päättämiä. Kirjoittamalla lokituskomentoja koodiin, saadaan suorituksen aikainen lokitus tallennettua halutulla tavalla. Virhetilanteissa nämä lokitiedot ovat kultaakin kalliimpia, joten päätin aloittaa tarkastelemaan, miten voisin parantaa niiden informaatioarvoa entisestään. Tarkoitukseni on lisäillä lokituskomentoja sinne missä niitä ei vielä ole, ja parannella olemassa olevia lokituksia. Tämä työ on kuitenkin prioriteettiasteikolla alhainen, joten mikäli aamupalaverissamme tulee tietoon uusia tehtäviä, todennäköisesti siirryn niiden pariin.

Aamupalaverissa asiakkaamme esitteli ideansa, että miten saisimme parin viikon takaisen turhaksi osoittautuneen rajapinnan toimivaksi niin, että työtilaus saataisiin varmasti yksilöityä. Totesimme työparini kanssa idean toimivaksi, joskin se vaatii täydellisen erilaisen tulokulman, kuin millä sitä jo

aloitimme. Tulevassa versiossa sysäämme yksilöinnin palvelun käyttäjän vastuulle, joten muutoksia tulee tehdä niin palvelin- kuin selainpuolen koodeihin. Kirjoitin prosessikuvauksen itselleni puhtaaksi ja aloitin muutokset luomalla tietokantaan tarvittavat uudet taulut ja tietueet. Työ on vaativa, joten en aseta itselleni sen kummemmin tavoitteita. Päivän päätteeksi olin saanut tietokantamuutokset tehtyä, sekä toiminnallisuuden tietokantaan tallentamiselle. Tästä on hyvä jatkaa huomenna.

Torstai 27.5.2021

Päätimme heti aamusta tehdä kaikki valmiiksi tulevaa tuotanto- ja testiympäristön versiopäivitystä varten. Viimeistelin muutoslokit ja tein uudet versiotagit, kun työparini sai viimeiset yhdistämispyynnöt katselmoitua.

Seuraavalla viikolla on useampi uusi käyttöönotto, joten on hyvä testata, että kaikki varmasti toimii moitteettomasti. Ajoinkin testiympäristöön uuden version heti, kun se oli mahdollista. Sekä asiakkaamme että työparini aikoivat testata uusia toimintoja, joten itse päätin syventyä edellispäivänä aloittamaani rajapintatehtävään. Tänään haluaisin saattaa Back End -koodin niin valmiiksi kuin mahdollista, jotta voin huomenna alkaa tekemään selainpuolen muutoksia.

Ongelmat tietokantamigraatioiden kanssa viivästyttivät kuitenkin sen verran, että palvelinpuolen muutosten kanssa jatkan vielä huomennakin. Monesti tuntuu, että käytössä olevat ohjelmistot eivät toimi halutulla tavalla, mutta yleensä lopuksi paljastuu, että suurin ongelman aiheuttaja sijaitsee penkin ja tietokoneen välissä.

Perjantai 28.5.2021

Päivä alkoi edellispäivän ongelmien parissa. Tarkoituksenani oli tehdä vain puolikas työpäivä, joten en aio aloittaa selainpuolen muutoksia tänään. Päätin käyttää aikani testailemalla rajapintaa monilla eri skenaarioilla.

Tämä osoittautui järkeväksi, sillä huomasin ettei validointiehtoni toimineet halutulla tavalla. Olin asettanut rajapinnalle tulevaan dataan tiettyjä ehtoja. Käytin ehtojeni luomiseen säännöllistä lauseketta, jonka piti tarkkailla, että kahden putkimerkin välissä olisi kahdeksan merkkiä pitkä päivämäärä. Sitä en ollut osannut ottaa huomioon, että päivämäärän pitäisi olla myös tiettyä muotoa, jotta se voidaan ongelmitta tallentaa tietokantaan.

Saatuani ehtolauseet kohdilleen sekä siivoiltua muotoiluvirheet koodista, oli aika siirtää muutos versionhallintaan. Asioille lähdön johdosta työpäivä jäi lyhyeksi, mutta oli silti mielestäni hyödyllinen. Rajapinta toimii nyt halutulla tavalla ja ensi viikolla voin alkaa toteuttamaan selainpuolen muutoksia.

Vasta tapahtuneen yritysoston myötä olemme saaneet paljon uusia työkavereita, joihin tulemme tutustumaan illalla järjestettävässä henkilöstön virkistymistapahtumassa. Hienoa saada välillä muutakin ajateltavaa kuin ohjelmointi.

Viikkoanalyysi

Jälleen kerran viikko oli monipuolinen. Välillä sain perehtyä täysin ohjelmointiin ja välillä oli palavereita ja viikon kruunasi perjantainen henkilöstötapahtuma. Viikko oli myös merkityksekäs siinä mielessä, että se oli tämän päiväkirjan viimeinen viikko ja opiskeluprosessi tulee samalla päätökseensä.

Viime ajat olen pyrkinyt parantamaan tarkkuuttani muun muassa muotoiluvirheiden osalta ja huomaan sen vaikutuksen. Olenkin kirjannut itselleni ylös muitakin kehityskohteita ja uskon, että vain omat heikkoutensa tunnistamalla on mahdollista kehittyä. Viikolta parhaiten jäi mieleen oma päätökseni puuttua huonon suorituskyvyn omaamaan funktion toteutukseen. Omaa työtään tulee aina tarkastella kriittisesti ja refaktorointi on todella tärkeää, jotta saadaan pidettyä koodipohja ajan tasalla ja suorituskykyisenä. Algoritmien tehokkuus kiinnostaa itseäni aiheena suuresti ja siksipä se valikoituikin päiväkirjan viimeisen viikon analyysin aiheeksi. Algoritmien tehokkuudesta puhuttaessa usein erotetaan algoritmin tilakompleksisuus ja aikakompleksisuus. Tässä analyysissä perehdytään vain aikakompleksisuuteen ja sen mittayksikkönä toimivaan asymptoottiseen suoritusaikaan.

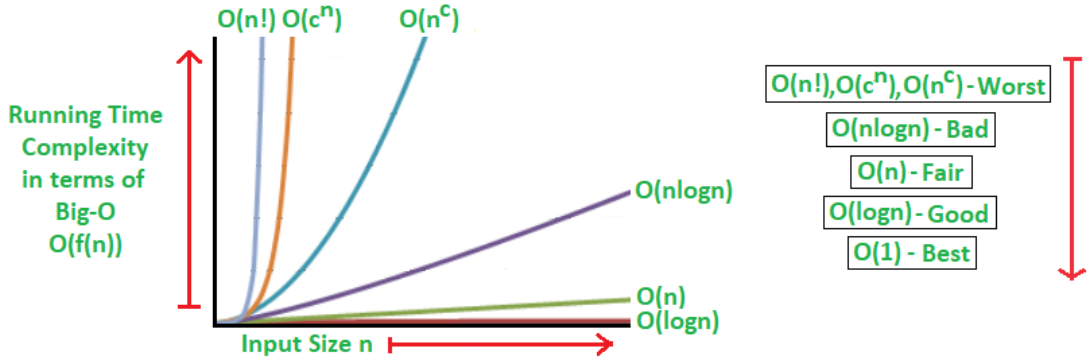
Asymptoottinen suoritusaika, eli Big O Notation, kertoo algoritmin tekemien operaatioiden määrän. Nimen ”Big O Notation” se saa esitystavastaan, jossa operaation arvioitua määrää kuvaavaa lukua edeltää iso O-kirjain. Nimestään huolimatta, se ei siis kerro algoritmin suoritukseen kuluvaa aikaa, vaan ainoastaan tarvittavan operaatioiden määrän.

Algoritmeja voidaan vertailla arvioimalla niiden suorituskykyä ongelman instanssiin, joka on kooltaan *n*. Näin tekemällä voidaan arvioida mitkä algoritmit skaalautuvat suoriutumaan ongelmista vertaamalla niiden suoritusaikaa suhteessa syötteen kokoon. (Heineman, Pollice & Selkow 2015, luku 2.)

Algoritmien suorituskyky voidaan jakaa seuraaviin seitsemään nopeusjärjestyksessä olevaan luokkaan:

- Vakio: O(1)
- Logaritminen: O(log n)
- Sublineaarinen: O(nd) for d < 1
- Lineaarinen: O(n)
- Linearitminen: O(n log n)
- Kvadraattinen: O(n^2)
- Eksponentiaalinen: O(2^n)

Algoritmin suorituskykyä arvioitaessa on pidettävä mielessä, että täytyy tunnistaa kaikkein kallein laskutapahtuma algoritmissa, jotta voidaan päätellä sen luokka. Mikäli esimerkiksi algoritmi koostuu kahdesta tehtävästä, joista toinen luokitellaan lineaariseksi ja toinen kvadraattiseksi, algoritmin suorituskykyluokka on silloin kvadraattinen. (Heineman ym. 2015, luku 2.)



KUVIO 5. Algoritmien nopeusluokkien vertailua syötteen kasvaessa (Debnath 2021).

5 POHDINTA

Ihmismieli toimii oudolla tavalla. Olisi helppo ajatella, että en muka olisi oppinut tämän päiväkirjan kirjoittamisen aikana juurikaan uusia asioita. Kuitenkin, kun luin päiväkirjan läpi ensimmäisen kerran, tajusin olevani ohjelmistokehittäjänä aivan eri tasolla kuin aloittaessani. Useat asiat, jotka aiemmin olivat vain hienoja termejä, ovat minulle nykyään arkipäivää. Parin sivun mittaisten viikkoanalyysien eteen jouduin usein tutkimaan asioita tuntikaupalla, ja uskonpa, että nuo opit eivät hevillä mielestäni lähde. Opinnäytetyön alkuperäinen tarkoitus oli tuoda esille, millaisia työtehtäviä aloitteleva ohjelmistokehittäjä työssään kohtaa, sekä toimia itselleni tukena työssäni kehittymisessä. Koen noiden tavoitteiden täyttyneen, ja toivon, että tästä työstä olisi oikeasti hyötyä tulevillekin ohjelmoijille.

Päiväkirjamuotoisessa opinnäytetyössä tunnettu haaste on päivien samankaltaisuus, ja tähän törmäsin itsekin. Alkuvaiheen viikkoja kestävät refaktoroinnit aiheuttivat haasteita sisällöntuotannon kannalta, eikä ole myöskään helppoa päättää, kuinka syvällisesti päivistään kertoo. Olisi helppo täyttää päiväkirjan sivut itsestäänselvyyksillä ja puuduttaa lukija jo ensimmäisen viikon teksteillä, mutta onneksi joka viikolle löytyi kantava teema, joka auttoi pitämään sisällön loogisena.

Päiväkirjan kirjoittaminen toi myös mukavasti rakennetta viikkoihin. Huomasin lopulta suunnittelevani päivieni sisältöä etukäteen ihan vain sen takia, että niistä olisi helpompi kirjoittaa. Opitut asiatkin pysyvät paremmin mielessä, koska ne on joutunut ensin kirjoittamaan muistiinpanoihin ja sieltä vielä puhtaaksi. Olenkin vankkumaton käsin kirjoitettujen muistiinpanojen kannattaja ja niiden tekemistä aion jatkaa tulevaisuudessakin.

Ennen tämän prosessin alkua, pidin itseäni taitavana suoriutujana. Nyt kaiken tämän oppimisen jälkeen ymmärrän, kuinka optimistinen näkökanta tuo olikaan. Ehkä olenkin elävä esimerkki Dunning-Kruger -efektin oikeellisuudesta, ja olen nyt käyrän puolivälissä ja tiedostan oman vajavaisuuteni. Ohjelmointitaitoni ovat selkeästi parantuneet prosessin aikana, mutta suurimman kehityksen koen tapahtuneen kaikilla muilla työn osa-alueilla. Ohjelmointia olin tehnyt kohtalaisen paljon jo ennen töiden aloittamistakin, mutta työhön vahvasti liittyvät DevOps, versionhallinta, virtuaalikoneet sekä ohjelmistokonttiteknologiat olivat vieraampia. Kehittyminen on tapahtunut pääasiassa

kokemuksen kautta, mutta päiväkirjaa kirjoittaessani asioita on myös joutunut analysoimaan huomattavasti enemmän, mikä on johtanut asioiden syvempään ymmärtämiseen.
On ollut hienoa havaita oma kehityksensä monilla eri osa-alueilla ja huomata kuormittavani työpariani kysymyksilläni yhä vähenevissä määrin. Haen tietoa itsenäisesti ja uskallan ottaa osaa keskusteluihin esimerkiksi palaverien aikana. Muutama esittämäni mielipide on myös otettu erittäin hyvin vastaan, ja niistä on poikinut muutoksia niin sovellukseemme kuin toimintatapoihimmekin. Olen saanut myös positiivista palautetta projektityöskentelystäni.

Aion tulevaisuudessakin käyttää viikkoanalyyseissa käyttämiäni tutkintatapoja. Tällä alalla oppiminen ei lopu koskaan, mikäli aikoo pysyä relevanttina. Seuraava askel urallani on päästä eroon Junior-etuliitteestä ja tulevaisuudessa aion suunnata enemmän ohjelmistoarkkitehtuurin pariin. En kuitenkaan aio luopua ohjelmoinnista täysin missään vaiheessa.

LÄHTEET

Atlassian 2021. What is Git. Hakupäivä 19.4.2021. https://www.atlassian.com/git/tutorials/what-is-git

Banda, Francisco 2020. The Four Pillars of Object Oriented Programming. Hakupäivä 27.4.2021. https://info.keylimeinteractive.com/the-four-pillars-of-object-oriented-programming

Blanton, Sean 2021. What is User Management? Jumpcloud 2.3.2021. Hakupäivä 8.4.2021. https://jumpcloud.com/blog/what-is-user-management

Chacon, Scott, Straub, Ben 2014. Pro Git. Second edition. New York: Apress. Hakupäivä 13.4.2021. https://git-scm.com/book/en/v2

Chandel, Munish 2018. What are four basic principles of Object Oriented Programming? Hakupäivä 27.4.2021. https://medium.com/@cancerian0684/what-are-four-basic-principles-of-object-oriented-programming-645af8b43727

Costa, Ricardo 2021. Asynchronous vs. Synchronous Programming: When to Use What. Hakupäivä 25.5.2021. https://www.outsystems.com/blog/posts/asynchronous-vs-synchronous-programming/

Danturthi, Sarma R.2020. 70 tips and tricks for mastering the CISSP exam. 1st edition. New york: Apress. Hakupäivä 9.4.2021. O'Reilly Online Learning. Vaatii käyttöoikeuden.

Debedureka, Sayantini 2020. What is Debugging and Why is it important? Hakupäivä 2.5.2021. https://www.edureka.co/blog/what-is-debugging/

Debnath, Soumyadeep 2021. Valokuva. Artikkelissa Analysis of Algorithms | Big-O analysis. Geeks For Geeks. Hakupäivä 30.5.2021. https://www.geeksforgeeks.org/analysis-algorithms-big-o-analysis/

Fielding, Roy, Thomas 2000. Architectural Styles and the Design of Network-based Software Architectures. Kalifornian yliopisto. Väitöskirja.
Hakupäivä 12.5.2021. https://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf

Flanagan, David 2020. JavaScript: The Definitive Guide. 7th edition. Sebastopol: O'Reilly Media Inc. Hakupäivä 27.5.2021. O'Reilly Online Learning. Vaatii käyttöoikeuden

Fowler, Martin 2021. Refactoring. Refactoring website.
Hakupäivä 1.4.2021. https://refactoring.com/

Guru99 2021. What is Class and Object in Java OOPS? Learn with Example.
Hakupäivä 27.4.2021. https://www.guru99.com/java-oops-class-objects.html

Heineman, George, Pollice, Gary & Selkow, Stanley 2015. Sebastopol: O'Reilly Media Inc.
Hakupäivä 30.5.2021. O'Reilly Online Learning. Vaatii käyttöoikeuden

Highsmith, Jim 2001. History: The Agile Manifesto.
Hakupäivä 30.5.2021. http://agilemanifesto.org/history.html

Lamelas, Ana 2018. Top 5 main Agile methodologies: advantages and disadvantages.
Hakupäivä 30.5.2021. https://www.xpand-it.com/blog/top-5-agile-methodologies/

LeMay, Matt 2018. Agile for Everybody. 1st edition. Sebastopol: O'Reilly Media inc. Hakupäivä 29.5.2021. O'Reilly Online Learning. Vaatii käyttöoikeuden.

MuleSoft. What is an API?(Application Programming Interface)
Hakupäivä 10.5.2021. https://www.mulesoft.com/resources/api/what-is-an-api

Metwalli, A, Sara 2020. All about debugging: The techniques.
Hakupäivä 3.5.2021. https://betterprogramming.pub/all-about-debugging-the-techniques-920b06d61a9e

Pieters, Achim 2021. Valokuva. Artikkelissa Software – Rubber duck debugging
Hakupäivä 2.5.2021. https://www.studiopieters.nl/software-rubber-duck-debugging/

Preston-Werner, Tom 2011. Semantic Versioning 1.0.0.
Hakupäivä 17.5.2021. https://semver.org/spec/v1.0.0.html

Rubber duck debugging 2008. Rubber duck debugging.
Hakupäivä 2.5.2021. https://rubberduckdebugging.com/

Suryanarayana, Girish, Samarthyam, Ganesh & Sharma, Tushar 2014. Refactoring for Software Design Smells : Managing Technical Debt. San Francisco: Elsevier Science & Technology.