Osaamista
ja oivallusta
tulevaisuuden
tekemiseen

Henri Kekkonen

Sovelluksen laadun varmistaminen

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tieto- ja viestintätekniikka

Insinöörityö

20.11.2019

Tekijä Otsikko Sivumäärä Aika	Henri Kekkonen Sovelluksen laadun varmistaminen 52 sivua 20.11.2019
Tutkinto	Insinööri (AMK)
Tutkinto-ohjelma	Tieto- ja viestintätekniikka
Ammatillinen pääaine	Mediatekniikka
Ohjaaja	Yliopettaja Kari Aaltonen

Opinnäytetyössä perehdyttiin sovelluskehitykseen laadun näkökulmasta. Laatu on yksi tärkeimmistä kehitystyön ominaisuuksista, jota monet haluavat mutta jossa harvat onnistuvat. Varmistamalla laadukas sovellus ja sovelluksen kehitystyö hankkeessa säästetään rahaa, aikaa ja hermoja.

Laatu on tärkeä osa onnistunutta sovelluskehitystä. Opinnäytetyössä perehdyttiin siihen, mitä laatu on, mitä hyötyä laadusta on ja miten sovelluskehityksen laatua voidaan parantaa. Opinnäytetyön tavoite oli tarkastella laatua sovelluskehityksen ominaisuutena ja etsiä tapoja kehittää sovellusta laadun näkökulmasta. Laadukkaan sovelluksen tulee olla käyttäjäkokemukseltaan sulava, sisäiseltä rakenteeltaan luettavaa ja ylläpidettävää, ulkoisessa testauksessa toimiva ja projektikehitykseltään laadukkaasti johdettu. Laadukkaasti toteutettuna sovellusprojekti säästää merkittävästi kustannuksissa, sillä asioiden tekeminen kerralla oikein on aina tehokkaampaa ja edullisempaa.

Kehitystiimi voi valita haluamansa laadun parantamisen menetelmät sovelluskehitysprojektiin. Sovelluksen ylläpidon ja ymmärrettävyyden kehittämiseksi pyritään parantamaan koodin luettavuutta ja loogista yhtenäisyyttä sekä säilyttämään koodin kyky kestää jatkokehitystä ja muutoksia. Testaamalla ja vertaisarvioinnilla koodin sisäinen rakenne pidetään yhtenäisenä ja samalla varmistetaan uusien ominaisuuksien toimivuus.

Opinnäytetyö tehtiin yritykselle, joka oli tuottamassa laajaa kehitysprojektia. Opinnäytetyö keskittyi kehitysprojektiin kuuluvaan sovelluskehitysprojektiin, johon sisältyi käyttäjä- ja tapahtumanhallintatyökalu. Sovellusta työstettäessä laadun ominaisuuksia tarkasteltiin testien, projektihallinnan ja koodin työstämisen näkökulmasta.

Kehitysprojektissa huomattiin, että laadun keinoja voidaan soveltaa projektissa monin eri keinoin. Kehitysprojektin laadun parantaminen nopeutti prosesseja ja vähensi sovellusprojektin jälkeenpäin vaatimaa korjailua ja uudelleen työstöä.

Avainsanat	sovelluskehitys, laatu, Angular, Laravel

Author Title Number of Pages Date	Henri Kekkonen Ensuring Software Quality 52 pages 20 November 2019
Degree	Bachelor of Engineering
Degree Programme	Information Technology
Professional Major	Media Technology
Instructor	Kari Aaltonen, Principal Lecturer

In this thesis I look at the quality aspect of software development. Quality is one of the most important attributes that many development teams wish for, but few succeed at. By ensuring quality in the software and in the development process, the project can save money, time and patience.

Quality is a crucial part of successful software development. In this thesis I try to find answer to questions what quality is, what can we benefit of quality and how can we improve the quality of software development. The software should have a good user experience, logical and understandable inner structure, be externally faultless and have a well-organized project management. When successfully implemented, the software project is more cost-effective, as doing things right the first time is always more efficient and economical.

The project team can choose how they want to implement quality features into the software project. The software's maintenance and understandability can be increased by improving the code's readability, logical coherence and maintainability. Testing and peer reviewing the code guarantees that the inner structure will be kept consistent and that new functionalities can be deployed to the code safely.

This thesis was done for a company working on a major development project. Thesis focuses on a user and event management software project that is a part of the larger development project. The properties of software quality are examined in the project by taking a closer look at the tests, project management and code of the project.

The thesis concluded that the properties of software quality can be utilized in many ways. Improving the quality of the software project had a positive impact on the project time frames and decreassed the need for refactoring code afterwards.

Keywords	Software, Quality, Angular, Laravel

Sisällys

Lyhenteet

ORM	Object-relational mapping. Oliomallin mukaisen esityksen kuvaus relaatiomallin mukaiseksi esitykseksi.
TKHJ	Tietokannan hallintajärjestelmä. Ohjelmisto, jonka avulla hallinnoidaan tietokantoja.
PRE	Punaisen Ristin Ensiapu Oy.
CSS	Cascaded Style Sheets. HTML-sivun tyylittelytiedosto.
API	Application programming interface. Ohjelmointirajapinta, jonka avulla esimerkiksi verkkosovelluksissa selain- ja palvelinohjelmistot keskustelevat.
MVC	Model-View-Controller. Malli-näkymä-ohjain. Ohjelmistoarkkitehtuurityyli.
UI	User-Interface. Käyttöliittymä. Käyttäjän näkemä ja käyttäjän kanssa vuorovaikuttava osa sovellusta.
PHP	PHP: Hypertext Preprocessor. Palvelinohjelmointikieli.
HTML	Hypertext Markup Language. Selainohjelmoinnissa käytettävä merkintäkieli.
SQL	Structured Query Language. Relaatiotietokannoissa käytettävä kyselykieli.
DOA	Definition of Done. Valmiin määritelmä. Tehtävän ehdot, mitkä määrittelevät milloin tehtävä voidaan katsoa valmiiksi.
DAKI	Drop, add, keep, improve. Pudota, lisää, pidä, paranna. Projektinhallinnassa käytettävä lajittelutapa.

1 Johdanto

Opinnäytetyössä tarkastellaan sovelluskehityksen laatua ja sen varmistamista. Sen lisäksi mietitään keinoja, miten uutta sovellusta kehitettäessä voidaan varmistaa tuotteen laatu ja kestävä kehitys. Opinnäytetyö on osa laajempaa asiakkaalle valmistettavaa kehitysprojektia. Opinnäytetyössä keskitytään tutkimaan kehitysprojektiin kuuluvaa sovellusprojektia ja sen kehitystä. Työssä tutustutaan myös sovelluksen laadun periaatteisiin ja katsotaan, miten niitä sovellettiin asiakasprojektia kehitettäessä.

Kehitysprojektin tarkoitus on luoda uusi selainpohjainen hallintatyökalu Punaisen Ristin Ensiapu Oy:lle. Hallintatyökalussa PRE:n hyväksymät kouluttajat voivat luoda, hallinnoida ja seurata ensiapukoulutuksia.

Opinnäytetyö tuotetaan Kiwa Inspectalle. Kiwa on suuri kansainvälinen yritys, joka tarjoaa monenlaisia testaus-, tarkastus-, koulutus-, teknologia- ja sertifikaatiopalveluita. Suomessa Kiwa toimii myös nimellä Kiwa Inspecta, kun vanha tarkastuspalveluita tarjonnut Inspecta liittyi Kiwaan vuonna 2015. LIS Group, jossa projektia tuotetaan, liittyi Kiwaan vuonna 2018. LIS Group on erikoistunut tarjoamaan erilaisia tarkastus- ja koulutuspalveluita. [Tietoa Kiwasta 2019; LIS Group Oy 2019.]

Suomen Punainen Risti on Suomen suurin humanitaarista apua tarjoava kansalaisjärjestö. Punainen Risti auttaa katastrofien ja onnettomuuksien sattuessa ja tarjoaa monipuolisesti erilaisia koulutuksia. Suomessa Punainen risti tarjoaa useita erilaisia ensiapukursseja, jotka valmentavat osallistujia onnettomuuksien varalle. [Tutustu Punaiseen Ristiin 2019.]

Punaisen Ristin tavoitteena on ensiapukoulutusten kehittäminen ja koulutettavien määrän kaksinkertaistaminen seuraavien lähivuosien aikana. Kiwa Inspecta ja Punainen Risti aloittivat alkuvuodesta 2019 yhteistyön näiden tavoitteiden saavuttamiseksi. Ensimmäisenä työlistalla on tarve uudistaa Suomen Punaisen Ristin ensiapukoulutusjärjestelmä. [Punainen Risti ottaa suuren kehitysloikan Kiwa Inspectan kanssa 2019; Tietoa Kiwasta 2019.]

Sovellusta käytettäessä saatetaan joskus kuvailla, että tuote on laadukas. Laatu on tuotekehityksessä käytettävä ominaisuus, jolla pyritään ennaltaehkäisemään tuotteen kehityksen ongelmia. Tuotteen kehityksessä ilmaantuvia ongelmia voi olla monia erilaisia, ja ne ilmentyvät projektin eri vaiheissa, esimerkiksi tuotteen tavoitteiden epärealistinen arviointi, tuotteen suunnitteluvirheet ja tuotteen viat. Yleinen harhakäsitys on, että laadukas tuote on synonyymi hyvälle tuotteelle. Laadukas tuote on usein hyvä, mutta hyvä tuote ei ole välttämättä tarkoita laadukasta. Laatu on siis tuotteen vikoja ennaltaehkäisevä ominaisuus, ja laadukas tuote on suunnitelmallisesti toteutettu kokonaisuus, joka täyttää kaikki sille asetetut tavoitteet onnistuneesti. [Crosby 1979: 4,18–20, 135.]

Opinnäytetyössä tutkitaan, kuinka laajan verkkosovelluksen kehittäminen onnistuu projektin alusta ensimmäiseen julkaistuun tuoteversioon. Opinnäytetyössä keskitytään tutkimaan, miten kehitystyötä voisi parantaa laadukkaan sovelluksen tuottamiseksi. Työssä käydään läpi projekti ja sen vaiheet, mutta keskitytään enimmäkseen selainohjelmoinnin kehitystyöhön, koska omat tehtävänkuvani painottuivat enemmän selainohjelmoinnin tehtäviin.

2 Kehitysprojektin taustat

Tässä luvussa käydään läpi PRE:n Kiwa Inspectalta tilaamaa sovellusta ja sen taustoja. Koska projekti on tätä kirjoitettaessa yhä kehitystyön alla, sovelluksesta puhutaan yleiskattavasti menemättä liikaa yksityiskohtiin. Ketterän kehityksen ansiosta PRE oli aktiivisesti mukana kehitysprosessissa.

2.1 Sovelluksen kehityksen tarpeet

PRE:n tavoite on kaksinkertaistaa ensiapukoulutettujen määrä Suomessa. Tätä kunnianhimoista tavoitetta varten järjestö haluaa uusia ensiapukoulutusohjelmien suorittamisen ja pätevyyksien hallinnan. Ensimmäinen askel tähän on uusia verkko- ja lähikoulutuksena toimivat ensiapukurssit.

Punaisen Ristin Ensiapu järjestää ensiapukoulutuksia ja kouluttaa ensiapukouluttajia. Sovelluksen myötä Kiwa aloittaa koulutustapahtumien ja kouluttajien ensiapupätevyyden

laadunvalvonnan. Koulutuksien järjestäminen ja hallinnointi halutaan yhdistää yhden palvelun alle. Uudistuksen myötä olisi tarve myös päivittää käyttäjähallinnointitietokanta. Päivitetyllä tietokannalla PRE voisi saada ajankohtaisempaa tietoa vanhenevista sertifikaateista. [Lehtomäki 2019.]

2.2 Sovelluksen tavoitteet

Sovellusprojektin päämäärä on täyttää valmiilla sovelluksella seuraavat tavoitteet:

- Virallisten koulutusten tasoa ja saatavuutta halutaan parantaa.
- Nykyinen malli halutaan yhdenmukaistaa koulutuksien hallinnointia varten.
- Sovelluksesta halutaan saada laadukas työkalu kouluttajille.
- Ensiaputietoisuutta halutaan lisätä ja tarjota enemmän mahdollisuuksia hankkia ensiapukoulutus. Tätä varten tarvitaan koulutuksien tehokkaampi esille tuonti.

Järjestämällä monipuolisesti lähiopintoja ja verkossa suoritettavia kursseja pyritään tarjoamaan isommalle osalle väestöstä mahdollisuus hankkia ensiapukoulutus varallisuuden ja aikataulun mukaan. Hallinnointityökalun myötä halutaan saada parempi kontrolli ja tietoisuus järjestetyistä koulutuksista, käyttäjärekisteröinnistä ja ensiapukurssin suorittaneista osallistujista. [Lehtomäki 2019.]

2.3 Sovelluskehityksen projektitiimi

Projektia oli tekemässä kooltaan vaihteleva monikansallinen kehitystiimi. Pääkielenä dokumentoinnissa ja kommunikaatiossa projektissa käytettiin englannin kieltä. Ketterän kehityksen menetelmien ansiosta tiimi kommunikoi paljon keskenään, eikä palavereissa pelätty kysyä epäselviä asioita.

Projektitiimiläiset toimivat seuraavissa rooleissa:

- Scrum master toimii kehitystiimin ryhmän vetäjänä, joka toiminnallaan varmistaa scrumin toteutumisen ja opastaa ketterän kehityksen tekniikoita. Scrum master toimii palavereissa seremoniamestarina ja suunnittelee käyttäjätarinoita projektin omistajan ja johtajan kanssa.

- Projektin omistaja valvoo kehitystyössä projektin asiakkaan tavoitteita, kommentoiden projektia asiakkaan näkökulmasta.

- Projektin johtaja valvoo projektin etenemistä ja toimii viimeisenä päättävänä tahona ongelmatilanteissa.

- Käyttöliittymän ja käyttökokemuksen suunnittelija vastaa sovelluksen ulkoasusta.

- Ohjelmoijat vastaavat sovelluksen kehitystyöstä. Kehittäjien määrä vaihtelee neljästä kymmeneen riippuen projektin vaiheesta. Ohjelmoijat jakautuvat lisäksi selain- ja palvelinohjelmoijiin riippuen erityisosaamisestaan. [Lehtomäki 2019.]

3 Kehitystekniikat

3.1 Ketterä kehittäminen

Ketterä kehittämien on iteratiivinen projektin kehitysmalli, jossa projektin julkaisut pilkotaan yhden suuren julkaisun sijasta moniksi pienemmiksi versiojulkaisuiksi. Tämä antaa asiakkaalle kykyä seurata ja vaikuttaa paremmin tuotteen kehitykseen jo prosessivaiheessa. Projektin toimintasuunnitelmiin ja vaatimuksiin palataan joka iteraatiolla, ja toimintasuunnitelma päivitetään aina muutosten tullessa. Näin kehitystyö pysyy joustavana ja valmiina muutoksille. [Agile Coach 2019.]

Ketterän ohjelmistokehityksen pääperiaatteet on tiivistetty neljään osaan Ketterän ohjelmistokehityksen julistuksessa:

> Yksilöitä ja kanssakäymistä enemmän kuin menetelmiä ja työkaluja.
>
> Toimivaa ohjelmistoa enemmän kuin kattavaa dokumentaatiota.
>
> Asiakasyhteistyötä enemmän kuin sopimusneuvotteluja.
>
> Vastaamista muutokseen enemmän kuin pitäytymistä suunnitelmassa. [Manifesto for Agile Software Development 2001.]

Ketterän kehittämisen työtavat saivat juurensa 1940-luvulla Toyotan tehtaalla, kun Taiichi Ohno kehitti järjestelmän, jonka avulla voisi parantaa tehtaan tehokkuutta. Ketterän kehittämisen tekniikoita on sen jälkeen hyödynnetty useissa projektin kehittämiseen tarkoitetuissa rajapinnoissa, kuten scrumissa, extreme programmingissä ja kanbanissa. [Agile Coach 2019.]

PRE-projektia suunniteltaessa projektikehitykseen haluttiin ottaa käyttöön ketterän kehittämisen työtapoja. Ketterän kehityksen rajapinnaksi valittiin sekä scrum että kanban, yhdistäen työtapojen tekniikoita keskenään.

3.2 Työnhallintarajapinta scrum

Scrum on työnhallintarajapinta, joka on kevytrakenteinen ja helppo oppia, mutta vaikea hallita. Scrumin kehittivät sen nykyiseen muotoon Ken Schwaber ja Jeff Sutherland 1990-luvulla, ja sitä on sovellettu maailmanlaajuisesti niin sovelluskehityksessä kuin tutkimuksissa, tuotteen ylläpidossa ja julkaisussa. [Scrum Guides 2018.]

Scrumissa projektin tehtävät on listattu kahdessa tilauskannassa (Backlog). Ensimmäinen tilauskanta on projektin haltijan, ja siinä on hänen haluamiaan ominaisuuksia ja päivityksiä. Toinen on sprint-tilauskanta, joka on aktiivisen sprintin tehtävälista. Tähän tilauskantaan otetaan uusia tehtäviä projektin haltijan tilauskannasta ennen sprintin alkua. Sprintin alkaessa tilauskanta kertoo sen työrupeaman tavoitteen.

Tuotteen kehitys jaetaan kahden viikon sprintteihin, joissa rakennetaan tehtävälista, joka tiimin on sprintin aikana suoritettava. Tehtävänannot pyritään pitämään lyhyinä, 1–2 päivää kestävinä suorituksina. [What is Scrum 2019.]

Scrum voidaan jakaa neljään osioon, jotka ajavat projektin kehitystä eteenpäin:

- Sprinttien suunnittelu (Sprint Planning): palaveri, jossa koko tiimi yhdessä osallistuu suunnittelemaan, mitä tulevassa sprintissä on määrä tehdä.

- Päivittäinen palaveri (Daily Standup): päivittäinen lyhyt kokous, jossa käydään läpi, mitä jokainen on parhaillaan tekemässä, onko jotain ongelmia ilmennyt ja

mitä osallistujat aikovat tehdä seuraavaksi. Kommunikaatio tiimissä on aktiivista ja jopa odotettua. Päivittäiset palaverit pyritään pitämään lyhyinä, noin 15 -minuutin pituisina.

- Sprinttidemo (Sprint Demo): Sprintin lopuksi kehitystiimi käy ryhmässä läpi sprintin tuotokset. Sprinttidemossa esitellään jokainen sprintin aikana valmistunut aikaansaannos. Palaverin tavoitteena on saada koko tiimille kokonaiskuva projektista ja sen tilasta sekä herättää tiimin keskuudessa keskustelua toteutuksesta ja esittää kysymyksiä.

- Sprintin analyysi (Retrospective): Kehitystiimi kokoontuu analysoimaan viime sprintin tuloksia. Sprintin epäonnistumiset käydään läpi ja pohditaan parannuskeinoja, jotta seuraavalla sprintillä onnistuttaisiin välttämään vastaavat kompastuskivet. Palaverissa katsotaan myös sprintin onnistumiset ja miten sama saataisiin toistettua vastaisuudessakin. [What is Scrum 2019.]

3.3 Kanban board, työnhallintajärjestelmä

Kanban on työnhallintajärjestelmä, jota käytetään ketterän kehityksen työympäristössä. Kanban tulee japanin kielestä ja tarkoittaa ”visuaalista työtä”. Nimi on sujuva, sillä kanbanin tehtävä on tehdä tiimin työnhallinnasta visuaalista ja nopeasti ymmärrettävää. Tärkeä osa kanbania on kanban-taulu, joka jakaa tehtävät kategorioihin, joihin tiimin jäsenet päivittävät omien tehtäviensä tilan. Tehtävät kulkevat taululla kanban-korteissa. [What is kanban 2019.]

Osa Jiran ketterän kehityksen palveluita on verkossa toimiva kanban-taulu, jota tiimi ylläpitää yhdessä. Jokainen ryhmän jäsen näkee, mitä toiset ovat parhaillaan tekemässä, onko ongelmia ilmennyt ja mitkä tehtävät ovat valmistumassa. Tehtävän tila jaetaan viiteen osaan: Backlog, In Progress, Blocked, Review ja Done.

Jokainen tehtävä on osa suurempaa yhteen kuuluvien tehtävien kokonaisuutta, tarinaa. Tarinat ovat käyttäjätarinoita, jotka ennen sprinttiä puretaan ja eritellään pienemmiksi tehtävänannoiksi kanban-korteille. Kun kaikki tarinan tehtävänannot ovat valmiit, käyttäjä pystyy toimimaan tarinan mukaisesti ja tarina on valmis. [Jira Software 2019.]

Kanban-kortit määrittelevät tehtävää suorittavalle henkilölle kaiken, mitä tehtävästä tulee tietää:

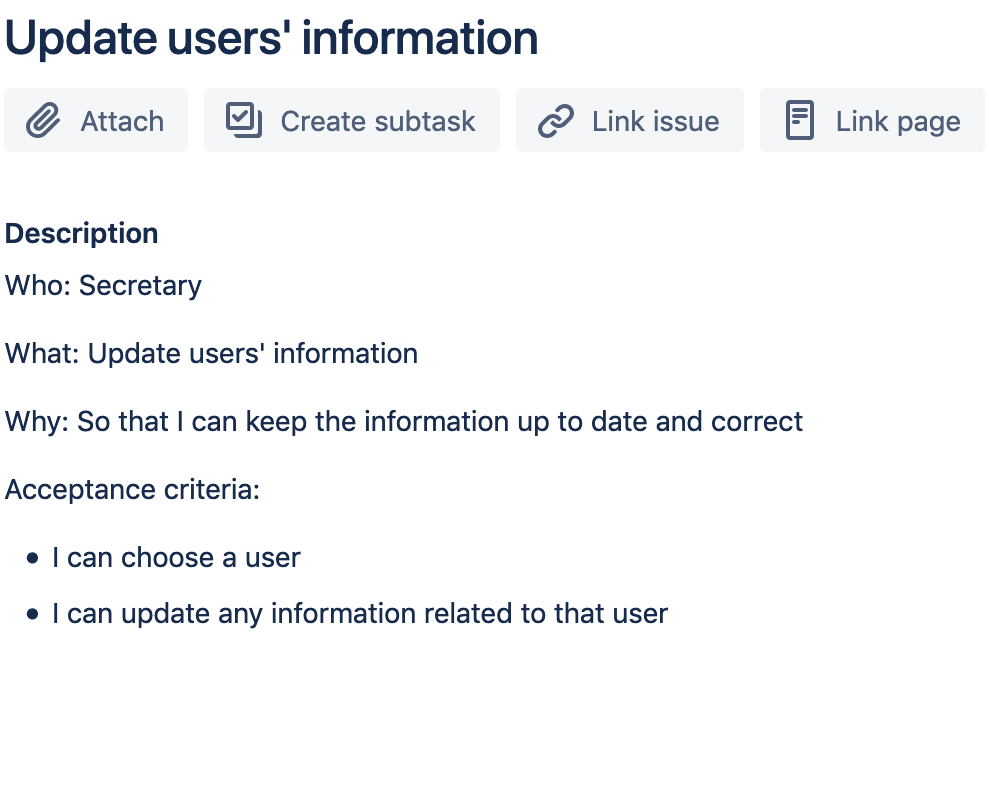
- Tehtävästä vastuussa oleva henkilö. Kortit kertovat, kuka on tehtävästä vastuussa.

- Tehtävän tyyppi. Minkä tyyppinen tehtävä on, esimerkiksi palvelinohjelmointi, selainohjelmointi, suunnittelu vai virhe.

- Tehtävän skaala ja kesto. Tehtävää suunniteltaessa tehdään aika-arvio sen suorittamiseen kuluvasta ajasta. Mitä suurempi tehtävänanto on, sitä enemmän siihen on varattava aikaa. Tehtävä pilkotaan pienemmiksi työtehtäviksi, jos se osoittautuu skaalaltaan liian isoksi.

- DoD (Definition of Done) eli valmiuden määritelmä. Valmiuden määritelmä kertoo, mitä kaikkea tehtävässä pitää olla tehtynä, jotta se voidaan siirtää eteenpäin vertaisarviointiin. [What is kanban 2019.]

Kuvassa 1 näkyy esimerkkinä projektin tehtävänanto. Käyttäjätarina esitellään kortissa vastaamalla kysymyksiin kuka, mitä ja miksi: kuka tarinan käyttäjä on ja missä roolissa hän ohjelmassa toimii, mitä käyttäjä haluaa tehdä ja miksi käyttäjän on pystyttävä tekemään kyseinen tehtävä. Tehtävän valmiuden määritelmässä listataan yksinkertaiset tavoitteet, jotka pitää saavuttaa, jotta tehtävä olisi valmis.



Kuva 1. Esimerkki projektin Käyttäjätarinasta.

Sprintin Backlog sisältää kaikki tehtävät, jotka tiimi on sprintin alussa yhdessä päättänyt ottaa tavoitteekseen. Tilauskannasta jokainen tiimin jäsen voi ottaa itselleen vapaan tehtävän ja siirtää sen In Progress -sarakkeeseen.

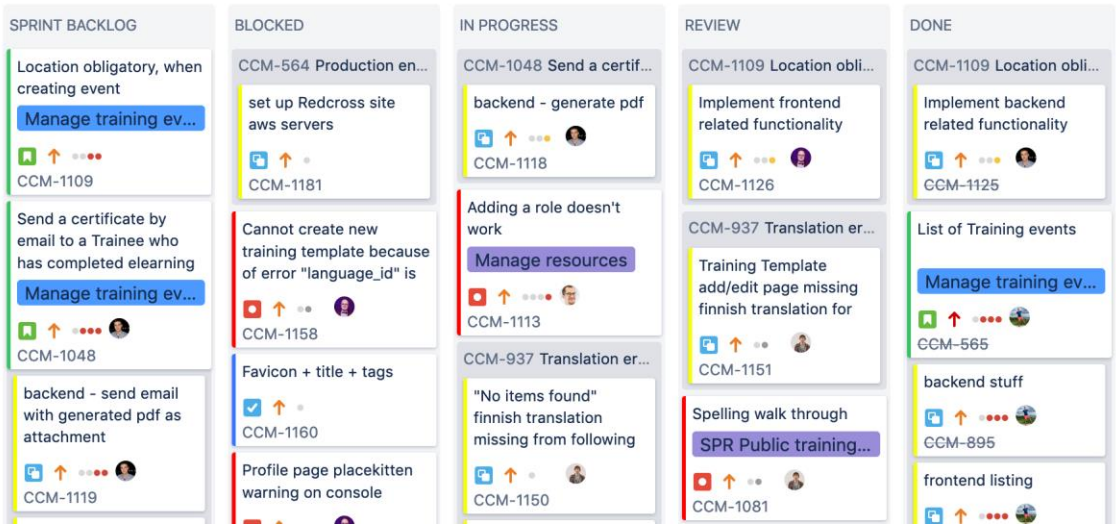
In Progress -sarakkeessa olevat tehtävät ovat parhaillaan kehitteillä. Jokaisessa kortissa näkyy, kuka on ottanut tehtävän omakseen, joten kysymysten tullessa yhteyshenkilö löytyy korttia katsomalla. Kortti voi liikkua In Progress -sarakkeesta kahdella eri tavalla: ongelmien ilmentyessä Blocked-tilaan odottamaan selvitystä tai In Review -sarakkeeseen vertaisarviointia varten.

Blocked -tilassa olevat tehtävät ovat syystä tai toisesta estyneitä. Niiden saamiseksi takaisin työstöön on jokin ongelma ratkottava. Ongelman kohdannut jäsen kirjoittaa ongelmansa tehtävänantoon. Tämä voi olla esimerkiksi jokin toinen tehtävä, joka estää työstämisen, tai mahdollinen virhe, joka on ratkaistava, ennen kuin tehtävään voidaan palata. Kortti siirretään takaisin In Progress -sarakkeeseen, kun ongelma on ratkaistu. [Lehtomäki 2019.]

Tehtävä siirretään Review-sarakkeeseen eli arviointiin, kun se koetaan valmiiksi vertaisarviointia varten. Jokainen tehtävä pitää tarkastaa ja hyväksyä vertaisarvioinnissa. Parannusehdotukset ja kommentit ovat tavallisia ja jopa toivottuja. Jos tehtävä saa vertaisarvioinnissa parannusehdotuksia, se siirretään takaisin In Progress -vaiheeseen. Kun tehtävä läpäisee vertaisarvioinnin, se siirretään Done-sarakkeeseen.

Done-sarakkeessa olevat tehtävät ovat valmistuneet. Sprintin lopussa katsotaan kaikki Done-sarakkeessa olevat tehtävät. Jos tehtävä ei ole sprintin lopussa valmis, se lasketaan keskeneräiseksi ja siirretään seuraavalle sprintille. [Lehtomäki 2019.]

Kuvassa 2 on kesken sprinttiä otettu kuvakaappaus projektin kanban-taulusta. Taulussa näkyy tehtäviä eri sarakkeissa, mikä kertoo nopealla silmäyksellä, missä vaiheessa kukin tehtävänanto parhaillaan on. Kehittäjillä voi olla useampia tehtävänantoja samanaikaisesti: tehtävän ollessa estynyt tai odottaessa arviointia voidaan jo seuraava tehtävä aloittaa.

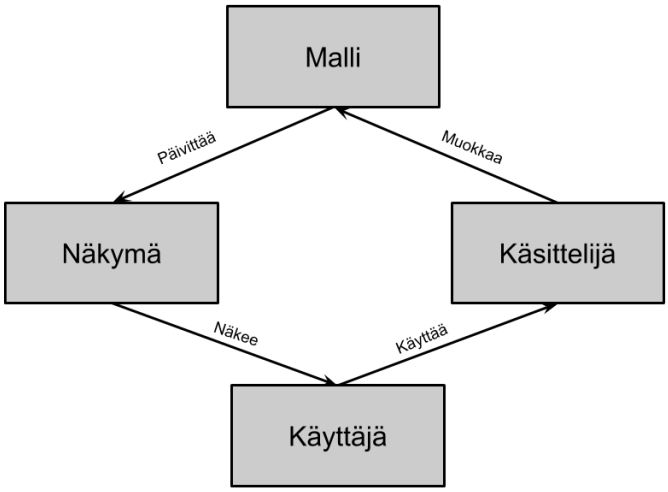


Kuva 2. Projektissa käytetty sprintin kanban-taulu.

3.4 MVC-arkkitehtuuri

MVC-arkkitehtuuri eli malli, näkymä ja käsittelijä (model, view ja controller) on yksi eniten käytetyistä ohjelmistoarkkitehtuurityylistandardeista. MVC-ohjelmistoarkkitehtuurissa projektin järjestelmä jaetaan kolmeen osaan ja tehdään selkeä ero osien välille.

Järjestelmän osat toimivat omina kokonaisuuksinaan ja kommunikoivat keskenään arkkitehtuurin sääntöjen mukaisesti. Tämän avulla ohjelmiston rakenne pysyy selkeänä ja helposti laajennettavana. Kuvassa 3 esitetään kaaviona, kuinka arkkitehtuurin osat keskustelevat keskenään. Mallin, näkymän ja käsittelijän lisäksi kuvaan on lisätty käyttäjä, joka toimii näkymän ja käsittelijän välisenä tiedon siirtäjänä. [Lampinen 2019.]



Kuva 3. Kaavio MVC-arkkitehtuurin toiminnasta [Lampinen 2019].

Malli-komponentti (Model) pitää sisällään kaiken ns. ”pellin alla” olevan logiikan, mitä loppukäyttäjän ei ole tarkoitus nähdä. Komponentti sisältää tietojenkäsittelylogiikan ja ylläpitää ja käsittelee tietokantaa. Malli-komponentti ei tee itsenäisesti mitään, vaan toimii käsittelijästä tulevien pyyntöjen mukaisesti. Mallin sisällä on kaikki toiminnallisuus, joka on yhteydessä tietokantaan, joten tiedon hakeminen, asettaminen ja päivittäminen tapahtuu kaikki mallin kautta.

Näkymä-komponentissa (View) ovat käyttäjän puolella olevat toiminnot. Tämä sisältää UI:n eli käyttöliittymän (User Interface) ja sen sisältämän toiminnallisuuden. Näkymään kuuluvat täten mm. käyttäjän täytettävissä olevat lomakkeet, painikkeet jne. [Lampinen 2019.]

Käsittelijä (Controller) toimii mallin ja näkymän välissä ohjaten näkymästä tulevat viestit malliin ja ohjaten palautteen takaisin. Käsittelijä saa syötteen käyttäjältä ja ohjaa sen

eteenpäin käsiteltäväksi malliin. Käsittelijä päivittää tarvittaessa myös näkymää mallin palautteella.

Sovellusprojekti suunniteltiin noudattamaan MVC-arkkitehtuuria. Selainohjelmistoon on Angularin avulla rakennettu näkymä ja käsittelijä, ja palvelinohjelmistossa on Laravelin tukemat käsittelijä ja malli. Angular ja Laravel molemmat tukevat MVC-arkkitehtuuria. Selain- ja palvelinohjelmistolla on molemmilla omat käsittelijänsä, koska ne toteutettiin omina itsenäisinä kokonaisuuksinaan. [Lampinen 2019.]

4 Selainohjelmoinnin teknologiat ja palvelut

Projektin kehitystyössä käytettiin laajaa kirjoa teknologioita ja palveluita. Tässä luvussa esitellään teknologioita, niiden käyttötarkoituksia ja syitä, miksi ne valittiin käytettäväksi projektissa. Teknologioiden käyttöä projektissa käsitellään luvussa 8.

4.1 Angular-ohjelmistokehys, selainsovelluksen sydän

Angular-ohjelmistokehys on vuonna 2016 julkaistu Googlen omistama avoimen lähdekoodin ohjelmistokehys. Angularin avulla voidaan rakentaa selainohjelmisto noudattaen MVC-mallin periaatteita HTML:llä (Hypertext Markup Language) ja TypeScriptillä. Angular on rakennettu TypeScriptillä, ja se tukee kolmannen osapuolen JavaScript-kirjastoja. [Angular Docs 2019.]

Angularin avulla kehittäjä pystyy laajentamaan HTML-merkintäkieltä ja tekemään sivustosta järjestelmällisen, luettavan ja dynaamisen. Angularin avulla on mahdollista saada loppukäyttäjän selaimen käyttöön työkaluja ja tekniikoita, joiden käyttö on aiemmin ollut mahdollista vain palvelinpuolella. Angular on vahvimmillaan sivustoissa, jotka on suunniteltu yksisivuisiksi ohjelmistoiksi (Single-page Application). [Pro Angular 6 2017.]

Käyttäjän navigoidessa web-sivustolle hän saa palautteena HTML-dokumentin. Toisin kuin tavallisessa sivustossa, jossa käyttäjän navigoidessa sivusto ladataan aina uudelleen, yksisivuisessa ohjelmistossa käyttäjä saa vastaukseksi täydentäviä HTML-dokumentin palasia, jotka päivitetään suoraan sivustolle vaatimatta sivuston päivittämistä.

Sivuston asynkroninen päivittäminen vähentää käyttäjän odottamista ja saa sivuston vaikuttamaan suoraviivaisemmalta ja nopealta.

Angular on suuri kokonaisuus, jonka hyödyt alkavat näkyä suurissa projekteissa, jotka haluavat implementoida tuotteestaan yksisivuisen ohjelmiston ominaisuuksia. Angular on rakennettu noudattamaan MVC-arkkitehtuuria, joka pitää ohjelmiston osat helposti hallittavassa järjestyksessä. [Pro Angular 6 2017.]

Projektin selainohjelmoinnin päätyökaluksi valittiin Angular-ohjelmointikehys, koska se oli valmiiksi tuttu koko kehitystiimille. Ohjelmistokehyksen etukäteen tietäminen nopeutti projektin kehitystyötä, sillä kehittäjillä oli valmis ymmärrys ohjelmistokehyksen työkaluista, ominaisuuksista ja työtavoista. Projektissa käytetään Angularin versiota 8.0.

4.2 TypeScript Angularin avustajana

TypeScript on vuonna 2012 julkaistu avoimen lähdekoodin ohjelmointikieli, joka on rakennettu JavaScriptin jatkeeksi. TypeScriptin kieli mukailee suurilta osin JavaScriptia muutamia poikkeuksia lukuun ottamatta. Nimensä mukaisesti TypeScriptin näkyvin ominaisuus on sen tuoma parametrien tyypitys. JavaScriptissä ei oletuksena ole tyyppejä, vaan ohjelmointikieli itse vaihtaa parametrin tyyppiä tarvittaessa. TypeScriptissä ohjelmoija itse määrittelee parametrin tyypin sitä luotaessa. Kovalla tyypityksellä pystytään välttämään arvojen sekoittumisista johtuvat virheet ohjelmiston koon kasvaessa.

TypeScript ajetaan sivustoa kootessa puhtaaksi JavaScriptiksi, joten se tukee JavaScriptin omia ja kolmannen osapuolen kirjastoja. [TypeScript Documentation 2019.]

Projektissa käytetään TypeScriptin versiota 3.4. Ohjelmointiympäristön avustajana käytettiin TSLint-työkalua oikolukemaan ja huomauttamaan ohjelmointikielen virheistä ja parantamaan luettavuutta.

4.3 SCSS ja Bootstrap helpottamaan tyylittelyä

SCSS (Syntactically Awesome Style Sheet) eli toiselta nimeltään Sass on CSS (Cascading Style Sheets) tyyliohjeiden esiprosessori. SCSS käyttää CSS:n mukaista syntaksia ja laajentaa sitä konventioilla, jotka ovat monille jo tuttuja muista ohjelmointikielistä. Nämä ominaisuudet ovat mm. koodin kommentointi, parametrit ja funktiot. Niiden avulla tyyliohjeiden kirjoittaminen on nopeampaa ja johdonmukaisempaa. Sivustoa käsiteltäessä Sass käännetään puhtaaksi CSS:ksi. [Sass, CSS with superpowers 2019.]

Bootstrap on avoimen lähdekoodin CSS-kehys, jonka tarkoitus on tehdä CSS:n käytöstä nopeaa ja intuitiivista. Bootstrap tuo mukanaan laajan tyylikirjaston, jonka avulla erillisen CSS-tiedoston sijaan ohjelmoija voi määritellä tyylejä suoraan elementin luokkaan. Bootstrapissä on valmiiksi rakennettu sivustokehikko, joka antaa ohjelmoijalle pohjan, jonka avulla asetella sivuston elementit dynaamisesti sivustolle. Bootstrap on rakennettu tukemaan responsiivisuutta, mikä helpottaa sivuston suunnittelua sekä työpöytäselaimelle että mobiililaitteelle.

Bootstrapista on saatavilla Angular-ohjelmistokehykselle suunniteltu Ng-Bootstrap, joka tuo Bootstrapin toiminnallisuudet saataville Angular-ympäristöön. Ng-Bootstrap sisältää Bootstrapiin pohjautuvan Angular-natiivin kirjaston ja sovittaa Bootstrapin JQueryä ja JavaScriptiä vaativat toiminnallisuudet Angulariin yhteensopivaksi. [Bootstrap widgets the angular way 2019.]

Projektissa käytetään pääasiassa Bootstrapia sivuston ulkoasun luomiseen. Sen lisäksi tarvittaessa yksityiskohtia hiotaan tapauskohtaisesti SCSS:n avulla.

4.4 Cypress kattavaan testaukseen

Cypress on avoimen lähdekoodin moderni end-to-end-testausympäristö. Cypress on laajasti tuettu täysi testauskirjasto, joka ei tarvitse muita testausohjelmistoja avukseen. Cypress mukailee loppukäyttäjän toimintaa etsien painikkeita, täyttäen lomakkeita ja navigoiden pitkin sivua. Cypress on rakennettu käyttäjälähtöisesti, pääpyrkimyksenä tehdä testaamisesta mahdollisimman helppolukuista ja nopeaa kirjoittaa.

Cypres pitää tallessa historiansa, mikä antaa käyttäjälle mahdollisuuden käydä läpi testin vaiheet pala palalta. Loki näyttää testaajalle kaikki käyttäjäympäristön sivut. Virheitä korjatessa käyttäjä pystyy katsomaan virhelokin lisäksi käyttöliittymästä suoraan, missä virhe on tapahtunut. Cypress pystyy tallentamaan testeistä tulleen datan tulokset ja ottamaan kuvakaappauksia ja videoita testeistä. [Cypress 2019.]

Projektissa selainohjelmiston testaus otettiin käyttöön verrattain myöhään. Projektiin valittiin testausympäristöksi Cypress, koska se todettiin helposti kirjoitettavaksi ja ymmärrettäväksi. Angular käyttää oletuksena Jasmine-testiympäristöä ja testien suorittamiseen Karma-testiohjelmaa [Angular Testing 2019]. Jasmine ei ollut entuudestaan tuttu suurelle osalle kehitystiimiä. Angularin ohjelmoijien oli helpompi ottaa käyttöön Cypress kuin kirjoittaa testit Jaminella.

5 Palvelinohjelmiston teknologiat ja palvelut

Palvelinohjelmisto on web-sovelluskehityksen puolisko, joka on vähemmän näkyvillä mutta jonka toiminnallisuus on sitäkin suurempi. Tässä luvussa käydään läpi teknologioita ja palveluita, joita käytettiin projektikehityksessä. Luvussa keskitytään erityisesti tutkimaan projektin kehitystyön kulmakivenä toiminutta Laravel-ohjelmistokehystä.

5.1 PHP palvelinpuolen kielenä

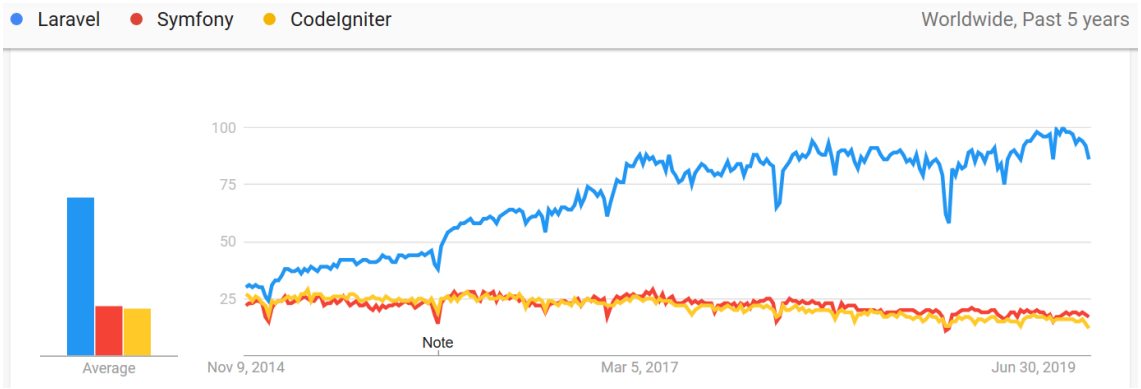
PHP eli PHP: Hypertext Preprocessor on vuonna 1995 julkaistu vapaan lähdekoodin ohjelmointikieli, jota käytetään laajasti verkkosivujen suunnittelukielenä.

Projektissa PHP on palvelinpuolen pääohjelmointikieli, joka ylläpitää koko palvelinohjelmiston toiminnallisuutta. PHP oli selvä valinta projektiin, sillä sen kanssa työstämisestä kehitystiimillä oli jo valmiiksi vankka kokemus. Lisäksi näin päästiin käyttämään PHP:ta hyödyntävää ohjelmistokehystä Laravelia. Projektissa käytetään testaukseen Phpunit-testausrajapintaa. Kehitystiimin PHP-kielen yhtenäisyyden varmistamiseen käytetään PHPCodeSnifferiä.

5.2 Laravel-ohjelmistokehys helpottamaan palvelinohjelmointia

Laravel on vuonna 2011 julkaistu vapaan lähdekoodin PHP:lle tarkoitettu web-sovellusohjelmistokehys. Se on rakennettu valmiiksi kaikenkattavaksi pakkaukseksi, joka tukee MVC-mallista kehitystyötä. [Laravel 2019.]

PHP:lle on saatavilla useita erilaisia ohjelmistokehyksiä, kuten Symfony, CodeIgniter ja Laravel. Kuten kuvasta 4 nähdään, Laravelin suosio on kuitenkin jatkanut tasaisesti nousuaan verrattuna muihin PHP:n ohjelmistokehyksiin [Compare Laravel, Symfony, CodeIgniter 2019]. Laravelin suuri suosio heijastuu suoraan aktiivisena kehittäjäyhteisönä [Laravel from Scratch 2019].



Kuva 4. Google Trendsin Symfonyn, CodeIgniterin ja Laravelin suhteutettu suosio [Compare Laravel, Symfony, CodeIgniter 2019].

Laravelin ympäristö on rakennettu toimimaan MVC-mallin mukaisesti käyttäen malleja, käsittelijöitä ja näkymiä. Laravel käyttää hyödykseen myös OOP:n (Object-oriented Programming eli Olio-ohjelmoinnin) pääperiaatteita, jossa MVC:n osat on luotu oliorakenteella ja ne kommunikoivat näin toistensa avulla. [What is Laravel Framework 2019.]

Laravelissa tulee mukana laaja autentikointikirjasto, joka tekee käyttäjähallinnasta nopeaa ja helppokäyttöistä. Laravelin autentikointijärjestelmä koostuu kahdesta osasta: suojista (Guards) ja tuottajista (Providers).

Suojat varmistavat käyttäjän oikeudet jokaiseen tämän esittämään pyyntöön. Esimerkkinä Laravelissa on asennettaessa oletuksena mukana session suojaus, joka sisältää session tilan ja evästeiden tilan ylläpitäminen. [Laravel Authentication 2019.]

Tuottajat määrittelevät, kuinka käyttäjätietoja haetaan tietokannasta ja miten tietoa käsitellään. Laravelissa on valmiiksi mukana laajamittainen tietokantojen kyselyrakentaja-komponentti Eloquent ORM (Object Relational Mapping). Eloquent-komponentti toimii mallina Laravelin ja tietokannan välillä ja tekee tietokannan tauluista olioita, joille voidaan tehdä kyselyitä olio-ohjelmoinnin mukaisesti. Eloquent on tietokannasta riippumaton, joten se toimii samalla lailla kaikilla tuetuilla tietokantaohjelmistoilla. Tietokantamalli tarvitsee vain määritellä sovelluksen konfiguraatiotiedostossa [Eloquent ORM 2019]. Eloquent yhdistää haluttuun tietokantaan asetusten mukaisesti ja hoitaa tietokannan kanssa keskustelun tietokantamallin mukaisesti. Käyttäjän ei tarvitse itse muuttaa ”pellin alla” tapahtuvia tietokantakutsuja. [Laravel from Scratch 2019; Laravel Authentication 2019.]

Vaikka Laravel pystyy tarjoamaan käyttöön koko MVC-arkkitehtuurin, projektissa kuitenkin päädyttiin käyttämään Laravelia RESTful API-ohjelmointirajapintana, joka dokumentoitiin Stoplight-palvelun avulla. Näin sovelluksen palvelin- ja selainpuoli pystyttiin erottamaan omiksi irrallisiksi osuuksikseen, joita pystytään kehittämään puuttumatta toiseen. Palvelinpuolella Laravel kuitenkin pitää arkkitehtuurin mukaisesti malleja ja niiden käsittelijöitä. Laravelin malleissa on tärkeää pitää tarkkaa huolta joukkomäärittelyn säännöistä, sillä projektissa käytetään paljon kutsuja, joissa haetaan suuria datamääriä. Käsittelijöihin on liitetty datan validointi ja käyttäjävaltuuttaminen. Vain näkymä ja näkymän käsittelijät on siirretty selainpuolelle Angularin alle. Angularin ja Laravelin käsittelijät kommunikoivat keskenään siirtäen dataa loppukäyttäjän ja palvelimen välillä. Laravelin kanssa käytettiin mallien ja ohjainten lisäksi palvelutasoa (Service) auttamaan ohjainten toimintaa sekä muuntajat-tasoa (Transformer) muokkaamaan ohjainten palautedata oikeaan muotoon.

Laravelin kanssa käytettiin muutamaa kolmannen osapuolen kirjastoa nopeuttamaan ohjelmistokehyksen kanssa työskentelyä. Näistä tärkeimmät ovat Composer ja Artisan. Composer on ulkoisten kirjastojen riippuvuuksien käsittelyyn tehty hallintatyökalu. Artisanin avulla pystytään tekemään koodista migraatioita suoraan tietokantaan. Projektin kehittyessä tietokantaan tulee yhtenään muutoksia. Artisan vähentää tietokannan muutoksista johtuvia virheitä yhtenäistämällä jokaisen kehittäjän tietokannan jokaisen päivityksen yhteydessä. [Lampinen 2019.]

6 Projektin julkaisu, dokumentointi ja hallinnointi

Sovelluskehitysprojektin koon kasvaessa sen hallinnointi saattaa nopeasti vaikeutua. Projektin kehitystyön keskinäinen osa on kattava ja ajankohtainen dokumentaatio. Ohjelmoijien valmistama lopullinen koodi voi olla vain yhtä hyvää kuin heitä ohjeistanut dokumentaatio. Sovelluskehityksessä myös projektin ohjaamisen pitää olla sujuvaa. Tässä luvussa esitellään työkaluja, joilla tähän tavoitteeseen pyrittiin.

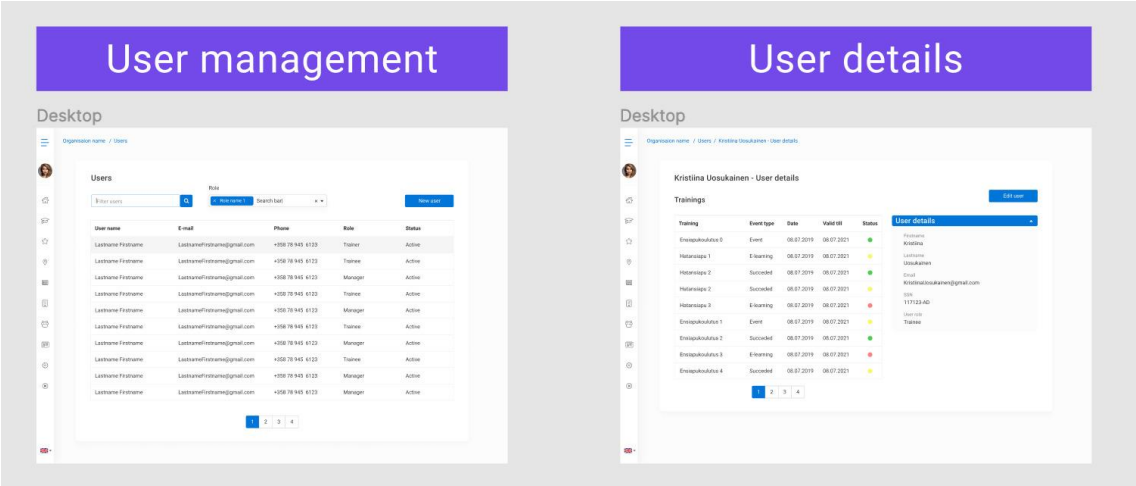
6.1 Projektin julkaisu ja käyttöönotto

Projekti julkaistiin käyttäen Amazonin Simple Storage Servicea (Amazon S3). Projektien versioiden julkaisu hoidettiin Envoyer-käyttöönottopalvelun avulla. Envoyer on PHP-ohjelmistojen käyttöönottoon tarkoitettu palvelu, jonka avulla sivuston julkaisu tehdään asteittain ja turvallisesti ilman häiriöaikaa [Envoyer – The PHP deployer is now launched 2016].

6.2 Figma, suunnittelun apuväline

Figma on koko kehitystiimille tarkoitettu selaimessa toimiva suunnitteluohjelmisto. Sen avulla koko tiimi pystyy ottamaan osaa ja kommentoimaan käyttäjäympäristön suunnittelua. Figma on rakennettu toimimaan selaimessa osana kehitystyökalujen sarjaa. Se ei vaadi käyttäjältä asentamista tai jatkuvia päivityksiä, mutta siitä on saatavilla myös työpöytäsovellus. [Figma 2019.]

Kehitystiimin käyttöliittymän suunnittelija käytti Figmaa projektin suunnittelussa. Jokaisesta ohjelmiston näkymästä on rakennettu kuvan 5 mukainen suunnitelma, joka on projektin Figma-sivulla. Jokaisella kehitystiimin jäsenellä oli mahdollisuus käydä katsomassa ja kommentoimassa näkymien sommittelua.



Kuva 5. Esimerkkinäkymä projektin Figma-sivun näkymistä.

6.3 Stoplight dokumentoinnin avustajana

Stoplight on ohjelmointirajapinnan eli API:n (Application Programming Interface) suunnitteluun luotu hallintaohjelmisto, jonka tarkoitus on helpottaa selain- ja palvelinohjelmoinnin välistä työskentelyä. Stoplightin tarkoitus on koota projektin API-dokumentaatio helposti tiimin saataville yhteen paikkaan.

Projektissa tiimin koon ja tehtävien jakautumisen vuoksi selkeälukuinen API-dokumentaatio oli hyvin tärkeää. Kun API-päätepisteet määritellään jo suunnitteluvaiheessa, se vähentää kommunikoinnin epäselvyyksistä johtuvia virheitä palvelin- ja selainohjelmistojen välillä. [Stoplight 2019.]

6.4 Jira Software- ja Confluence-tehtävänhallintatyökalut

Confluence ja Jira Software ovat molemmat Atlassian omistamia selaimessa toimivia tehtävänhallintatyökaluja. Jira Software tarjoaa kehitystiimille kaikki ketterässä kehityksessä tarvittavat ominaisuudet ja työkalut julkaisusuunnitelmista kanban-tauluun. Confluence on projektin dokumentointityökalu, jonka avulla kehitystiimi voi jakaa tietoa ja hallinnoida projektin dokumentaatiota. [Atlassian Products 2019.]

Projektissa Jira Software ja Confluence olivat projektihallinnoinnin ja tiimityöskentelyn kulmakiviä. Projektin aikataulun hallinnointi ja dokumentointi saatiin pidettyä ajan tasalla ja koko ajan käden ulottuvilla.

6.5 Gitlab versiohallintaratkaisuna

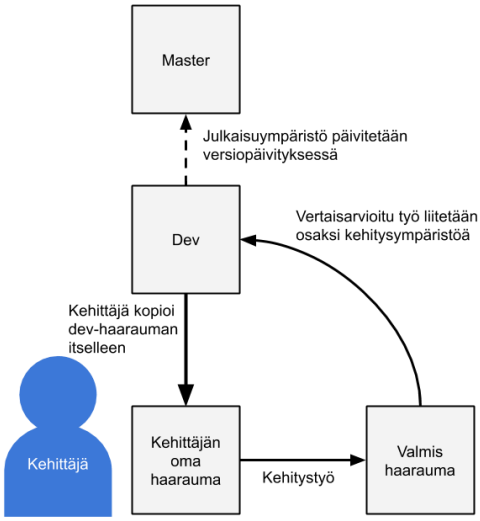
GitLab on vuonna 2011 julkaistu avoimen lähdekoodin DevOps-toimintamallin kehitysalusta. Se tarjoaa kattavan sovelluskehittämisen hallinnointiin tarkoitetun pilvipalvelun yhdessä sovelluksessa. GitLab:in versiohallinta pohjautuu Git-versiosäilöntäjärjestelmään [What is GitLab 2019; Git 2019]. DevOps (Software development and information-technology operations) on yhdistelmä tekniikoita ja työtapoja, joiden tarkoitus on automatisoida projektin julkaisuprosesseja, helpottaa kehitystyötä ja parantaa laadunvarmistusta [Mikä on DevOps 2019].

Projektia varten kehitystiimi pohti kahden vaihtoehdon välillä: GitHub ja GitLab. GitHub oli tiimille tuttu aikaisemmista kehitystöistä, mutta lopulta GitLab todettiin palveluiltaan kattavammaksi ja valittiin projektihallintatyökaluksi. GitLabin tärkein tehtävä projektissa on versionhallinta.

Projektin versiohallinnassa käytetään kahta eri ympäristöä: dev- eli kehitysympäristö ja master- eli julkaisuympäristö. Sekä master- että dev-ympäristöjen ohjelmistoista on verkossa testattavana olevat versiot. Kehitysympäristöä käytetään kehitystyössä ja projektin uusien komponenttien testauksessa. Julkaisuympäristö on asiakkaan käytössä oleva vakaa ohjelmaympäristö.

Projektissa versiohallinnan käyttökartta on kuvan 6 mukainen: Kehittäjä luo tiketilleen kehitysympäristöstä oman haarauman (branch), jota hän työstää lokaalisti. Kun kehittäjä on valmis siirtämään tehtävänsä vertaisarvioitavaksi, hän puskee haaraumansa versiohallintaan. GitLabissa kehittäjä luo yhdistämispyynnön (merge request) dev-ympäristöön ja jakaa pyynnön muille kehittäjille. Toinen kehittäjä tutustuu tikettiin ja kommentoi mahdollisista parannusehdotuksista. Kun yhdistämispyyntö on todettu tavoitteet täyttäväksi ja toimivaksi, arvioija merkitsee pyynnön hyväksytyksi. Kehittäjä voi nyt yhdistää haaraumansa yhteiseen kehitysympäristöön. Kehitysympäristö pusketaan julkaisuympäristöön,

kun sprintin tavoitteet on saavutettu ja versio on valmis luovutettavaksi ulkopuolisille tahoille.



Kuva 6. Projektin versiohallinnan käyttökartta.

Jos kehittämisen aikana dev-ympäristöön on tehty muutoksia, on mahdollista, että kehittäjän haaraumassa on päällekkäisyyksiä nykyisen ympäristön kanssa (merge conflicts). Tällöin kehittäjä voi korjata päällekkäisyydet itse tai uudelleen pohjustaa (rebase) haaraumansa kehitysympäristöön. [Lampinen, 2019.]

7 Sovelluksen laatu

Sovellukset ovat monimutkaisia ja hankalia kehittää. Sovellusprojektin onnistumisen määrittelee kolme kantavaa tekijää: laatu, aika ja hinta. Toisin kuin ajan ja hinnan, laadun korjaaminen on hidas ja vaikea prosessi. Huonolaatuinen sovellus saattaa rampauttaa koko projektin ja vaatia korjaamiseen mittavia taloudellisia ja ajallisia ponnisteluja. [Spinellis 2016.]

Uutta tuotetta valmistettaessa halutaan aina, että tuote on laadukas, mutta tähän pyritään ilman tarkkaa suunnitelmaa laadun takaamiseksi. Laatu vain mielletään tuotteen abstraktiksi ihanteeksi, mihin kehitettäessä pyritään. Laatu on kuitenkin tuotteen

mitattava ominaisuus ja mittayksikkönä toimii raha. Mittaamalla tuotteen kehityksessä koituneita kustannuksia virheiden ehkäisystä, valvonnasta ja vikojen korjaamisesta, voidaan saada hyvä kuva tuotteen todellisesta laadusta. Kitkemällä tuotantovaiheen ongelmia ja parantamalla kehitystyötä säästetään arvokasta työaikaa ja vältetään myöhemmin syntyviä ongelmia, jotka viivästyttävät tuotteen julkaisua ja hankaloittavat tavoitteiden saavutusta. Laadukas työ säästää rahaa, sillä asioiden tekeminen kerralla oikein on aina halvempaa. [Crosby 1979: 4,18–20, 135.]

Philip B. Crosby tiivistää teoksessaan Laatu on ilmaista laadun käsitteen seuraavasti:

> Laatu on kokonaisuus, joka voidaan saavuttaa, jota voi mitata, joka on kannattava ja joka voidaan ottaa käyttöön heti kun kaikki ovat mukana, ymmärtävät sen merkityksen ja ovat valmiita tekemään lujasti töitä [Crosby 1979: 6].

7.1 Laadukkaan sovelluksen ominaisuudet

Laatu syntyy aktiivisen toiminnan tuloksena. Suunnitteluvaiheessa määritellään tulevan tehtävän tavoitteet ja laadun suuntaviivat. Hyvällä ja kattavalla suunnittelulla vältetään ohjelmoijan tulkinnan varaan jäävät mustat aukot tehtävässä. Tehtävän suunnitelma on valmis toteutukseen, kun tehtävän tavoitteet ja valmiuden määritelmä eli DOD (Definition of Done) ovat selkeät kaikille osapuolille. [Crosby 1979: 80.]

Laadukkaan sovelluksen ominaisuudet voidaan jakaa neljään eri luokkaan:

1. Käytön laatu (Quality in use) eli käyttäjäkokemus (User experience). Käytön laadussa tutkitkaan, miten loppukäyttäjä kokee tuotteen. Onnistuneessa käyttäjäkokemuksessa käyttäjä voi saavuttaa haluamansa tuloksen käyttäessään sovellusta normaalisti. Yleensä käytön laadussa laatu tarkastetaan hallitussa ympäristössä jokin tietty päämäärä mielessä. Sovelluksen pitäisi toimia laadukkaasti ja ongelmitta niin kauan, kuin loppukäyttäjä toimii näiden vaatimusten sisällä. Jos käyttäjä poikkeaa suunnitellusta tavasta, saattaa käyttäjäkokemus olla täysin erilainen. Käyttäjäkokemusta kehitettäessä keskitytään varmistamaan virheetön ja sujuva käyttö ja jätetään monimutkaisempien virheiden selvitys toissijaiseksi, jos käyttäjä ei törmää niihin normaalissa käytössä.

2. Ulkoiset laatuvaatimukset (External quality attributes). Ulkoisissa laatuvaatimuksissa tutkitaan, miltä tuote vaikuttaa ulkoisesti. Tuotetta käydään läpi katsoen, näyttävätkö sen osat toimivilta ja toimiiko sovellus oletetusti kompastelematta. Sovelluskehityksessä tämä voidaan varmistaa hyvällä testauksella. Ulkoisissa laatuvaatimuksissa mietitään, millaisiin virheisiin loppukäyttäjä voisi törmätä ja miten näitä virheitä voisi välttää. Onnistuneissa ulkoisissa laatuvaatimuksissa tuote antaa loppukäyttäjälle vaikutelman laadukkaasta ja virheettömästä tuotteesta.

3. Sisäiset laatuvaatimukset (Internal quality attributes). Sisäisissä laatuvaatimuksissa pyritään etsimään virheitä tutkimalla ja uudelleen rakentamalla sovelluksen lähdekoodia. Sisäisissä laatuvaatimuksissa pyritään siihen, että jokainen projektin kehittäjä pystyisi osallistumaan koodin tulkintaan. Tätä varten koodi halutaan pitää luettavana ja helposti lähestyttävänä. Yhtenäistämällä koodia ja kommentoimalla toiminnallisuutta kehittäjä, joka ei ollut alun perin rakentamassa kyseistä toiminnallisuutta, pääsee nopeasti perille siitä, mitä koodilla halutaan saavuttaa.

4. Prosessin laatu (Process quality). Projektin laatu vaikuttaa merkittävästi lopputuotteen laatuun. Prosessin laatua tutkiessa tarkastellaan sovelluksen ja lähdekoodin sijaan itse projektia ja sen kehitystapoja. Prosessin laadussa katsotaan sovelluksen kehitystapoja, projektiin valittuja teknologioita sekä palveluita, aikataulua ja dokumentaatiota. Kehittämällä prosessien laatua voidaan vaikuttaa merkittävästi sovelluksen kehitystyöhön. [Spinellis 2016.]

Sovelluksen laatua voidaan tarkastella myös kansainvälisten standardien avulla. ISO/IEC 912 -standardissa mallin laatu määritellään kuuteen kategoriaan: funktionaalisuus, luettavuus, käytettävyys, tehokkuus, ylläpidettävyys ja siirrettävyys. Tässä opinnäytetyössä tutustutaan näistä neljään: funktionaalisuuteen, luettavuuteen, käytettävyyteen ja ylläpidettävyyteen.

Sovelluksen funktionaalisuutta tarkasteltaessa tutkitaan sitä, mitä sovellus oikeastaan tekee. Funktionaalisuudessa ei ole tärkeätä, miten sovellus suorittaa sille annetun tehtävän, vaan se, miten tarkasti sovellus suorittaa sille annetun tehtävän.

Sovelluksen luotettavuudessa seurataan sitä, miten varmasti sovellus pystyy toimimaan erilaisissa tilanteissa, miten sovellus välttää ongelmatilanteita ja miten ongelmatilanteet

käsitellään ja ongelmista palaudutaan. Sovelluksen luotettavuudesta puhuttaessa voidaan myös puhua sovelluksen kypsyydestä. Sovelluksen kypsyyttä voidaan mitata tutkimalla mitä ongelmia sovelluksessa on. Mitä kypsempi sovellus on, sitä varmemmin se suorittaa sille annetut tehtävät.

Sovelluksen käytettävyydessä tutkitaan käyttäjän ja sovelluksen vuorovaikutusta. Tarkastelu painottuu sovelluksen ulkoisiin tekijöihin: miten helppo sovellusta on käyttää ja miten helppoa sovelluksen käyttö on oppia. Myös sovelluksen ulkonäöllä on merkitystä, sillä se voi vaikuttaa siihen, houkutteleeko se käyttäjää käyttämään sovellusta. [Stephen 2015.]

7.2 Sovelluksen ylläpito

Kuten luvussa 7.1 huomattiin, sovelluskehityksen laatua voi tarkastella monesta eri näkökulmasta. Projektin kehitystiimi päättää aloittaessaan ohjelmistoprojektin, mitä laadun osia halutaan ottaa huomioon sovellusta työstettäessä. Tässä luvussa käydään läpi joitain tapoja, joita soveltamalla kehittäjätiimi voi parantaa tuotteensa laatua. Laadukkaan ohjelmistokehityksen osat kulkevat jokainen käsi kädessä toistensa kanssa: parantamalla esimerkiksi sovelluksen luettavuutta helpottuvat vastaisuudessa sekä arviointi että ylläpidettävyys.

Moderni sovelluskehitys ei lakkaa sovelluksen julkaisuun, vaan vaatii jatkuvaa ylläpitoa ja päivittämistä. Sovelluksesta voidaan julkaisun jälkeen korjata vielä virheitä, parantaa vanhoja ominaisuuksia tai jopa tuoda täysin uusia. Puhuttaessa sovelluksen kyvystä kestää päivityksiä puhutaan sovelluksen ylläpidosta (maintainability). Laadukas sovellus vaatii ylläpitoa pysyäkseen käyttökelpoisena ja säilyttääkseen käyttäjänsä. Usein jatkuva ylläpitäminen saattaa kuitenkin rasittaa alkuperäistä ohjelmiston laatua ja samalla tehdä koodista vähitellen vaikeampaa ylläpitää. Ylläpitävä toiminta voidaan jakaa rakentavaan kehittämiseen ja epäregressiiviseen kehittämiseen. Rakentavassa kehittämisessä luodaan uusia ominaisuuksia ja parannetaan ohjelmiston toiminnallisuutta. Epäregressiivisessä kehittämisessä koodia korjataan ja refaktoroidaan toimivammaksi ja paremmin luettavammaksi. [Spinellis 2016.]

Ohjelmat elävät jatkuvassa muutoksen tilassa, eikä ohjelma ole koskaan täysin valmis. Koodin muutettavuus (changeability) kertoo ohjelman kyvystä kestää muutoksien tekeminen lähdekoodiin ja siitä, miten laajoja korjauksia on tehtävä muutoksen myötä. Jos koodi on rakennettu laadukkaasti, sen pitäisi pystyä kestämään muutoksien tekeminen vähäisillä korjauksilla. Jos koodiin on hankala tehdä muutoksia, voidaan puhua sovelluksen sitkeydestä (software viscosity). Jos muutoksien tekeminen on erittäin vaikeaa ja resursseja syövää, puhutaan sovelluksen kankeudesta (software rigidity). Tällöin tulee miettiä, onko muutoksien tekeminen sovellukseen enää kannattavaa.

Muutoksia tehdessä on mietittävä muutoksen laajuutta ja sen vaikutusta muuhun ohjelmistoon. Laadukkaassa sovelluksessa mahdollisiin muutoksiin on varauduttu, eivätkä muutokset aiheuta laajaa epävakautta sovelluksessa. Koodin muutettavuuteen vaikuttaa muun muassa kovakoodatut arvot, koodin turha toistuvuus ja elementtien nimeämiskonventiot. [Spinellis 2016.]

7.3 Sovelluksen laadun ongelmat

Ongelmat ovat osa tavallista sovelluskehitystä. Viat sovelluksessa eivät välttämättä tarkoita, että koodi ei olisi laadukas. Vioissa on ensiksi syytä tutkia lähemmin ongelmaa ja sen lähteitä, jotta ongelma voidaan ratkaista onnistuneesti.

Sovelluksen logiikassa ilmenevät ongelmat ovat hankalia paikantaa, sillä ne saattavat ilmaantua ongelmana vasta myöhemmin tapahtumaketjussa. Logiikkaongelmat häiritsevät ohjelmiston toiminnan virtaa ja voivat pahimmillaan haitata kriittisiä operaatioita ja datansiirtoa.

Tarpeeton funktionaalisuus on toiminnallisuutta, jolle ei löydy lopputuotteessa enää paikkaa tai tarvetta. Koodin uudelleenkäsittelyssä ohjelmoija saattaa epähuomiossa jättää paloja entisestä koodista ja luoda näin tarpeetonta funktionaalisuutta. Virheiden etsimiseen tarkoitettu vanha koodi saattaa luoda tarpeetonta funktionaalisuutta, häiritä luettavuutta ja tukkia koodia.

Kun ohjelmisto on moniosainen, sen elementit käyvät jatkuvaa tiedonvaihtoa keskenään. Esimerkkinä verkkosivujen suunnittelussa palvelin- ja selainohjelmistot käyvät usein

keskustelua API-rajapinnan avulla. Molemmilla keskustelun osapuolilla on omat odotukset, minkälaista dataa toinen toiselle lähettää. Ongelmia aiheutuu silloin, kun data on normista poikkeavaa. Virhe saattaa esimerkiksi olla vääränmuotoinen viesti tai puutteellinen lähete. Ohjelman on kyettävä käsittelemään virheellinen viesti ja antamaan toiselle osapuolelle tieto ongelmasta. Kestävän rajapinnan end pointit ovat yksinkertaisia toiminnallisuuksia monimutkaisten kokonaisuuksien sijaan.

Joihinkin ongelmiin ei löydy ratkaisua. Silloin on tärkeää miettiä, kuinka paljon resursseja ongelman ratkaisuun voidaan käyttää. Joskus on vain parempi hyväksyä ongelma ja jatkaa eteenpäin. [Crosby 1979: 91.]

Jokaisesta sovelluksesta löytyy ennemmin tai myöhemmin ongelma, johon ei löydy selvää ratkaisua. Tällainen virhe voi pahimmillaan johtaa käytön estymiseen ja järjestelmän kaatumiseen. Virhetoleranssin (Fault Tolerance) omaava sovellus on suunniteltu toimimaan virheestä huolimatta. Sovelluksesta löytyvät virheet ovat osa kehittämisprosessia, eikä kaikkia virheitä voida ennakoida. Ohjelmisto voidaan kuitenkin suunnitella kestämään virhetilanne ja jatkaa eteenpäin. [Spinellis 2016.]

7.4 Sovelluksen ymmärrettävyyden parantaminen

Virheen sattuessa on tärkeää pystyä nopeasti paikantamaan virheen sijainti ja sen koko. Vikaa jäljitettäessä voidaan puhua ylimalkaisesti koodin ymmärrettävyydestä. Koodin ymmärrettävyys riippuu siitä, miten helppo uuden ohjelmoijan on tutustua lähdekoodiin. Parantamalla lähdekoodin ymmärrettävyyttä sovelluksen ongelmien korjaus nopeutuu ja ylläpidettävyys paranee. Koodin ymmärrettävyyttä voidaan lähteä parantamaan yksinkertaisin keinoin. Kahden samankaltaisen funktion tulisi muistuttaa toisiaan. Suuremmissa tiimeissä ohjelmoijan henkilökohtaisen kädenjäljen tulisi mukautua yhteisesti sovitun tyylin puitteisiin, sillä koodin yhtenäinen kirjoitustapa helpottaa luettavuutta: selkeä parametrien ja funktioiden nimeäminen, koodin kattava kommentointi ja funktioiden suunnitelmallisuus, jossa monimutkaiset toiminnallisuudet jaetaan pienempiin osiin.

Kommentointi on ensimmäinen keino, mitä harkita, kun sovelluksen lähdekoodin ymmärrettävyyttä halutaan parantaa. Kommenttirivillä toiminnallisuuden alkuperäinen kirjoittaja voi avata ajatusmaailmaansa lukijalle ja kertoa, mitä alla oleva koodi pitää sisällään.

Koodissa olevien kommenttien tulisi kuitenkin täydentää ja perustella lukijalle mitä koodissa tapahtuu ja miksi. Kommentti, joka vain toistaa sen, mitä funktiossa tapahtuu, ei auta lukijaa ymmärtämään toiminnallisuuden tarkoitusta. Jos koodi on monimutkainen, sen kommentoinnista ei ole välttämättä hyötyä. Sen sijaan tulisi harkita koodin refaktorointia tai toiminnallisuuden pilkkomista pienempiin osiin.

Joskus toiminnallisuutta muutettaessa osa vanhasta koodista halutaan säilyttää varmuuden varalle tai myöhempää käyttöä varten. Tätä kutsutaan versiokommentoinniksi. Isot pätkät kommentoituja koodiosuuksia häiritsevät koodin luettavuutta ja selkeyttä. Kaiken lisäksi nämä versiokommentit saattavat jäädä lojumaan koodiin pitkiksi ajoiksi niiden käytön epäselvyyden takia. Versiokommentointia tulisi välttää, jotta koodi pysyisi ajantasaisena, selkeänä ja luettavana. Versiokontrollipalvelut, kuten tässä projektissa käytetty GitLab, pitävät kirjan kehityshistoriasta, jonne tarvittaessa kehittäjä voi palata tutustumaan vanhoihin versioihin. [Spinellis 2016.]

7.5 Koodin refaktorointi laadukkaammaksi

Tässä luvussa avataan lyhyesti keinoja, joilla ohjelmoija voi parantaa kirjoittamansa koodin laatua. Näitä keinoja ovat koodin lyhentäminen, nimeämiskonventiot ja kovakoodattujen arvojen sekä turhan toistuvuuden välttäminen.

Lyhyt koodi on helpommin luettavaa kuin pitkät ja monimutkaiset kokonaisuudet. Lukijan mielenkiinto herpaantuu helposti pidempää kokonaisuutta lukiessa, jolloin sen tarkka toiminnallisuus saattaa jäädä hämäräksi. Koodi on siis parempi jakaa pienempiin loogisiin osuuksiin, kuin yrittää pitää yhdessä suuressa kokonaisuudessa. Vaikeasti purettavia isoja kokonaisuuksia voi jakaa luettavuuden parantamiseksi useiksi toiminnallisuuden palikoiksi, joilla kaikilla on oma tehtävänsä.

Koodissa käytettävien nimien tulisi auttaa ohjelmoijaa tunnistamaan nimetyn osan roolia koodissa. Mitä selvempi nimi elementillä on, sitä helpompi sen tarkoitusta ja sen muuttamisen seurauksia on ottaa huomioon. Koodissa ilmentyvän nimen pituuden pitäisi olla suhteutettuna sen tärkeyteen; iteroidessa käytettävä parametrin nimi ei tarvitse yhtä sanaa pidempää nimeä, kun taas luokan olisi hyvä kertoa nimessään sen käyttötarkoitus.

Kovakoodatut arvot ovat parametrien arvoja, jotka on kirjoitettu suoraan lähdekoodiin. Kovakoodattuja arvoja on vaikea paikantaa koodista ja muuttaa, mikä hidastaa kehitystyötä. Vaihtoehtoisesti arvot voidaan kutsua erillisestä kokoonpanotiedostosta.

Turha koodin toistuvuus hankaloittaa muutoksen suunnittelua ja toteutusta. Ohjelmoidessa pitäisi pyrkiä käyttämään DRY:n (Don't Repeat Yourself, "Älä toista itseäsi") periaatteita. Älä toista itseäsi -periaatteen pääviesti on: "Jokaisella tiedolla systeemissä pitäisi olla yksittäinen, yksiselitteinen ja arvovaltainen kuvaus". [Spinellis 2016.]

7.6 Testit ja vertaisarviointi

Tuotteen testauksessa valmistetaan järjestelmä, jolla voidaan teknisesti mitata, toimiiko testattava ominaisuus. Vertaamalla testin tulosta suunnitteluvaiheessa määriteltyihin hyväksymiskriiteereihin voidaan objektiivisesti todeta, täyttääkö komponentti sille asetetut laatuvaatimukset. Sovelluksen testaukseen voidaan myös soveltaa laadun periaatteita. [Crosby 1979: 73–74.]

Testit eivät kuitenkaan yksinään voi ratkaista kaikkia ohjelmiston ongelmia. Testit voivat osoittaa virheellisiä tuloksia ja löytää virheitä, mutta syvempiä toiminnallisuuden ja suunnittelun ongelmia etsittäessä on implementoitua koodia tulkittava projektin dokumentaation ja toimintasuunnitelmien kanssa. Laadukkaan koodin on oltava toimivaa, ymmärrettävää ja dokumentoinnin mukaista. Testien lisäksi on siis hyvä käydä läpi lähdekoodia tutkien, miten se vastaa tehtävän tavoitteita, yhteisiä sovittuja käytäntöjä ja tietenkin itse koodin toimivuutta.

Ohjelmoijan esittäessä työtään toisille arvioitavaksi puhutaan koodin arvioinnista (code review) ja arvioitavuudesta (reviewability). Vertaisarvioinnissa ohjelmoija saa ulkopuolisen silmäparin tutkimaan kokonaisuutta. Ohjelmoija saa vertaisarvioinnissa rakentavaa palautetta työstään ja parannusehdotuksia. Koodin ymmärrettävyys on arvioinnin tärkeä kulmakivi. Jos koodin luettavuus on heikko, arvioijalla kuluu enemmän aikaa koodin ymmärtämiseen kuin itse arviointiin. [Spinellis 2016.]

Arvioija tutkii koodia miettien, onko työn jälki laadukasta. Voisiko esimerkiksi toiminnallisuuden erottaa erilliseksi funktioksi tai ratkeaako ongelma kattavammalla

kommentoinnilla? Koodin arviointi on kuitenkin yhtä subjektiivista kuin koodin kirjoittaminenkin. Lisäksi koodi ja projektin dokumentaatio on kirjoitettu eri kielillä, mikä tekee arviointityöstä paikoin hyvinkin tulkinnanvaraista. Kommunikaatio ja aktiivinen dialogi arvioijan ja ohjelmoijan kesken pitää molemmat samalla aaltopituudella ja helpottaa ongelmien ratkaisemisessa.

Koodin vertaisarvioinnissa on hyvä miettiä seuraavia seikkoja:

- Mikä on kirjoitetun koodin tehtävä?
- Onko kaikki tavoitteet täytetty?
- Tuoko koodi mukanaan turhaa toistoa tai jotain muuta tarpeetonta?

Näihin kysymyksiin pitäisi löytyä vastaus projektin dokumentaatiosta. Dokumentaation pitäisi kertoa lukijalle tehtävän taustat, vaatimukset ja suunnitteluvedokset. Kun koodin arvioija tutkii koodia näiden taustojen pohjalta, ovat arvioijan tavoitteet arvioinnille selvät. Jos koodi on luettavaa ja vastaa kaikkiin kysymyksiin onnistuneesti, voi arvioija todeta sen laadukkaaksi. [Spinellis 2016.]

8 Palasia sovellusprojektin tehtävistä

Koulutuksenhallintaohjelmistoprojekti oli sen verran kookas, että koko koodin purkaminen ja esittely veisi liikaa aikaa ja sivuja. Avaan tässä luvussa muutaman esimerkkitapauksen, joita olin itse mukana työstämässä, ja tutkin niitä laadun kehittämisen näkökulmasta.

8.1 Sprinttipalaverit

Scrumiin kuuluu useita palavereita, jotka järjestetään joka sprintillä. Tässä luvussa käydään muutamia projektin sprinttien aikana pidettyjä palavereita läpi ja tarkastellaan, mitä hyötyä niistä on sovelluksen laadun kehittämisen kannalta.

Sprintti 10:n arviointipalaveri

Sprintti 10:n viimeisenä päivänä pidettiin arviointipalaveri, jonka kesto oli noin 30 minuuttia. Koko projektiryhmä oli saapunut kokoustilaan läsnä, ja loput osallistuvat palaveriin videopuhelun kautta.

Scrum-mestarin aloitteesta palaveri alkaa ja ryhmä ryhtyy katsomaan Done-sarakkeen tehtäviä. Kaikki sprintin tarinat on etukäteen jaettu kehitystiimin kesken. Tiimin jäsenet ovat käyneet heille jaetut tarinat läpi ja testanneet käyttäjätarinoiden toiminnallisuuden kehitysympäristössä ennen arviointipalaveria. Arviointipalaverissa jokainen kehittäjä esittelee käyttäjätarinan muille ja näyttää valmiin toiminnallisuuden. Testatessa ilmenneet tarinan ongelmat tuodaan palaverissa esille, ja niille mietitään yhdessä jatkotoimenpiteitä. Jos tarina on puutteellinen, se merkitään takaisin tehtäväksi. Tarinan toiminnallisuuden esittelyn jälkeen muut osallistujat, kuten projektin omistaja, kommentoivat tuotosta: vastaako se heidän odotuksiaan vai tarvitseeko tehtävä vielä jatkokehitystä.

Arviointipalaverit ovat mielestäni erittäin hyödyllisiä ja edesauttavat laadukasta projektikehitystä. Palavereissa koko projektitiimi pääsee samalle aaltopituudelle muiden kanssa siitä, missä vaiheessa projekti realistisesti tällä hetkellä on. Arviointipalaverissa ovat myös kaikki projektin sisäiset sidosryhmät paikalla ja projektin tilasta käydään vilkasta dialogin vaihtoa. Näin kehityksessä piilevät epäselvyydet paljastuvat jo hyvissä ajoin ja niihin voidaan vaikuttaa ajoissa.

Sprintti 10:n retrospektiivinen palaveri

Retrospektiivisessä palaverissa halutaan katsoa kriittisesti juuri päättyvää sprinttiä ja miettiä, olisiko kehitystyössä jotain parantamisen varaa. Palaveri järjestettiin videopuheluna, ja sen kesto oli noin 45 minuuttia.

Edellisessä retrospektiivisessä palaverissa sprintin 10 tavoitteeksi oli sovittu lähdekoodin kattavampi kommentointi. Paremmalla koodin kommentoinnilla saataisiin parannettua sovelluksen luettavuutta. Ryhmän arvioidessa tavoitteen onnistumista päädyttiin lähes yksimielisesti siihen, että sprintti 10:ssä tähän tavoitteeseen ei päästy. Tavoitteeseen haluttiin silti pyrkiä, joten päätettiin, että ensi sprintillä yritetään arvioida toisten koodia vielä ahkerammin ja vaatia koodin selvennystä vertaisarvioinnissa.

Tavoitteiden jälkeen palaverissa siirryttiin keskustelemaan sprintin onnistumisista ja epäonnistumisista. Tähän käytettiin pudota, lisää, pidä ja paranna -taulua eli DAKI:a (drop, add, keep, improve). DAKI:ssa jokainen osallistuja yrittää keksiä jokaiseen kategoriaan jotain kirjoitettavaa liittyen sprinttiin. DAKI:lla voidaan esimerkiksi karsia huonoksi todettuja käytäntöjä ja esittää parannusehdotuksia kehityksen laadun parantamiseksi. Sprintti 10:n DAKI:ssa keskityttiin kuvan 7 mukaisesti kehittämään yhteisiä käytäntöjä projektin vertaisarvioinnin parantamiseksi.

3. Drop Add Keep Improve

Drop	Add	Keep	Improve
	▪ more stuff on backlog ▪ let's add #number_of_task to commit message, like this "#123 commit message" (where 123 - number of Jira task), so it will be easier for developer to find the task which is related to this commit	▪ posting links to slack when mr is ready for review	▪ Marking the MR:s with eyes when taking a look at them

Kuva 7. Sprintti 10:n retrospektiivinen palaverin DAKI.

Vertaisarvioinnissa on ollut epäselvyyksiä siitä, kuka vertaisarvioinnin suorittaa. Tähän sovitaan ratkaisuksi, että arvioija merkitsee itsensä aktiiviseksi versiohaarauman arvioijaksi. Kun arviointi on tehty, sama arvioija merkitsee koodin hyväksytyksi antamalla haaraumalle peukalo ylös -hymiön.

Toinen epäselvyys arvioinnissa on ollut tiedon hajautuminen. Arvioijalla on kulunut turhaan aikaa etsien arvioitavan versiohaarauman dokumentaatioita. Ratkaisuksi tähän päätetään, että kaikki tehtävään tarvittava dokumentaatio, kuten suunnittelumalli ja käyttäjätarina, liitetään vertaisarvioinnin versiohaarauman yhdistämispyyntöön.

Asiakkaiden on määrä päästä testaamaan kehitysympäristöä seuraavalla viikolla. Palaverissa sovitaan, että kehitysympäristön vakauteen on painotettava, jotta sovellus olisi mahdollisimman kestävä ensimmäiseen ulkopuoliseen julkaisuun. Samalla otetaan puheeksi projektin end-to-end-testaus. Sovelluksen vakauden takaamiseksi halutaan ottaa käyttöön testiympäristö, jolla testit voidaan hoitaa luotettavasti. Cypress-testausohjelmaa lähdetään tutkimaan sopivana ratkaisuna.

Mielestäni retrospektiivinen palaveri on hyvä lisä suuren tiimin kehitystyöhön, sillä palaverissa voidaan nostaa esille kehitystyön ongelmakohtia ja miettiä yhdessä ratkaisuja niihin. Jokainen kehittäjä sitoutuu päätöksiin paremmin, koska he ovat itse olleet palaverissa paikalla ja ovat voineet vaikuttaa päätöksiin.

Sprintti 11:n suunnittelupalaveri

Sprintti 11:n suunnittelupalaveri aloitetaan käymällä läpi sprintti 11:n tilauskantaa. Kaikki sprintti 10:n keskeneräiseksi jääneet käyttäjätarinat siirrettiin sprintti 11:n tilauskantaan. Jokainen tilauskannan tehtävä käydään läpi ja katsotaan, mitä on vielä tekemättä ja onko jokin tehtävässä muuttunut. Sprintti 11 oli viimeinen sprintti ennen sovelluksen ensimmäistä julkaisua asiakkaan testattavaksi.

Ensimmäiseksi sprintin tehtävistä katsotaan käyttäjätarinaa korttien tulostuksesta. Korttien tulostus on aiemmin ollut pätevyyssivulla, mutta testauksen jälkeen on todettu, että korttien tulostus on parempi siirtää omalle sivulleen. Korttien tulostuksen vaatima komentopöytä sekoittui testatessa helposti pätevyyden toiminnallisuuksien kanssa. Samalla pätevyyssivua saataisiin yksinkertaistettua poistamalla sinne kuulumaton korttien tulostus. Korttien tulostus on laajempi kokonaisuus, joten tarina jaetaan useaksi pieneksi tehtäväksi.

Tähän mennessä projektin esittely asiakkaalle on tehty kehitysympäristössä. Julkaisun lähestyessä tuotantoympäristö on saatava pystyyn. Odottamattomien virheiden varalle tuotantoympäristö olisi hyvä saada pystyyn mahdollisimman ajoissa. Päädytään kuitenkin odottamaan, että asiakas toimittaa tietokantoihin määriteltävän pohjadatan, jonka tuotantoympäristö tarvitsee toimiakseen.

End-to-end-testejä on vielä runsaasti tekemättä ennen ensimmäistä julkaisua. Testien rakentamiseen allokoidaan enemmän kehittäjiä, sillä testien tekeminen on hieman jäljessä aikataulusta.

Tulevan sprintin tehtävät jakautuvat karkeasti kolmeen kategoriaan: uusiin ominaisuuksiin, testeihin ja vikojen korjaukseen. Viat ovat prioriteettina, sillä ohjelmistoa halutaan siistiä julkaisua varten ja pahimmat viat karsia pois. Testit tuottavat lisää ratkaistavia

vikoja, joten niiden kirjoittaminen koetaan tärkeämmäksi kuin uusien ominaisuuksien luominen. Uusien ominaisuuksien lisääminen tuo myös riskin uusista vioista.

Palaverin lopuksi vielä keskustellaan sprintin tehtävien määrästä ja puuttuvista tehtävistä. Sprintin tarinat priorisoidaan tärkeysjärjestykseen tilauskantaan, ylhäältä alas. Puuttuvien toiminnallisuuksien käyttäjätarinoiden suunnittelua varten sovitaan erillinen palaveri. Seuraavalle sprintille jokaiselle kehittäjälle jaetaan vastuualue, jota manuaalisesti testata ja mistä etsiä mahdollisia vikoja.

Sprintti 12:n kehityspalaveri

Kehityspalaverissa lähdetään kategorioittain käymään läpi tilauskannan olevia käyttäjätarinoita. Käyttäjätarinasta käydään lyhyt keskustelu, pitääkö tehtävän kuvaus vielä voimansa vai tarvitseeko tarina jotain täydennettävää. Tehtävistä haetaan vain yleinen kuva, implementaation yksityiskohtien tarkempi pohtiminen käydään myöhemmin läpi ryhmissä. Valmiiksi ratkaistut tai vanhentuneet tehtävät karsitaan pois tilauskannasta.

Kehityspalaveri pidetään samana päivänä, kuin sovellus on julkaistu asiakkaalle testikäyttöön. Asiakkaalta on jo ehtinyt saapua joitain parannusehdotuksia, jotka otetaan huomioon kokouksessa. Seuraavan sprintin tavoitteita ovat muun muassa saada ruotsinkieliset käännökset, sivuston tilan monitorointi ja käyttäjien oikeuksien hallinnoinnin kehittäminen.

Katsauksen jälkeen jakaudutaan tiimeihin, joissa on sekä palvelin- että selainohjelmoijia. Sprintin käyttäjätarinat, tehtävät ja virheet jaetaan ryhmille, ja ne alkavat käydä niitä läpi tarkentaen kuvausta tarvittavilla yksityiskohdilla ja määrittelyillä.

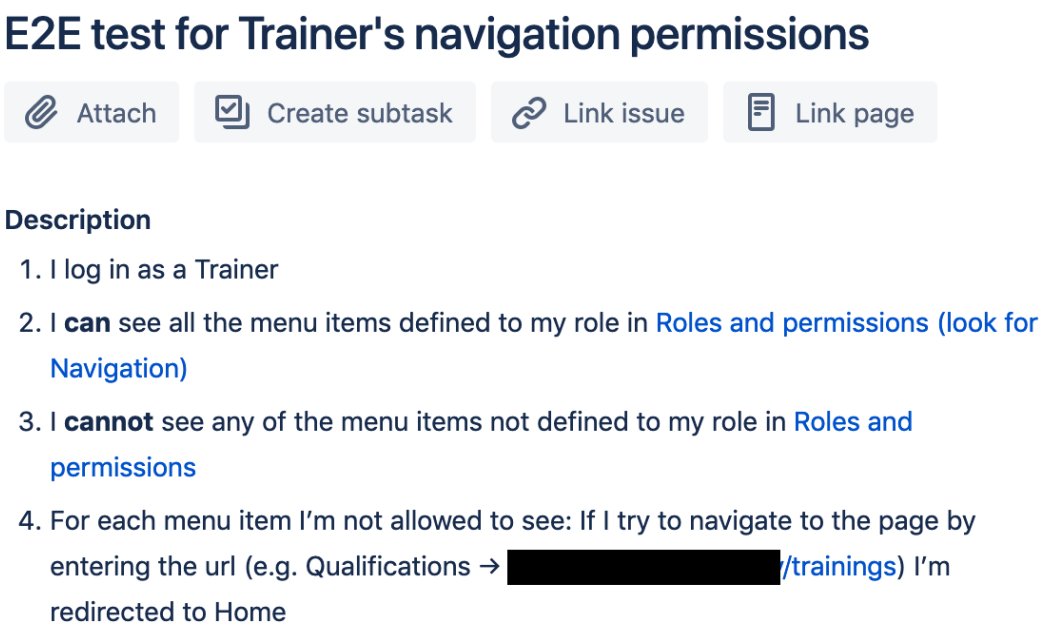
Koko tiimin osallistuminen tehtävien läpikäyntiin ja määrittelyyn on minusta erittäin tehokas keino varmistaa kehitystyön sulavuus ja tiedon jako. Käyttäjätarinat on usein kirjoitettu pääpiirteittäin jättäen paljon tulkinnanvaraa. Käyttäjätarinoissa ohjelmoijat ovat mukana luomassa valmiuden määritelmiä, jolloin tehtävän tavoitteet saadaan kirjoitettua muotoon, josta työtehtävän kehittäminen on sprintin aikana luontevaa. Näin sprintin aikana vältytään tilanteita, jossa tehtävä vaatii lisäselvitystä epäselkeyden vuoksi.

8.2 Käyttäjän navigoinnin oikeuksien testaus

Tehtäväni oli valmistaa selainohjelmistoon end-to-end- eli E2E-testit, joissa varmistetaan käyttäjän oikeudet liikkua sivustolla vain käyttäjän rooliin määrätyillä alueilla, ja tarkistaa, että navigaatiopalkissa näkyy oikeat linkit. Testit valmistettiin käyttäen Cypress-rajapintaa.

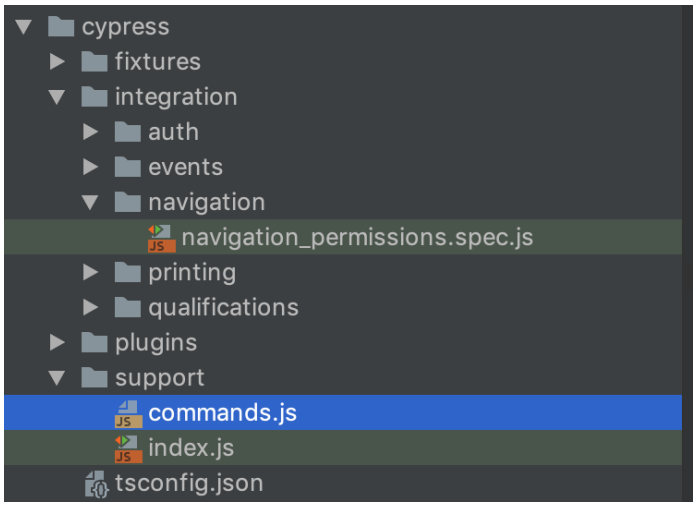
Testissä testataan niin kutsuttu ”onnellinen polku”, jossa varmistetaan elementin toiminta ihannetapauksena. Tässä testissä onnellinen polku tarkoittaa, että käyttäjä näkee vain sille tarkoitetut linkit ja hän ei pääse pois suljettuihin osiin. Jos jompikumpi näistä ehdoista ei täyty, testi epäonnistuu ja ilmoittaa virheen syyn.

Kuvassa 8 näkyvät kouluttajan navigoinnin testauksen määritelmät. Tehtävänantoon on kirjoitettu alkutoimenpiteet, mistä alkupisteestä tehtävää aletaan testata. Tässä tehtävässä ainoa alkuasetelman vaatimus oli kyseisessä roolissa olevan käyttäjän kirjaaminen sisään. Testin määrittelyssä kerrotaan testin onnistumisen rajat: missä halutaan onnistua ja mitä ei saa tapahtua. Testi on kaksiosainen, joten harkinnanvaraisesti tehtävän olisi voinut jakaa kahteen pienempään toimintoon. Molemmissa käsitellään kuitenkin käyttäjän rooliin kuuluvia oikeuksia, joten yksi tehtävä oli perusteltu.



Kuva 8. Testin kouluttajan tehtävänanto.

Tehtäväni oli testata toiminnallisuus jokaiselle sovelluksen käyttäjäroolille. Välttääkseni turhaa iteraatiota päätin käyttää Cypressin tarjoamaa ominaisuutta valmistaa räätälöityjä komentoja. Niitä varten tuli luoda oma commands.js-tiedosto supports-kansioon, kuten esitettynä kuvassa 9.



Kuva 9. Cypress-kansion arkkitehtuuri.

Testejä suorittaessa Cypress tulkitsee räätälöidyt komennot.

Ensimmäisessä testissä halutaan käydä navigointipalkki läpi ja verrata sieltä löytyvien linkkien nimiä käyttäjän roolin sallimiin linkkeihin. Loin räätälöidyn komennon nimeltä ”testNavigation”, jota pystytään kutsumaan testeissä kuin mitä tahansa Cypressin omaa komentoa. Komentoon annetaan parametrina lista käyttäjän roolin sallimista linkeistä.

Funktiossa kutsutaan get-metodilla html-sivun listaelementtiä, jonka sisältä haetaan kaikkien listalla olevien linkkien nimet. Listan pituutta verrataan parametrina saatuun listan pituuteen ja varmistetaan, että listat ovat yhtä pitkät. Tämän jälkeen listaelementin linkkien nimet iteroidaan läpi. Kuvan 10 mukaisesti jokaista navigaation linkkiä verrataan parametrina saatuun listaan ja katsotaan includes-metodilla, löytyykö samanniminen linkki listasta. Jos jokainen listan elementti löytyy, räätälöity komento päättyy onnistuneesti.

```
Cypress.Commands.add( name: 'testNavigation', fn: (permissions) => {
    cy.visit( url: '/home');
    cy.get( selector: 'ul').within( fn: ($list) => {
        cy.get( selector: 'li a span').its( propertyName: 'length').should( chainer: 'eq', permissions.length);
        cy.get( selector: 'li a span').each( fn: function (value) {
            expect(permissions.includes(value.text())).to.eq(true);
        })
    });
});
```

Kuva 10. Navigaatiopalkin testaus -funktio.

Testissä ei testata erikseen kiellettyjen linkkien näkyvyyttä. Tämän pitäisi varmistua testissä poissulkevana menettelynä: jokainen funktioon syötetyn listan linkki pitää löytyä navigoinnin listasta, eikä lista saa olla parametrin listaa pidempi, joten kiellettyjä linkkejä ei voi löytyä navigoinnin listasta.

Testin toisessa osassa testataan, että käyttäjä ei pääse navigoimaan url-palkin kautta sivuston osiin, mihin hänellä ei ole oikeuksia. Tätä varten aluksi luotiin lista nimeltä allPermissions kaikista sivun osista, joihin voi url-palkin kautta navigoida. Kuvan 11 mukaisesti lista iteroidaan läpi. Ehtolauseella erotetaan tilapäiseen urlsToTest-listaan kaikki käyttäjälle kielletyt osoitteet.

```
const urlsToTest = [];
allPermissions.map( allValue => {
    if (!permissions.includes(allValue.name)) {
        urlsToTest.push(allValue.url);
    }
});
cy.visit( url: '/home');

urlsToTest.map( url => {
    cy.visit(url);
    cy.url().should( chainer: 'eq',  value: Cypress.config( key: 'baseUrl') + '/');
});
```

Kuva 11. Käyttäjän url-navigoinnin testausfunktio.

Lista urlsToTest käydään läpi yrittäen aina vierailla kyseisessä osassa sivustoa laittamalla sivuston osoite suoraan url-palkkiin. Tämän jälkeen sivuston osoite tarkistetaan, varmistaen, että käyttäjä on uudelleenohjattu takaisin etusivulle.

Itse testien rakentaminen Cypressillä oli tämän jälkeen helppoa, sillä kaikki testin toiminnallisuus tuli juuri rakennetuista apufunktioista. Jokaiselle käyttäjäroolille tarvitsi vain määritellä lista sille sallituista sivuston osista. Kuvassa 12 on kouluttajan navigoinnin

testaus kokonaisuudessaan. Testissä käyttäjä kirjataan sivulle käyttäen kolmatta apufunktiota, joka on implementoitu projektiin jo aiemmin. Sen jälkeen tarvitsee vain kutsua juuri rakennettuja testNavigation- ja testUrls-komentoja ja välittää parametrina lista oikeuksista.

```
it('can navigate as a trainer', () => {
    const permissions = [
        'Home',
        'Qualifications',
        'Events',
        'Log out'
    ];

    cy.login('trainer').then(() => {
        cy.testNavigation(permissions);
        cy.testUrls(permissions);
    });
});
```

Kuva 12. Testi, jolla tarkastetaan kouluttajan navigointipalkin linkit.

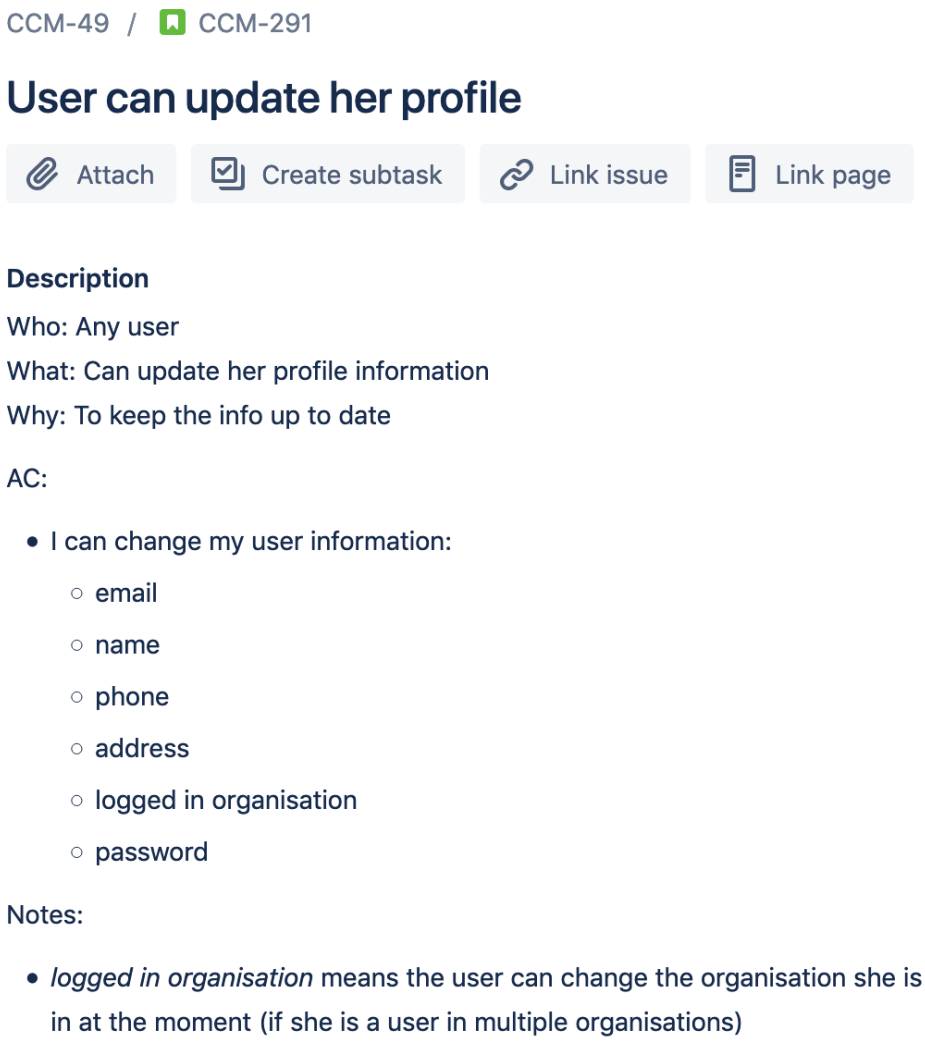
Tämän testin avulla havaittiin useita virheitä liittyen käyttäjän navigoinnin oikeuksiin. Testillä saadaan myös helposti testattua jatkossakin oikeudet, jos muutoksia käyttäjien oikeuksiin tulee.

8.3 Profiilisivun implementointi

Profiilisivun implementoinnin käyttäjätarinassa oli tarkoitus toteuttaa käyttäjän profiilisivusto. Profiilisivustolla käyttäjä pystyy muokkaamaan henkilökohtaisia tietojaan ja vaihtamaan salasanansa. Tehtävä oli kokonaisuudessaan laaja: se sisälsi sekä suunnittelun että palvelin- ja selainohjelmiston tehtäviä. Tästä syystä käyttäjätarina oli jaettu viiteen pienempään osaan. Minun tehtäväni oli implementoida sivuston selainohjelmiston osuus.

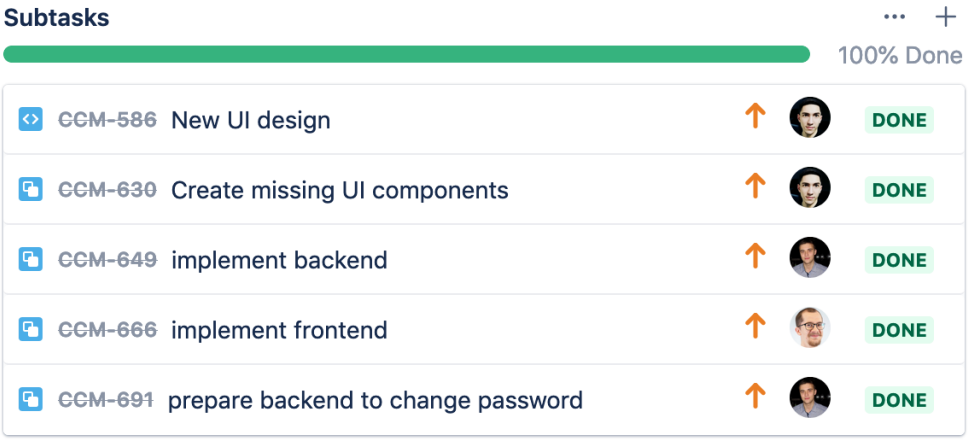
Kuvassa 13 on profiilisivuston käyttäjätarina. Käyttäjätarinasta nähdään kaikki tehtävän oleellinen informaatio. Syy profiilisivun luomiseen on yksiselitteinen: käyttäjien täytyy pystyä päivittämään tietojaan. Valmistumiskriteereihin on määritelty, mitä kaikkea

käyttäjän tulee voida päivittää sivullaan. Tämän tarinan pohjalta lähdettiin kehittämään profiilisivua.



Kuva 13. Profiili-sivun käyttäjätarina ja DOA.

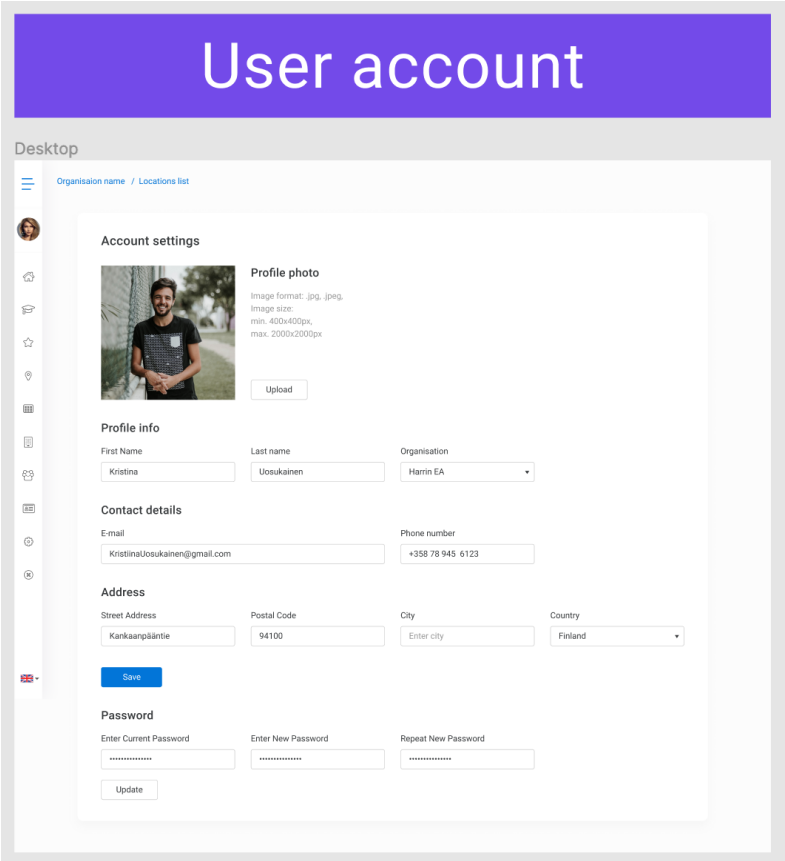
Käyttäjätarinan pohjalta luotiin viisi eri tehtävää. Kun jokainen tehtävä on tehty, profiilisivuston pitäisi olla valmis julkaisuun. Kuvassa 14 nähdään kaikki tarinan pohjalta luodut tehtävät. Profiilisivulla tapahtui toteutuksen aikana useampi muutos, minkä vuoksi se vaati käyttöliittymän uudelleen suunnittelua. Salasanan vaihto vaati myös uutta toteutusta palvelinohjelmistoon, minkä vuoksi sille piti luoda oma tehtävänsä.



Kuva 14. Käyttäjätarinan pohjalta valmistetut tehtävät.

Käyttöliittymää suunniteltiin useita viikkoja sivuston toteutusta edellä. Tämän ansiosta tehtävän saapuessa sprintille käyttöliittymä oli jo suunniteltu ja nähtävillä Figma-sovelluksessa.

Kuvassa 15 on Figma-sovellukseen tehty malli profiilisivuston käyttöliittymästä. Profiilisivuston kehittäminen käyttäen mallia pohjana oli nopeaa. Sivustolle oli tarkoitus rakentaa kolme eri ominaisuutta.



Kuva 15. Suunnittelijan valmistama malli käyttäjän profiilisivusta.

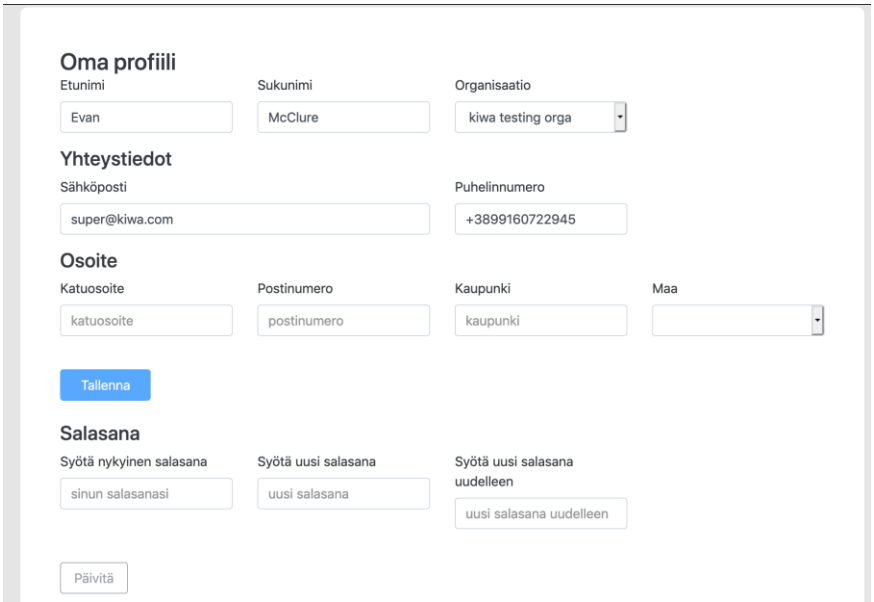
Ensimmäinen ominaisuus oli profiilikuvan lisääminen ja vaihtaminen. Profiilikuvaa oli tarkoitus hallita käyttäen Bootstrapin modal-ikkunaa. Sprintillä ei kuitenkaan ollut otettu huomioon, että tiedostojen lähettäminen ja käsittely tulee vasta myöhemmin implementoitavaksi, joten profiilikuva jätettiin toistaiseksi tekemättä.

Toinen ominaisuus oli käyttäjätietojen muokkaaminen. Käyttäjätiedoille tehtiin Angularin reaktiivinen lomake, ja se sommiteltiin Bootstrapin verkkojärjestelmän avulla. Käyttäjätiedot validoidaan sekä selainohjelmistossa että myöhemmin palvelinohjelmistossa. Jotta reaktiivisen lomakkeen virheviestien kääntäminen onnistuisi aktiiviselle kielelle, oli rakennettava virheviesteille oma palvelu, joka palauttaa oikealle kielelle käännettävän virheviestin.

Kolmas ominaisuus oli salasanan vaihto. Alun perin salasana oli osa käyttäjätietolomaketta, mutta arvioinnissa todettiin, että sen siirtäminen omaksi osuudeksi olisi parasta. Käyttäjän vaihtaessa salasanansa on hänen kirjoitettava oma nykyinen salasanansa

sekä uusi salasana kahdesti. Selainohjelmistoon oli tätä varten rakennettava oma validointi uuden salasanan yhtenäisyyden tarkastamiseksi.

Arvioinnin ja muokkauksien jälkeen käyttäjäprofiilisivustosta tuli kuvan 16 mukainen. Profiilikuva jätettiin toistaiseksi sivuun, ja alkuperäisestä suunnitelmasta eroten salasana on erillisenä osuutena.

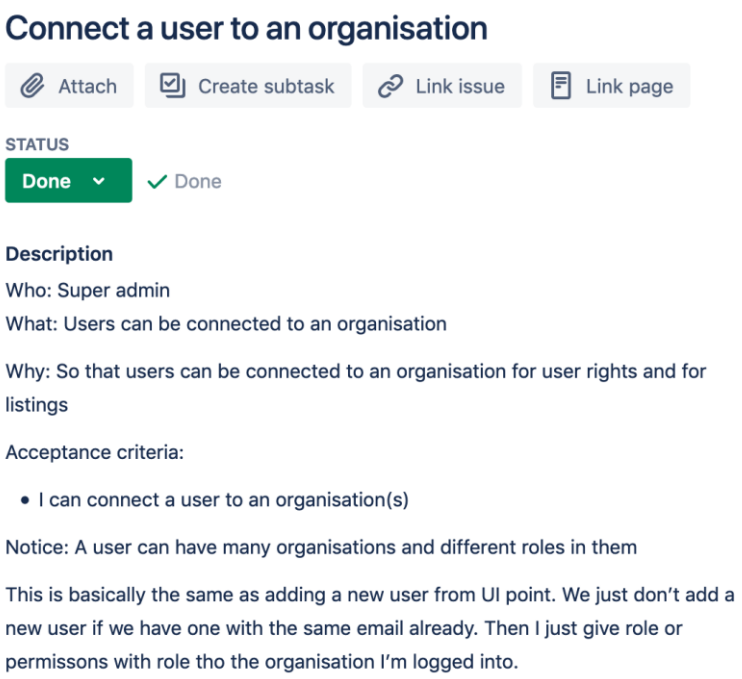


Kuva 16. Lopullinen projektiin tuotu versio.

8.4 Käyttäjän sähköpostin tarkastus

Käyttäjän sähköpostin tarkastus tehtiin osana suurempaa käyttäjän hallinnan valmistusta. Toiminto on tarkoitettu pääkäyttäjälle, joka pystyy itse lisäämään uusia henkilöitä järjestelmään. Tarinan tarkoitus on tarkentaa käyttäjähallintaa antamalla pääkäyttäjälle mahdollisuus muokata, mihin organisaatioihin käyttäjä kuuluu.

Kuvan 17 mukaisessa käyttäjätarinassa tarkennetaan lopussa, että käyttäjää luodessa tulee tämän sähköposti tarkastaa. Sähköpostiosoitteet ovat sovelluksessa uniikit, joten usealla käyttäjällä ei voi olla sama sähköpostiosoite. Tämän välttämiseksi on sähköposti tarkastettava ennen käyttäjän luontia.

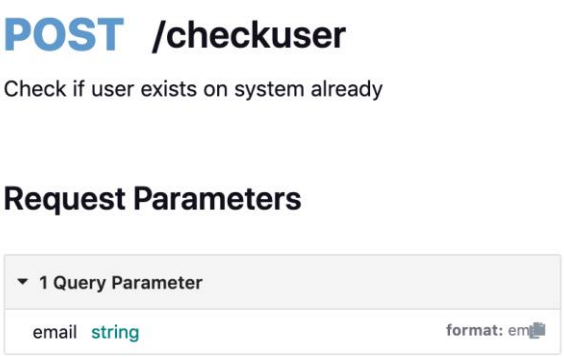


Kuva 17. Käyttäjän liittäminen organisaatioon -käyttäjätarina.

Tehtävä jaettiin useaan pienempään työtehtävään, joista yksi oli palvelinpuolella tapahtuva käyttäjän sähköpostin tarkistus. Ensimmäinen vaihe tehtävää aloitettaessa oli tarkastaa, millainen rakenne API-päätepisteelle oli suunniteltu.

Palvelinpuolen toiminnallisuuden rakentamiseen kuului myös tehtävän testien rakentaminen. Tässä työnannossa oli rakennettava onnistumis- ja epäonnistumistestit sekä ohjaimelle että palvelulle.

Stoplight-ohjelmaan on dokumentoitu kaikki sovelluksen käyttämät päätepisteet. Käyttäjän sähköpostin tarkastamiselle oli luotu kuvan 18 mukainen osoite. Osoitteeseen tulee lähettää POST-muotoinen kysely, johon annetaan parametrina käyttäjän sähköposti. Palautteena saadaan joko käyttäjän nimi ja käyttäjätunniste tai virheviesti, jos käyttäjällä on jo olemassa sähköposti järjestelmässä.



Kuva 18. Stoplight-ohjelman API-määritelmä.

Toiminnallisuuden varmistamiseksi projektissa palvelimelle saapuva data täytyy validoida, ennen kuin sitä voidaan hyödyntää. Samoin pyynnön tekevällä käyttäjällä on oltava valtuudet tehtävän suorittamiseksi. Jos molemmat ehdot täyttyvät, voidaan kysely tehdä.

Palvelinpuolen toiminnallisuus mukailee MVC-mallin toimintaa. Ohjain ottaa viestin vastaan, validoi ja käsittelee sen. Sen jälkeen ohjain kutsuu käyttäjienhallintapalvelua, joka tekee tarvittavan tietokantakyselyn käyttäjän sähköpostista. Jos sähköposti löytyy tietokannasta, palautteesta otetaan käyttäjän nimi ja tunnistenumero ja palautetaan takaisin ohjaimelle. Ohjain käsittelee datan ja antaa sen takaisin käyttäjälle joko onnistuneena palautteena tai virheviestinä ”Käyttäjää ei löytynyt”.

9 Yhteenveto

Insinöörityössä huomattiin, että sovelluksen laatua voi parantaa monella eri tavalla: paremmalla dokumentaatiolla, luettavammalla koodilla ja huolellisella testauksella. Monet ohjelmoijat käyttävät laadun kehittämisen menetelmiä niitä itse välttämättä tiedostamatta.

Laadun teoriassa tutustuttiin siihen mitä laatu sovelluskehityksessä tarkoittaa ja miten se voi parantaa projektin kehitystyötä. Laadun eri ominaisuudet sovelluskehityksessä purettiin ja mietittiin, miten näitä laadun keinoja voidaan hyödyntää kehitysprojektissa.

Sovellusprojektissa laatua pyrittiin parantamaan monin eri keinoin: projektinhallinta suoritettiin scrumilla, sovelluksen uudet ominaisuudet testattiin ja vertaisarvioitiin, ja kehitysprojektista ylläpidettiin ajankohtaista dokumentaatiota. Scrumin hyödyntäminen projektissa lisäsi tiimin sisäistä kommunikaatiota ja teki sovelluskehityksestä määrätietoisempaa. Vertaisarvioinnilla ja testaamalla koodista saatiin karsittua virheitä ja epäkäytännöllisyyksiä ennen sen implementointia sovellusprojektiin. Ajankohtainen projektidokumentointi helpotti projektikehitystä ja selkeytti sovelluskehityksen tavoitteita. Laadun parantamisen keinot eivät hidastaneet kehitystyötä, vaan päinvastoin pidemmällä aikavälillä nopeuttivat sitä.

Laatu olisi hyvä ottaa sovelluskehityksessä huomioon mahdollisimman aikaisin. Ohjelmointitiimin tulisi jo projektia aloittaessa sopia yhdessä käytännöt, joita soveltaa kehitystyössään. Käytäntöjen tuominen aikaisin projektiin säästää kehitystyön edetessä rahaa, aikaa ja hermoja. Laatu on sovelluskehityksessä arvo, josta ei kannata tinkiä, sillä laatu voittaa aina määrän.

Lähteet

Agile Coach. 2019. Verkkoaineisto. Atlassian. <https://www.atlassian.com/agile>. Päivitetty 2019. Luettu 20.10.2019.

Amazon S3. 2019. Verkkoaineisto. Amazon. <https://aws.amazon.com/s3/?nc2=h_ql_prod_st_s3>. Päivitetty 2019. Luettu 25.9.2019.

Angular Testing. 2010. Verkkoaineisto. Google. <https://angular.io/guide/testing>. Päivitetty 2019. Luettu 1.11.2019.

Atlassian Products. 2019. Verkkoaineisto. Atlassian. <https://www.atlassian.com/software>. Päivitetty 2019. Luettu 20.10.2019.

Bootstrap widgets the angular way. 2019. Verkkoaineisto. Ng-bootstrap Team. <https://ng-bootstrap.github.io/>. Päivitetty 2019. Luettu 20.10.2019

Compare Laravel, Symfony, CodeIgniter. 2019. Verkkoaineisto. Google Trends. <https://trends.google.com/trends/explore/TIMESERIES/1570986000?hl=en-US&tz=-180&date=today+5-y&geo=US&q=%2Fm%2F0jwy148,%2Fm%2F09cjcl,%2Fm%2F02qgdkj&sni=3>. Päivitetty 2019. Luettu 1.11.2019.

Cypress. 2019. Verkkoaineisto. Cypress.io. <https://www.cypress.io/>. Cypress.io. Päivitetty 2019. Luettu 2.11.2019.

Eloquent ORM. 2019. Verkkoaineisto. Laravel. <https://laravel.com/docs/5.0/eloquent>. Päivitetty 2019. Luettu 24.10.2019.

Envoyer – The PHP deployer is now launched. 2016. Verkkoaineisto. Laravel News. <https://laravel-news.com/envoyer-the-php-deployer-is-now-launched> Luettu 17.10.2019.

Figma. 2019. Verkkoaineisto. <https://www.figma.com/>. Päivitetty 2019. Luettu 1.10.2019.

Git. 2019. Verkkoaineisto. <https://git-scm.com/>. Päivitetty 2019. Luettu 1.10.2019.

Jira Software. 2019. Verkkoaineisto. Atlassian. <https://www.atlassian.com/software/jira>. Päivitetty 2019. Luettu 20.10.2019.

Laravel from Scratch. 2018. Verkkoaineisto. Laracasts. <https://laracasts.com/series/laravel-from-scratch-2018>. Päivitetty 2019. Luettu 19.10.2019.

LIS Group Oy. 2016. Verkkoaineisto. LIS Group Oy. <www.lis.fi>. Päivitetty 2019. Luettu 1.10.2019.

Lampinen, Heikki. 2019. Kehittäjä, Kiwa Inspecta Oy, Helsinki. Keskustelu 15.10.2019.

Lehtomäki, Antti. 2019. Liiketoiminnan esimies, Kiwa Inspecta Oy, Helsinki. Keskustelu 14.10.2019.

Laravel. 2019. Verkkoaineisto. Laravel. <https://laravel.com/docs/6.x>. Päivitetty 2019. Luettu 24.10.2019.

Laravel Authentication. 2019. Verkkoaineisto. Laravel. <https://laravel.com/docs/5.8/authentication>. Päivitetty 2019. Luettu 24.10.2019.

Manifesto for Agile Software Development. 2001. Verkkoaineisto. Ward Cunningham. <http://agilemanifesto.org/ >. Päivitetty 2001. Luettu 12.10.2019.

Mikä on DevOps? 2019. Verkkoaineisto. Eficode. <https://www.eficode.com/blogi/blogi/mita-on-devops>. Päivitetty 2019. Luettu 1.10.2019.

Punainen Risti ottaa suuren kehitysloikan Kiwa Inspectan kanssa. 2019. Verkkoaineisto. Kiwa. <https://www.kiwa.com/fi/fi/Tiedotus/uutiset/punainen-risti-ottaa-suuren-kehitysloikan-kiwa-inspectan-kanssa--tavoitteena-kaksinkertaistaa-ensiapukoulutettujen-maara-suomessa/>. Päivitetty 2019. Luettu 1.10.2019.

Sass, Css with superpowers. 2006. Verkkoaineisto. Sass. <https://sass-lang.com>. Päivitetty 2019. Luettu 20.10.2019.

Spinellis, Diomidis. 2016. Code Quality. E-kirja. Addison-Wesley Professional.

Scrum Guides. 2008. Verkkoaineisto. Scrum Guide. <https://www.scrumguides.org/>. Päivitetty 2018. Luettu 2.10.2019.

Stoplight. 2019. Verkkoaineisto. Stoplight. <https://stoplight.io>. Päivitetty 2019. Luettu 21.10.2019.

Tietoa Kiwasta. 2019. Verkkoaineisto. Kiwa. <https://www.kiwa.com/fi/fi/tietoa-kiwasta>. Päivitetty 2019. Luettu 1.10.2019.

Tutustu Punaiseen Ristiin. 2019. Verkkoaineisto. Suomen Punainen Risti. <www.punainenristi.fi/tutustu-punaiseen-ristiin>. Päivitetty 2019. Luettu 1.10.2019.

TypeScript Documentation. 2019. Verkkoaineisto. Microsoft. <https://www.typescript-lang.org/docs/home.html>. Päivitetty 2019. Luettu 24.10.2019.

Vance, Stephen. 2015. Quality Code: Software Testing Principles, Practices, and Patterns. E-kirja. Addison-Wesley.

What is GitLab. 2019. Verkkomateriaali. GitLab <https://about.gitlab.com/what-is-gitlab/>. Luettu 22.10.2019.

What is kanban? 2019. Verkkoaineisto. Atlassian. <https://www.atlassian.com/agile/kanban>. Päivitetty 2019. Luettu 20.10.2019.

What is Laravel Framework. 2019. Verkkoaineisto. Educba. <https://www.educba.com/what-is-laravel-framework/>. Päivitetty 2019. Luettu 22.10.2019.

What is Scrum? 2019. Verkkoaineisto. Scrum.org. <https://www.scrum.org/resources/what-is-scrum/>. Päivitetty 2019. Luettu 10.10.2019.