Hannu Karstu

Raportinluontipalvelu testiautomaatiosovelluk-seen

Insinööri

Tieto- ja viestintätekniikan koulutus

Syksy 2022

KAMK • University of Applied Sciences

Tiivistelmä

Tekijä(t): Karstu Hannu

Työn nimi: Raportinluontipalvelu testiautomaatiosovellukseen

Tutkintonimike: Insinööri (AMK), tieto- ja viestintätekniikka

Asiasanat: tuotekehitys, raportointi, automaatio, mikropalvelut, Python, Docker, Pandoc

Opinnäytetyön tavoitteena oli kehittää mikropalvelu testiajoraporttien automaattisen luomiseen osaksi testiautomaatiosovellusta. Mikropalvelu kehitettiin tuotekehitysprosessin vaiheita noudattaen. Vaiheet ja niiden sisältö vaihtelevat lähteestä riippuen. Tämän opinnäytetyön puitteissa tuotekehitysprosessi tiivistettiin kuuteen vaiheeseen, jotka soveltuivat parhaiten ohjelmistokehityksen tuotekehitykseen.

Opinnäytetyön teoreettiseksi viitekehykseksi valittiin tuotekehitysprosessi ja sen vaiheet. Tuotekehitysprosessin tarkoitus on tehdä tuotteen kehityksestä mahdollisimman tehokasta ja edullista. Raportissa käydään läpi valitut tuotekehitysprosessin vaiheet ja peilataan niitä kehitysprosessin etenemiseen.

Opinnäytetyön raportissa kuvataan seikkaperäisesti valmiin tuotteen arkkitehtuuri ja sen yhteys testiautomaatiosovelluksen muihin toimintoihin. Raportissa esitellään myös valmiin tuotteen toiminnallisuus ja sidokset muihin testiautomaatiosovelluksen osiin. Raportti ei sisällä ohjelmakoodia eikä lopullista esimerkkiä lopullisesta PowerPoint-raportista, mutta mukana on kuitenkin esimerkkeinä syötedataa ja niistä muokattua dataa.

Opinnäytetyön raportin lopussa käydään läpi lopullisesta tuotteesta saatu palaute, työn tekijän itsearviointi sekä jatkokehityssuunnitelmat.

KAMK • University of Applied Sciences

Abstract

Author(s): Karstu Hannu

Title of the Publication: Report Creation Service for Test Automation Tool

Degree Title: Bachelor of Engineering, Information technology

Keywords: product development, reporting, automation, microservices, Python, Docker, Pandoc

The goal of the thesis was to develop a microservice for automatically creating test run reports. The microservice would be implemented as a part of a test automation application. The development process was done by following product development guidelines. The phases of the product development process vary based on the source. In the scope of this thesis, the process was reduced to six phases most suitable for software development process.

Product development process and its phases were chosen as the theoretical background of the thesis. The goal of the product development process is to make the development of new product as efficient and affordable as possible. The chosen phases are mirrored against the actual progress of the development process.

The architecture and its connections to other parts of the test automation application are thoroughly explained in the thesis. The functionality of the service is described with detail. Images and diagrams are also used where appropriate. The thesis doesn't contain screenshots of the PowerPoint report or any source code of the actual service, but examples of input and processed data are presented.

Received feedback, self-evaluation and further development plans of the report creation service are all presented as a conclusion.

Sisällys

1 Johdanto 1

2 Testiautomaatiosovellus 2
2.1 Arkkitehtuuri 2
2.2 Käyttöliittymä 3

3 Tuotekehitysprosessi 5
3.1 Vaatimusten määrittely 5
3.2 Toimintamallien valinta 8
3.3 Soveltuvuusselvitys 9
3.4 Tuotteen kehitys 9
3.5 Testaus 11
3.6 Jatkokehitys 12

4 Raportinluontipalvelu 13
4.1 Toiminnallisuus 13
4.1.1 Syötedata, sisällysluettelot ja sivupohjat 13
4.1.2 Raporttien luonti 16
4.1.3 Visualisointipalvelu 18
4.2 Arkkitehtuuri 19

5 Yhteenveto 22
5.1 Jatkokehityssuunnitelmat 24
5.2 Itsearviointi ja saatu palaute 24

Lähteet 26

1 Johdanto

Työn toimeksiantaja Devecto Oy:n pääosaaminen on älykkäiden laitteiden ja koneiden ohjelmistokehityksessä ja -testauksessa sekä testausjärjestelmissä. Devecto on vuoden 2022 alusta ollut osa Gofore-teknologia- ja konsulttiyritystä. [1.]

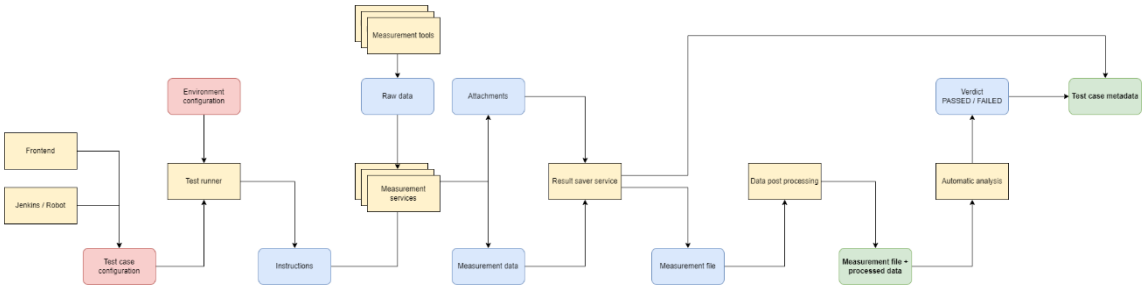
Devecton asiakasyrityksessä suoritetaan laitevalmistusta ja -testausta. Testaaminen on erittäin aikaa vievä ja vaativa prosessi. Sitä nopeuttamaan on sisäisesti kehitetty testiautomaatiosovellusta. Tämä sovellus nopeuttaa testausprosessia ajamalla automaattisesti testit suunnitelmien mukaan. Testiajon aluksi sovellus käynnistää mittalaitteet ja -palvelut. Testiajon aikana sovellus kerää dataa mittauslaitteilta. Lopuksi sovellus jatkojalostaa dataa prosessoimalla sitä sekä analysoi testiajon hyväksymistä kerätyn datan pohjalta. Testaajan on mahdollista myös visualisoida kerättyä dataa erilaisin keinoin.

Seuraava askel testausprosessin automatisoinnissa on testiajoraporttien automaattinen luominen. Ilman testiautomaatiosovellusta testaajat ovat keränneet testiajoista syntyvän datan muun muassa itse kehittämillään työkaluilla, ja yhdistelleet ja visualisoineet sitä taulukkolaskentaohjelmilla. Lopuksi he ovat luoneet raportteja testiajoista leikkaa-liimaa-tekniikalla grafiikkaa ja dataa yhdistellen esimerkiksi PowerPoint-esitysgrafiikkaohjelmalla. Tämä on ollut erittäin aikaa vievä prosessi. Testiajoraporttien automaattinen luominen testiautomaatiosovelluksen tuottaman datan pohjalta tulisi siis nopeuttamaan ja helpottamaan testausprosessia entisestään.

Opinnäytetyön tekijä on työskennellyt opintojen ohella Devecton asiakasyrityksellä osana testiautomaatiosovelluksen kehitystiimiä reilun vuoden ajan ennen opinnäytetyöprosessin aloittamista. Tulevia ominaisuuksia suunniteltaessa nousi esiin tarve testiajoraporttien automaattiselle luomiselle. Aiheen laajuus, vaativuus ja toteutustavan vapaus tekivät siitä erinomaisen aiheen opinnäytetyölle.

2 Testiautomaatiosovellus

Kuvassa 1 esitetään opinnäytetyöhön liittyvän testiautomaatiosovelluksen toiminnallisuus yksinkertaistettuna. Testiajot käynnistetään web-pohjaisen käyttöliittymän tai Jenkins-automaatioserverin käyttöliittymän kautta. Testiajon aluksi testiautomaatiosovellus lähettää käynnistyskäskyt eri mittauslaitteille ja -palveluille. Kun laitteet ovat käynnistyneet, alkavat palvelut tallettaa laitteilta tulevaa dataa. Tämä data lähetetään erillisen tallennuspalvelun käsiteltäväksi, ja se lisää datan yhteyteen esimerkiksi puuttuvat aikaleimat. Kun mittaukset on suoritettu, pysäyttää testiautomaatiosovellus mittauslaitteet ja -palvelut. Tämän jälkeen tallennettu mittausdata jälkikäsitellään. Lopuksi mittausdatasta suoritetaan automaattinen analyysi, joka kertoo, oliko testiajo onnistunut vai ei.



Kuva 1. Testiautomaatiosovelluksen toiminnallisuus yksinkertaistettuna.

2.1 Arkkitehtuuri

Ohjelmiston arkkitehtuuri on erään määritelmän mukaan käsite, joka tarkoittaa ohjelmiston perusorganisaatiota sisältäen kaikki ohjelmiston osat ja niiden keskinäiset suhteet sekä osien suhteet ympäristöön. Ohjelmiston arkkitehtuuri voidaan myös tiivistää niiksi asioiksi, joita on haastavaa muuttaa jälkikäteen. [2.]

Opinnäytetyöhön liittyvä testiautomaatiosovellus on rakennettu mikropalveluarkkitehtuurin pohjalta. Mikropalveluarkkitehtuurissa useat pienet autonomiset palvelut kommunikoivat verkon yli keskenään ja toteuttavat yhdessä järjestelmän toimivuuden. Mikropalveluista pyritään usein tekemään mahdollisimman itsenäisiä, ja ne eivät esimerkiksi kutsu suoraan toistensa funktioita. [2.] Osa testiautomaatiosovelluksen mikropalveluista käyttää kuitenkin yhteisiä, samassa repositoriossa eli lähdekoodivarastossa, määriteltyjä funktioita niiltä osin kuin on järkevää.

Mikropalveluiden pohjalta toimivaa sovellusta on helppo skaalata ja laajentaa tarpeiden mukaan. Mikään ei esimerkiksi estä ajamasta useita mikropalveluja rinnakkain mahdollisten pullonkaulojen vähentämiseksi. Mikropalveluissa voidaan myös käyttää useita eri ohjelmointikieliä toisin kuin tavanomaisissa arkkitehtuureissa. [2.]

Testiautomaatiosovelluksen mikropalvelut kommunikoivat toistensa kanssa viestinvälityspalvelun avulla. Tämä palvelu voi esimerkiksi välittää viestin mikropalvelulta toiselle, odottaa vastauksen ja välittää sitten sen takaisin. Mikropalvelut voivat tilata tietyllä otsikolla olevat viestit, jolloin viestinvälityspalvelu välittää viestit oikeaan paikkaan. Viestien lähetys on asynkronista, eli mikropalvelu jatkaa eteenpäin koodissa heti viestin lähetettyään. On kuitenkin myös mahdollista jäädä odottamaan vastausta viestiin ennen eteenpäin jatkamista.

Mikropalvelut tuovat eduistaan huolimatta mukanaan myös haasteita. Sovelluksen toiminnallisuus täytyy esimerkiksi jakaa järkevän kokoisiin mikropalveluihin. Vääränlainen jako voi johtaa tilanteeseen, jossa palvelut joutuvat jatkuvasti keskustelemaan verkon yli, mikä taas voi aiheuttaa suorituskykyongelmia sovelluksessa. [2.]

Mikropalveluissa käytetään nykyään usein konttiteknologiaa. Kontit ovat yksinkertaisesti sanottuna virtuaalikoneita, joita on mahdollista olla käynnissä useita yhdellä fyysisellä koneella. Yleisin konttiteknologian käytön mahdollistava ohjelmisto on Docker. [3.] Mikropalveluiden jakaminen kontteihin tekee koko sovelluksen testaamisen ja virheiden etsimisen haastavaksi etenkin viestinvälityspalveluita käyttäessä [2].

2.2 Käyttöliittymä

Automaattiset testiajot voidaan käynnistää kahta eri reittiä, selainpohjaisen käyttöliittymän tai Jenkins-automaatioserverin käyttöliittymän kautta. Testiajon vaatimukset määrittelevät kumpaa käyttöliittymää testaajan tulee käyttää.

Selainpohjaisen sovelluksen käyttöliittymä koostuu kaikesta siitä, mitä käyttäjä näkee ja voi käyttää selaimen kautta. Selainpohjainen käyttöliittymä, frontend, koostuu useimmiten HTML:stä, CSS:stä ja JavaScriptistä. [4.]

HTML on lyhenne sanoista Hypertext Markup Language, suomeksi hypertekstin merkintäkieli. HTML on kuvauskieli, jonka avulla kuvataan verkkosivun sisältö ja rakenne. [5.] CSS taas on lyhenne sanoista Cascading Style Sheets, tyylisivut. CSS-tiedostossa kuvataan verkkosivun tyyli.

Siinä voidaan määritellä esimerkiksi käytetyt fontit, värit ja sisennykset. [6.] JavaScript on skriptaus- tai ohjelmointikieli, jonka avulla voidaan luoda monimutkaisia ominaisuuksia ja dynaamisesti päivittyviä komponentteja verkkosivulle. [7.]

Testiautomaatiosovelluksen selainpohjainen frontend on rakennettu HTML:ää, CSS:ää sekä Javascriptiä ja React-kirjastoa käyttäen. Se mahdollistaa testiajon valinnan, konfiguroinnin ja käynnistämisen. Frontendin kautta voi myös konfiguroida ympäristön parametreja ja seurata eri mikropalveluiden tuottamia logeja. Selaimen kautta voi myös tarkastella ajettujen testien tuloksia ja esimerkiksi avata testiajon mittaustulosten visualisointeja. Frontend kommunikoi testiautomaatiosovelluksen kanssa REST-rajapinnan kautta.

Jenkins toimii toisena reittinä testiajojen käynnistämiseen. Jenkins on automaatiotyökalu, joka mahdollistaa sovellusten jatkuvan rakentamisen, testaamisen ja toimittamisen. Se nopeuttaa kehittäjien työtä automatisoimalla monia usein toistuvia kehitystyön vaiheita. [8.] Testiajossa Jenkins käyttää Robot Frameworkia. Robot Framework on Python-pohjainen automaatiotestaukseen käytetty sovelluskehys. Robot Frameworkin koodi rakentuu avainsanoista, joita lukemalla harjaantumatonkin silmä pystyy ymmärtämään, mitä koodi tekee. [9.]

3 Tuotekehitysprosessi

Opinnäytetyön teoreettiseksi viitekehykseksi valittiin tuotekehitysprosessi vaiheineen. Tuotekehitysprosessin tavoitteena on kehittää uusi tuote tai parantaa olemassa olevaa [15, s. 105].

Tuotekehitysprosessi voidaan jakaa toimintavaiheisiin. Teoksessaan "Tuotekehitys" Tapani Jokinen jakaa tuotekehitysprosessin neljään vaiheeseen: tuotekehitysprosessin käynnistäminen, tuotteen luonnostelu, sen kehittäminen ja viimeistely. [10., s. 14.] Antti Elo artikkelissaan "Tuotekehityksen vaiheet - näin onnistut tuotekehityksessä" taas jakaa prosessin kahdeksaan eri vaiheeseen. Näitä ovat vaatimusten määrittely, ratkaisujen määrittely, markkinoiden testaaminen, valitun ratkaisun konseptointi ja simulointi, lopullinen ratkaisu ja yksityiskohdat, prototyypin testauksen, valmistaminen ja laadunvarmistuksen suunnittelu, prototyypin testaus sekä jatkokehitys ja tuotannon suunnittelu. [11.]

Tämän opinnäytetyöprosessin puitteissa tuotekehitysprosessi tiivistettiin kuuteen vaiheeseen: vaatimusten määrittely, toimintamallien valinta, proof of conceptin kehitys tai soveltuvuusselvitys, tuotteen kehitys, testaus ja jatkokehitys.

Tuotetta kehittäessä on tärkeää seurata tarkkaan prosessin kaikkia vaiheita. Näin toimien huolehditaan siitä, että tuotteen kehityksen suuntaa on mahdollista muuttaa tai koko prosessi jopa lopettaa, jos huomataan ettei idea toimikaan. [11.] Tämän opinnäytetyön myötä tehdyn tuotekehitysprosessin vaiheet tarkentuivat useasti työn etenemisen myötä. Työn etenemistä vaiheesta toiseen arvioitiin myös huolellisesti.

3.1 Vaatimusten määrittely

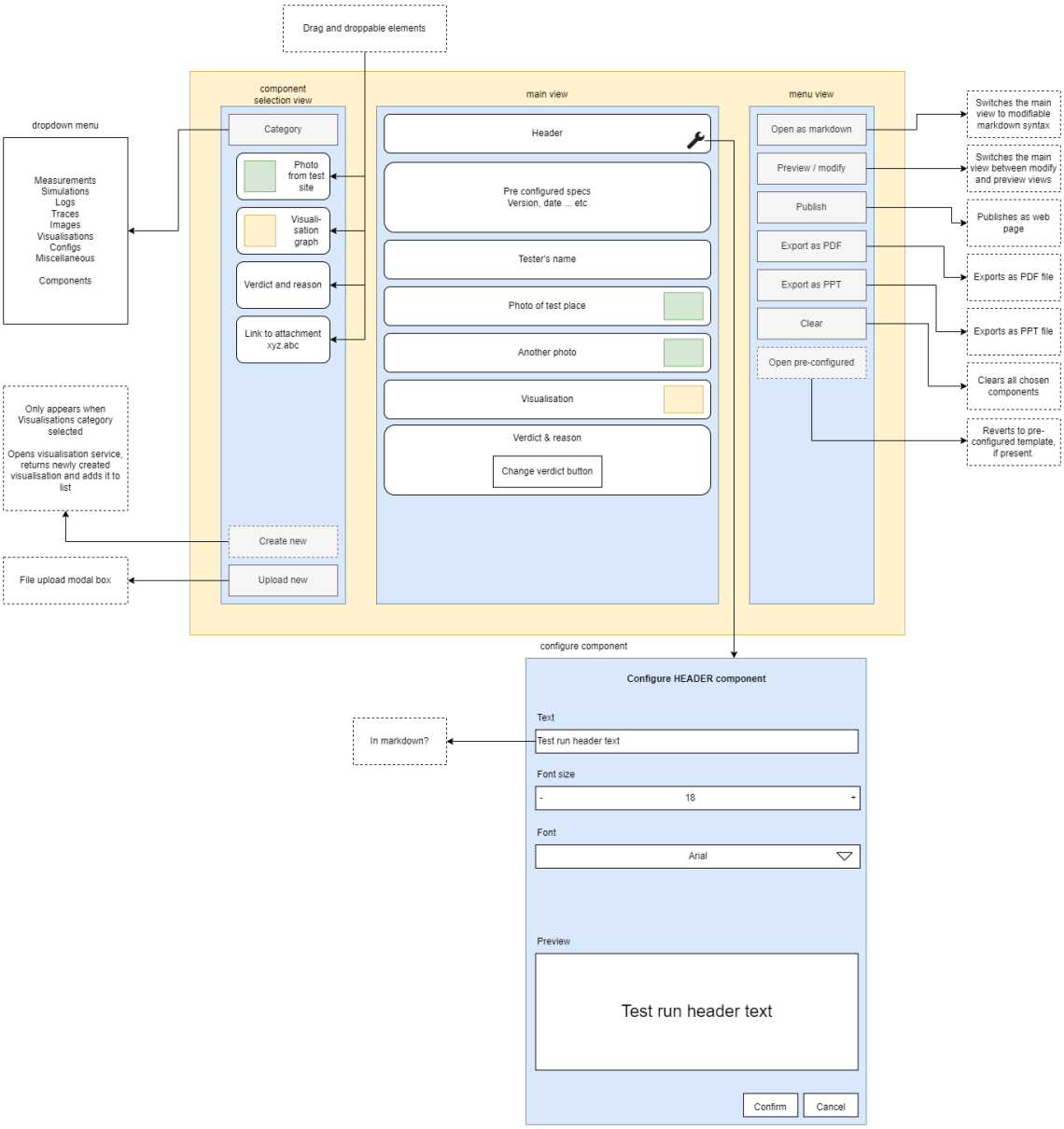
Kuten johdannossa kerrotaan, tarve automaattiselle raporttien luomiselle oli havaittu. Ratkaisut tämän tarpeen täyttämiseksi eivät kuitenkaan olleet selviä, eivätkä myöskään asiaan liittyvät vaatimukset. Näin ollen tuotekehitysprosessi päädyttiin aloittamaan vaatimusten määrittelyllä.

Vaatimusten määrittely on ehkä keskeisin asia rakennettavalle ohjelmistolle. Ohjelmistojen vaatimukset jakaantuvat kahteen osaan: Toiminnallisiin ja ei-toiminnallisiin vaatimuksiin. Toiminnal-

liset vaatimukset tarkoittavat niitä asioita, mitä ohjelmistolla voi tehdä. Ei-toiminnalliset taas tarkoittavat esimerkiksi käytettävyyteen tai tietoturvaan liittyviä asioita. Myös ohjelmiston toimintaympäristö voi asettaa ohjelmistolle rajoitteita. [2.]

Vaatimusten kartoittamiselle on olemassa erilaisia menetelmiä. Näitä voivat olla esimerkiksi sidosryhmien haastattelut tai työn toimeksiantajan / asiakkaan kanssa pidetyt suunnittelukokoukset. [2.] Raportinluontipalvelun vaatimukset määriteltiin yhdessä sidosryhmien kanssa ja niitä tarkennettiin työn edetessä.

Vaatimusten määrittelyn yhteydessä sovelluksen käyttöliittymästä kannattaa tehdä luonnoksia ja suunnitelmia, joiden avulla voidaan tarkentaa näkemystä vaatimuksista [2]. Raportinluontipalvelun kehitystyön aikana luotiin muun muassa useita sekvenssikaavioita, joiden avulla palvelun toiminnallisuutta pystyttiin selittämään. Palvelulle laadittiin myös kuvassa 2 esitelty käyttöliittymäsuunnitelma, mutta käyttöliittymän toteuttaminen päädyttiin siirtämään jatkokehitysideaksi työn mittakaavan tarkentumisen myötä.



Kuva 2. Käyttöliittymäsuunnitelma.

Valmiille tuotteelle asetettiin useita toiminnallisia vaatimuksia. Näistä olennaisimpia olivat raporttien ulostuloformaatit PowerPoint-tiedosto ja HTML-sivupresentaatio, raporttien ulkoasun konfiguroitavuus sekä toteutuksen geneerisyys ja monikäyttöisyys. Olennaisimpien vaatimusten testaaminen olisi tärkeää jo soveltuvuusselvitysvaiheessa. Palvelun pitäisi myös pystyä keskustelemaan muiden mikropalveluiden kanssa ja hakemaan tarvitsemaansa dataa niistä.

Toteutuksen geneerisyydellä tarkoitetaan sitä, että palvelun tulee pystyä luomaan raportti käyttäen hyväkseen mitä tahansa dataa. Alussa syötedataksi rajattiin yksittäisen testiajon tuottama data, mutta sopivilla muokkauksilla raporttipohjaan ja ohjelmakoodiin pystyy palvelulla luomaan raportteja mistä tahansa datasta.

Valmiiden raporttien formaateiksi rajattiin PowerPoint ja HTML-pohjainen esitys. PowerPointin valinta perustuu testaajien tottumukseen ja helppoon muokattavuuteen. HTML-pohjainen esitys ei ole muokattavissa jälkikäteen, mutta sen olemassaolo mahdollistaa testiajon tietojen helpon tarkastelun jäsennellyssä muodossa.

Yhteys muihin mikropalveluihin mahdollistaa datan hakemisen tai käsittelyn myös niitä hyödyntäen. Tässä yhteydessä visualisointipalveluun oli luotava mahdollisuus pyytää visualisointeja ilman graafista käyttöliittymää.

Ei-toiminnalliset vaatimukset liittyivät pääasiassa tekniseen toteutukseen. Lopullisen palvelun tuli olla helppo- ja monikäyttöinen, monipuolisesti konfiguroitavissa ja noudattaa muiden mikropalveluiden tyyliä toimia Docker-kontissa.

Raportin ulkoasun muokattavuus haluttiin mukaan, jotta luodut raportit olisivat esittelykelpoisia myös ulkoisille sidosryhmille. Toimeksiantajan asiakasyrityksellä on olemassa valmis PowerPoint-pohja, joka tulisi ottaa malliksi raportin ulkoasulle.

3.2 Toimintamallien valinta

Kun vaatimukset oli selvitetty, täytyi selvittää, millä keinoilla ne saataisiin täytettyä. Tämä vaati perehtymistä eri ohjelmistoihin ja teknologioihin. Tuli selvittää, kuinka syötedatasta luotaisiin raportin tiedot sisältävä tiedosto ja kuinka tämä tiedosto muunnettaisiin PowerPoint-tiedostoksi ja HTML-sivuksi.

Raportinluontipalvelun toimintamallit päädyttiin valitsemaan toiminnallisten vaatimusten perusteella. Toiminnalliset vaatimukset kuvaavat, mitä toimintoja ohjelmistossa on oltava. Näiden dokumentointi voi tapahtua esimerkiksi ominaisuuslistauksena tai käyttäjätarinoina, jotka kuvaavat, mitä käyttäjä haluaa tehdä. [2.]

Toimintamallien selvityksen jälkeen vaihtoehdot esiteltiin sidosryhmille ja niistä valittiin yhdessä toimivimmat. Prosessin aikana tutustuttiin myös toimeksiantajan asiakasyrityksen muiden osastojen tekemiin ratkaisuihin raportointimielessä. Näistä saadut toimivat ideat pyrittiin yhdistämään raportinluontipalveluun.

3.3 Soveltuvuusselvitys

Kun vaatimukset oli määritelty ja toimintamallit valittu, tuli aika kehittää proof of concept eli tehdä soveltuvuusselvitys. Tämän avulla tulisi esitellä sidosryhmille idean toimivuutta käytännössä yksinkertaisen, mutta toimivan prototyypin avulla.

Soveltuvuusselvitys on tuotekonseptin testausvaihe, jonka avulla nähdään, toimiiko idea käytännössä. Soveltuvuusselvityksen avulla tulee pystyä päättämään, lähdetäänkö ideaa jatkokehittämään. Soveltuvuusselvityksen lopputuloksena voi olla myös se, ettei tuoteidea vastaakaan tarpeeseen. Tällöin on järkevää lopettaa kehitys tai tehdä tarvittavia muutoksia. Soveltuvuusselvityksen avulla nähdään myös, kuinka paljon tuotteen kehitys tulisi vaatimaan resursseja. [12.]

Proof of concept esiteltiin valmistuttuaan sidosryhmille. Se oli useista pienemmistä paloista koostuva kokonaisuus, jonka avulla pystyi kuitenkin esittämään ohjelman tulevan toiminnallisuuden. Saadun palautteen perusteella vaatimuksia tarkennettiin ja lupa edetä seuraavaan vaiheeseen myönnettiin.

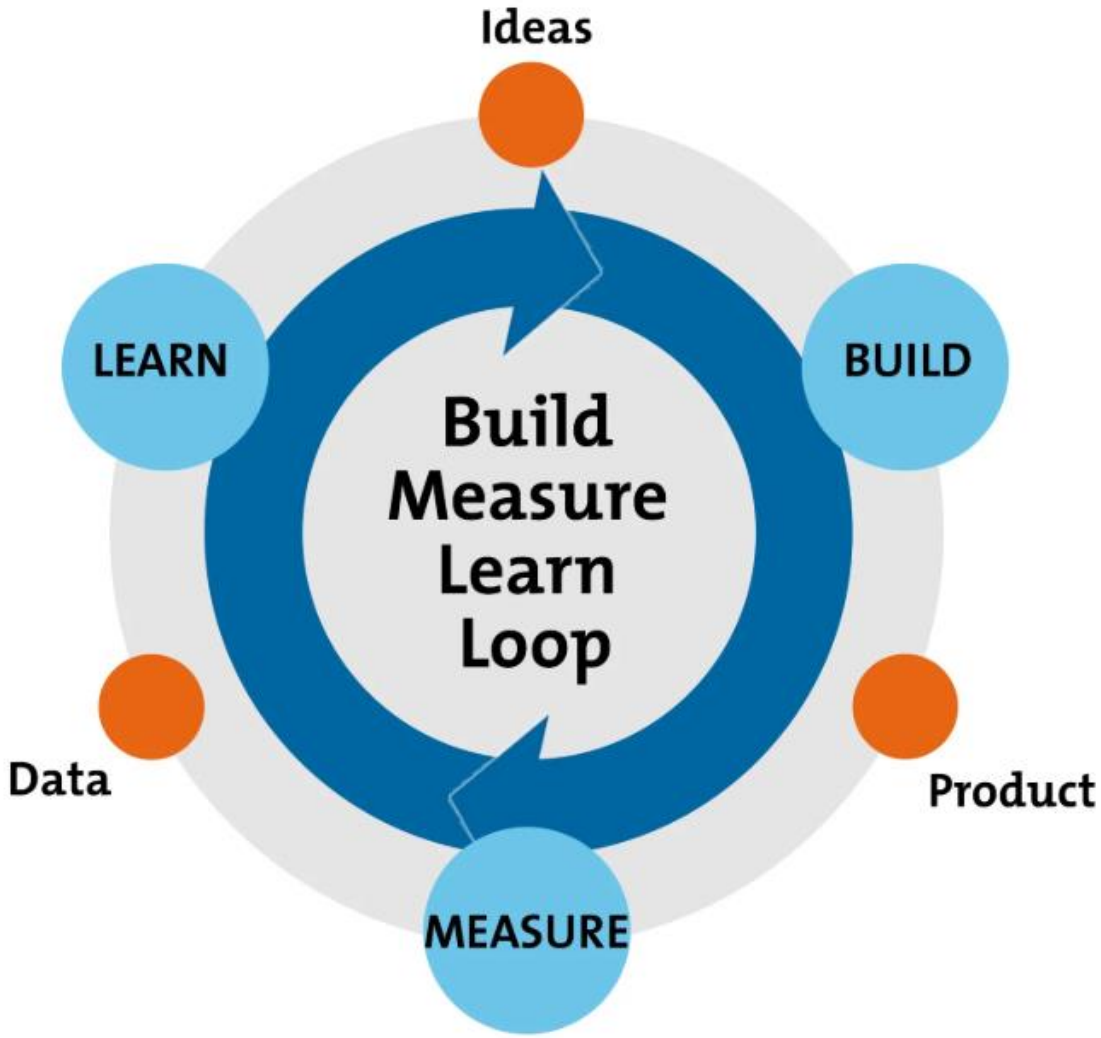
3.4 Tuotteen kehitys

Kun proof of concept oli hyväksytty, oli aika kehittää konseptin pohjalta toimiva tuote. Tuotteen ei kuitenkaan tässäkään vaiheessa tarvitsisi vielä olla kaiken kattava. Riittävä taso olisi, että tuote integroituisi uutena mikropalveluna testiautomaatiosovellukseen ja sitä pystyisi käyttämään rajatuissa käyttötapauksissa.

Aiemmin yrityselämässä tuotteen vaatimukset on haluttu määrittää mahdollisimman kattavasti ja ristiriidattomasti. On pidetty tärkeänä, että kaikki toimeksiantajan vaatimukset on kerätty heti projektin alussa, eikä niitä ole sen jälkeen muutettu. Tämä ei kuitenkaan toimi ohjelmistojen tuotekehityksessä, koska alan organisaatioiden toimintaympäristö muuttuu jatkuvasti, eikä kaikkia vaatimuksia voida mitenkään osata kuvata riittävällä tarkkuudella. Toimeksiantajan mielipide voi myös muuttua työn välivaiheita tai lopputuloksen nähdessään. [2.]

Nykyaikaiseen ketterään ja iteratiiviseen ohjelmistokehitykseen kuuluu tapa, että jokaiseen tuotteen iteraatioon valitaan toteutettavaksi vain ne vaatimukset, jotka tuottavat eniten lisäarvoa toimeksiantajalle. Vaatimusten toteuttamiseen vaadittava työmäärä ja tärkeys arvioidaan, ja niiden perusteella tehdään tarvittavat toimenpiteet. [2.]

Uusimpien trendien mukainen tuotekehitysprosessi jakaantuu rakenna-mittaa-opi-syklin toistamiseen. Ajatuksena on rakentaa ideoista tuote, mitata eri tavoin tuotteen toimivuutta ja syntyneestä datasta saaduilla opeilla luoda uusia ideoita. [2.]



Kuva 3. Build-measure-learn -sykli [16.]

Raportinluontipalvelun kehityksen aikana hiottiin vaatimuksia ja toimintamalleja vastaamaan prosessin aikana selkiintyviin tarpeisiin. Toimivan tuotteen lopullista käyttötarkoitusta tarkennettiin myös, ja joitain asioita siirrettiin kehitysvaiheesta jatkokehitysvaiheeseen. Joitain osia jouduttiin myös rakentamaan kokonaan uusiksi vaatimusten tarkentuessa tai huomatessa niiden rajoittavan muuta toiminnallisuutta.

Test driven development on kehitysmenetelmä, jossa pyritään luomaan testit koodille jo ennen koodin kirjoittamista. Tämä ei kuitenkaan välttämättä tarkoita käytännössä testien kirjoittamista ensin, vaan kyseessä on pikemminkin tekniikka, jonka tuotteena syntyy kattava joukko automaattisesti suoritettavia testejä. [2.]

Raportinluontipalvelua ja visualisointipalvelun muutoksia tehdessä pyrittiin luomaan riittävä määrä erilaisia yksikkötestejä, jotka testaisivat komponenttien toimivuuden lisäksi myös sen kykyä käsitellä mahdollisia virhetilanteita tai käyttäjän antamia virheellisiä syötteitä. Testien luominen oli haastavaa kehitysprosessin aikana, sillä ohjelman toiminnallisuus oli jatkuvassa muutostilassa. Tämä vaati myös testien muokkaamista tarvittaessa.

Koska raportinluontipalvelu on riippuvainen muista testiautomaatiosovelluksen osista, tuli yksikkötestausta varten riippuvuudet korvata niin sanotuilla tynkäkomponenteilla (engl. mock). Tynkäkomponentit tai mockit korvaavat ulkopuolisten komponenttien palauttamat arvot kovakoodatuilla arvoilla. [2.] Mockeja käytettäessä on tärkeää tietää, millaisilla parametreilla korvattua komponenttia on kutsuttava ja millaisessa muodossa se palauttaa vastauksen. Mockit on myös muistettava päivittää ajan tasalle, mikäli tehdään muutoksia korvattavan komponentin koodiin.

3.5 Testaus

Huolellisesti testatun tuotteen toiminnallisuus vahvistettu ja tutkittu [14, s. 88]. Testaus jakaantuu tasoihin sen mukaan, mikä osa testauksen kohteena on. Ohjelmistokehityksessä testauksen tasoina ovat usein katselmointi, yksikkötestaus, integraatiotestaus, järjestelmätestaus ja käyttäjän hyväksymistestaus. [2.]

Osa ohjelmistokehitysprosessia on, että toiset kehittäjät katselmoivat koodia. Katselmoinnin avulla ohjelmakoodista voidaan löytää ongelmia, joita ei välttämättä muulla testauksella havaita. Katselmoinnilla voidaan esimerkiksi tarkistaa, onko koodi riittävän ylläpidettävää ja noudattaako se sovittuja tyylivalintoja. [2.]

Raportinluontipalvelun koodia esiteltiin sidosryhmille useaan otteeseen työn etenemisen myötä. Etenkin toteutuksen geneerisyys ja lisäominaisuuksien helppo lisättävyys olivat tarkastelun alaisina. Työn toimeksiantajan asiakasyrityksen käytäntöihin kuuluu, että kaikki testiautomaatiosovelluksen koodin katselmoi vähintään yksi kehittäjä.

Yksikkötestien tarkoituksena on taata sovelluksen yksittäisten komponenttien toimivuus halutulla tavalla. Integraatiotestauksessa taas testataan yksittäisistä komponenteista koostuvaa kokonaisuutta. Yksikkö- ja integraatiotestauksesta on vastuussa yksittäinen kehittäjä. [2.]

Raportinluontipalvelulle ja visualisointipalvelun muutoksille kirjoitettiin yksikkötestit, joiden avulla haluttu toiminnallisuus saatiin varmistettua. Integraatiotestausta varten ei luotu automaattista selainpohjaista testiä, mutta toimivuus varmistettiin useaan otteeseen manuaalisesti testaamalla.

Järjestelmätestaus varmistaa, että ohjelmisto toimii kokonaisuudessaan halutulla tavalla. Ohjelmistoa testataan siinä ympäristössä, jonka kautta käyttäjät sitä käyttävät. Järjestelmätestauksesta vastaa usein erillinen laadunvalvonta. Sovelluksen oikeat käyttäjät, esimerkiksi pieni käyttäjäjoukko, tekevät hyväksymistestauksen. [2.]

Testausvaihetta ei tämän opinnäytetyöprosessin puitteissa saatettu loppuun asti, vaan sekä kehittäjät että testaajat tulevat jatkamaan testaamista tulevaisuudessa. Osaa tuotteen tuloksista tullaan hyödyntämään jatkokehityssuunnitelmissa, mutta osa toiminnallisuudesta tullaan mahdollisesti luomaan uudestaan uudessa ympäristössä.

3.6 Jatkokehitys

Tuotteen testauksen käynnistyttyä aloitettiin myös jatkokehityssuunnitelmat. Tuotteen jatkokehittäminen tarkoittaa tuotteen kehittämistä sen valmistumisen jälkeen. Tuotteesta voidaan esimerkiksi haluta tehdä tehokkaampi tai edullisempi. Jatkokehitykselle voi olla tarvetta esimerkiksi, kun tuotteen käyttäjiltä tulee toiveita tai kehitysehdotuksia. [13.] Jatkuvalla kehityksellä vältetään uusien tuotekehitysprojektien aloittaminen ja siitä koituvat turhat kustannukset [14, s. 94].

Opinnäytetyön raportin kirjoitushetkellä raportinluontipalvelulle on muutamia jatkokehityssuunnitelmia. Ensinnäkin tavoitteena on tehdä palvelusta entistäkin geneerisempi ja automaattisempi, jotta se pystyy minimaalisin muutoksin luomaan raportteja kaikista erilaisista testiajoista. Suunnitelmissa on myös kopioida raportinluontipalvelun toiminnallisuus laajempaan data-analytiikka-ympäristöön.

4 Raportinluontipalvelu

Kuten johdannossa kuvataan, testiautomaation tuloksista halutaan luoda raportteja. Tähän asti testaajat ovat raportoineet tuloksistaan muutamilla eri tavoilla, joista yksi on ollut PowerPointilla luodut raportit. Näihin PowerPoint-raportteihin testaajat ovat koostaneet testiajon tuloksista luotuja visualisointeja, valokuvia esimerkiksi testauspaikalta ja tietoa testilaitteiden konfiguraatioista.

Testiautomaatiosovelluksen osana mikropalveluna toimivan raportinluontipalvelun tarkoituksena on helpottaa testaajien työtä keräämällä kaikki testiajoon liittyvä olennainen data valmiiseen muokattavissa olevaan pohjaan.

4.1 Toiminnallisuus

Raportinluontipalvelun toiminnallisuus rakentuu syötedatasta, raporttien sisällysluetteloista, sivupohjista, tyyli-mallipohjista, raportin luonnista sekä yhteydestä visualisointipalveluun.

4.1.1 Syötedata, sisällysluettelot ja sivupohjat

Raportinluontipalvelulle annetaan syötteenä taulukon 1 tyyppinen syötedata ja ohjelmakoodi 2:n tyyppinen raportin sisällysluettelo. Syötedata on yksirivinen Pandas-dataframe, joka muunnetaan ohjelmakoodi 1:n kuvaamaksi Pythonin dictionaryksi, josta raportinluontipalvelu hakee tarvitsemansa tiedot.

info.name	info.description	info.productName	…
test name	test description	product name	…

Taulukko 1. Esimerkki syötedatasta.

```
{
    "info.name": "test_name",
    "info.description": "test description",
    "info.productName": "product name"
}
```

Ohjelmakoodi 1. Esimerkki syötedatasta muunnettuna dictionaryksi.

Sisällysluettelo on JSON-muotoinen tiedosto, joka sisältää tiedot raportin jokaisesta sivusta sekä käytettävästä PowerPoint-mallipohjasta ja CSS-tyylitiedostosta. Sivukohtaisesti on annettu käytettävän sivupohjan tiedostonimi ja sarakkeet, joiden tiedot syötedatasta siihen täytetään. Kuvia sisältäville sivuille taas määritellään kansio, mistä kuvat haetaan ja mitkä ovat sallitut tiedostopäätteet.

```
{
    "pages": [
        {
            "template": "header.md"
        },
        {
            "template": "table.md",
            "data:": {
                "header": "Test info",
                "columns": [
                    {
                        "name": "Test name",
                        "input": "info.name"
                    },
                    {
                        "name": "Description",
                        "input": "info.description"
                    }
                ]

            }
        },
        {
            "template": "images.md",
            "data": {
                "header": "Images",
                "folder": "images",
                "extensions": ["png", "jpg"]
            }
        }
    ],
    "visualisations": [
        {
            "summary_numerical.variables": [
                "column_name_x"
            ]
        }
    ],
    "powerpoint_template": "powerpoint_template.pptx",
    "css": "css_file.css"
}
```

Ohjelmakoodi 2. Esimerkki sisällysluettelosta. Hannu Karstu.

Sisällysluetteloiden käyttämät sivupohjat ovat Markdown-muotoisia tiedostoja. Nämä sisältävät Jinja2-syntaksilla kirjoitetut käskyt, kuinka sivupohjan saama data sivulle täytetään. Ohjelmakoodi 3:ssa on esimerkkinä liitetiedostoja sisältävien sivujen luonti.

```
{% for ref_type in func.get_ref_types(result) -%}
 - {{ ref_type -}}
    {%- for storage_id in func.get_storage_ids(ref_type, result) %}
    - [{{ storage_id[0] -}}]({{ storage_id[1] -}})
    {%- endfor %}
{% endfor %}
```

Ohjelmakoodi 3. Esimerkki sivupohjan tyylistä.

4.1.2 Raporttien luonti

Raportinluontipalvelu vastaanottaa siis syötedatan ja sisällysluettelon ja hyödyntää valmiiksi konfiguroituja sivupohjia. Näistä luodaan Jinja2-kirjastoa käyttäen Markdown-muotoinen tekstitiedosto. Ohjelmakoodi 4:ssä on esimerkki tiedoston tyylistä.

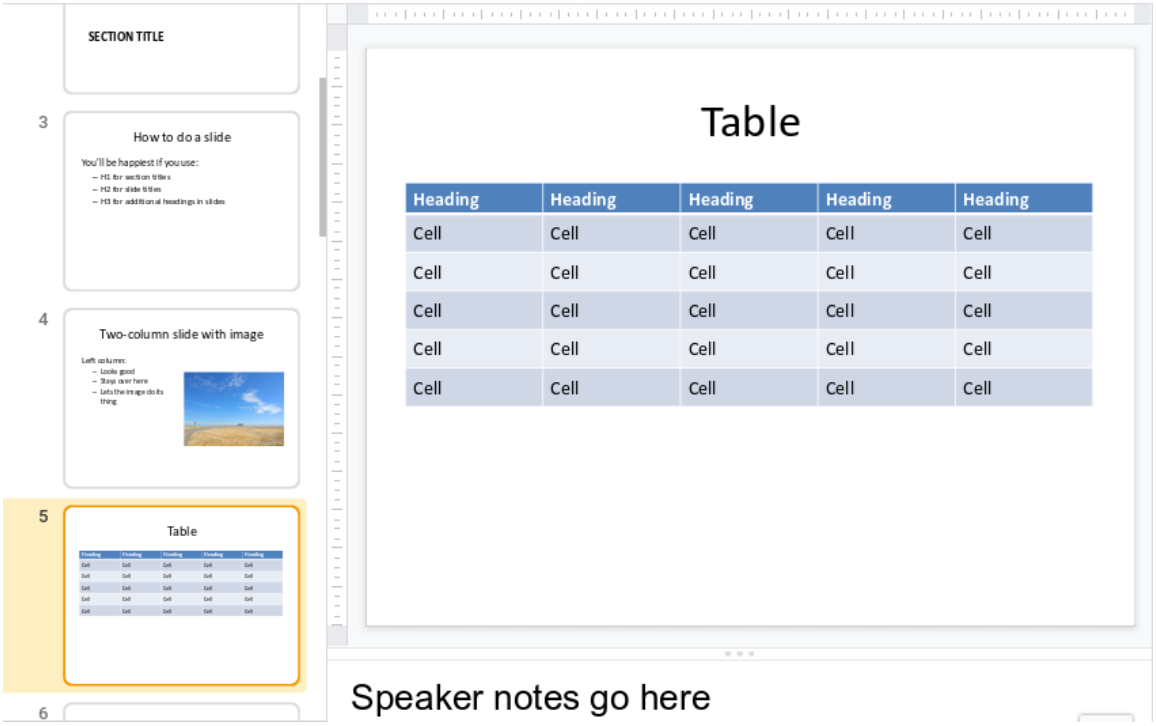
```
title: Test report
subtitle: Test report subtitle
author:
    - John Doe
date: 2022-07-27
---
# Test info
|              |                 |
| ------------ | --------------- |
| Name         | test name       |
| Description  | test description |
| Product name | product name    |
| ...          | ...             |
---
```

Ohjelmakoodi 4. Esimerkki Markdown-pohjaisesta raportista.

Tämän Markdown-muotoisen tekstitiedoston muuntamiseksi PowerPoint-tiedostoksi ja HTML-pohjaiseksi esitykseksi käytetään Pandoc-sovellusta. Tuotekehitysprosessin aluksi tutkittiin ja vertailtiin eri mahdollisuuksia muuntaa Markdown-muotoinen tiedosto haluttuihin formaatteihin. Pandoc todettiin kaikista monipuolisimmaksi ja eniten konfiguraatiomahdollisuuksia sisältäväksi.

Pandocin konfigurointi oli haastava työ. Kun ulostuloformaattina on PowerPoint, tulee Pandocille antaa PowerPoint-mallipohja, joka sisältää halutut fontit, värit ja asettelut. Tämä mallipohja taas täytyi itse rakentaa yhdistellen Pandocin oletusmallipohjaa ja asiakasyrityksen PowerPoint-pohjaa. Mallipohjan muokkaus oli haastavaa, koska satunnaiset muutokset saattoivat yllättäen rikkoa sen yhteensopivuuden Pandocin kanssa. Lisäksi Pandocin eri versiot tuottivat erilaisia lopputuloksia. Lopulta toimiva kokonaisuus saatiin kuitenkin muokattua.

Opinnäytetyön raportti ei työn luottamuksellisuuden vuoksi sisällä esimerkkejä lopullisen PowerPoint-raportin ulkoasusta. Kuvassa 4 on esimerkkinä Pandocin oletusmallipohjalla luotu PowerPoint-presentaatio.



Kuva 4. Pandocin oletusmallipohjalla luotu PowerPoint-presentaatio [17.]

HTML-pohjaisen esityksen luomisessa tuli taas ensin valita sopiva slideshow-esitystapa Pandocin tukemista formaateista. Näitä ovat Slideous, Slidy, DZSlides, RevealJS ja S5. Näistä päädyttiin valitsemaan Slidy sen sopivan yksinkertaisen käyttöliittymän ja luotettavan toiminnallisuuden

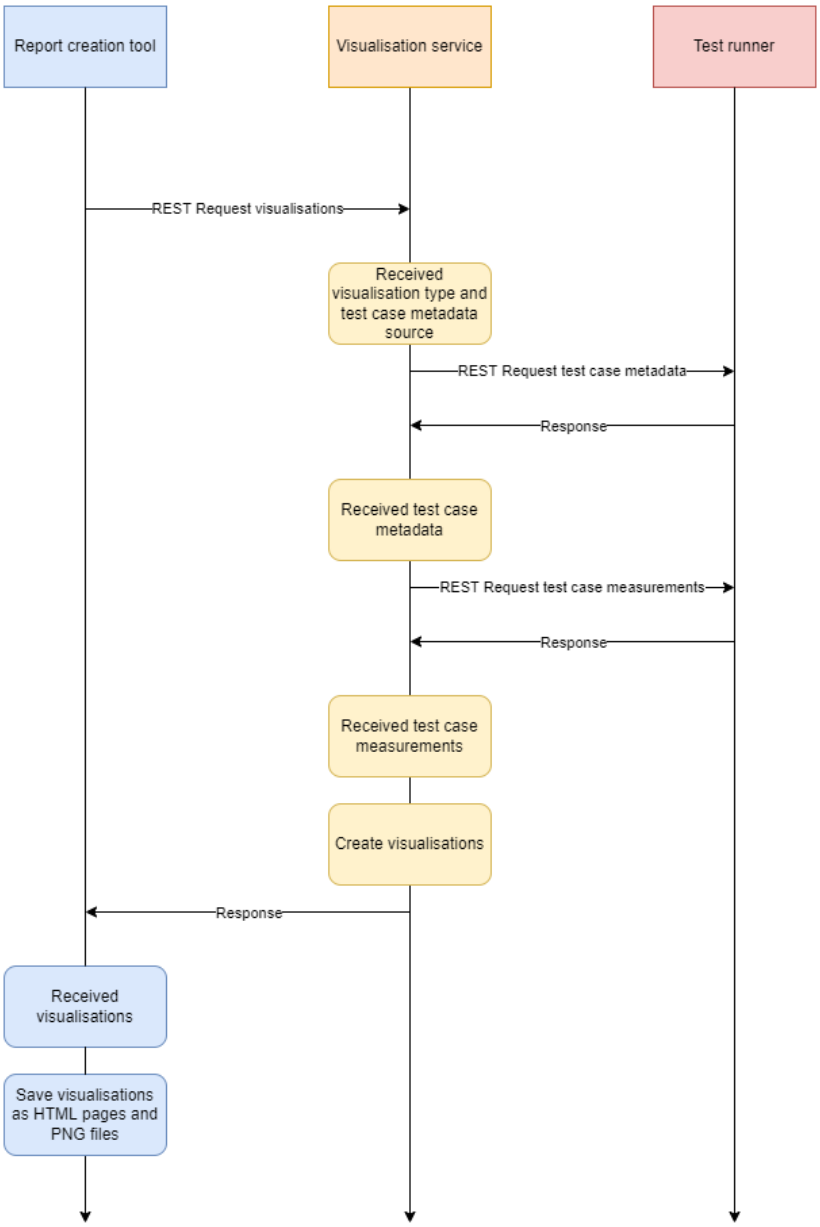
vuoksi. Jotta HTML-pohjainen esitys olisi riippumaton ulkopuolisista lähteistä, tuli Slidyn käyttämät skriptit ja tyylitiedostot ladata ja määritellä Pandoc käyttämään niitä. Muokattavuus taas piti mahdollistaa lisäämällä ylimääräinen tyylitiedosto, jolla raportin fontit ja värit saatiin kohdalleen.

Pandocille tuli lisäksi määritellä kansiot, joista se hakee esimerkiksi raportteihin lisättävät kuvat ja visualisoinnit. Nämä tuli tietenkin ensin hakea esimerkiksi testiautomaatiosovelluksen web apin kautta tai visualisointipalvelulta.

4.1.3 Visualisointipalvelu

Testiautomaatiosovelluksessa mikropalveluna toimiva visualisointipalvelu on alun perin luotu puhtaasti käyttöliittymäpohjaiseksi palveluksi. Jotta raportinluontipalvelu pystyisi pyytämään visualisointeja visualisointipalvelulta, tuli ohjelmakoodia muokata sen verran, että HTML-komponentit ja raakafiguurit eroteltiin toisistaan.

Visualisointipalveluun tuli myös lisätä ohjelmointirajapinta, mikä mahdollisti visualisointien pyytämisen joko niin sanottua suosikkistruktuuria tai suosikkitiedoston nimeä käyttäen. Visualisoinnit piti myös saada pyydettyä haluamassaan formaatissa, interaktiivisena HTML-sivuna, kuvana tai JSON-muotoisena datana. Lisäksi tuli mahdollistaa kuvan koon määrittäminen ja tietysti luoda yksikkötestit koko toiminnallisuudelle. Kuvassa 5 esitetään visualisointien hakeminen sekvenssikaavion muodossa.



Kuva 5. Sekvenssikaavio visualisointien hakemisesta.

4.2 Arkkitehtuuri

Raportinluontipalvelu toteutettiin erillisenä mikropalveluna. Tämä tarkoittaa sitä, että palvelu pystytetään Docker-kontissa pyöriväksi Python-skriptiksi, johon voi ottaa yhteyttä ohjelmointirajapinnan kautta. Mahdollisia rajapintoja palveluun luotiin alun perin kaksi: web-pohjainen testi-

automaatiosovelluksen ulkopuolista käyttöä varten sekä NATS-pohjainen sisäistä, Docker-konteissa pyörivien mikropalveluiden välistä käyttöä varten. NATS-pohjainen rajapinta päädyttiin kuitenkin hylkäämään, koska palvelu päädyttiin ensisijaisesti sijoittamaan ulkoiseen sovellusvarastoon.

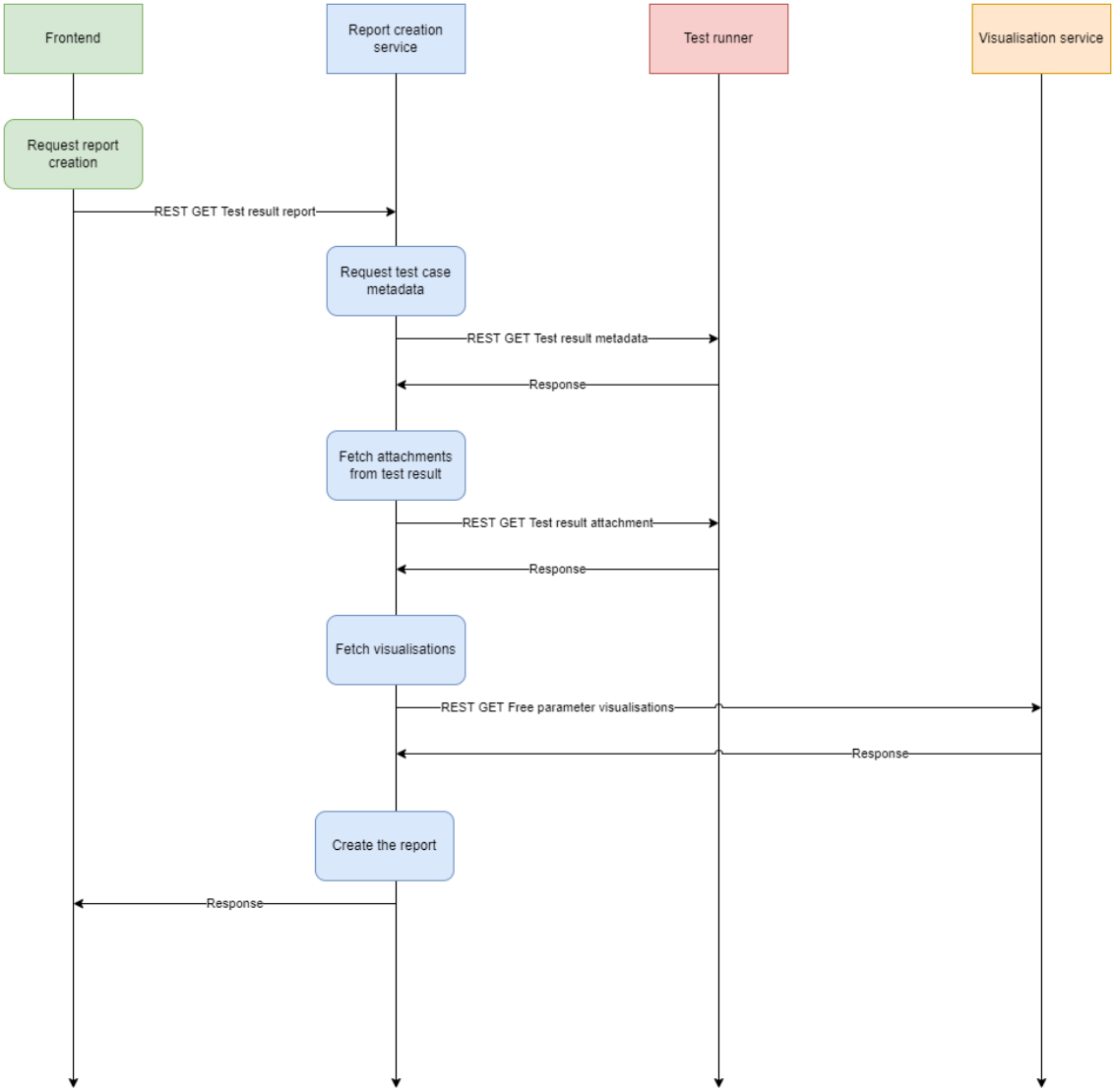
Raportinluontipalvelu vastaanottaa siis linkin syötedataan ja pyytää sen testiautomaatiosovellukselta. Lisäksi palvelu vastaanottaa sisällysluettelon tai sen tiedostonimen. Aluksi palvelu avaa sisällysluettelon ja tarkistaa, onko siihen määritelty visualisointipalvelulta pyydettäviä visualisointeja. Mikäli on, palvelu lähettää viestin visualisointipalvelulle. Viesti sisältää linkin syötedataan, haluttujen visualisointien tyypit sekä halutun ulostuloformaatin, joka voi olla kuva, HTML-sivu tai visualisointifiguuri JSON-muodossa. Tässä tapauksessa pyydetään visualisoinnit raakana JSON-muodossa. Ne saatuaan palvelu tallentaa visualisoinnit kuvatiedostoina sekä interaktiivisina HTML-sivuina. Kuvassa 6 esitetään sekvenssikaaviona raportinluontipalvelun yhteyden muihin palveluihin.

Mahdolliset visualisoinnit haettuaan raportinluontipalvelu hakee testiajoon mahdollisesti lisätyt liitetiedostot, kuten kuvat ja muilla tavoin luodut visualisoinnit. Kuten visualisoinnit, ne tallennetaan tiedostoina raportin resursseiksi.

Seuraavaksi raportinluontipalvelu käy läpi sisällysluettelon sivut käyttäen määriteltyjä sivupohjia. Sivupohjiin täytetään sisällysluettelossa määritelty data. Sivupohja voi olla esimerkiksi dynaamisesti luotava taulukko tai useammasta sivusta koostuva kokonaisuus, joka sisältää esimerkiksi kaikki luodut datan visualisoinnit.

Kun palvelu on luonut sisällysluettelosta Markdown-pohjaisen tekstitiedoston, muuntaa Pandoc sen PowerPoint-tiedostoksi käyttäen määriteltyä mallipohjaa, tai HTML-pohjaiseksi presentaatioksi käyttäen hyväkseen sisällysluettelossa määriteltyä tyylitiedostoa. Luotu raportti annetaan käyttäjälle suoraan ladattavaksi.

Raportinluontipalvelulle kirjoitettiin yksikkötestit, joiden avulla toimivuutta oli helppo testata. Lisäksi testiautomaatiosovelluksen web-käyttöliittymään lisättiin valinnat raportin luomiseksi ja sisällysluettelon muokkaamiseksi.

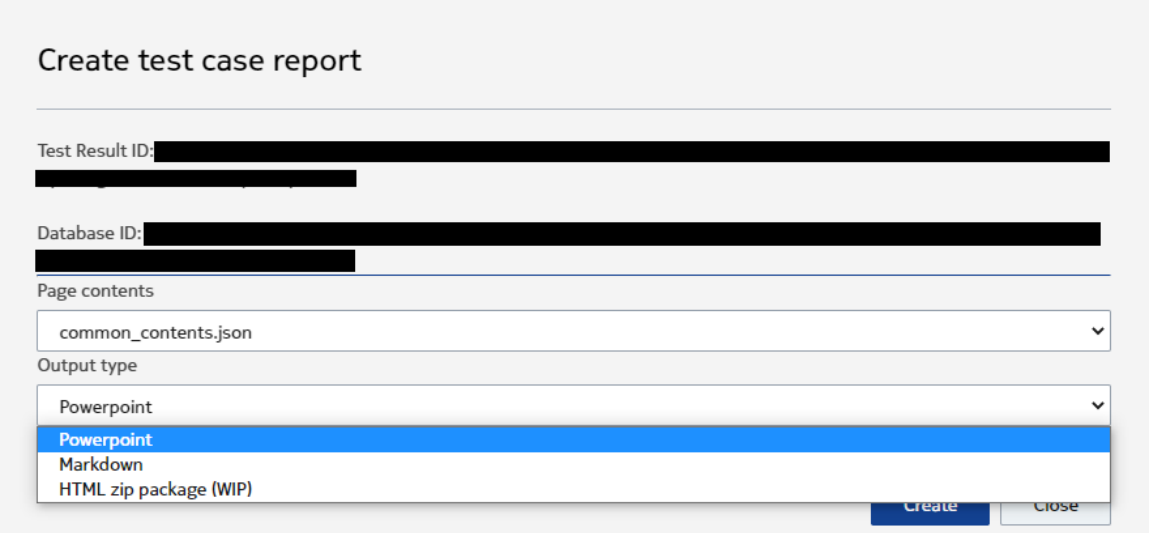


Kuva 6. Sekvenssikaavio raportinluontipalvelun yhteyksistä muihin palveluihin.

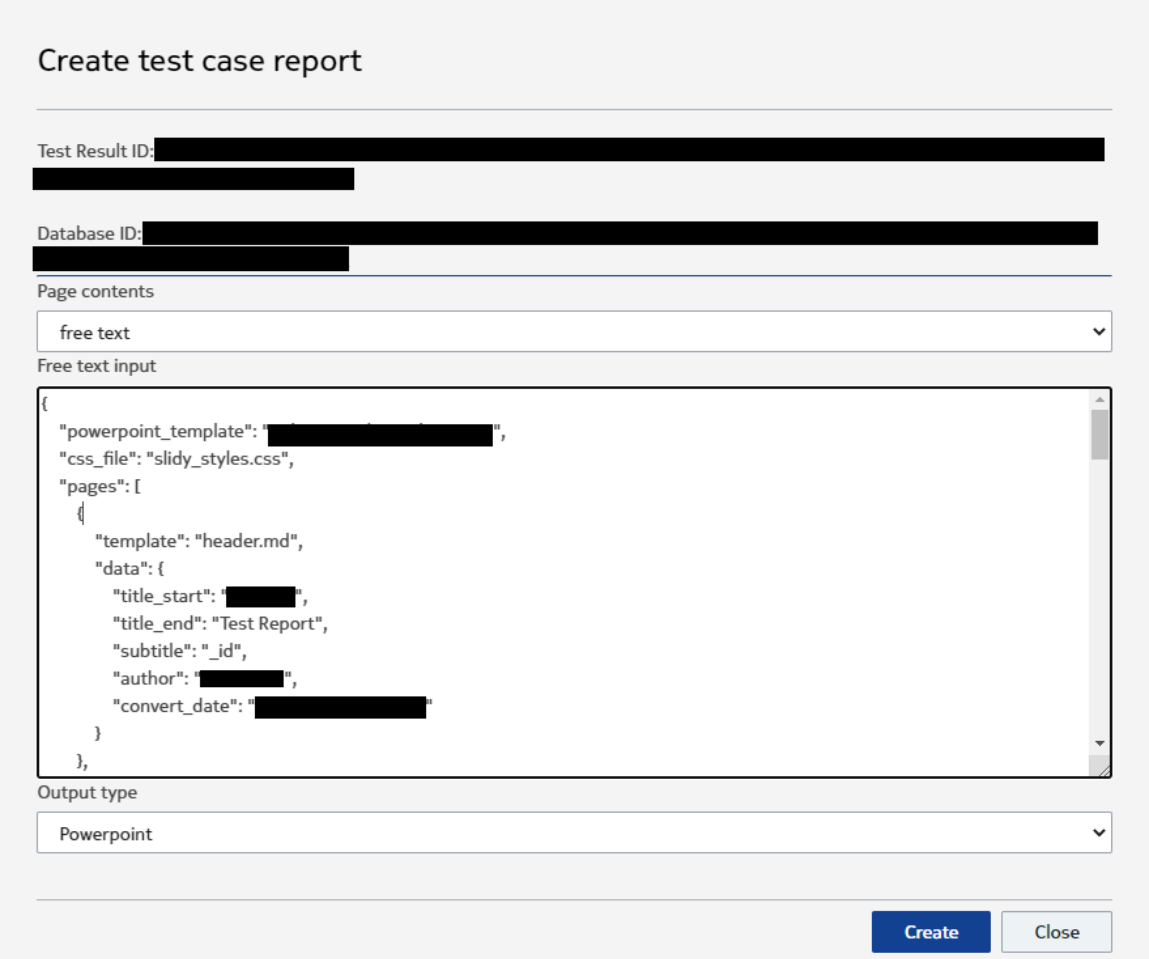
5 Yhteenveto

Raportin kirjoitushetkellä raportinluontipalvelu on julkaistu testaajien käytettäväksi. Palvelu päädyttiin eriyttämään testiautomaatiosovelluksesta siten, että sen ohjelmakoodi on kokonaisuudessaan omassa repositoriossaan. Itse palvelu toimii pilvipohjaisella alustalla, johon eri testilinjastojen testiautomaatiosovellukset ottavat tarvittaessa yhteyttä. Kehittäjän koneella kehitysympäristössä on palvelu mahdollista pystyttää paikallisesti Docker-konttiin. Molemmat mahdollisuudet on huomioitu testiautomaatiosovelluksen konfiguraatiossa.

Palvelun perustoiminnallisuus on kunnossa. Käyttäjä pystyy luomaan raportteja haluamistaan testiajoista käyttäen joko esikonfiguroitua raporttipohjaa tai syöttämällä käyttöliittymään JSON-muotoisena datana muokatun raporttipohjan. Kuvissa 7 ja 8 näytetään raportinluontipalvelun web-pohjaista käyttöliittymää. Palvelu palauttaa muutamassa sekunnissa valitun tyyppisen raportin. Virhetilanteissa palvelu palauttaa käyttäjälle virhekoodin ja geneerisen virheilmoituksen. Palvelun lokeista pystyy näkemään virheen syyn tarkemmin.



Kuva 7. Raportinluontipalvelun käyttöliittymä ja ulostulovalinnat.



Kuva 8. Raportinluontipalvelun käyttöliittymä ja raporttipohjan vapaa muokkaus.

Kun raportinluontipalvelun käyttö alkaa, täytyy kirjoittaa valitulle testityypille sopiva sisällysluettelo, jotta raportti sisältää tarvittavat tiedot. Tämän työn tulee todennäköisesti hoitamaan testaaja, joka tuntee testityypin ja sen erikoispiirteet parhaiten. Testaaja pystyy käyttöliittymän avulla kokeilemaan, minkälaisella sivupohjalla syntyy halutun kaltainen raportti, ja sitten talletta-maan sen ohjelmakoodiin tekemällä lisäyspyynnön raportinluontipalvelun repositorioon.

HTML-pohjaisten raporttien tarjoaminen käyttäjälle jäi keskeneräiseksi. Lopullisena tavoitteena oli alun perin pystyä jakamaan raportteja itsenäisesti tarjottuina sivustoina. Tämä tarkoittaa, että jokaisen raportin tulisi sisältää tarvittavat skriptit, tyylitiedostot, kuvat sekä mahdolliset HTML-pohjaiset visualisoinnit. Kehitysprosessin loppuvaiheessa painoalue haluttiin siirtää PowerPointin toiminnallisuuden takaamiseen ja HTML-pohjainen raportti jäi näin taka-alalle. Lopputilanteena HTML-pohjaiset raportit tarjotaan zip-paketteina, jotka sisältävät kaikki raportin käyttämät kuvat, visualisoinnit, tyylitiedostot ja skriptit.

5.1 Jatkokehityssuunnitelmat

Raportinluontipalvelun vaatimukset ja käyttötarkoitus tarkentuivat jatkuvasti tuotekehitysprosessin myötä. Tällä hetkellä jatkokehityssuunnitelmana on tukea raporttien luomista yksittäisten testiajojen kohdalla. Tämä mahdollistaa nopean vilkaisun testiajon tärkeimpiin osiin, kuten konfiguraatioihin, datan visualisointeihin ja mittausdatasta tehtyyn automaattiseen analytiikkaan. PowerPoint-formaatti takaa myös sen, että raportteja on helppo muokata ja jakaa eteenpäin sidosryhmille.

Mikäli raportinluontipalvelua päädytään kehittämään eteenpäin aktiivisesti, olisi jatkokehityslistalla muutamia ideoita. Esimerkiksi raporttipohjamäärän kasvaessa olisi hyvä miettiä jonkinlaista kategorisointia, jonka mukaan ne olisivat lajiteltuina. Raporttien luontia voisi myös entisestään suoraviivaistaa ja toiminnallisuuden virheensietokykyä parantaa. Suuremmat raportit tultaisiin kuitenkin todennäköisesti jatkossa luomaan toisenlaisella systeemillä, kenties Microsoftin Power BI -alustalla. Työ toteutetaan myöhemmällä ajanjaksolla.

5.2 Itsearviointi ja saatu palaute

Raportinluontipalvelun kehitysprojekti oli haastava tehtävä. Palvelun toteuttamiseksi annettiin suhteellisen vapaat kädet. Mahdollisten toimintatapojen suuri määrä pakotti perehtymään aiheeseen huolella ja käyttämään runsaasti aikaa suunnitteluun sekä asioiden tutkimiseen ja testaamiseen.

Prosessin edetessä luotiin useita eri kaavioita palvelun toiminnallisuudesta ja sen yhteyksistä muihin palveluihin. Nämä kaaviot auttoivat asioiden demonstroinnissa sidosryhmille, ja niiden avulla saatiin hiottua lopullinen toiminnallisuus halutulle tasolle. Joitain asioita, kuten graafinen käyttöliittymä raporttien luomiseksi, päädyttiin hyllyttämään, jotta ydintoiminnallisuus ehtisi valmistua tuotekehitysprojektin puitteissa.

Kun Pandoc valittiin työkaluksi Markdown-tekstitiedoston muuntamiseen PowerPoint-tiedostoksi ja HTML-sivuksi, vei paljon aikaa saada kaikki pienetkin osaset toimimaan yhdessä. Pandoc on erittäin monipuolinen työkalu, mutta myös suhteellisen monimutkainen käyttää tehokkaasti. Etenkin tyylitiedostojen, skriptien ja mallipohjien konfigurointi oli aikaa vievää ja haastavaa saada toimimaan kunnolla.

Sisällysluettelon ja sivupohjien formaattien kanssa joutui myös käyttämään aikaa ja iteroimaan, kunnes sopiva lopputulos löytyi. Jo alusta lähtien oli selvää, että sivuluettelo ja sivupohjat eroteltaisiin toisistaan, mutta molempien formaattia ja sisältöä hiottiin loppuun asti. Tavoitteena oli saada tehtyä sivupohjista mahdollisimman geneerisiä ja sisällysluetteloista taas mahdollisimman helposti muokattavia. Esimerkiksi Jinja2-kirjaston käyttö sivupohjissa tuli mukaan vasta myöhemmässä vaiheessa. Siihen saakka käytettiin sivujen täyttöön varta vasten itse kirjoitettua koodia.

Visualisointipalveluun vaaditut muutokset söivät suuren osan työajasta. Toteutuksen geneerisyys kuitenkin mahdollistaa rajapinnan käyttämisen muissakin yhteyksissä kuin raporttien luomisessa. Haasteellista oli myös palvelun lopullinen julkaisu. Palvelun julkaisemin tuotantokäyttöön vaati perehtymistä muun muassa Kubernetekseen, Terraformiin ja asiakasyrityksen palomuurisääntöihin.

Lopputuloksena syntyi suhteellisen hyvin toimiva palvelu, joka kuitenkin kaipaa vielä jatkokehitystä ja käyttäjätestausta päätyäkseen päivittäiseen käyttöön testausprosessissa. Tietyt vaatimukset, kuten PowerPoint-tiedosto ulostuloformaattina, pakottivat käyttämään tiettyjä työkaluja. Se määritti myös HTML-sivun ulkoasun presentaatiomuotoiseksi. Mikäli raportin ei olisi tarvinnut olla presentaatiomuotoinen, olisi HTML-sivusta voinut luoda huomattavasti interaktiivisemman ja dynaamisemman.

Kaiken kaikkiaan tuotekehitysprosessin myötä saatiin sekä kartutettua opinnäytetyön tekijän ammattitaitoa että ymmärrettiin mihin suuntaan raportointia halutaan jatkossa viedä. Prosessin alussa tämä oli vielä epäselvää, ja siksi työn valmistuminen oli tärkeää.

Raportinluontipalvelusta saatiin runsaasti palautetta jo työn ollessa kesken. Saadun palautteen avulla päädyttiin esimerkiksi hyllyttämään graafinen käyttöliittymä ja siirtämään painoalue kohti PowerPoint-raportin hiontaa. Palautteen avulla myös koodista saatiin hiottua toimivampaa ja taattiin toteutuksen geneerisyys ja monikäyttöisyys.

Loppupalaute sekä työn toimeksiantajalta että asiakasyrityksen edustajalta oli kiitettävää. Työn toiminnallisuus täytti vaatimukset ja dokumentointi oli suoritettu hyvin. Opinnäytetyön raportin teksti oli toimeksiantajan mukaan sujuvaa ja tarina eteni loogisesti opinnäytetyön rakenteita noudattaen. Toimeksiantaja kiitteli kokonaisuutta ja raportin kuvausta työn sisällöstä, toteutuksesta ja lopputuloksesta.

Lähteet

1. Devecto. Gofore ostaa Devecton. 2022 [viitattu: 28.6.2022]. Saatavilla: https://devecto.com/gofore-ostaa-devecton/

2. Luukkainen M. Ohjelmistotuotanto. 2021 [viitattu: 28.6.2022]. Saatavilla: https://ohjelmistotuotanto-hy.github.io/

3. Docker. Docker overview. 2022 [viitattu: 28.6.2022]. Saatavilla: https://docs.docker.com/get-started/overview/

4. AltexSoft. What is Front-End Development: Key Technologies and Concepts. 2020 [viitattu 27.7.2022]. Saatavilla: https://www.altexsoft.com/blog/front-end-development-technologies-concepts/

5. W3 Schools. HTML Introduction. 2017 [viitattu: 27.7.2022]. Saatavilla: https://www.w3schools.com/html/html_intro.asp

6. MDN Contributors. What is CSS. 2022 [viitattu: 27.7.2022]. Saatavilla: https://developer.mozilla.org/en-US/docs/Learn/CSS/First_steps/What_is_CSS

7. MDN Contributors. What is JavaScript. 2022 [viitattu: 27.7.2022]. Saatavilla: https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/What_is_JavaScript

8. Edureka. What is Jenkins. 2022 [viitattu: 27.7.2022]. Saatavilla: https://www.edureka.co/blog/what-is-jenkins/

9. Robocorp. Basic concepts of Robot Framework. 2022 [viitattu: 27.7.2022]. Saatavilla: https://robocorp.com/docs/languages-and-frameworks/robot-framework/basics

10. Jokinen T. Tuotekehitys. Helsinki: Otatieto Oy Yliopistokustannus University Press Finland Ltd HYY-Yhtymä; 2001.

11. Elo A. Tuotekehityksen vaiheet - näin onnistut tuotekehityksessä. 2022 [viitattu 28.6.2022]. Saatavilla: https://matricomp.fi/artikkelit/tuotekehityksen-vaiheet/

12. Meom. Proof of Concept eli PoC on tuotekehityksen ensimmäinen askel. 2022 [viitattu 28.6.2022]. Saatavilla: https://www.meom.fi/venture/proof-of-concept-ja-tuotekehitys/

13. Suomi.fi-verkkotoimitus. Tuotteen jatkokehittäminen. 2022 [viitattu: 27.7.2022]. Saatavilla: https://www.suomi.fi/yritykselle/tuotteiden-ja-palveluiden-kehittaminen/tuotteistaminen/opas/tuotekehitys/tuotteen-jatkokehittaminen

14. Auer A, Auer L, Heinäsmäki M, Hölttä J, Kalliala E, Laanti M, Laine K, Lekman L, Miinalainen P, Naski H, Piiparinen T, Puhakka H, Pyhäjärvi M, Pääkkönen T, Räisänen S, Sora H, Taipale M, Talvio J, Tanninen A, Toikkanen T, Toivola T, Toro K, Valsta A, Väyrynen V, von Weissenberg M. Ketterää kehitystä. Vantaa: Finn Lectura; 2013.

15. Helsingin seudun kauppakamari. Tuotteista tähtituotteita. Viro: Helsingin Kamari Oy: 2016.

16. Ries E. The Lean Startup. New York: Crown Business; 2011.

17. Conrad P. What? Slides? From Markdown?. 2019 [viitattu: 25.10.2022]. Saatavilla: https://stymied.medium.com/what-slides-from-markdown-5239ed31e7ac