Haaga-Helia
ammattikorkeakoulu Oy

Työaikakirjanpitosovelluksen luominen nykyaikaisilla sovelluskehitystyökaluilla

Matias Ranta

Opinnäytetyö
Tietojenkäsittelyn koulutusohjelma
2018

Tiivistelmä

Haaga-Helia ammattikorkeakoulu Oy

Tekijä(t)
Matias Ranta

Koulutusohjelma
Tietojenkäsittely

Opinnäytetyön otsikko
Työaikakirjanpitosovelluksen luominen nykyaikaisilla sovelluskehitystyökaluilla

Sivu- ja liitesivumäärä
45 + 8

Opinnäytetyön tavoitteena on selvittää, miten verkkosovellus kannattaa toteuttaa suosituimpia web-ohjelmointityökaluja hyödyntäen. Opinnäytetyössä syntyy produktina työaikakirjanpitosovellus, jonka onnistumista selvitetään mittaustutkimuksen avulla. Tutkimustulosten pohjalta syntyy johtopäätös produktin kehityksen onnistumisesta. Opinnäytetyön tuloksista hyötyvät kaikki ohjelmoijat ja yritykset, jotka haluavat parantaa sovelluskehitysprosessiaan ja nykyaikaistaa web-ohjelmointityökalujaan.

Opinnäytetyön tietoperustassa selvitetään työaikaan liittyviä taustoja ja työajanseurantaa. Työajan lisäksi, tietoperustassa selvitetään sovelluskehitykseen liittyviä asioita, kuten sovellusarkkitehtuuria ja erilaisia ohjelmointityökaluja. Opinnäytetyössä ei käsitellä erikseen frontend-ohjelmointiin liittyviä asioita eikä sovelluksen julkaisu- tai tuotteistamisprosesseja.

Opinnäytetyön empiria koostuu kahdesta osasta: opinnäytetyössä syntyneen produktin kehitysvaiheiden raportoinnista ja produktin mittaamisesta. Opinnäytetyön produktina syntyi työaikakirjanpitosovellus, josta on raportoitu projektin taustat sekä suunnittelu- ja ohjelmointivaiheita. Produktin toteutus alkoi marraskuun alussa vuonna 2017, joulukuu oli taukoaikaa ja kehitys päättyi tammikuun lopulla vuonna 2018.

Empirian mittausosiossa taas selvitetään produktin ja vastaavanlaisen tuotteistetun sovelluksen toiminnallisuuksiin kuluvaa aikaa. Mittaustutkimukseen osallistui viisi henkilöä (n = 5), joista syntynyt aineisto on analysoitu kvantitatiivisten ja kvalitatiivisten analysointi menetelmien avulla. Mittaustilaisuudet järjestettiin 19.3. – 25.3.2018 välisenä aikana.

Mittaustulosten pohjalta lopulta ilmenee, että tärkeimmät toiminnallisuudet toimivat nopeammin opinnäytetöiden produktissa. Tuloksista ilmeni myös, että osa toiminnallisuuksien suorituksista onnistui nopeammin vertailtavassa sovelluksessa. Tulosten pohjalta syntyi lopulta johtopäätös, jonka mukaan produktin toteutus onnistui paremmin kuin mitä oli odotettu sekä käytetyt kehitysmenetelmät ja työkalut sopivat projektiin.

Asiasanat
Ohjelmistokehitys, web-ohjelmointi, sovelluskehykset, Python

Sisällys

1 Johdanto

Perinteisten projektinhallintamenetelmien käyttö sovelluskehityksessä on erityisen haastavaa, varsinkin kun koko projekti pitäisi suunnitella kerralla valmiiksi. Olisi suoraviivaista vain aluksi päättää, mitä tehdään ja kun päätös on tehty, niin se vain tehtäisiin. Kuitenkin, sovelluksesta puhuttaessa on vaikea tietää etukäteen lopputulema ennen kuin sitä on alettu luomaan – ikään kuin sovelluskehitysprosessin hallinnointi olisi mahdotonta. Sovelluskehitystä vaikeuttaa hallinnoimisen lisäksi ohjelmointikielten ja sovellusviitekehysten paljous. Miten valita ohjelmointikieli ja sen jälkeen vielä kielelle yhteensopiva viitekehys? (Gleeson 2017, luku 2.)

Tämän opinnäytetyön tarkoituksena onkin selvittää tutkimustyyppisesti, kuinka sovelluskehitysprojekti kannattaa suunnitella ja toteuttaa, jotta projekti pysyy hallittavana. Sovelluskehitysprosessia havainnollistetaan toiminnallisen tutkimusmenetelmän avulla luomalla työaikakirjanpitosovellus. Sovellus luodaan hyödyntäen nykyaikaisia projektinhallintamenetelmiä ja ohjelmointityökaluja. Sovelluksen kehittämisen jälkeen tutkitaan vielä sovelluksen onnistumista triangulaatio-menetelmällä, joka on kvantitatiivisen ja kvalitatiivisen tutkimusmenetelmien yhdistelmä.

Kvantitatiivisessa eli määrällisessä tutkimuksessa mitataan produktina syntyneen työaikakirjanpitosovelluksen sekä vastaavanlaisen tuotteistetun sovelluksen toiminnallisuuksien suorittamiseen kuluvaa aikaa. Kvalitatiivisessa eli laadullisessa tutkimuksessa taas selvitetään mittaustutkimukseen vaikuttavia tekijöitä. Selvityksien pohjalta saadaan lopulta vastaukset seuraaviin kysymyksiin:

1. Miten sovelluskehitysprosessi kannattaa suunnitella ja toteuttaa, jotta se pysyy hallittavana?
2. Mitkä ovat suosituimpia backend-ohjelmointityökaluja ja miten valita sopivimmat työkalut ohjelmointiin?

1.1 Opinnäytetyön tavoitteet ja rajaukset

Opinnäytetyön tavoitteena on ratkaista kaksi tutkimusongelmaa ja parantaa opinnäytetyön tekijän ohjelmointi- ja analysointitaitoja. Ratkaistavat tutkimusongelmat ovat: miten sovellus kannattaa kehittää hallitusti ja mitkä ovat nykyajan suosituimpia ohjelmointityökaluja. Ratkaisu jakautuu kahteen osioon, joista ensimmäisessä osiossa luodaan toiminnallisin menetelmin työaikakirjanpitosovellusprodukti. Toiminnallisessa osiossa raportoidaan, miten sovellus on kehitetty nykyaikaisia kehitysmenetelmiä hyödyntäen. Raportissa selvitetään, kuinka sovellus on hahmoteltu, suunniteltu ja lopuksi ohjelmoitu.

Raportoinnin jälkeen siirrytään toiseen osioon, jossa on selvitetty määrällisin ja laadullisin tutkimusmenetelmin, kuinka toimiva produkti lopulta on ollut. Produktin toimivuutta on selvitetty mittaustutkimuksella, jossa on mitattu kuinka paljon aikaa produktin sekä vastaavanlaisen tuotteistetun sovelluksen toiminnallisuuksien suorittamiseen kului viideltä tutkimukseen osallistuneelta henkilöltä. Tutkimuksen tulosten pohjalta luodaan lopulta johtopäätökset opinnäytetyössä kehitetyn sovelluksen onnistumisesta. Tuloksia ja produktin raportointia voivat hyödyntää kaikki ohjelmoijat ja yritykset, jotka haluavat päivittää sovelluskehitysmenetelmiään ja ohjelmointityökalujaan.

Opinnäytetyössä on pääosin keskitytty backend-teknologioihin sekä tietoperustassa että toiminnallisessa osiossa, jonka seurauksena opinnäytetyössä ei erikseen käsitellä sovelluksen frontendiin liittyviä työkaluja. Opinnäytetyössä ei myöskään käsitellä sovelluksen julkaisuun tai tuotteistamiseen liittyviä vaiheita. Ilman kyseenomaisia rajauksia, opinnäytetyön laajuus olisi kasvanut liian isoksi, jolloin työ ei olisi valmistunut toivotussa aikataulussa.

1.2 Opinnäytetyön rakenne

Rakenteeltaan opinnäytetyö noudattaa perinteistä Haaga-Helian tutkimustyyppistä raportointimallia. Opinnäytetyö koostuu johdannosta, tietoperustasta, empiriasta sekä pohdinnasta. Opinnäytetyön ensimmäisessä luvussa eli johdannossa, selvitetään opinnäytetyön tavoitteet, tutkimusongelmat, rajaukset sekä käsitteet. Tietoperusta jakautuu lukuihin kaksi ja kolme, joista luvussa kaksi, perehdytään työajan kirjaamissääntöihin ja seurantaan. Kolmannessa luvussa taas selvitetään sovelluksen suunnitteluun liittyviä menetelmiä sekä esitellään suosittuja backend-ohjelmointityökaluja.

Tietoperustan jälkeen opinnäytetyössä siirrytään empiriaan, joka on jaettu lukuihin neljä ja viisi. Neljännessä luvussa kuvaillaan opinnäytetyön toiminnallista osiota, jossa ilmenee, miten produktina syntynyt työaikakirjanpitosovellus on suunniteltu ja luotu tietoperustan pohjalta. Toisessa empirian osassa eli opinnäytetyön viidennessä luvussa tutkitaan produktin toimivuutta mittaamalla produktin ja vastaavanlaisen tuotteistetun sovelluksen toiminnallisuuksien suorittamiseen kuluvaa aikaa. Opinnäytetyön kuudennessa luvussa pohditaan produktin onnistumista peilaten mittaustuloksiin. Pohdinta-luvussa selvitetään myös tutkimuksen eettisyyttä ja luotettavuutta, sekä opinnäytetyön onnistumista sekä tekijän omaa oppimista.

1.3 Käsitteet

Backend	Sovelluksen palvelinpuoli, jossa hoidetaan mm. tietokantakyselyjen määritys, sovelluksen bisneslogiikka ja datan reitittäminen näkymäpuolelle eli frontendille.
Frontend	Sovelluksen näkymäpuoli, joka koostuu mm. datan näyttämislogiikan käsittelystä sekä käyttöliittymästä eli kaikesta näkymistä ja interaktiivisuuksista.
Käyttötapaus	Osa sovelluksen vaatimusmäärittelyä, jossa kuvataan käyttäjän ja palvelun välistä vuorovaikutusta.
MVP	Ideologia, joka on lyhenne sanoista minimum-viable-product. Sen tarkoituksena on luoda yksinkertaistettu toimiva tuote mahdollisimman tehokkaasti.
Sovellus	Tietokoneohjelma, jonka tarkoitus on ratkaista jokin ongelma tai helpottaa jonkin asian tekemistä.
URL-osoite	Merkkijono, joka kertoo missä osoitteessa jokin tieto sijaitsee.

2 Työaika

Suomen työaikalaki määrittää työajaksi ajan, jonka työntekijä on käyttänyt työn tekemiseen. Lisäksi, työaikalain mukaan työajaksi lasketaan aika, jona työntekijä on ollut työnantajan saatavilla. Säännöllistä työaikaa saa järjestää työntekijälle vuorokaudessa enintään kahdeksan tuntia ja viikossa yhteensä 40 tuntia. Viikoittaista säännöllistä työtä voi järjestää työntekijälle 40 tuntia tasoittuen keskimääräisesti 52 viikon ajalle. Työnantaja ja työntekijä voivat soveltaa sopimuksen yksityiskohtia haluamallaan tavalla, niin kauan kuin molemmat osapuolet hyväksyvät sopimuksen ehdot ja ehdot noudattavat työehtosopimusta. (Työaikalaki 605/1996.)

Työaikalaissa säännöllisen työn lisäksi on määritelty erikseen lisä-, yli-, hätä- ja sunnuntaityö. Lisätyötä on säännöllisen työn lisänä tehty työ, joka pysyy säännöllisen työn tunti- ja viikkokohtaisissa rajoissa. Ylityöllä taas tarkoitetaan aikaa, joka ylittää säännölliseksi työksi määritetyn ajan. Työnantaja saa teettää ylitöitä enintään neljän kuukauden aikana 138 tuntia ja kaiken kaikkiaan 250 tuntia koko vuoden aikana. Hätätyöksi taas on määritelty työaika, jona työntekijä on ollut pelastamassa jonkun hengen tai terveyden, tai turvannut omaisuutta. Sunnuntaityö on työaikaa, jota on teetetty kirkollisena juhlapäivänä tai nimensä mukaisesti sunnuntaina. Sunnuntaityötä saa teettää, jos se on määritelty työsopimuksessa säännölliseksi työajaksi tai jos työntekijä on erikseen antanut siihen suostumuksensa. (Työaikalaki 605/1996.)

2.1 Joustava työaika

Teknologian kehityksen seurauksena työn tekemisen joustavuus on parantunut, jonka seurauksena tehdyn työn ajankohta ja paikka voivat vaihtua työntekijän tai työnantajan tarpeiden mukaan. Sen seurauksena työntekijä voi tehdä etätöitä esimerkiksi työnantajan eri toimipisteissä, kotona tai vaikka junassa. Eri työskentelypaikkojen lisäksi, työtä voi tehdä myös osissa; esimerkiksi kolme tuntia heti aamusta ja loput vaadittavista töistä myöhemmin illalla tai vaikka seuraavana päivänä. Sen takia tehdyn työn kestot voivat vaihdella huomattavasti. (Työterveyslaitos 2016.)

Joustava työaika sopii varsin hyvin työhön, jossa on kyse hallinnoinnista, keskittymisestä tai luovuudesta. Työntekijä voi hyötyä joustavasta työajasta muun muassa siten, että työrauha ja keskittyminen voi parantua paikan vaihtamisella. Keskittymisen parantuessa, työteho paranee, josta hyötyy suoraan myös työnantaja. Joustavan työajan ansiosta työntekijä voi säästää myös aikaa ja rahaa, jotka muuten kuluisivat matkustamiseen. Työnteki-

jän tehdessä töitä muualla kuin työnantajan toimipisteellä, työnantaja säästää myös toimitilojen kustannuksissa. (Työterveyslaitos 2016.)

Joustavalla työajalla voi olla myös haittavaikutuksia. Työntekijän työajat saattavat venähtää sekä työ ja vapaa-aika voivat sekoittua, jonka seurauksena työstä voi olla vaikea irtautua. Liiallinen työskentely etänä saattaa myös loitontaa työyhteisöstä, jolloin työntekijä voi kokea olonsa yksinäiseksi. (Työterveyslaitos 2016.)

2.2 Työaikakirjanpito

Työaikalain 7 luvun § 37 mukaan työnantajan täytyy yksitellen kirjata kunkin työntekijän työtunnit sekä korvaukset seurauksena tehdyistä työtunneista. Työaikakirjanpidossa pitää lain mukaan näkyä työntekijän tekemät säännölliset työajan työtunnit. Säännöllisten työajan työtuntien lisäksi työaikakirjanpidossa pitää merkata erikseen työntekijän lisä-, yli-, hätä- ja sunnuntaityötunnit. (Työaikalaki 605/1996.)

Työnantajan tulee työaikalain mukaan säilyttää työaikakirjanpitoa vähintään 2 vuotta sopimuksen päättymisen kalenterivuoden jälkeen. Työaikakirjanpitoa pitää vaadittaessa näyttää työntekijöiden edustajalle sekä työsuojelutarkastuksen toimittajalle. Työnantajan tulee myös työntekijän vaatiessa luoda selvitys työntekijää koskevista merkinnöistä työaikakirjanpidossa. (Työaikalaki 605/1996.)

Työaikakirjanpito on tärkeää tilanteissa, joissa työnantajan ja työntekijän pitää selvittää erimielisyystilanteita. Erimielisyyttä saattaa syntyä muun muassa työntekijän tekemistä työtunneista ja niistä saaduista korvauksista. Mikäli työnantaja on hoitanut työaikakirjanpidon huonosti, tuomioistuin useimmiten katsoo tilanteen työntekijän eduksi, jos työntekijällä on riittävän hyvät selvitykset ja muistiinpanot työtunneistaan ja korvauksistaan. (Työsuojelu 2017.)

2.3 Työajanseuranta

Kehitystiimin kasvaessa ja asiakkaiden liittyessä mukaan projekteihin, tuotannon seurannan tärkeys korostuu. Tuolloin projektin eteneminen ei kiinnosta ainoastaan työnantajaa vaan myös asiakasta, sillä projektin osien valmistuessa, projektista voi tarvittaessa eritellä osia esimerkiksi uudelleenkäyttöä tai vaikka asiakkaan vakuuttamista varten. Seurannan avulla on mahdollista selvittää, pysyykö projekti aikataulussa ja budjetissa, ja jos ei niin hyvä seuranta auttaa selvittämään miksi esimerkiksi budjetti on ylittynyt. (Zwerman & Okun 2017.)

Projekti on jaettavissa useampiin yksittäisiin tehtäviin (engl. task), jossa jokaiselle tehtävälle on arvioitavissa oma valmistumisaikansa. Tehtävien valmistumisarvioiden avulla määrittyy kokonaisuus projektin budjetista ja aikataulusta. Mikäli esimerkiksi tiettyyn tehtävään arvioitu aika on ylittynyt tai on ylittymässä, se on seurannan avulla havaittavissa. Tällöin on mahdollista järjestää aputoimenpiteitä, jotka pitävät vahingot pienimillään. (Zwerman & Okun 2017.)

3 Sovelluskehitys

Sovelluskehitys tarkoittaa sovelluksen elinkaarta, eli toisin sanoen työvaiheita, joita tarvitaan sovelluksen luomiseen. Sovelluskehitys on jaettavissa neljään eri päävaiheeseen, jotka ovat jaettavissa vielä pienemmiksi tehtäviksi. Ensimmäinen päävaihe koostuu hahmottelusta, jonka tarkoituksena on selvittää sovelluksen käyttäjät ja käyttötapaukset (engl. use case). (García 2017.)

Käyttäjien ja käyttötapausten hahmotuttua, sovelluskehityksessä siirrytään seuraavaan vaiheeseen, jossa suunnitellaan sovelluksen arkkitehtuuri sekä sovelluksen osien sisäiset suhteet. Suunnittelun jälkeen käynnistyy kolmas vaihe, jossa luodaan sovellus ohjelmoimalla eli koodia kirjottamalla. Neljäs ja viimeinen vaihe taas kattaa sovelluksen käytön mahdollistamisen, eli esimerkiksi sovelluksen asentamisen, käynnistämisen, testauksen ja julkaisemisen loppukäyttäjille tai asiakkaalle. (García 2017.)

3.1 Sovelluskehitysmenetelmät

Aiemmin sovelluskehityksessä korostui luotavan systeemin hallittavuus ja vakaus. Sen seurauksena luodut systeemit ja niiden kehitykseen käytetyt työkalut olivat paljon yksinkertaisempia. Nykyään taas ideat joiden pohjalta systeemit kehitetään ovat monimutkaisempia ja ideoita on huomattavasti enemmän kuin aikoinaan. Ideoiden toteuttaminen toimiviksi palveluiksi tulee myös tapahtua nopeasti ja joustavasti, ilman että palvelun hallittavuus tai vakaus kärsii. Sen seurauksena tuotteistamiseen on kehitetty uudenlaisia kehitysmenetelmiä, joiden avulla tuotteen kehitys on nopeaa ja joustavaa, sekä lopputuloksena syntynyt tuote on hallittavissa ja vakaa. (Knaster & Leffingwell 2017, luku 3.)

Suosituimpia sovelluskehityksen tehostusmenetelmiä ovat Lean-tuotekehitys ja ketterä kehitys. *Lean-tuotekehitys* (engl. Lean product development) on yhdistelmä Lean-ajattelua ja Toyotan tuotekehityssysteemiä. Lean-ajattelun pääidea perustuu ylijäämien ja viivästyksien poistamiseen arvioimalla käytössä olevia prosesseja jatkuvasti. Lean-ajattelumallia on sovellettu tuotekehitykseen, jonka pohjalta on lopulta syntynyt Lean-tuotekehitys. Lean-tuotekehityksessä korostuu keskittyminen idean arvoon ja arvon säilyttäminen, kun ideaa kehitetään tuotteeksi. (Knaster & Leffingwell 2017, luku 3.) Kehitettävän asian arvoon Lean-tuotekehityksessä Knasterin ja Leffingwellin (2017, luku 3) mukaan vaikuttaa:

- Turhien vaiheiden ja hidasteiden minimointi.
- Kehityksen hallinnointi hyvällä rytmityksellä ja kehityksessä olevien asioiden rajoittaminen sopivan suuruiseksi.
- Työntekijöiden ja heidän kulttuurinsa kunnioitus, sillä he lopulta tekevät työn.

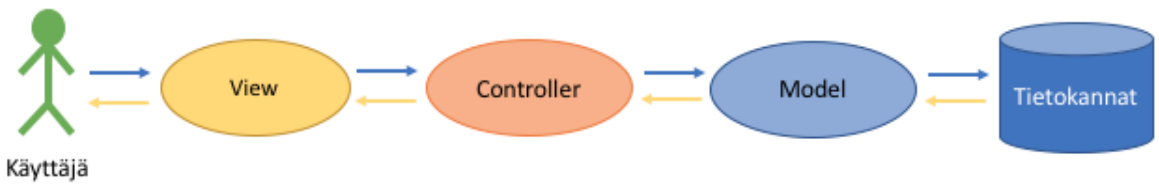
- Innovointiprosessin toteutus MVP-mallia (minimum viable product) hyödyntäen, jonka tarkoituksena on ideoida tuotteen tarvittavat ominaisuudet nopeasti ja halvalla.

Ketterä kehitys (engl. agile development) on useiden sovelluskehitysviitekehysten pohjalta syntynyt menetelmä, joka yhdistää useat filosofiat ja käytänteet yhdeksi malliksi. Yhtenäinen ketterän kehityksen manifesti luotiin vuonna 2001, jolloin sitä on myös viimeksi päivitetty. Ketterän kehityksen manifestissa on korostettu neljä pääarvoa. Ensimmäisenä arvona manifestissa on määritetty yksilön ja vuorovaikutuksen priorisointi ennen työkaluja ja menetelmiä. Toisena arvona on ohjelmiston toimivuuden takaaminen ennemmin kuin kattavan dokumentaation saatavuus siitä. Kolmannen arvon mukaan asiakkaan kanssa työskennellessä yhteistyö on neuvottelua tärkeämpää. Lopuksi neljäntenä arvona on muutoksiin sopeutumisen ensisijaistaminen alkuperäisessä suunnitelmassa pysymisen sijaan. (Ketterän ohjelmistokehityksen julistus 2001; Knaster & Leffingwell 2017, luku 3.)

3.2 Sovellusarkkitehtuuri

Verkkosovellukset rakentuvat arkkitehtuurillisesti yleensä data-, logiikka- ja näyttämistasoista. Data-taso vastaa tietokantoja, joihin tallentuu sovelluksen käyttämä tieto. Logiikkataso sisältää laitteen ja sovelluksen välisen kommunikaatioon tarvittavan koodin. Näyttämistaso taas kuvaa sovelluksen näkymä-osaa. (Haviv, Mejia & Onodi 2016, luku 1.1.)

Sovellusten arkkitehtuurillisten tasojen mallinnukseen on olemassa erilaisia malleja, joista yksi suosituimmista on *MVC-malli*. Mallin lyhentyy osista Model-View-Controller, joista jokaisella on oma tehtävänsä. Model heijastaa sovelluksen tietokantojen rakennetta sekä vastaa datan käsittelystä ja kulusta. View on sovelluksen näkyvä ja interaktiivinen osa, jonka käyttäjä näkee ja klikkailee. Controllerin tehtävä on välittää käskyjä ja dataa eli tietoa Modelin ja Viewin välillä. (Haviv ym. 2016, luku 1.1.) Kuvassa 1 on selvitetty MVC-mallin, tietokantojen sekä käyttäjän välisiä yhteyksiä ja tiedon kulkua. Kuvan nuolet havainnollistavat tiedon kulkua eri osioiden välillä: siniset nuolet kuvaavat pyyntöjä (engl. request) ja keltaiset vastauksia (engl. response).



Kuva 1: Kuvio MVC-mallin, käyttäjän ja tietokantojen välisistä suhteista (Haviv ym. 2016, luku 1.1)

MVC-mallin pohjalta on kehitetty ajan saatossa muita arkkitehtuurimalleja, kuten esimerkiksi *MVT-malli*. MVT-mallin (Model-View-Template) idea on samanlainen kuin MVC-mallissa, mutta osioiden roolit ja nimet ovat hieman erilaiset. MVT-mallissa View-osio vastaa MVC-mallin Controller-osiota ja Template korvaa MVC-mallin Viewin. MVT-mallin rakenteen idea pohjautuu työelämään, jossa suunnittelijat (engl. designer) voivat keskittyä ainoastaan templaatteihin, jotka sisältävät sovelluksen HTML-koodin. Tuolloin suunnittelijoiden ei tarvitse välittää kehittäjien (engl. developer) koodista, joka taas sijaitsee Model- ja View-osioissa. (Dauzon, Bendoraitis & Ravindran 2016, luku 1.1.)

3.3 Suosituimmat web-ohjelmointikielet

IEEE Spectrumin selvityksen mukaan viisi suosituinta web-ohjelmointikieltä alkaen suosituimmasta kielestä ovat Python, Java, C#, JavaScript ja PHP. Taulukossa 1 on listattuna viisi suosituinta web-ohjelmointikieltä IEEE Spectrumin tekemän selvityksen mukaan. Selvityksen mukaan eniten pisteitä on saanut Python pistein 100. Taulukkoon on kerätty kaikki web-ohjelmointikielet, jotka ovat Pythonin lisäksi saaneet yli 80 pistettä. (IEEE Spectrum 2017a.)

Taulukko 1: Viisi suosituinta web-ohjelmointikieltä IEEE Spectrum (2017a) luoman selvityksen mukaan

Sijoitus	Kieli	Pisteet
1.	Python	100.0
2.	Java	99.4
3.	C#	88.6
4.	JavaScript	85.5
5.	PHP	81.4

Ohjelmointikielen suosio koostuu 12 eri mittarista, jotka taas perustuvat kymmeneen eri lähteeseen. Lähteet on luokiteltu kolmeen eri kategoriaan: sosiaalinen keskustelu, avoimen lähdekoodin kehitys ja avoimet työpaikat. Sosiaalisen keskustelun lähteisiin lukeutuivat muun muassa Google Search, Twitter ja Reddit, joissa mitattiin ohjelmointikieliin liittyvien hakujen ja keskustelujen määriä. (IEEE Spectrum 2017b.)

Avoimen lähdekoodin kehityksen lähteisiin taas lukeutuivat GitHub ja Stack Overflow -sivustot. GitHubissa laskettiin ohjelmointikielillä luotujen uusien ohjelmavarastojen (engl. repository) määriä, sekä jo olemassa olleiden ohjelmavarastojen aktiivista käyttöä. Stack Overflow:ssa laskettiin sekä kysymysten että vastausten lukumääriä ohjelmointikielten perusteella. (IEEE Spectrum 2017b.)

Kolmas sovelluksen lähdetyyppi eli avoimet työpaikat, perustuu Dice ja CareerBuilder -sivustoihin. Molemmissa lähteissä laskettiin enintään 30 päivää vanhojen työpaikkailmoitusten lukumäärät. Lukumäärien laskeminen tapahtui yhdistämällä haussa ohjelmointikielen nimi ja sana ohjelmointi (engl. programming), eli esimerkiksi yksi hakusyötteistä oli ”Java ohjelmointi” (engl. Java programming). (IEEE Spectrum 2017b.)

3.4 Web-viitekehysten taustaa

Web-viitekehys tarkoittaa työkalua, joka ohjaa kehittäjää luomaan sovellusta tietyllä tavalla. Web-viitekehys koostuu erilaisista työkaluista, joilla verkkosivuston luominen onnistuu ilman, että jo olemassa olevia asioita tarvitsisi keksiä uudelleen. Web-viitekehys hoitaa lukuisia asioita ohjelmoijan puolesta, jonka seurauksena sovelluksen kehittämisvaiheessa kehittäjän tarvitsee keskittyä ainoastaan lopullisen ratkaisun kannalta olennaiseen. (Romano, Phillips & van Hattem 2016, luku 1.10.)

Web-viitekehys on jaettavissa erilaisiin rooleihin: käyttäjiin, selaimeen ja backendiin eli palvelimeen ja sen palveluihin. Verkkosovelluksessa käyttäjät luovat sovelluksen vaatimukset, joiden pohjalta syntyvät ominaisuudet, sillä sovellus luodaan juuri sen käyttäjiä varten. Käyttäjien tehtävä viitekehyksissä on esimerkiksi klikkailla käyttöliittymää ja antaa syötettä tekstikenttiin, eli toisin sanoen käyttää sovelluksen interaktiivista ja näkyvää osaa. (Dayley, Dayley & Dayley 2017, luku 1.)

Selaimen tarkoituksena taas on huolehtia käyttäjän ja palvelimen välisestä kommunikaatiosta, joka syntyy käyttäjän interaktiosta. Interaktion pohjalta selain lähettää serverille dataa ja tulostaa serverin vastauksen näytölle nähtäväksi sopivaksi muotoiltuna. Kommunikaatio selaimen ja palvelimen välillä tapahtuu HTTP (Hypertext Transfer Protocol) ja HTTPS (Hypertext Transfer Protocol Secure) –protokollien avulla. Protokollat määrittelevät data-kutsujen tyypit, sekä muokkaavat selaimesta lähetettävän ja palvelimelta vastaanotettavan datan oikeaan muotoon. (Dayley ym. 2017, luku 1.)

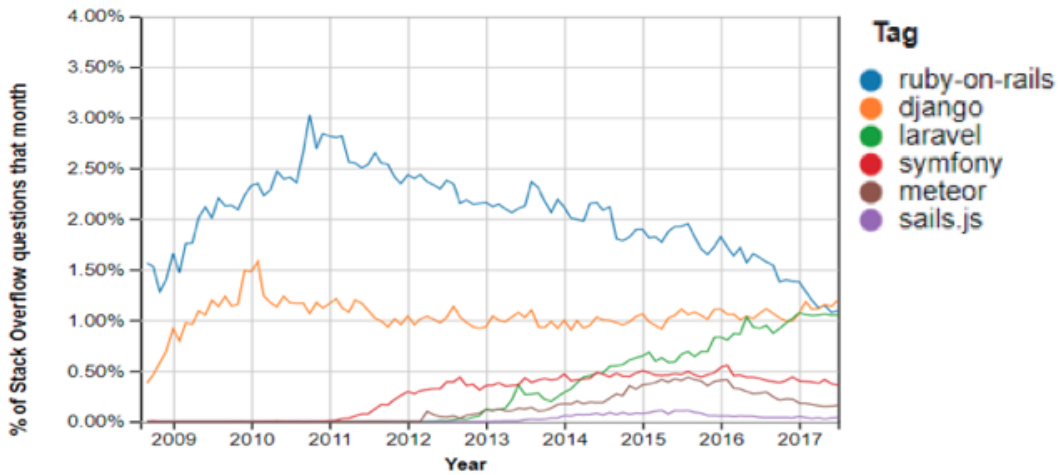
Palvelimen tehtävä viitekehyksessä on käsitellä selaimen lähettämiä pyyntöjä HTTP- tai HTTPS-protokollan perusteella. Pyynnön tyypin perusteella palvelin voi esimerkiksi noutaa, muuttaa tai lisätä dataa, joka sijaitsee tietokannoissa. Hyvä verkkopalvelin viitekehys voi helpottaa huomattavasti pyyntöjen sitomista esimerkiksi sopiviin tietokantakyselyihin ja palvelinpuolella sijaitsevien koodipätkien ajamiseen. (Dayley ym. 2017, luku 1.)

Seuraavaksi tässä luvussa on selvitetty suosituimpia web-viitekehyksiä GitHub ja Stack Overflow -sivustojen selvitysten pohjalta. Taulukossa 2 on listattuna GitHub-sivuston (2018a) ilmoittamat web-viitekehykset tähtien mukaan. Taulukossa on myös ilmoitettu viitekehyksen käyttämä ohjelmointikieli. GitHubin (2018b) mukaan tähtien lukumäärällä kuvataan, kuinka suosittu ja arvostettu kyseenomainen tähdellä merkattu asia on, ja tässä tapauksessa asialla tarkoitetaan web-viitekehystä. Taulukon 2 mukaan Meteor, Ruby on Rails, ja Laravel ovat saaneet yli 38 000 tähteä. Lähellä kolmen kärkeä taulukossa seuraa Django 31 000 tähdellä. Jokainen neljästä suosituimmasta viitekehyksestä on kirjoitettu eri ohjelmointikielellä, jotka ovat JavaScript, Ruby, PHP ja Python.

Taulukko 2: Fullstack web-viitekehykset GitHubin (2018a) selvityksessä, jotka ovat saaneet vähintään 15 000 tähteä

Sijoitus	Tähtien lkm.	Nimi	Kieli
1.	39 086	Meteor	JavaScript
2.	38 392	Ruby on Rails	Ruby
3.	38 274	Laravel	PHP
4.	31 322	Django	Python
5.	18 318	Sails	JavaScript
6.	16 429	Symfony	PHP

Kuva 2 selvittää viivadiagrammin muodossa web-viitekehyksien suosion vaihtelua ajalta 2009 – 2017. Diagrammi selvittää web-viitekehyksien osuutta kaikista Stack Overflow –sivustolla esitetyistä kysymyksistä ja vastauksista. Kuva 2 perustuu Stack Overflow Trends -selvitykseen, jonka mukaan kolme suosituinta web-viitekehystä ovat olleet Django, Ruby on Rails ja Laravel. Keskustelu liittyen Ruby on Railsiin on ollut selvässä laskussa vuodesta 2011 alkaen. Huomattavaa nousua sen sijaan on tehnyt Laravel vuodesta 2012 alkaen. Djangolla keskustelujen muutos on taas ollut viitekehyksistä tasaisinta.



Kuva 2: Kuvakaappaus Stack Overflown (2018) selvityksestä viivadiagrammin muodossa, joka selvittää web-viitekehysten suosiota Stack Overflow -sivustolla

Ruby on Rails, joka tunnetaan myös nimellä Rails, on web-viitekehys, joka pohjautuu Ruby-ohjelmointikieleen. Railsin ensimmäinen beeta-versio julkaistiin vuonna 2004 ja viimeisin versio viitekehyksestä 5.0 julkaistiin vuonna 2016. Rails on rakennettu noudattaen MVC-sovellusarkkitehtuurimallia ja hyödyntää lukuisia hyviä ohjelmointiperiaatteita. Yksi oleellisimmista periaatteista on ”convention over configuration”, jonka idea on vähentää ohjelmoidessa tehtävien päätösten määrää. (Goodrich & Lenz 2016, luku 1.)

Vuonna 2003 julkaistu *Django* on yksi suosituimmista web-viitekehyksistä. Django julkaistiin BSD-lisenssillä tehden sen käytöstä ja muokkaamisesta rajoituksetonta sekä täysin ilmaista. Djangon tarkoitus on toimia alustana, jolla pystyy nopeasti luomaan turvallisen ja toimivan verkkosivuston vähäisellä koodin määrällä. Vähäisen koodin kirjoittamisen määrä perustuu DRY-periaatteeseen (Don’t Repeat Yourself), jossa vältetään kirjoitetun koodin toistamista useissa eri paikoissa. Eloisan yhteisön omaava Django pohjautuu Python-ohjelmointikieleen, ja se tukee Pythonin versioita 2 ja 3. Python-kielen ansiosta Django-sovelluksessa voi käyttää muita Python-kirjastoja, sekä Pythonilla kirjoitettu sovellus on luonnostaa helppolukuista. Sovellusarkkitehtuurimallinaan Django käyttää MVT-mallia, joka taas perustuu MVC-malliin. (Dauzon, Bendoraitis & Ravindran 2016, luku 1.1.)

Laravel on vuonna 2011 julkaistu PHP-ohjelmointikieleen pohjautuva web-viitekehys. Laravel käyttää MVC-arkkitehtuurimallia, jonka seurauksena Laravel-viitekehyksellä luodut verkkosovellukset ovat syntyessään vankkarakenteisia. Laravelin päätarkoitus on palvella kehittäjiä yksilöinä, jonka takia viitekehyksessä on keskitytty alustan koodin ja toiminnallisuuksien selkeyteen ja yksinkertaisuuteen. Selkeän ja yksinkertaisen koodin tarkoituksena on tehdä viitekehyksen oppimisesta ja soveltamisesta helppoa ja nopeaa, sekä kannustaa ohjelmoijia luomaan selkeää, yksinkertaista ja kestävää koodia. (Gore 2017; Stauffer 2016, luku 1.)

Kaksi tärkeintä Laraveliin liittyvää arvoa ovat kehittäjän onnellisuus sekä kehitysnopeus. Laravel luo onnellisuutta keskittymällä viitekehyksen käytön ja soveltamisen helpottamiseen. Kehitysnopeutta taas Laravel tehostaa vähentämällä tarvittavia vaiheita sovelluksen kehitysprosessissa. Laravelin arvojen seurauksena viitekehyksen taustalle on muodostunut rikkaan energinen yhteisö. Yhteisö koostuu sekä uusista kehittäjistä että vanhoista yksilöistä jotka ovat olleet web-viitekehyksen kehityksen mukana alusta alkaen. (Stauffer 2016, luku 1.)

Vuonna 2011 lopulla julkaistu Skybreak, joka sai noin kuukauden myöhemmin nimekseen *Meteor*, on JavaScript-ohjelmointikieleen perustuva web-viitekehys. Meteorin lähtökohta-

na on ollut uudistaa vuosikymmeniä vanhoja sovelluskehitystekniikoita. (DeBergalis 1.12.2011; DeBergalis 20.1.2012; Robinson, Gray & Titarenco 2016, luku 2.) Tekniikat pohjautuvat seitsemään eri periaatteeseen Robinsonin ja muiden (2016, luku 2) mukaan seuraavasti:

- Anna frontendin hoitaa palvelimelta tulevan datan visualisointi.
- Vähennä käyttöliittymän käytön viiveitä hyödyntämällä ennakoivia hakuja frontendissä.
- Päivitä tietokantoja ja käyttöliittymää reaaliaikaisesti.
- Kirjoita verkkosovelluksen frontend ja backend -osiot yhdellä kielellä; JavaScriptillä.
- Keskustele tietokantojen, front- ja backendin välillä yhdenmukaisella API:lla.
- Hyödynnä Node.js:n ekosysteemiä ennen kuin lähdet kehittämään uudelleen jo jotakin olemassa olevaa.
- Pyri yksinkertaisuuteen, jotta Meteorin käytettävyys säilyy.

Idea datan visualisoinnista frontendissä perustuu selainten kykyyn lähettää kyselyjä takaisin palvelimelle ilman sivun uudelleenlatausta. Sen seurauksena sivusta tarvitsee päivittää ainoastaan tietty osa, joka taas parantaa sovelluksen käyttökokemusta. Sovelluksen käyttökokemusta parantaa myös ennakoivien tietokantahakujen toteuttaminen frontendissä, jolloin käyttäjien ei tarvitse odottaa sovelluksen vastauksia niin pitkään. (Robinson ym. 2016, luku 2.)

Käyttöliittymän käyttökokemuksen parannusperiaatteiden lisäksi Meteor keskittyy myös ohjelmoijan työskentelyn helpottamiseen. Yhden kielen käyttäminen sekä front- että backendissä vähentää opeteltavien kielten määrää, helpottaen koko sovelluksen ymmärtämistä. Kehitystyötä nopeuttaa ja helpottaa yhden kielen lisäksi myös periaate yhdenmukaisuudesta tietokantojen, back- ja frontendin välillä. Kaiken kaikkiaan yksinkertainen lähestymis- ja kehitystapa nopeuttaa sekä helpottaa asioiden tekemistä ja vähentää työstressiä. (Robinson ym. 2016, luku 2.)

3.5 Tietokannan hallintajärjestelmät

Tietokanta tarkoittaa ydinajatukseltaan datan eli tiedon säilyttämistä järjestelmällisesti tietyn rakenteen mukaan. Datan hallinnointiin on olemassa erilaisia hallinnointijärjestelmiä (engl. database management system, tästä eteenpäin DBMS), jotka hoitavat muun muassa tiedon siirtelyä ja muokkaamista ennalta määritetyn datasanaston (engl. data dictionary) perusteella. DBMS hallinnoi tietokannan dataa datasanaston pohjalta, josta ilmenee datan määritykset, suhteet, alkuperä, käyttötarkoitus ja formaatti. (Southekal 2017, luku 3.)

Tietokantaa ei voi yleensä siirrellä DBMS:ien välillä, mutta eri DBMS:ät voivat kuitenkin hyödyntää samaa tietokantaa erilaisten työkalujen avulla. Yksi tunnetuimmista työkaluista,

joka mahdollistaa työskentelyn useamman DBMS:än välillä on SQL. SQL on lyhenne sanoista Structured Query Language, joka tarkoittaa vapaasti käännettynä rakenteellista kyselykieltä. (Southekal 2017, luku 3.)

Kaksi yleisintä olemassa olevaa tietokantatyyppiä ovat SQL ja NoSQL. SQL-tietokanta, joka tunnetaan myös nimellä relaatiotietokanta (engl. relational database), käsittelee dataa tietyn rakenteen mukaan. SQL-tietokannan rakenne perustuu suunniteltuun malliin (engl. schema), jossa on määritettynä esimerkiksi tietokantataulujen väliset suhteet sekä taulun kenttien tyypit. Tietokannan mallin ja sen määritysten tarkoituksena on vähentää turhan datan tallentamista ja varmistaa, että data pysyy luotettavana. (Southekal 2017, luku 3.)

NoSQL-tietokanta (engl. non-relational database) taas käsittelee dataa, jonka jäsentely ei perustu malliin, vaan kaikki data sijaitsee yhdessä tiedostossa. Sen seurauksena NoSQL-tietokanta on nopeampi ja joustavampi kuin perinteinen SQL-tietokanta. NoSQL-tietokanta vaatii kuitenkin enemmän laskentatehoja ja tallennustilaa, ja datan säilyvyys ei ole samalla tasolla kuin SQL-tietokannoissa. Molemmissa tietokantojen hallinnointijärjestelmätyypeissä on omat vahvuutensa, jonka seurauksena useat yritykset hyödyntävät molempia vaihtoehtoja. (Southekal 2017, luku 3.)

Tietokannoilla luotettavuus korostuu eteenkin, kun on kyse useammasta kuin yhdestä samanaikaisesta tietokannan käyttäjästä. Tuolloin, tietokannoissa voi tapahtua useita transaktioita eli tietokannan suorittamia tehtäviä yhtä aikaa. SQL tyyppiset tietokannanhallintajärjestelmät pyrkivät tekemään transaktioista luotettavia noudattamalla ACID-mallia, joka lyhentyy sanoista atomisuus, eheys, eristyneisyys ja pysyvyys (engl. atomicity, consistency, isolation ja durability). (Harrington 2016, luku 22.) Harringtonin (2016, luku 22) mukaan sanat tarkoittavat seuraavaa:

- Atomisuus tarkoittaa, että transaktio joko onnistuu tai epäonnistuu täysin, jolloin osittainen transaktion onnistuminen ei ole mahdollista.
- Eheys vastaa tietokantojen tilan säilyvyydestä, eli tietokantojen tila pysyy eheänä koko ajan.
- Eristyneisyys huolehtii, että samanaikaisesti tapahtuvat transaktiot eivät vaikuta toisiinsa.
- Pysyvyys tarkoittaa, että transaktio ei peruunnu tai muutu sen valmistuttua.

NoSQL-tietokannanhallintajärjestelmät hyödyntävät ACID-mallin sijaan BASE-mallia tietokantojen luotettavuuden takaamiseksi. BASE on lyhenne sanoista Basically Available, Soft state ja Eventual Consistency. Idealtaan sanat tarkoittavat seuraavaa: tieto on saatavilla, tieto saattaa muuttua transaktioiden aikana ja lopulta tieto on aina oikeellista. Verrat-

tuna ACID-malliin BASE siis asettaa tiedon saatavuuden luotettavuuden edelle. (Harrington 2016, luku 28.)

Seuraavaksi tässä luvussa on käsitelty suosituimpia tietokannan hallinnointijärjestelmiä. Suosituimmat hallinnointijärjestelmät perustuvat DB-Enginesin (2018a) tekemään selvitykseen, joista viisi suosituinta hallinnointijärjestelmää on otettu mukaan tarkasteluun. Taulukon 3 mukaan suosituimmat DBMS:ät ovat Oracle, MySQL ja Microsoft SQL Server, joista kaikki ovat saaneet yli 1100 pistettä. Jokaisen kärkikolmikossa olevan DBMS:än suosio on laskenut viime vuodesta lähes kymmenen prosenttia. Neljäntenä taulukossa on PostgreSQL, jonka suosio on taas kasvanut huomattavasti vuoden takaisesta. Kaikki neljä suosituinta DBMS:ää ovat tyypiltään SQL- eli relaatiotietokantoja. Taulukossa on myös yksi NoSQL-kantoja edustava DBMS, joka on MongoDB viidennellä sijalla. MongoDB:n suosion muutos viime vuodesta on ollut positiivinen, mutta pieni verrattuna PostgreSQL:n muutokseen.

Taulukko 3: Vuoden 2018 suosituimmat tietokannan hallintajärjestelmät DB-Enginesin (2018a) selvityksen mukaan

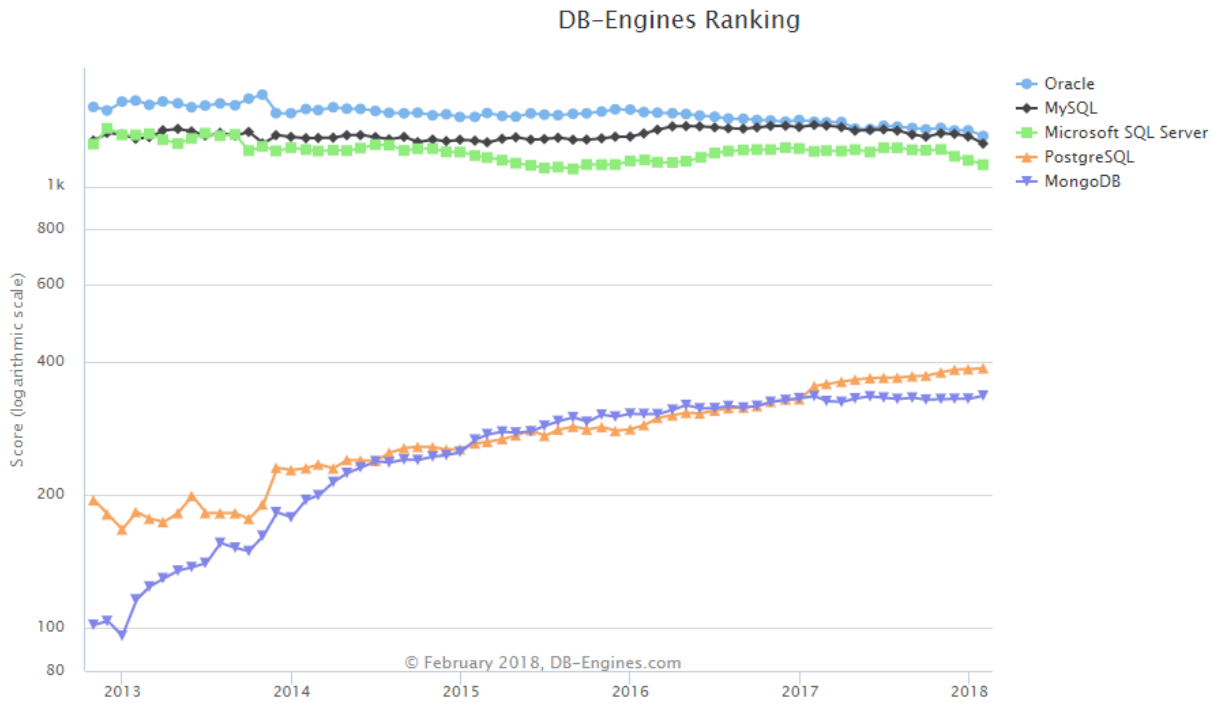
DBMS	Tyyppi	Pisteet	
		Helmikuu 2018	Muutos helmikuusta 2017
Oracle	SQL	1303.28	-100.55
MySQL	SQL	1252.47	-127.83
Microsoft SQL Server	SQL	1122.04	-81.42
PostgreSQL	SQL	388.38	+34.70
MongoDB	NoSQL	336.42	+0.92

DB-Enginesin (2018b) mukaan suosion pisteytys tietokannan hallinnointijärjestelmistä muodostuu kuuden eri kategorian perusteella seuraavasti:

- Google, Bing ja Yandex -hakukoneiden ilmoittamien hakujen tulosten lukumäärät.
- Google Trends -palvelun selvitykset hakujen tiheydestä tietyllä aikavälillä.
- Teknisten keskustelujen lukumäärä ja tiheys Stack Overflow sekä DBA Stack Exchange -sivustoilla.
- Työpaikkojen lukumäärä Indeed ja Simply Hired -sivustoilla.
- Profiilien lukumäärä, joissa on mainittuna kyseenomainen DBMS LinkedIn ja Upwork -sivustoilla.
- Twitter-sivustolla julkaistujen viestien eli ”twiittien” lukumäärä.

Kunkin hallinnointijärjestelmän pisteytys on laskettu kategorioissa ilmoitettujen lähteiden standardisoidun arvon ja keskiarvon perusteella. Selvityksessä ei oteta huomioon tietokannan hallinnointijärjestelmien asennusten lukumääriä eikä käyttöön liittyviä asioita. (DB-Engines 2018b.)

DB-Engines (2018c) on myös tehnyt visuaalisen selvityksen tietokannan hallinnointijärjestelmien suosioista vuosien varrella. Kuva 3 havainnollistaa DB-Enginesin seuraamaa DBMS:ien suosion muutosta. Kuvaan on otettu mukaan taulukon 3 listaamat tietokannan hallintajärjestelmät ja muut vähemmät suositut hallintajärjestelmät on rajattu pois. Kuvassa 3 ilmenevän viivadiagrammin x-akseli määrittää aikavälin ja y-akseli DBMS:än saamat pisteet.



Kuva 3: Kuvakaappaus DB-Enginesin (2018c) viivadiagrammista, joka kuvaa DBMS:ien suosiota ajalta 2013 – 2018

Kuvan 3 mukaan Oraclen ja MySQL:n suosio on ollut tasaisessa laskussa vuodesta 2016 alkaen ja päätyen selvään laskuun vuoden 2017 lopulla. Microsoft SQL Serverillä taas vuosi 2016 oli nousujohteinen, mutta senkin suosio muuttui laskevaksi vuoden 2017 lopulla. Selvää nousua ovat tehneet PostgreSQL ja MongoDB, joista PostgreSQL:n suosio on ollut nousussa vuodesta 2014 alkaen. MongoDB:n suosio alkoi kuvan 3 mukaan vuonna 2013, mutta suosion kasvu on hiipunut tasaisesti vuoden 2015 puolesta välistä alkaen.

Seuraavaksi tässä luvussa on selvitetty taulukossa 3 listattujen tietokannan hallintajärjestelmien taustaa.

Oracle, joka tunnetaan myös nimellä Oracle RDBMS tai Oracle Database, on maksullinen Oracle Corporations -yrityksen omistama relaatiotietokanta hallintajärjestelmä. Oracle on tietokantamarkkinoiden yksi suurimmista ja menestyneimmistä alustoista. Oracle on hal-

linnut markkinoita noin 30 vuoden ajan, ja se on ollut edelläkävijä keskeisimpien tietokanta-arkkitehtuuriratkaisujen osalta. Oraclella on OTN-lisenssi, jonka seurauksena hallintajärjestelmän käytöllä on rajoituksia. Oracle noudattaa tietokantojen transaktioissa ACID-mallia. (Harrison 2016.)

MySQL on nopea ja vankkarakenteinen relaatiotietokanta hallintajärjestelmä. MySQL:ää on kehitetty vuodesta 1979 alkaen ja se on vapautettu julkiseen käyttöön vuonna 1996. MySQL:llä on lisenssi, jonka mukaan alustaa voi käyttää ilmaiseksi niin kauan, kunnes sitä ei käytetä myyntitarkoituksiin. Mikäli MySQL:ää hyödynnetään myyntitarkoituksessa alustaa varten tulee ostaa lisenssi. MySQL:n vahvuuksia ovat: hyvä suorituskyky, halpa hinta, asetusten muokkaamisen helppous, yhteensopivuus useiden käyttöjärjestelmien kanssa, lähdekoodin saatavuus ja muokattavuus, sekä hyvä tuki. (Welling & Thomson 2016.)

Microsoft SQL Server, joka tunnetaan myös nimellä SQL Server, on tietokannan hallintajärjestelmä. Se toimii ainoastaan Windows-käyttöjärjestelmillä, jonka seurauksena kehittäjät, jotka haluavat käyttää esimerkiksi Linux-käyttöjärjestelmää, eivät voi hyödyntää Microsoft SQL Serveriä. Microsoft SQL Serverillä on hyvä suorituskyky, sekä se on luotettava, sillä useat yritykset ja yksittäiset käyttäjät käyttävät sitä datan käsittelyyn. Microsoft SQL Server on myös integroitavissa muiden Microsoftin kehittämien palveluiden, kuten esimerkiksi Azure-pilvipalvelu kanssa. Microsoft SQL Server on myös helppo asentaa ja käynnistää. (Forta 2016, luku 2.)

PostgreSQL on avoimen lähdekoodin relaatiotietokanta hallintajärjestelmä. Se noudattaa kaikkia ACID-periaatteita sekä se tukee suurinta osaa olemassa olevista SQL-datatyypeistä. PostgreSQL tukee myös kuvien, äänten ja videoiden tallentamista tietokantoihin. PostgreSQL:llä on vapaa avoimen lähdekoodin lisenssi, jonka seurauksena PostgreSQL:ää voi muokata haluamallaan tavalla ilman lisäkustannuksia ja pelkoa laillisista seuraamuksista. Alusta on myös hyvin tuettu ja ylläpidetty, jonka ansiosta yrityksen tai yksittäisen kehittäjän ei tarvitse itse huolehtia ylläpitokustannuksista. (PostgreSQL 2018a; PostgreSQL 2018b.)

MongoDB on NoSQL-tyyppinen tietokannan hallintajärjestelmä, josta on tullut ’de facto’ eli ainut oikea NoSQL-tietokantahallintajärjestelmistä. MongoDB tarjoaa joustavan ratkaisun tietokantojen mallin määrityksien suhteen, sillä siihen ei tarvitse etukäteen määritellä tietokantojen rakennetta. Sen seurauksena MongoDB tekee datan tallentamisesta kohtuullisen helppoa myöhempää analysointia varten. Kuitenkin, datan tallentaminen tietokantoihin, joissa ei ole rakennetta, altistaa MongoDB datan käsittelyn ongelmille, kun dataa hae-

taan tietokannoista. Vaarana MongoDB:n käytössä on myös samanaikaisten transaktioiden hallinnointi, sillä MongoDB sallii transaktioiden osittaisen onnistumisen tai epäonnistumisen. Tuolloin tietokantojen eheys on kyseenalaista, ellei kyseenomaisia transaktioita erikseen huomioida kooditasolla. (Giamas 2017.)

4 Työaikakirjanpitosovelluksen toteutus

Tämän opinnäytetyön empiria on jaettu kahteen osaan eri pääluvuiksi. Ensimmäinen empiriaa koskeva osa on käsitelty tässä luvussa ja toinen osa taas luvussa 5. Tämän opinnäytetyön yhdeksi menetelmäksi valittiin produktityyppinen eli toiminnallinen toteutus, jossa ohjelmoitiin työaikakirjanpitosovellus. Toiminnallinen toteutus katsottiin parhaat tavaksi selvittää, miten sovelluksen sovellus kannattaa suunnitella ja ohjelmoida tehokkaasti. Tässä luvussa on selvitetty sovelluksen suunnittelun ja kehityksen vaiheita peilaten tämän opinnäytetyön tietoperustaan. Toteutuksen tavoitteena oli luoda suosituilla avoimen lähdekoodin työkaluilla työaikakirjanpitosovellus hallitusti ja kohtuullisessa ajassa.

Selvityksestä hyötyvät erityisesti ohjelmistoyritykset ja kehittäjät, sillä toiminnallisesta toteutuksesta ilmenee sovelluksen suunnittelun ja kehityksen vaiheita, joiden pohjalta muun muassa yritykset voivat kehittää omaa sovelluksen kehityskaartaan. Selvityksestä hyötyvät myös ohjelmoijat, jotka haluavat tietää mitkä ovat suosituimpia backend-ohjelmointityökaluja sekä kuinka niitä voi soveltaa käytännössä.

Seuraavaksi on selvitetty projektin taustatietoa, josta selviää sovelluksen kehittäjät ja aikataululliset asiat. Sen jälkeen on selvitetty, kuinka produktina syntynyt sovellus on suunniteltu, mitkä ovat sen toiminnallisuudet ja mitä ohjelmointityökaluja on käytetty. Työkalujen jälkeen on perehdytty itse sovelluksen ohjelmointiin, jossa on selvitetty kuvakaappauksien ja liitteiden avulla ohjelmointiprosessin vaiheita. Pääluvussa 5 on lopulta mitattu, kuinka hyvin produktina syntynyt sovellus on onnistunut.

4.1 Projektin taustatietoa

Sovelluksen toteutus tapahtui yhdessä toisen Haaga-Helian opiskelijan Teemu Salmisen kanssa. Molemmat kehittäjät tekivät sovelluksen palvelin- ja käyttöliittymäohjelmointia, mutta opinnäytetyön tekijä keskittyi enemmän sovelluksen backend-puoleen ja Salminen vuorostaan frontend-puoleen. Kehittäjät nimesivät sovelluksen Flexeriksi, joksi sovellusta on tässä tutkimuksessa tästä eteenpäin myös kutsuttu selkeyden vuoksi.

Sovelluksen suunnittelussa ja toteutuksessa hyödynnetiin Lean-tuotekehitysmallia ja ketterän kehityksen periaatteita, joita on selvitetty tarkemmin tämän opinnäytetyön tietoperustassa luvussa 3.1. Resurssien puutteen takia, sovellus on suunniteltu ja kehitetty yhdelle käyttäjälle, jotta sovellus oli mahdollista toteuttaa kohtuullisessa ajassa kahdella kehittäjällä. Kehittäjät kokivat, että kohtuullinen kehitysaika projektille oli kaksi kuukautta, jossa viikkoa kohden yhdellä kehittäjällä oli kehitysaikaa noin 15 tuntia, eli kaksi kokonais-

ta työpäivää. Kaiken kaikkiaan sovelluksen kehitykselle oli varattuna aikaa noin 120 tuntia kehittäjää kohden, eli yhteensä 240 kehitystuntia. Sovelluksen kehitys alkoi marraskuun alussa vuonna 2017 ja päättyi tammikuun lopulla vuonna 2018. Vuoden 2017 joulukuu suunniteltiin taukoajaksi.

4.2 Kehityksen suunnittelu

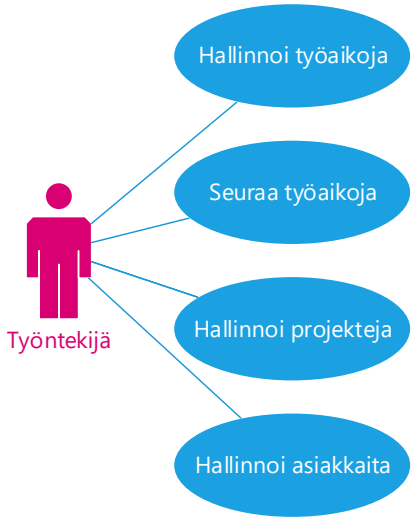
Projektin kehitys alkoi suunnittelulla, johon hyödynnettiin Lean-tuotekehitysmallia ja ketterän kehityksen menetelmiä. Lean-menetelmän mukaisesti turhia vaiheita pyrittiin minimoimaan selvittämällä aluksi projektin riskit, molempien kehittäjien osaaminen ja kiinnostusten kohteet. Riskikartoituksessa (ks. liite 1) taulukoitiin kaikki mahdolliset asiat, jotka saattoivat vaikuttaa projektin etenemiseen. Jokaista taulukoitua asiaa kohden selvitettiin, kuinka vakava kyseenomainen riski asteikolla pieni, keskisuuri ja suuri sattuessaan olisi ollut, miten se olisi vaikuttanut projektiin sekä kuinka siihen varauduttiin. Esimerkiksi, yksi riski oli sairastuminen, jolle annettiin riskiasteikoksi keskisuuri, koska se olisi hidastanut projektin etenemistä huomattavasti. Sairastumisen ehkäisemiseksi kehittäjien piti pitää huolta terveydestään syömällä terveellisesti, nukkumalla hyvin ja urheilemalla.

Riskien selvittämisen jälkeen kehittäjät arvioivat osaamisensa, jonka pohjalta luotiin projektin alustava aikataulu. Aikatauluksi määritettiin kaksi kuukautta eli noin kahdeksan viikkoa. Kaksi viikkoa säästettiin sovelluksen suunnitteluun ja käytettävien työkalujen valintaan. Yksi viikko kehitysympäristön luomiseen, neljä viikkoa sovelluksen ohjelmointiin ja viimeinen viikko ohjelmointivirheiden korjailuun. Aikataulussa pyrittiin rytmittään Lean-mallin mukaisesti projektin osioiden valmistuminen. Lisäksi, suunnittelussa huomioitiin, että aikataulu ei aiheuttanut ylimääräisiä paineita kehittäjille, joka myös noudattaa ketterän kehityksen manifestin mukaisesti periaatetta liittyen yksilöiden ja vuorovaikuttamisen huomiointiin.

4.3 Käyttötapaukset

Tässä alaluvussa on kuvattu sovelluksen käyttötapaukset, joiden pohjalta kehittäjät määrittivät sovelluksen päätoiminnot ja käyttäjän. Käyttötapaukset määriteltiin tämän opinnäytetyön luvun 2 pohjalta. Luvussa 2 on selvitetty, että työaikakirjanpitoon liittyy useimmiten asiakas, jolle kehitetään jotakin projektiluonteisesti. Sen seurauksena, Flexeriin määriteltiin, että sovelluksella voi hallinnoida asiakkaita sekä projekteja. Luvussa 2 on myös selvitetty, että projektin etenemistä seurataan työntekijöiden tekemän työmäärän ja tehtävien mukaan, jonka takia sovellukseen katsottiin tärkeäksi myös työaikojen hallinnointi sekä erillinen seuranta.

Lean-malli kannusti pitämään kehitettävän sovelluksen sopivan kokoisena ja siten hallittavana, jonka takia muut mahdolliset käyttötapaukset on rajattu pois. Kuvassa 4 on lopuksi selvennetty käyttäjän ja valittujen käyttötapausten väliset suhteet. Kuvan 4 mukaan, Flexerissä työntekijä, eli sovelluksen käyttäjä, voi hallinnoida ja seurata työaikoja, sekä hallinnoida projekteja ja asiakkaita. Hallinnointi käyttötapauksen kohdalla tarkoittaa katselua, lisäämistä, muokkaamista ja poistamista.



Kuva 4: Flexerin käyttötapauskaavio

Taulukossa 4 on eriteltynä kuvan 4 käyttötapauskaavion pohjalta määritetyt toiminnallisuudet, jotka kehitettiin Flexeriin. Jokainen hallinnointiin liittyvä käyttötapaus koostui näkemisestä, lisäämisetä, muokkaamisesta ja poistamisesta.

Taulukko 4: Sovelluksen toiminnallisuudet

Toiminnallisuus	Kuvaus
Työaikamerkintöjen näkeminen	Työntekijä näkee järjestelmään kirjatun työaikamerkinnän.
Työaikamerkinnän lisääminen	Työntekijä lisää järjestelmään uuden työaikamerkinnän.
Työaikamerkinnän muokkaaminen	Työntekijä muokkaa olemassa olevaa työaikamerkintää.
Työaikamerkinnän poistaminen	Työntekijä poistaa järjestelmästä työaikamerkinnän.
Työaikamerkintöjen seuranta	Työntekijä näkee kuukausikohtaisesti työaikamerkinnät.
Projektien näkeminen	Työntekijä näkee projektit, jotka on lisätty järjestelmään.
Projektin lisääminen	Työntekijä lisää projektin järjestelmään.
Projektin muokkaaminen	Työntekijä muokkaa olemassa olevaa projektia.
Projektin poistaminen	Työntekijä poistaa järjestelmästä projektin.
Asiakkaiden näkeminen	Työntekijä näkee järjestelmään lisätyt asiakkaat.
Asiakkaan lisääminen	Työntekijä lisää järjestelmään uuden asiakkaan.
Asiakkaan muokkaaminen	Työntekijä muokkaa olemassa olevan asiakkaan tietoja.
Asiakkaan poistaminen	Työntekijä poistaa järjestelmästä asiakkaan.

4.4 Ohjelmointityökalut

Tässä alaluvussa on selvitetty, mitä ohjelmointityökaluja sovelluksen toteutukseen on valittu. Tutkijalla oli jo ennestään kokemusta ohjelmoinnista työelämän parista. Sen seurauksena laite, koodieditori, käyttöjärjestelmä, terminaali- ja versionhallintatyökalut on valittu aikaisempien kokemuksiensa pohjalta, jotta itse ohjelmointiin kuluvaa aikaa on ollut käytössä enemmän. Web-viitekehys ja tietokantahallintajärjestelmä on valittu tämän tutkimuksen tietoperustan pohjalta.

Web-viitekehyksen valinnassa ohjelmointikielen suosio on koettu tärkeäksi. Tämän tutkimuksen tietoperustasta ilmenee, että suosittujen ohjelmointikielten osaajille on tarjolla työpaikkoja, sekä kielelle löytyy sovellusta kehittäessä ja ylläpitäessä tukiverkosto. Sen seurauksena Web-viitekehykseksi on valittu Pythonilla kirjoitettu Django. Djangon valintaan vaikutti IEEE Spectrumin (2017a) selvitys, jonka mukaan Python on suosituin ohjelmointikieli.

Muita syitä Djangon valinnalle olivat sen suosio, suosion tasaisuus vuosien ajan, dokumentaation paljous ja kattavuus, sekä integraatiotuki usealle eri DBMS:lle. Lisäsyy Djangon valinnalle oli myös se, että Python muistuttaa syntaksiltaan JavaScriptiä, jolla työaikakirjanpitosovelluksen frontend-osa on pääosin toteutettu. Tällöin sovellusta on jatkossa helpompi hallita ja jos sovellusta mahdollisesti ohjelmoi joskus uusi kehittäjä, hänen on helppo oppia sovelluksen arkkitehtuuri.

Tietokannanhallintajärjestelmäksi on valittu PostgreSQL. Syitä PostgreSQL:n valintaa olivat lisenssi, tietokantojen koko, sekä yhteensopivuus Djangon ja käyttöjärjestelmän integraation kanssa. Lisäksi, PostgreSQL:n lisenssin ansiosta sovelluksen voi tuotteistaa tulevaisuudessa ilman lisäkuluja. Muut tietoperustassa selvitetyt SQL-tyyppiset DBMS-ratkaisut vaativat lisäkuluja sovelluksen tuotteistamistarkoituksiin siirtyessä. Tutkijan resurssien vähyyden seurauksena, tutkija ei voinut valita myöskään MongoDB:tä, koska NoSQL-tyyppinen MongoDB vaatii enemmän tallennustilaa ja tehoja, kuin SQL-tyyppinen DBMS.

Taulukossa 5 on vielä lueteltuna työkalut, jotka on valittu sovelluksen toteutukseen. Ensimmäisessä taulukon sarakkeessa on työkalun rooli ja toisessa sarakkeessa roolia vastaavan työkalun nimi ja sen versio.

Taulukko 5: Sovelluksen kehityksessä käytetyt työkalut

Työkalu rooli	Työkalun nimi
Laite	Macbook Pro 13”
Käyttöjärjestelmä	Mac OSX El Capitan v10.11.6
Terminaali	iTerm2 v3.1.5
Koodieditori	Visual Studio Code v1.20.0
Web-viitekehys	Django v1.11.7
Syntaksi	Python v3.6
DBMS	PostgreSQL v9.5
Versionhallinta	GitHub

4.5 Sovelluksen suunnittelu ja ohjelmointi

Flexer-sovelluksen kehitys alkoi näkymien visuaalisella mallintamisella, jonka pohjalta sovelluksen tietokannan arkkitehtuuri on suunniteltu. Esimerkkinä suunnitteluprosessista on liite 2, joka on visualisoitu malli sovelluksen työaikamerkintänäkymästä. Mallin tarkoitus oli selvittää Flexeriin ohjelmoidut näkymät, niiden rakenteen ja syötekentät. Mallin avulla vältettiin Lean-mallin mukaisesti turhia kehitysvaiheita. Lisäksi, käyttötapaukset pysyivät selkeinä, jolloin myös sovelluksen päätarkoitus ja arvot olivat siten selvät.

Liitteessä 2 esiintyvän mallin pohjalta on suunniteltu Flexerin työaikamerkintään liittyvät tietokantarakenteet. Tietokantarakenteita on selvennetty kuvassa 5, joka on relaatiokaavio sovelluksen taulukohtaisista kentistä ja taulujen välisistä suhteista.

Kuva 5: Flexer tietokantataulujen väliset suhteet ja niiden kentät

Flexer-sovelluksen tietokannat koostuvat kolmesta eri taulusta kuvan 5 mukaisesti. Jokaisen tietokantataulun pääavain (engl. primary key) on id. Projekti-taulussa viiteavaimena on asiakas-taulun id ja tehtävä-taulussa viiteavaimena on projektin id. Tarkemmat taulujen kenttäkohtaiset määritykset on luettavissa kuvasta 7, joka on tässä luvussa hieman alempana.

Seuraavaksi on tarkasteltu, kuinka sovelluksen kehitysympäristön työkalut on asetettu toimimaan keskenään. Tarkastelua on havainnollistettu ottamalla kuvakaappauksia Visual Studio Code –koodieditorista, jolla Flexer kirjoitettiin. Ensimmäiseksi, kehitysvaiheessa asetettiin Django-viitekehyksen ja PostgreSLQ:n välinen yhteys. Yhteyden asetukset määritettiin settings.py-tiedostossa DATABASES objektin sisään kuvan 6 mukaisesti.

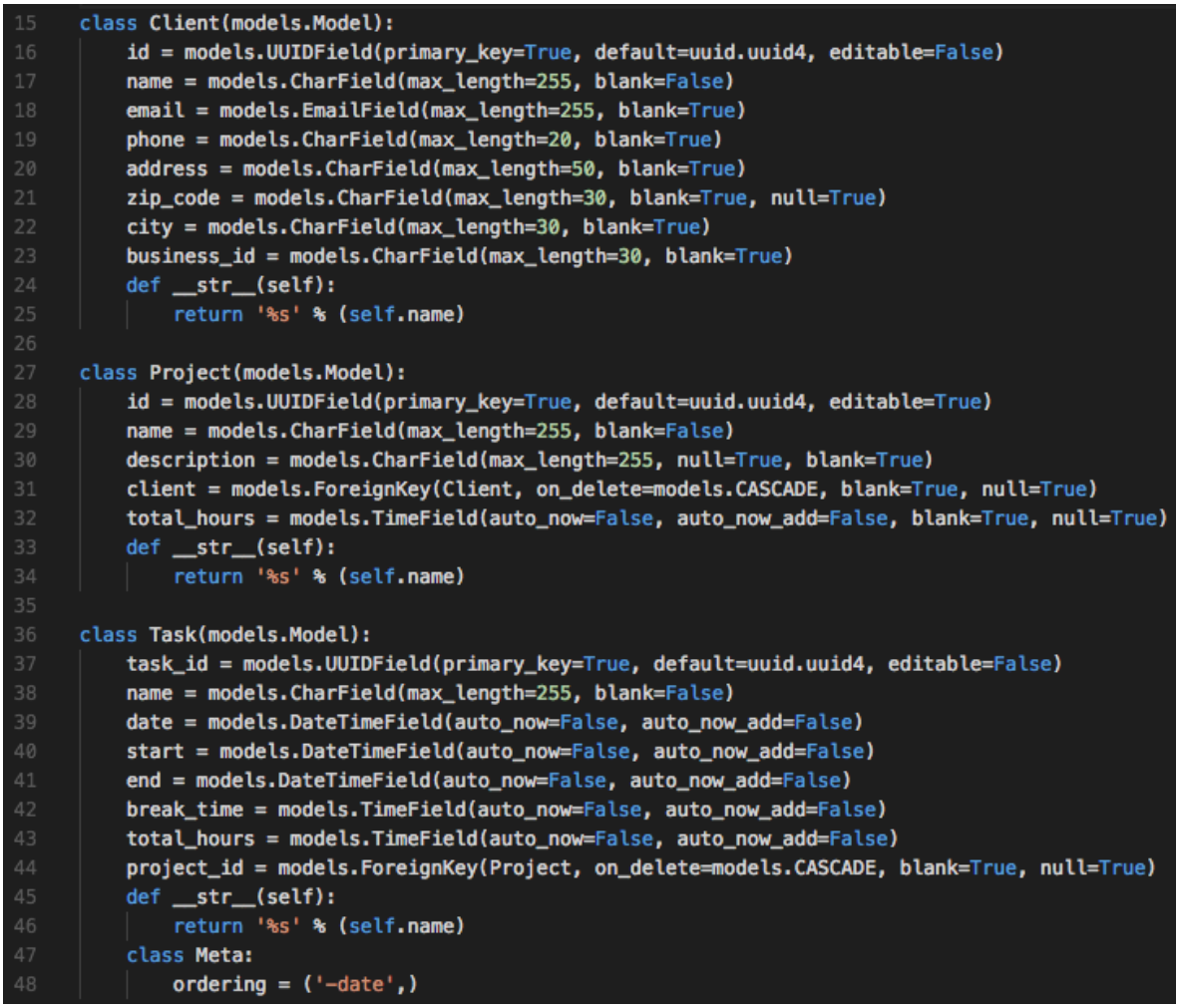
```
80  DATABASES = {
81      'default': {
82          'ENGINE': 'django.db.backends.postgresql_psycopg2',
83          'NAME': 'flexer',
84          'USER': 'flexeruser',
85          'PASSWORD': 'flexerpassu',
86          'HOST': '127.0.0.1',
87          'PORT': '5432',
88      },
89  }
```

Kuva 6: Tietokannan asetusten määrittely Django-viitekehyksen settings.py-tiedostossa

Kuvan 6 rivillä 82 on määritetty, että Django käyttää oletuksena tietokannanhallintajärjestelmänään PostgreSQL:ää. Tietokannalle annettiin nimeksi flexer, tietokannan käyttäjäksi flexeruser ja käyttäjän salasanaksi flexerpassu. Djangon ja PostgreSQL:n välinen keskustelu määritettiin käytäväksi portin 5432 kautta osoitteessa 127.0.0.1, joka tarkoittaa localhostia eli paikallista isäntää.

Seuraavaksi on selvitetty, kuinka lopulta sovelluksen toiminnallisuuksien ohjelmointi tapahtui Djangon käyttämän MVT-sovellusarkkitehtuurimallin mukaisesti. MVT-sovellusarkkitehtuurista on kerrottu tarkemmin tämän opinnäytetyön tietoperustassa luvussa 3.2. Djangon tietokantatauluja vastaavat mallit kirjoitettiin kuvan 5 pohjalta yksikerrallaan models.py-tiedostoon (ks. kuva 7).

Tietokantataulujen kenttien tietotyypit määritettiin Djangon tarjoaman models-objektin avulla. Esimerkiksi työaikamerkinnän päiväys-kenttä on määritetty kuvan 7 rivillä 39 päiväystyyppiseksi, käyttämällä models-objektin DateTimeField-attribuuttia. Tarkemmat kenttien määritykset on merkattu avain ja arvo pareilla sulkeiden sisään. Taulujen pääavaimet on merkattu kuvan mukaisesti jokaisen taulun mallin ensimmäiselle sisennetylle riville. Muutamien taulujen väliset riippuvuudet on merkattu viiteavaimilla, josta esimerkkinä on kuvan 7 rivillä 44 oleva työaikamerkintä-mallin project_id-kenttä, joka on viiteavain projekti-taulusta.



```python
class Client(models.Model):
    id = models.UUIDField(primary_key=True, default=uuid.uuid4, editable=False)
    name = models.CharField(max_length=255, blank=False)
    email = models.EmailField(max_length=255, blank=True)
    phone = models.CharField(max_length=20, blank=True)
    address = models.CharField(max_length=50, blank=True)
    zip_code = models.CharField(max_length=30, blank=True, null=True)
    city = models.CharField(max_length=30, blank=True)
    business_id = models.CharField(max_length=30, blank=True)
    def __str__(self):
        return '%s' % (self.name)

class Project(models.Model):
    id = models.UUIDField(primary_key=True, default=uuid.uuid4, editable=True)
    name = models.CharField(max_length=255, blank=False)
    description = models.CharField(max_length=255, null=True, blank=True)
    client = models.ForeignKey(Client, on_delete=models.CASCADE, blank=True, null=True)
    total_hours = models.TimeField(auto_now=False, auto_now_add=False, blank=True, null=True)
    def __str__(self):
        return '%s' % (self.name)

class Task(models.Model):
    task_id = models.UUIDField(primary_key=True, default=uuid.uuid4, editable=False)
    name = models.CharField(max_length=255, blank=False)
    date = models.DateTimeField(auto_now=False, auto_now_add=False)
    start = models.DateTimeField(auto_now=False, auto_now_add=False)
    end = models.DateTimeField(auto_now=False, auto_now_add=False)
    break_time = models.TimeField(auto_now=False, auto_now_add=False)
    total_hours = models.TimeField(auto_now=False, auto_now_add=False)
    project_id = models.ForeignKey(Project, on_delete=models.CASCADE, blank=True, null=True)
    def __str__(self):
        return '%s' % (self.name)
    class Meta:
        ordering = ('-date',)
```

Kuva 7: Flexerin tietokantataulujen mallit Django-viitekehyksessä

Tietokannan arkkitehtuurillisten määritysten jälkeen on selvitetty, kuinka Flexerin HTTP-kutsut on kirjoitettu Djangon ja selaimen välillä. HTTP-kutsut on käsitelty Djangon views.py, serializers.py ja urls.py –tiedostoissa. Views.py-tiedostossa on käsitelty kutsujen pyyntöjen tyypit (engl. request method) ja vastausten (engl. response) lähetys. Serializers.py-tiedostossa on määritetty kenttien nimet, joiden avulla Django on muotoillut tietokantaan tallennetun ja tietokannoista haetun datan. Urls.py-tiedostossa on hoidettu HTTP-kutsujen reititys views.py-tiedoston funktioihin.

Kuvat 8 ja 9 selvittävät, kuinka HTTP-pyynnöt on käsitelty Flexer-sovelluksen views.py-tiedostossa. Kuvan 8 rivillä 101 on aluksi määritetty funktio, joka käsittelee työaikamerkintöjen GET ja POST -kutsuja, eli haku ja lisäys -kutsuja. GET-kutsun kohdalla riveillä 103 – 106 on ensiksi haettu kaikki työaikamerkinnät tietokantojen vastaavasta työaikamerkintätaulusta, jonka jälkeen tietokannasta haetut merkinnät on muotoiltu JSON-formaattiin ja lopulta lähetetty vastauksen mukana frontendin käsiteltäväksi.

```
 99  # TASKS
100  @csrf_exempt
101  def fetch_tasks(request):
102      # Fetch all tasks
103      if request.method == 'GET':
104          tasks = Task.objects.all()
105          serializer = TaskSerializer(tasks, many=True)
106          return JsonResponse(serializer.data, safe=False)
107
108      # Create new task
109      elif request.method == 'POST':
110          data = JSONParser().parse(request)
111          serializer = TaskSerializer(data=data)
112          if serializer.is_valid():
113              serializer.save()
114              return JsonResponse(serializer.data, safe=False)
115          return JsonResponse(serializer.errors, status=400)
```

Kuva 8: Työaikamerkintöjen GET ja POST -kutsut

Kuvan 8 riveillä 109 – 115, jotka selvittävät työaikamerkintään liittyvän POST-kutsun, on ensin käsitelty kutsun mukana tullut data. Sen jälkeen on tarkastettu, onko kutsun mukana saapunut data ollut oikeassa muodossa. Mikäli data on ollut oikeassa muodossa, tietokantoihin on tallennettu uusi työaikamerkintä frontendistä tulleen datan mukaan, jonka jälkeen frontendiin on lähetetty vastauksena tietokantoihin lisätty tietorivi JSON-muodossa. Jos kutsun mukana tullut data on ollut väärässä muodossa, frontendiin on lähetetty virheilmoitus ja vastauksen tila (engl. status), joka kuvaa epäonnistunutta kutsua.

Seuraavaksi on havainnollistettu, kuinka työaikamerkintöjen PUT ja DELETE, eli muokkaus ja poisto -kutsut on käsitelty yksilöivän tietokannan tietorivin pääavaimen avulla (ks. kuva 9).

```
117  @csrf_exempt
118  def manage_task(request, pk):
119      # Check if task exists
120      try:
121          task = Task.objects.get(pk=pk)
122      except Task.DoesNotExist:
123          return HttpResponse(status=404)
124
125      if request.method == 'DELETE':
126          task.delete()
127          return JsonResponse(pk, safe=False)
128
129      elif request.method == 'PUT':
130          data = JSONParser().parse(request)
131          serializer = TaskSerializer(task, data=data)
132          if serializer.is_valid():
133              serializer.save()
134              return JsonResponse(serializer.data)
135          return JsonResponse(serializer.errors, status=400)
```

Kuva 9: Työaikamerkintään liittyvät DELETE- ja PUT-kutsut

Kuvan 9 rivillä 118 on määritetty funktion pääavain-argumentti, joka on tullut HTTP-kutsun mukana. Pääavaimen avulla on ensiksi etsitty vastaava tietorivi tietokannasta. Mikäli tietorivi on löytynyt tietokannasta, on tietoriviä käsitelty HTTP-kutsun tyypin mukaan, muulloin vastauksena frontendiin on lähetetty virheilmoitus. Jos kutsu on ollut PUT- eli muokkaustyyppinen, tietokannan pääavainta vastaava työaikamerkintätietorivi on korvattu HTTP-kutsun mukana tulleella datalla. Jos taas kutsu on ollut DELETE- eli poisto-tyyppinen, työaikamerkintää vastaava tietorivi on poistettu tietokannasta pääavaimen avulla ja vastauksena frontendiin on lähetetty poistetun tietorivin pääavain.

HTTP-kutsujen käsittelyssä on käytetty muotoilijoita (engl. serializer), jotka on määritetty kuvan 10 mukaisesti. Kuvassa 10 on kolme muotoilijaa asiakas-, projekti- ja työaikamerkintä-tietokantataulujen malleille. Muotoilijoihin on määritetty taulukohtaisesti avainten nimet, joiden mukaan HTTP-kutsujen mukana saapuva ja lähetettävä data on käsitelty. Muotoilijoihin on siis määritetty, missä muodossa frontendistä saapuneen datan täytyy olla, jotta Django pystyy tallentamaan datan tietokantoihin.

```
9   class ClientSerializer(serializers.ModelSerializer):
10      class Meta:
11          model = Client
12          fields = ('id', 'name', 'email', 'phone', 'address', 'zip_code', 'city', 'business_id')
13
14  class ProjectSerializer(serializers.ModelSerializer):
15      class Meta:
16          model = Project
17          fields = ('id', 'name', 'description', 'client', 'total_hours' )
18
19  class TaskSerializer(serializers.ModelSerializer):
20      class Meta:
21          model = Task
22          fields = ('task_id', 'name', 'date', 'start', 'end', 'break_time', 'total_hours', 'project_id')
```

Kuva 10: Tietokantatauluja vastaavat datan muotoilijat, joiden avulla Flexer-sovelluksessa on käsitelty HTTP-kutsujen dataa

Lopuksi, kuvassa 11 on listattuna URL-osoitteiden päätteet, joiden avulla HTTP-kutsut on ohjattu oikeisiin paikkoihin.

```
5   urlpatterns = [
6       url(r'^clients/$', views.fetch_clients),
7       url(r'^clients/(?P<pk>[^/]+)/$', views.manage_client),
8       url(r'^projects/$', views.fetch_projects),
9       url(r'^projects/(?P<pk>[^/]+)/$', views.manage_project),
10      url(r'^tasks/$', views.fetch_tasks),
11      url(r'^tasks/(?P<pk>[^/]+)/$', views.manage_task),
12      url(r'^overview/$', views.tasks_overview),
13  ]
```

Kuva 11: URL-osoitteiden yhdistäminen HTTP-kutsuihin

Esimerkiksi kuvan 11 rivillä 10 on määritetty, että frontendistä tulevat kutsut jotka päättyvät ”tasks/”-muotoisesti on ohjattu views.py-tiedostossa määritettyyn fetch_tasks-funktioon (ks. kuva 8). Kuvassa 11 rivillä 7, 9 ja 11 on myös selvitetty, kuinka URL-osoitteista on eroteltu tietokantakyselyjä varten käsiteltävän tietorivin pääavain säännöllisellä lausekkeella (engl. regular expression).

Liitteessä 3 on vielä selvitetty Djangon models.py-tiedoston pohjalta luoma Flexer-sovelluksen käyttämä tietokannan rakenne. Liitteessä on myös kuvattu asiakas-, projekti- ja työaikamerkintä-taulun rakenne tietokantatasolla. Liitteessä esiintyvien taulujen tietorivien arvot on luotu ainoastaan havainnollistamisen vuoksi, ja arvoilla ei ollut tarkoitus vastata olemassa olevien henkilöiden tai yritysten tietoja.

5 Sovellusten mittaaminen

Tässä luvussa on selvitetty, kuinka produktina syntyneen työaikakirjanpitosovelluksen (Flexerin) sekä vastaavanlaisen tuotteistetun sovelluksen toiminnallisuuksien suorittamiseen kului aikaa. Flexerin kanssa mitattavaksi tuotteistetuksi sovellukseksi valittiin Toggl, sillä se on ollut tämän opinnäytetyön tekijän työpaikalla käytössä ja todettu siellä toimivaksi ratkaisuksi. Toiminnallisuuksien suorittamiseen kuluvan ajan mittaamisen tarkoituksena oli selvittää, kuinka onnistuneesti opinnäytetyön produkti on kaiken kaikkiaan onnistunut.

Seuraavissa alaluvuissa on selvitetty tutkimuksen taustatietoja, tutkimusmateriaalien analysointi menetelmiä sekä lopulta mittaustulokset. Mittaustulokset on aluksi avattu kvantitatiivisin menetelmin, jonka jälkeen niitä on tutkittu vielä tarkemmin kvalitatiivisin tavoin. Flexerin kehityksen onnistumista ja mittaustuloksia on pohdittu enemmän tämän opinnäytetyön kuudennessa luvussa.

5.1 Tutkimuksen taustatiedot

Sovellusten mittauksesta pyrittiin tekemään mahdollisimman tasavertainen valitsemalla tutkimukseen henkilöitä, joilla ei ollut aikaisempaa käyttökokemusta Flexer tai Toggl -sovelluksista. Aikaisempi kokemus jommankumman sovelluksen käytöstä olisi muuten voinut vääristää mittaustuloksia. Tutkimukseen osallistui viisi henkilöä (n = 5), joista jokaisella oli kokemusta toimistotyössä käytettävistä sovelluksista. Kokemus toimistotyössä käytettävistä sovelluksista katsottiin tarpeelliseksi, jotta osallistujat pystyivät systemaattisesti suoriutumaan heille annetuista tehtävistä. Mittaustilaisuudet järjestettiin maaliskuussa vuonna 2018.

Jokainen mittaustilaisuuteen osallistunut henkilö sai aluksi lomakkeen tutkimukseen suostumisesta (ks. liite 4) viimeistään neljä päivää ennen tutkimustilaisuutta. Osallistujat saivat lomakkeet sekä aluksi sähköpostitse ja myöhemmin vielä erikseen paikan päällä paperisessa muodossa mittaustilaisuuden yhteydessä. Kaikki osallistujat antoivat suostumuksensa tutkimukseen palauttamalla suostumuslomakkeen allekirjoitettuna. Jokaiselle osallistujalle selvitettiin sekä kirjallisesti että suullisesti seuraavat asiat:

- Tutkimuksen tarkoitus ja hyödyt, sekä kenelle tutkimus on tehty.
- Tutkimuksessa käytettävät työkalut, tilanteen nauhoitus ja äänitys.
- Tutkimuksesta syntyneen aineiston käyttö, säilytys sekä hävitys.
- Tutkimustulosten raportointi.
- Mahdollisuus keskeyttää tutkimus minä hetkenä tahansa.
- Paikka ja päivämäärä.

Mittaustilaisuudet pyrittiin järjestämään niin, että osallistujat saivat keskittyä tilaisuuteen rauhassa ilman ulkopuolisia häiriöitä. Mittaustilaisuudet pidettiin osallistujille tutussa ym-

päristössä, kuten esimerkiksi osallistujan kodissa. Tutun ympäristön avulla pyrittiin vähentämään tutkimustilaisuudesta mahdollisesti aiheutuvaa jännitystä, joka muutoin olisi saattanut vaikuttaa tutkimustuloksiin. Kaikki osallistujat käyttivät samoja työkaluja, jotka olivat: MacBook Pro 13” -kannettava tietokone Mac OS X El Capitan -käyttöjärjestelmällä, Logitech MX518 -hiiri, hiirimatto sekä Google Chrome -selain versiolla 64.0.3282.186. Nauhoitukseen käytettiin QuickTime Player -sovellusta, jonka versio oli 10.4. Nauhoitteet koostuivat kannettavan tietokoneen näytön kuvasta sekä tilaisuudessa kuuluvista äänistä.

Tutkimuksessa käytettävät Flexer ja Toggl -työaikakirjanpitosovellukset asetettiin samanlaiseen tilaan jokaiselle osallistujalle, jotta sovellusten käyttöliittymät näyttivät ja toimivat samalla tavoin. Toggl-sovellusta varten luotiin tutkimusta varten käyttäjätili ja työaikamerkinnät asetettiin manuaalisesti syötettäväksi. Molempiin sovelluksiin asetettiin työaikamerkintöjen hallinnointisivu aloitusnäkymäksi. Tutkimustilanteen havainnollistamiseksi opinnäytetyön liitteisiin on kerätty kuvakaappauksia molempien sovelluksien käyttöliittymistä, joita osallistujat käyttivät. Liitteessä 6 on kuvakaappauksia Flexerin käyttöliittymästä ja liitteessä 7 taas Togglen käyttöliittymästä.

Jokaiselle osallistujalle annettiin liitteen 5 mukainen kaksisivuinen tehtäväseloste, jossa on selvitetty mitä osallistujan piti tehdä molempien sovelluksien käyttöliittymien kautta. Kaikkien osallistujien piti suorittaa samat yhdeksän tehtävää selosteen mukaisesti ensiksi Flexerissä ja sitten Togglessa. Jokainen osallistuja sai tehtäväkseen lisätä yhden asiakkaan ja projektin sekä kaksi työaikamerkintää. Osallistujien tuli myös muokata yhden asiakkaan ja projektin nimeä sekä yhden työaikamerkinnän alkamisaikaa. Lopuksi osallistujien tuli poistaa molemmat työaikamerkinnät, sekä projekti ja asiakas. Mittaustilaisuus katsottiin päättyneeksi, kun osallistuja oli suorittanut viimeisen tehtävän, jossa piti poistaa asiakas.

5.2 Tutkimuksen analysointimenetelmät

Tutkimuksen analysoinnissa on käytetty triangulaatio-ratkaisumenetelmää, joka on monimenetelmäinen aineiston keräys- ja käsittelytapa. Aineistoa voidaan käsitellä esimerkiksi kvantitatiivisin ja kvalitatiivisin menetelmin, eli määrällisesti ja laadullisesti. Aineiston käsittely useammalla eri menetelmällä ehkäisee virhetilanteilta, joita saattaisi syntyä yhtä menetelmää käyttäessä. Lisäksi, useamman menetelmän hyödyntäminen ongelman tutkinnassa ja tulosten käsittelyssä lisää tulosten kattavuutta ja luotettavuutta. (Kananen 2014, 120-121.)

Kvantitatiivinen eli määrällinen menetelmä soveltuu Yhteiskuntatieteellisen tietoarkiston (2015) mukaan silloin, kun tuloksia halutaan havainnollistaa matriiseina. Matriisit koostuvat havainnoista, jotka on eritelty tietoriveiksi. Tietorivit taas koostuvat sarakkeista, joissa on yksi muuttuja. (Yhteiskuntatieteellinen tietoarkisto 2015.) Kvantitatiivinen menetelmä katsottiin soveltuvan tutkimuksen aineiston analysointiin, sillä tulokset olivat selkeästi eroteltavissa ja kirjattavissa matriiseiksi eli taulukkomuotoon. Lisäksi, tutkimuksessa haluttiin selvittää toiminnallisuuksiin kuluvaa aikaa, joka sopi yksiselitteiseksi muuttujaksi matriisin tietorivien sarakkeisiin.

Kvantitatiivisen menetelmän lisäksi aineiston havainnollistamisessa käytettiin referoivaa litterointia, joka on Yhteiskuntatieteellisen tietoarkiston (2017b) mukaan yksi laadullisen tutkimuksen aineiston litterointitavoista. Referoivaa litterointia käytetään, kun nauhoitteista halutaan avata esimerkiksi puheen osia suurpiirteisesti esimerkiksi ranskalaisin viivoin (Yhteiskuntatieteellinen tietoarkisto 2017). Referoivan litteroinnin huomattiin tuovan lisäarvoa tutkimuksen poikkeustilanteiden ja muiden tuloksiin mahdollisesti vaikuttavien tekijöiden havainnollistamisessa.

Mittaustilaisuuksien avaaminen luettaviksi tuloksiksi alkoi havainnoimalla ja litteroimalla nauhoitteita osallistuja kerrallaan. Osallistujien identiteettien sijaan materiaalien tallentamisessa ja käsittelyssä käytettiin osallistujakohtaisia numeroita. Jokaiselle osallistujalle luotiin Excel-sovellusta käyttäen oma xlsx-tiedosto, johon taulukoitiin tehtäväkohtaisesti osallistujien suorittamat toiminnallisuudet Flexerissä ja Togglessa. Jokainen taulukon rivi koostui toiminnallisuuden nimestä, suorituksen aloitus- ja päätösajasta sekä suorituksen kestosta. Kaikki osallistuja suoriutuivat mittaustilaisuudesta alle tunnissa, jonka seurauksena kaikki ajat merkattiin minuutteina ja sekunteina muodossa ”mm:ss”. Lisäksi, suoritusten kohdalle listattiin referoivaa litterointi -menetelmää käyttäen erillisiä nauhoitteista ilmeneviä huomioita, jotka saattoivat vaikuttaa mittaustuloksiin. Kuvassa 12 on havainnollistettu, kuinka yhden osallistujan nauhoitteesta on litteroitu Flexer-sovelluksen mittaustulokset.

	Flexer			
tehtävä	aloitus (mm:ss)	lopetus (mm:ss)	kesto (mm:ss)	muistiinpanot
asiakkaan lisäys	00:09	00:21	00:12	tottumista näppäimistöön
projektin lisäys	00:28	01:02	00:34	syötti tiedon aluksi kuvaus kenttään
1. työaikamerkinnän lisäys	01:07	01:52	00:45	tarkisti toiselta sivulta jatkuiko tehtävä
2. työaikamerkinnän lisäys	01:53	02:23	00:30	
työaikamerkinnän muokkaus	02:27	02:39	00:12	
projektin muokkaus	02:42	03:00	00:18	
asiakkaan muokkaus	03:01	03:10	00:09	
työaikamerkintöjen poisto	03:12	03:17	00:05	
projektin poisto	03:18	03:23	00:05	
asiakkaan poisto	03:23	03:27	00:04	
yhteensä	00:09	03:27	03:18	

Kuva 12: Kuvakaappaus osallistujan nauhoitteen Flexerin litteroinnista Excel-ohjelmalla

Kuvassa 12 esiintyvät toiminnallisuudet merkattiin alkaneeksi, kun osallistuja lopetti toiminnallisuutta vastaavan tehtävän lukemisen. Toiminnallisuus katsottiin päättyneeksi, kun osallistuja suoritti interaktion loppuun niin, että muutos oli nähtävissä sovelluksen käyttöliittymän kautta. Suorituksen kesto on laskettu vähentämällä toiminnallisuuden suorittamisen päätösajasta sen aloitusaika.

Kanasen (2014, 101) mukaan aineistoja voi käsitellä erilaisin analyysimenetelmin, jotta aineistoon kätkeytyvän salaisuuden saa esille. Aineiston käsittely on jaettavissa neljään eri vaiheeseen: yhteismitallistamiseen, koodaukseen, luokitteluun ja yhdistämiseen. Yhteismitallistaminen, joka tunnetaan myös litterointina, tarkoittaa aineistojen muuttamista tekstimuotoon. Koodauksessa litteroitu aineisto pelkistetään yhdistämällä aineiston osia yhteisillä tekijöillä, jotta aineisto olisi selkeämpää ja tiiviimpää. Luokittelussa ja yhdistämisessä taas koodauksen pohjalta syntyneet tekijät määritellään saman luokan piiriin ja yhdistetään pääkäsitteiksi. Lopuksi kyseenomaisten vaiheiden pohjalta luodaan tulkinta, joka vastaa tutkimusongelmaan tai -kysymyksiin. (Kananen, 101-118.)

Kanasen selvittämiä menetelmiä sovellettiin keräämällä nauhoitteiden litteroinnin pohjalta syntyneet muistiinpanot omaan tiedostoonsa. Sen jälkeen siirryttiin koodaukseen, jossa saman tyyppiset muistiinpanot niputettiin yhteen ja niille nimettiin yhdistävä tekijä. Sovellusten muistiinpanojen koodaus tehtiin erillisissä tiedostoissa selkeyden vuoksi. Taulukossa 6 on esimerkkinä selvitetty, kuinka Togglen muistiinpanot liittyen tehtävänantoon on ensiksi niputettu ja sitten koodattu.

Taulukko 6: Litteroitujen Toggl-sovelluksen tehtävänantoon liittyvien muistiinpanojen koodaus

Raakateksti	**Koodaus**
Osallistuja onnistui poistamaan merkinnät, mutta poistotoiminnallisuutta ei voitu mitata koska se tapahtui päällekkäin muiden tehtävien kanssa.	Tehtävänanto
Osallistuja onnistui poistamaan projektin, mutta poistotoiminnallisuutta ei voitu mitata koska se tapahtui päällekkäin muiden tehtävien kanssa.	
Osallistuja onnistui poistamaan asiakkaan, mutta poistotoiminnallisuutta ei voitu mitata koska se tapahtui päällekkäin muiden tehtävien kanssa.	
Tehtävänannon kieli ja termit poikkesivat käyttöliittymästä.	
Epäselvyyksiä tehtävänannon kanssa.	
Osallistuja selvitti 6s ajatuksiaan ennen työaikamerkinnän tallentamista.	

Koodauksen jälkeen tekijöille määritettiin yhteiset luokat, joiden pohjalta lopulta saatiin selville mittaustuloksiin vaikuttavat tekijät. Jokainen analyysivaihetta vastaava tiedosto nimettiin ”vaihe_N_vaiheen_nimi.xlsx”-nimen mukaisesti, eli esimerkiksi litterointivaihetta koskeva tiedosto sai nimekseen ”vaihe_1_litterointi.xlsx”. Poikkeuksena tiedostojen ni-

meämisessä oli Flexerin ja Togglen koodaus erillisissä tiedostoissa, joissa tiedostot nimettiin ”vaihe_2_flexer_koodaus.xlsx” ja ”vaihe_2_toggl_koodaus.xlsx”. Taulukossa 7 on selvitetty, kuinka Flexer ja Toggl -koodausten tehtävänanto ja meluhaitta on luokiteltu ohjeiksi ja häiriööäniksi, jonka jälkeen niille on määritetty yhdistäväksi luokaksi tuloksiin vaikuttavat asiat.

Taulukko 7: Esimerkki koodausten luokittelusta yhdistäväksi luokaksi

Koodaus	Taso 1	Yhdistävä luokka
Tehtävänanto	Ohjeet	Tuloksiin vaikuttavat asiat
Tehtävänanto		
Meluhaitta	Häiriööänet	
Meluhaitta		

Mitallistamisen pohjalta syntyneet tulokset on lopulta avattu käänteisessä järjestyksessä alkaen yleisestä yhdistävästä luokasta päättyen yksityiskohtaiseen muistiinpanoon. Esimerkiksi, taulukon 7 pohjalta voidaan luoda selvitys, jossa mittaustuloksiin saattoi vaikuttaa ohjeistus ja häiriööänet. Ohjeistukseen liittyi tehtävänanto ja häiriööääniin meluhaitat. Tarkemmat yksityiskohdat liittyen tehtävänannon ja meluhaitan ongelmiin on avattu seuraavassa luvussa 5.4 mittaustulosten jälkeen.

5.3 Mittaustulokset

Mittaustulokset on listattu seuraaviin taulukoihin toiminnallisuuden tyyppi kerrallaan. Jokaisessa taulukossa on neljä saraketta, joista ilmenee toiminnallisuus, osallistujan numero sekä suorittamiseen kulunut aika Flexerissä ja Togglessa. Toisessa ja kolmannessa sarakkeessa on selvitetty toiminnallisuuden suorittamiseen kulunut aika Flexerissä ja Togglessa. Kesto-sarakkeissa on korostettu vihreällä värillä tulos, joka on kestänyt lyhyemmän ajan. Sarake on värjätty siniseksi, jos toiminnallisuuksien suorittaminen on vienyt osallistujalta yhtä kauan aikaa molemmissa sovelluksissa. Taulukoissa on merkitty ”-”-merkillä sarakkeen sisältö, jos osallistuja on keskeyttänyt toiminnallisuutta vastaavan tehtävän tai suoritukseen kulunut aika on ollut muuten mahdotonta mitata.

Seuraavaksi taulukoissa 8 – 10 on raportoitu kvantitatiivisin menetelmin, kuinka kauan kullakin osallistujalla kului aikaa eri toiminnallisuuksien suorittamiseen. Tutkimuksessa oli neljä erilaista lisäystoiminnallisuutta koskevaa tehtävää. Kaikki osallistujat onnistuivat sekä asiakkaan että projektin lisäystoiminnallisuuksissa. Kolme viidestä osallistujasta suoritti asiakkaan lisäyksen nopeammin Togglessa kuin Flexerissä. Projektin lisäys hoitui neljältä osallistujalta nopeammin Togglessa ja yhdeltä taas Flexerissä. Ensimmäisen työaikamerkinnän lisääminen onnistui kaikilta Flexerissä ja neljältä Togglessa. Kaikki osallistujat suorittivat ensimmäisen työaikamerkinnän lisäämisen nopeammin Flexerissä kuin Togglessa.

Toisen työaikamerkinnän lisäys onnistui myös kaikilta Flexerissä ja neljältä Togglessa. Neljä osallistujaa suoriutui toisen merkinnän lisäämisestä nopeammin Flexerissä ja yksi Togglessa. Seuraavassa taulukossa 8 on listattuna tarkemmin lisäystoiminnallisuuksien kestot.

Taulukko 8: Osallistujien käyttämä aika lisäystoiminnallisuuksissa

Toiminnallisuus	Osallistujan nro.	Kesto Flexerissä (mm:ss)	Kesto Togglessa (mm:ss)
Asiakkaan lisäys	1	00:27	00:34
	2	00:38	00:09
	3	00:42	00:15
	4	00:18	00:20
	5	00:12	00:08
Projektin lisäys	1	00:43	00:40
	2	00:44	00:16
	3	00:54	00:24
	4	00:48	01:20
	5	00:34	00:20
Ensimmäisen työaikamer-kinnän lisäys	1	02:01	-
	2	00:49	02:26
	3	01:36	06:16
	4	00:52	01:20
	5	00:45	01:28
Toisen työaikamerkinnän lisäys	1	00:56	-
	2	00:24	00:36
	3	00:37	00:58
	4	00:55	00:26
	5	00:30	00:42

Tutkimuksessa oli muokkaustoiminnallisuuksien mittaamista varten laadittu kolme tehtävää liittyen asiakkaan, projektin ja työaikamerkinnän muokkaamiseen. Kaikki osallistujat onnistuivat muokkaamaan asiakkaan nimen Flexerissä ja Togglessa taas onnistui neljä. Kaksi osallistujaa suoriutui asiakkaan nimen muokkaamisesta nopeammin Flexerissä, kaksi Togglessa ja yksi osallistuja oli yhtä nopea molemmissa.

Projektin nimen muokkaus luonnistui kaikilta osallistujilta Flexerissä ja neljältä Togglessa. Kaksi osallistujaa suoritti toiminnallisuuden nopeammin Flexerissä ja kolme hoiti sen vikkelämmin Togglessa. Työaikamerkinnän aloitusajan muokkaus hoitui kaikilta Flexerissä ja kolmelta Togglessa. Yksi hoiti merkinnän muokkauksen nopeammin Togglessa ja loput neljä osallistujaa taas Flexerissä. Seuraavan sivun taulukossa 9 on selvitetty tarkemmin toiminnallisuuksiin kuluneet kestot osallistujakohtaisesti.

Taulukko 9: Osallistujien käyttämä aika muokkaustoiminnallisuuksissa

Toiminnallisuus	Osallistujan nro.	Kesto Flexerissä (mm:ss)	Kesto Togglessa (mm:ss)
Asiakkaan nimen muokkaus	1	00:18	-
	2	00:12	00:10
	3	00:18	00:10
	4	00:18	01:01
	5	00:09	00:09
Projektin nimen muokkaus	1	00:25	-
	2	00:16	00:13
	3	00:29	00:12
	4	00:14	00:38
	5	00:18	00:11
Työaikamerkinnän aloitusajan muokkaus	1	00:59	-
	2	00:18	00:44
	3	00:16	00:30
	4	00:24	-
	5	00:12	00:07

Poistamista mitattiin kolmella eri toiminnallisuudella: työaikamerkintöjen, projektin ja asiakkaan poistamisella. Kaikki osallistujat suoriutuivat Flexerissä kaikista poistotoiminnallisuuksista. Neljän osallistujan suoritusta viidestä pystyttiin mittaamaan Togglessa. Kaikki osallistujat suorittivat poistotoiminnallisuudet nopeammin Flexerissä kuin Togglessa yhtä osallistujaa lukuun ottamatta, sillä yksi osallistuja poisti asiakkaan yhtä nopeasti sekä Flexerissä että Togglessa. Taulukossa 10 on täsmennetty osallistujien poistotoiminnallisuuksista suoriutumista.

Taulukko 10: Osallistujien käyttämä aika poistotoiminnallisuuksiin

Toiminnallisuus	Osallistujan nro.	Kesto Flexerissä (mm:ss)	Kesto Togglessa (mm:ss)
Molempien työaikamerkintöjen poistaminen	1	00:25	-
	2	00:07	00:30
	3	00:05	00:09
	4	00:03	00:07
	5	00:05	00:06
Projektin poistaminen	1	00:06	-
	2	00:03	00:10
	3	00:03	00:06
	4	00:01	00:07
	5	00:05	00:13
Asiakkaan poistaminen	1	00:05	-
	2	00:03	00:10
	3	00:04	00:04
	4	00:02	00:06
	5	00:04	00:05

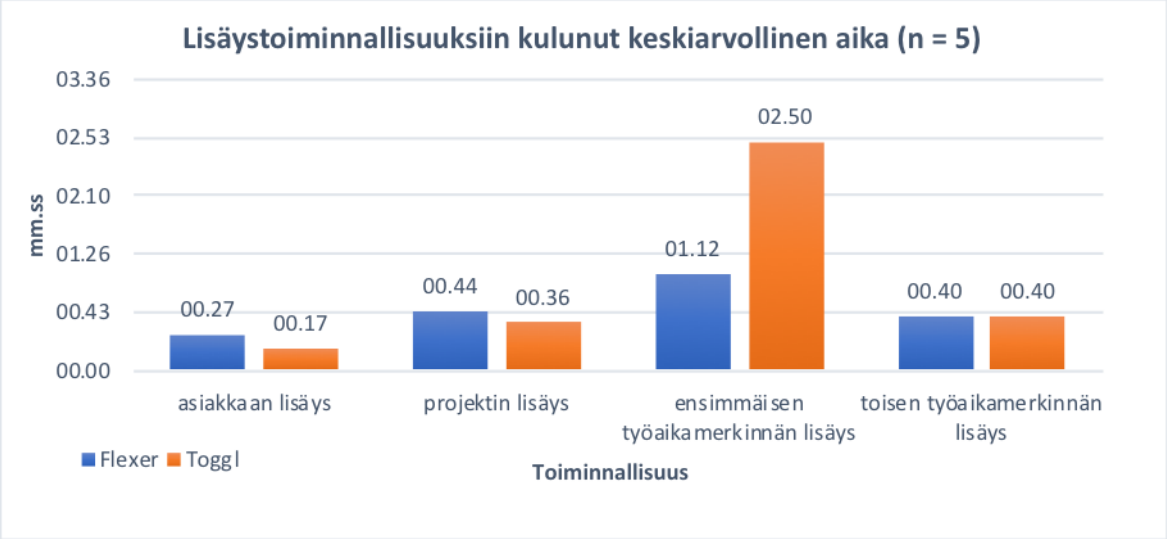
Taulukossa 11 on selvitetty osallistujakohtaisesti, kuinka paljon aikaa tehtävien suorittamiseen sovelluksissa kului. Kokonaiskestoon on laskettu mukaan tehtävien lukemiseen kulunut aika. Kaikki osallistujat suoriutuivat tehtävistä nopeammin Flexerissä.

Taulukko 11: Osallistujilla kulunut aika kaikkien toiminnallisuuksien suorittamiseen

Toiminnallisuus	Osallistujan nro.	Kesto Flexerissä (mm:ss)	Kesto Togglessa (mm:ss)
Kokonaiskesto	1	06:51	39:10
	2	04:10	06:03
	3	06:27	10:10
	4	04:34	07:24
	5	03:18	05:13

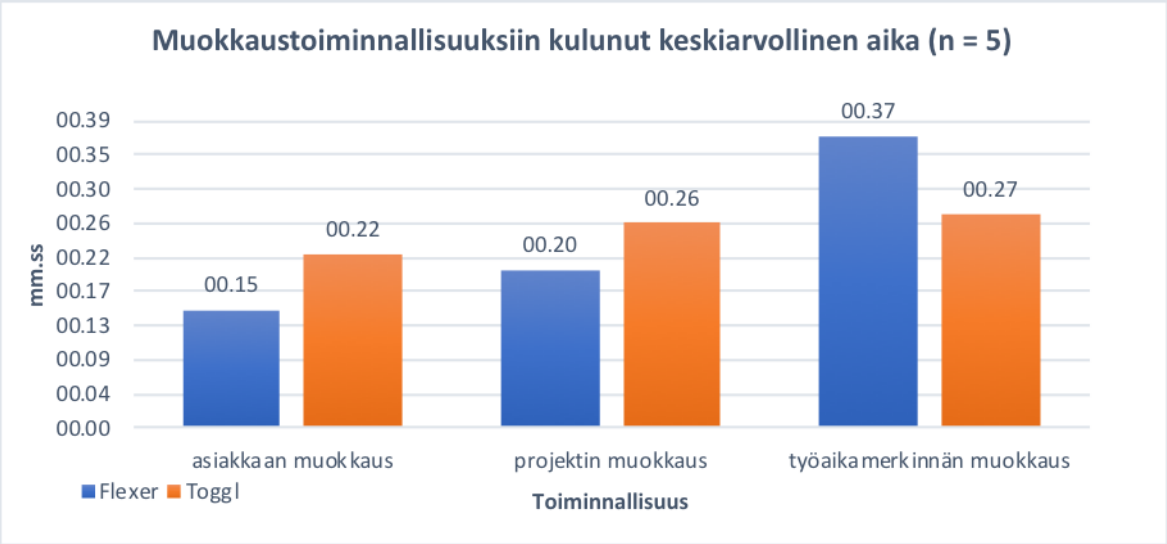
Seuraavaksi kuovioissa 1 – 3 on selvitetty lisäys-, muokkaus- ja poistotoiminnallisuuksiin kuluneet ajat keskiarvoina pylväsdiagrammeina. Jokaisen kuvion x-akselilla on ilmoitettu tarkkailtu toiminnallisuus ja y-akselilla toiminnallisuuteen kulunut aika minuutteina ja sekunteina muodossa ”mm.ss”. Kunkin toiminnallisuuden keskiarvo on selvitetty laskemalla onnistuneiden suoritusten kestojen summa jaettuna onnistuneiden suoritusten lukumäärällä. Kuvioissa toiminnallisuuksien kestoja kuvaa Flexerin kohdalla sininen palkki ja oranssin vuorostaan Togglen.

Lisäystoiminnallisuuksissa keskiarvollisesti osallistujilta kului Togglessa aikaa vähemmän asiakkaan ja projektin lisäyksessä. Asiakkaan lisäys onnistui kymmenen sekuntia nopeammin Togglessa ja projektin lisäys kahdeksan sekuntia nopeammin. Ensimmäisen työaikamerkinnän lisääminen sen sijaan onnistui Flexerissä yli puolitoista minuuttia nopeammin. Toisen työaikamerkinnän lisääminen onnistui sovelluksissa keskiarvollisesti samassa ajassa. Kuviossa 1 on havainnollistettu lisäystoiminnallisuuksien kestojen keskiarvoja.



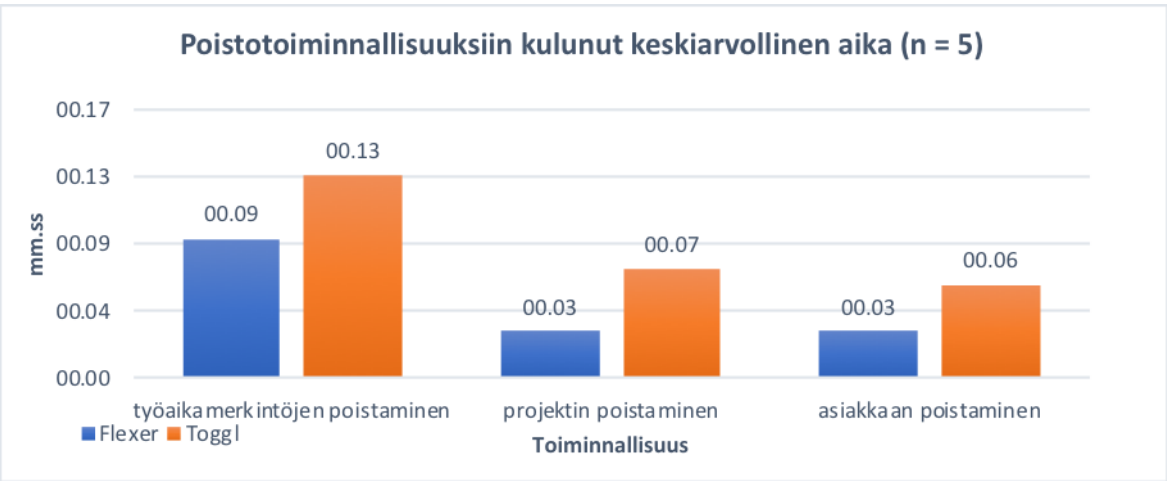
Kuvio 1: Lisäystoiminnallisuuksiin kulunut aika keskiarvoksi laskettuna

Muokkaustoiminnallisuuksissa Flexer oli keskiarvoiltaan nopeampi asiakkaan ja projektin muokkauksissa. Asiakkaan muokkaus luonnistui seitsemän sekuntia nopeammin ja projektin muokkaus taas kuusi sekuntia nopeammin. Toggl oli nopeampi työaikamerkinnän muokkauksessa kymmenellä sekunnilla. Kuviossa 2 on visualisoituna muokkaustoiminnallisuuksien keskiarvolliset tulokset.



Kuvio 2: Muokkaustoiminnallisuuksiin kulunut aika keskiarvoina

Kuviossa 3 on selvitetty keskiarvollisesti poistotoiminnallisuuksiin kuluneita kestoja. Flexer oli kaikissa poistotoiminnoissa Togglea nopeampi. Työaikamerkintöjen ja projektin poistaminen onnistui neljä sekuntia nopeammin Flexerissä. Asiakkaan poistamiseen kului keskimäärin kolme sekuntia vähemmän aikaa Flexerissä kuin Togglessa.



Kuvio 3: Poistotoiminnallisuuksiin kulunut aika keskiarvoksi laskettuna

5.4 Mittaustuloksiin vaikuttavia tekijöitä

Tässä luvussa on raportoitu mittausaineiston pohjalta syntyneitä huomioita, jotka saattoivat vaikuttaa mittaustilaisuuksiin ja sitä myötä mahdollisesti myös tuloksiin. Tuloksiin mahdollisesti vaikuttavat tekijät on avattu seuraavaksi pääluokka kerrallaan. Aineiston nauhoitteista havainnoitiin neljä pääluokkaa, jotka olivat: mittaustyökalut, häiriöäänet, ohjeet ja sovellusten käyttöliittymät. Mittaustyökaluihin liittyi kannettavan tietokoneen näppäimistöön totuttelu Flexer-sovelluksen tehtäviä suorittaessa sekä epähuomiossa tutkijalta jäänyt ”tag”-merkki Toggl-sovellukseen. Liittyen häiriööniin, yhden mittaustilaisuuden ajan oli äänekäs tiskikone päällä. Toisessa tilaisuudessa taas oli ikkuna auki ilmanvaihdon vuoksi, jonka takia ulkoa kuului selvästi lumen sulamisen seurauksena maahan putoavat vesipisarat.

Ohjeet-luokkaan liittyi tehtävänanto-ongelmia, joita listaantui nauhoitteista yhteensä Flexeristä 11 ja Togglesta kuusi. Yhden osallistujan Toggl-suorituksia ei voitu mitata, koska osallistuja suoritti tehtävät limittäin eikä järjestyksessä yksi kerrallaan. Tehtävänantoon liittyi myös eroavaisuudet tehtäväohjeiden ja käyttöliittymien kielten välillä, sillä tehtävät oli kirjoitettu osallistujille suomeksi ja sovellusten käyttöliittymät olivat englanniksi. Kieleen liittyviä muistiinpanoja kertyi Flexeristä kaksi ja Togglesta yksi. Kielten lisäksi, tehtävänantoon liittyi yhden tehtävän jakaminen kahdelle sivulle. Kyseenomainen tehtävän jakamiseen merkattiin kaksi muistiinpanoa Flexerin kohdalle, jossa osallistujat tarkistivat erikseen ennen toiminnallisuuden suorittamisen lopetusta toiselta tehtäväselostesivulta, että jatkuiko tehtävä. Tehtävänantoon liittyi myös epäselvyys, kumpi lisätyistä työaikamerkinnöistä piti muokata. Kyseenomainen epäselvyys liittyi ainoastaan Flexeriin.

Tuloksiin vaikuttavat ongelmat koskien sovellusten käyttöliittymiin, liittyivät tehtävien osittaiseen suorittamiseen ja tehtävänkuvausta vastaavien syötekenttien löytämiseen. Yksi osallistuja vaihtoi Flexerissä työaikamerkinnän tehtävänannon mukaisesti oikein, muttei tallentanut muutosta loppuun. Monella osallistujalla kului Flexerissä ylimääräistä aikaa projektiin kuuluvan asiakkaan valintakentän löytämisessä. Togglessa taas neljä osallistujaa ei merkannut työaikamerkintöihin liittyvää projektia. Kaikilla osallistujilla oli vaikeuksia löytää Togglen työaikamerkintänäkymä ja neljälle osallistujalle kerrottiin erikseen, missä työaikamerkintöihin liittyvät tehtävät tehdään. Kaksi osallistujaa ei vaihtanut Togglessa työaikamerkinnän päiväystä ollenkaan. Kaikkia osallistujia hidasti Togglessa 12-tunnin aikaformaatti työaikamerkintöjen aloitus- ja lopetuskenttien kohdalla.

6 Pohdinta

Tässä luvussa tarkastellaan aluksi luvussa neljä raportoitua sovelluksen kehitysprosessia sekä luvussa viisi selvitettyä mittaustutkimusta ja sen tuloksia. Tulosten tarkastelun jälkeen selvitetään opinnäytetyön eettisyyttä ja luotettavuutta viitaten Tutkimuseettisen neuvottelukunnan selvitykseen hyvistä tieteellisen käytännön menetelmistä. Sen jälkeen on avattu tutkimuksen johtopäätökset tulosten tarkastelun pohjalta ja pohdittu syvällisemmin tulosten merkitystä. Johtopäätösten lisäksi on annettu tutkimuksen parannusehdotuksia ja jatkotutkimusideoita. Viimeisessä alaluvussa on lopuksi selvitetty opinnäytetyön tekijän aatteita opinnäytetyöprosessista sekä omaa oppimista.

6.1 Tulosten tarkastelu

Tässä luvussa on tarkasteltu ensiksi produktina syntyneen Flexer-työaikakirjanpitosovelluksen luvussa 4 selvitettyä kehitysprosessia. Kehitysprosessin selvityksen jälkeen on tarkasteltu produktin toimivuutta avaamalla luvuissa 5.3 ja 5.4 raportoituja mittaustuloksia ja tuloksiin vaikuttavia tekijöitä. Luvussa 6.3 on lopulta pohdittu tarkemmin koko työn onnistumista tulosten tarkastelun pohjalta.

Flexer-sovelluksen kehitys oli jaettu selviksi kokonaisuuksiksi, jota tukee myös Garcían (2017) selvitys sovelluskehitysprosessin rakenteesta. Sovelluksen kehitys jaettiin Garcían tukeman selvityksen mukaisesti neljään vaiheeseen: hahmotteluun, suunnitteluun, ohjelmointiin ja käytön mahdollistamiseen. Hahmottelussa selvitettiin produktin taustat, käyttäjät ja käyttötapaukset peilaten tietoperustan työaikalukuun 2. Hahmottelu toteutettiin hyödyntämällä Knasterin ja Leffingwellin (2017, luku 3) kuvailemaa Lean-menetelmää sekä ketterän ohjelmistokehityksen julistuksen (2001) periaatteita. Hahmottelussa huomioitiin työajan kirjaamiseen liittyviä Työaikalain (605/1996) säädöksiä sekä Työterveyslaitoksen (2016) ja Zwermanin ja Okunin (2017) selvittämiä työajan seurannan käytänteitä.

Hahmotelma-vaiheen jälkeen luotiin suunnitelma Lean-menetelmää käyttäen sovelluksen toiminnallisuuksista, arkkitehtuurillisista rakenteista, käyttöliittymämallinnukset sekä selvitys käytettävistä ohjelmointityökaluista. Ohjelmointityökalujen valinnat perusteltiin peilaten tietoperustan luvun 3 alalukuihin. Sovelluksen ohjelmointi tapahtui Djangon käyttämää MVT-sovellusarkkitehtuurimallia noudattaen, jota tukee myös Dauzonin, Bendoraitiksen ja Ravindran (2016, luku 1.1) selvitys aiheesta. Sovelluksen tietokantojen rakenne ja toiminnallisuudet luotiin suunnitelmavaiheen pohjalta syntyneen dokumentaation mukaisesti.

Ohjelmointivaiheen jälkeen käynnistyi sovelluskehityksen viimeinen vaihe eli käytön mahdollistaminen. Käytön mahdollistamista selvitettiin tutkimuksella, jonka menetelmät ja tulokset on avattu luvussa 5. Tutkimuksen mittaustuloksista ilmeni, että produktina syntyneen työaikakirjanpitosovellus Flexerin toiminnallisuuksien suorittamiseen kului osallistujilta kaiken kaikkiaan vähemmän aikaa kuin tuotteistetussa vastaavanlaisen sovelluksessa nimeltä Toggl.

Asiakkaan ja projektin lisäystoiminnallisuuksissa Toggl oli Flexeriä nopeampi pienellä erolla. Työaikamerkinnän lisäys onnistui useammin ja nopeammin Flexerissä kuin Togglessa. Muokkaustoiminnallisuuksissa oli havaittavissa samaa trendiä, eli asiakkaan ja projektin muokkaaminen onnistui Togglessa aavistuksen nopeammin kuin Flexerissä, mutta työaikamerkinnän muokkaus sen sijaan oli paljon nopeampaa Flexerissä. Poistotoiminnallisuudet hoituivat kaikilta osallistujilta nopeammin Flexerissä kuin Togglessa.

Lisäksi yleishuomiona, kaikki osallistujat onnistuivat Flexerin kohdalla kaikissa kymmenessä tehtävässä Flexerissä niin, että suoritukset olivat mitattavissa. Togglessa taas yhdeltä osallistujalta jäi kahdeksan suoritusta mittaamatta, koska osallistuja ei suorittanut tehtäviä järjestyksessä turhautumisen takia. Toinen osallistuja taas päätti ohittaa työaikamerkinnän muokkaustehtävän, koska hän ei löytänyt miten toiminnallisuus suoritetaan Togglen käyttöliittymän kautta.

6.2 Tutkimuksen eettisyys ja luotettavuus

Tutkimuksen voi osoittaa luotettavaksi ja eettisesti hyväksi noudattamalla hyvää tieteellistä käytäntöä. Lisäksi, kun noudattaa hyvää tieteellistä käytäntöä, tutkimuksen tuloksista tulee uskottavia. (Tutkimuseettinen neuvottelukunta 2012, 6.) Tutkimuseettinen neuvottelukunta on jaotellut hyvän tieteellisen käytännön keskeiset lähtökohdat yhdeksään osaan. Jokainen lähtökohta tullaan seuraavaksi yksi kerrallaan esittelemään ja selvittämään, kuinka lähtökohdan käytäntöä ollaan tässä opinnäytetyössä noudatettu.

Tutkimuseettisen neuvottelukunnan (2012, 6) mukaan tutkimuksessa tulee noudattaa rehellisyyttä, huolellisuutta ja tarkkuutta tutkimusta tehdessä, tuloksia tallentaessa ja esittäessä sekä tulosten arvioinnissa. Tämän opinnäytetyön tutkija on pyrkinyt olemaan puolueeton, huolellinen ja systemaattisen tarkka tutkimuksen toteutuksessa, aineistojen käsittelyssä sekä tulosten raportoinnissa.

Hyvään tieteelliseen käytäntöön kuuluu myös tieteellisen tutkimuksen kriteerien mukainen ja eettisesti kestävien tiedonhankinta-, tutkimus- ja arviointimenetelmien käyttö (Tutkimus-

eettinen neuvottelukunta 2012, 6). Opinnäytetyön tietoperustan selvityksissä on pyritty käyttämään luotettavia tiedekirjallisuuden primäärilähteitä ja täydentämään sekundaarilähteillä tietoa, joka ei primäärilähteistä ollut saatavilla. Opinnäytetyön empiriassa käytetyiksi tutkimusmenetelmiksi on valittu menetelmiä, jotka on todettu ja hyväksytty tiedeyhteisöissä. Tutkimusmenetelmät ja -aineiston käsittely on kuvattu tarkemmin tämän opinnäytetyön luvuissa 4 ja 5.

Tutkimusta tehdessä, muiden tutkijoiden työtä ja saavutuksia tulee kunnioittaa viittaamalla heidän julkaisuihinsa asianmukaisin tavoin niin, että tutkijoiden saavutukset saavat niille kuuluvan arvon ja merkityksen (Tutkimuseettinen neuvottelukunta 2012, 6). Tässä opinnäytetyössä on pyritty viittaamaan lähteisiin pohjautuvaan kirjallisuuteen tutkijan parhaimman osaamisen mukaan vääristelemättä alkuperäisiä merkityksiä. Lisäksi, lähdeviittaukset on tarkistettu useampaan otteeseen opinnäytetyön luomisprosessin aikana, jotta tutkijat saavat heille kuuluvan arvonsa.

Tutkimus tulee suunnitella, toteuttaa ja raportoida sekä tutkimuksesta syntyneiden materiaalien tallentamisessa tulee noudattaa tieteellisiä asetusten mukaisia vaatimuksia (Tutkimuseettinen neuvottelukunta 2012, 6). Opinnäytetyössä teetetty tutkimus suunniteltiin, toteutettiin ja raportoitiin huolella. Tutkimuksen aineiston käsittelymenetelmät on raportoitu luvussa 5 sekä tutkimukseen osallistuneille annetut dokumentit on lisätty tämän opinnäytetyön liitteiksi.

Tutkimuseettisen neuvottelukunnan (2012, 6) mukaan tutkimusta tehdessä pitää hankkia tutkimusluvat. Ennen opinnäytetyöprosessin alkamista, Haaga-Helian ammattikorkeakoulu tarkasti ja hyväksyi opinnäytetyön aiheen. Lisäksi, opinnäytetyön tutkimukseen osallistuneilta henkilöiltä pyydettiin heidän suostumuksensa suostumuslomakkeen (ks. liite 4) avulla.

Ennen tutkimuksen aloittamista tulee kaikille tutkimuksen osapuolille selvittää heidän oikeudet, tutkimuksen tekijä, vastuut ja velvollisuudet. Osapuolille tulee myös selvittää tutkimuksen aineistojen käsittelyyn liittyvät yksityiskohdat niin, että osapuolet hyväksyvät ne. (Tutkimuseettinen neuvottelukunta 2012, 6.) Tutkimukseen osallistujille selvitettiin tutkimuksen kohde, tarkoitus ja hyödyt sekä mihin osallistujat sitoutuvat ryhtyessään tutkimukseen. Osallistujille selvitettiin, että tutkimus on vapaaehtoinen ja sen voi keskeyttää milloin tahansa. Heille myös kerrottiin, että tutkimustilaisuus nauhoitetaan ja nauhoitteista syntyvä aineisto tullaan käsittelemään ja lopulta hävittämään asianmukaisesti. Kaikki tutkimuksen osapuolet hyväksyivät tutkimuksen ehdot ja muut yksityiskohdat allekirjoittamalla heille annetun suostumuslomakkeen.

Tutkimukseen osallistuville tulisi myös selvittää tutkimuksen rahoituslähteet sekä muut merkittävät asiat (Tutkimuseettinen neuvottelukunta 2012, 6). Tutkimukseen osallistuneille ilmoitettiin suostumuslomakkeessa ja suullisesti, että tutkimus on osa tutkijan tekemää opinnäytetyötä sekä tutkimuksesta ei synny rahallisia hyötyjä.

Hyvään tieteelliseen käytäntöön kuuluu myös se, että tutkija pidättäytyy kaikista arviointi- ja päätöksentekotilanteista tutkijan ollessa esteellinen (Tutkimuseettinen neuvottelukunta 2012, 7). Haaga-Helian henkilökunta suorittaa opinnäytetyön arvioinnin ja päätöksenteon, jolloin opinnäytetyön tekijä ei osallistu kyseenomaisiin tilaisuuksiin.

Tutkimuseettisen neuvottelukunnan (2012, 7) mukaan tutkimusorganisaation tulee myös noudattaa hyvää henkilöstö- ja taloushallintoa, sekä huolehtia tietosuojaa koskevista kysymyksistä. Haaga-Helialla on täysi vastuu edellä mainituista asioista, jonka seurauksena opinnäytetyön tekijä ei ota opinnäytetyössään kantaa aiheeseen.

Opinnäytetyön aihe ja ratkaisuun käytetyt menetelmät tekevät työstä myös eettisesti hyväksytyn. Vastaavanlaisia opinnäytetöitä ei ole olemassa useita, joissa ohjelmointityökalujen valintakriteerinä on ollut suosio sekä toteutuksessa olisi käytetty nykyaikaisia ketteriä kehitysmenetelmiä. Suosittujen ohjelmointityökalujen ja nykyaikaisten kehitysmenetelmien käyttäminen on tärkeää, kuten myös Gleeson (2017, luku 2) toteaa, jonka seurauksena opinnäytetyö on ajankohtainen ja hyödynnettävä erityisesti ohjelmistokehityksen puolella.

6.3 Johtopäätökset, kehittämis- ja jatkotutkimusehdotukset

Opinnäytetyön produktina syntynyt Flexer-sovelluksen suunnittelu ja toteutus onnistui paremmin kuin mitä ennalta odotettiin. Onnistumisesta kertoo sovelluksen kehitykselle varatussa aikataulussa pysyminen, joka taas viestii sitä, että sovelluksen suunnitteluun, käytettyjen teknologioiden opetteluun sekä itse toteutukselle jätettiin riittävästi aikaa. Projektin jakaminen eri osiin mahdollisti keskittymisen yhteen kehitysvaiheeseen kerrallaan, jonka seurauksena suunnittelu ja toteutus -prosessit olivat erittäin suoraviivaisia ja helposti hallittavissa. Lisäksi, ohjelmointityökalujen valinta onnistui hyvin, koska kehitykseen valituista työkaluista ei erikseen syntynyt vaivaa, joka olisi vaikuttanut kehitysaikatauluun. Työkaluvalinnan epäonnistuessa, toteutukseen olisi muuten voinut kulua huomattavasti enemmän aikaa, jolloin aikataulussa pysyminen olisi ollut epärealistista. Lopputuloksena syntyi kaiken kaikkiaan produkti, joka täytti kaikki toiminnallisuusvaatimukset, jotka sille oltiin ennalta määritetty.

Onnistumisesta kertoo myös tutkimuksen tulokset. Kahden kuukauden aikana kahden kehittäjän luoma Flexer-sovellus pystyi haastamaan ilman erillistä käyttäjätestausta onnistuneesti Toggl-sovelluksen, joka on useamman ammattilaisten pitkälle kehittämä ja monien yritysten käyttämä ratkaisu. Kuten myös tutkimuksen muistiinpanojen analyysin tuloksista ilmeni, tutkimuksen tehtävänanto vaikutti enemmän Flexeriin, sillä tehtävät suoritettiin ensiksi Flexerissä. Sen seurauksena on todennäköistä, että osallistujilla kului ylimääräistä aikaa tottua tutkimustilanteeseen ja muistaa tehtävänanto. Tottuminen tilanteeseen ja tehtävien muistaminen, ovat saattaneet hidastaa osallistujien suorituksia Flexerissä, jolloin Flexerin suoritukset olisivat todellisuudessa vieläkin nopeampia. Tilanteeseen tottumisen lisäksi, aineistojen muistiinpanojen analyyseistä ilmeni, että joillakin osallistujilla kului ylimääräistä aikaa näppäimistöön tottumisessa, joka todennäköisesti vaikutti enemmän Flexerin tuloksiin.

Sovellusten järjestys saattoi toisaalta myös hidastaa osallistujien suorituksia Togglessa, koska osallistujat saattoivat tottua Flexerin käyttöliittymään. On mahdollista, että tottuessaan Flexerin käyttöliittymään, osallistujilla kului ylimääräistä aikaa Togglen käyttöliittymän ymmärtämiseen, jonka seurauksena Togglen suoritukset olisivat hitaammat kuin mitä tutkimus osoittaa. Mittaustuloksiin saattoi vaikuttaa myös Togglen koko sovelluksena, sillä kuten liitteen 7 kaikista kuvista ilmenee, Togglessa on huomattavasti enemmän linkkejä nähtävissä eri näkymiin kuin Flexerissä. Toisaalta on myös mahdollista, että näkymiin johtavien linkkien nimet saattoivat harhaanjohtaa osallistujia, joka kertoo Togglen puutteellisesta käyttöliittymäsuunnittelusta.

Flexerin mittaustuloksiin saattoi vaikuttaa myös validoinnin (engl. validation) eli käyttäjän syötteen vahvistamisen puute. Suoritukset Flexerissä olisivat saattaneet nopeutua, mikäli Flexerissä olisi ollut hyvät syötteen validointiominaisuudet. Samalla vahvisteiden puute todennäköisesti nopeutti Flexerin poistotoiminnallisuuksia, koska esimerkiksi asiakasta poistaessa sovellus ei erikseen kysynyt osallistujalta, halutaanko kyseinen asiakas varmasti poistaa. Togglessa taas asiakkaan poiston yhteydessä sovellus kysyi osallistujalta, haluttiinko valittu asiakas varmasti poistaa, joka saattoi hidastaa asiakkaan poistotoiminnallisuuden suorittamista.

Tutkimusta voisi siis tasapainottaa, lisäämällä Flexer-sovellukseen validointitoimintoja. Validoinnin lisäksi tutkimuksen tehtävänantoa voisi selventää esimerkiksi kirjallisesti tai näyttämällä sovellusten käyttöliittymistä osallistujille, missä näkymissä suoritettavat toiminnallisuudet toteutetaan. Siten pystyisi mahdollisesti vähentämään tutkimuksen tehtävänantoon totuttautumiseen kuluvaa aikaa sekä täsmentämään toiminnallisuuksien mittaamista. Tehtävänantoa voisi myös parantaa lisäämällä tehtäväselosteeseen toiminnalli-

suuksia vastaavia sanoja englanniksi tehtävien ohelle, jotta niitä vastaavien toiminnallisuuksien tunnistaminen käyttöliittymistä onnistuisi osallistujilta helpommin. Toisaalta, kyseenomaiset muutokset tehtävänantoon liittyen, keskittäisivät tutkimuksen painon sovelluksen yleistuntumasta toiminnallisuuksien käytettävyyteen.

Jatkotutkimuksena voisi siis luoda tutkimustilanteen, jossa selvitettäisiin enemmän sovellusten käytettävyyttä. Käytettävyyteen liittyvät tutkimuksen teettäminen olisi tuolloin järkevää toteuttaa kvalitatiivisesti eli laadullisia menetelmiä käyttäen, kuten esimerkiksi haastattelemalla osallistujia. Tutkimuksen voisi jatkossa toteuttaa myös esimerkiksi niin, että muutama osallistuja käyttäisi tutkittavia sovelluksia viikon ajan. Kuluneen viikon jälkeen, osallistujia haastateltaisiin ja siten saataisiin tuloksena syvällinen ymmärrys sovelluksen toimivuudesta ja käyttäjien tarpeista.

Toinen jatkotutkimustapa voisi olla selvitys sovelluksen tuotteistuksesta ja ylläpidosta. Selvityksestä voisi ilmetä, mitä sovelluksen pitää sisältää, jotta sitä voisi kutsua tuotteeksi. Tuloksena voisi syntyä esimerkiksi raportti sovelluksen julkaisuprosessista. Mikäli sovellukselle saataisiin oikea asiakas, voitaisiin myös selvittää, miten projektia hallinnoitaisiin asiakkaan, kehittäjien ja muiden mahdollisten sidosryhmien kanssa.

6.4 Opinnäytetyöprosessin ja oman oppimisen arviointi

Opinnäytetyö alkoi suunnittelulla, jossa selvitin opinnäytetyön aiheen, rajaukset ja tutkimusmenetelmiä. Suunnitelman pohjalta syntyi aikataulu, josta selvisi viikkokohtaisesti projektin toteutuksen vaiheita ja opinnäytetyön runko. Aikataulun yksityiskohtainen suunnittelu mahdollisti projektin toteutuksen systemaattisesti, jonka seurauksena minulle oli pääosin koko ajan selvää, mitä minun piti missäkin vaiheessa tehdä.

Eniten opinnäytetyössä haastetta loi toiminnallisen osion pohjalta syntyneen produktin onnistumisen konkretisointi mitattavaksi kokonaisuudeksi. Onneksi kuitenkin produktin mittaustutkimuksen toteutus- ja analysointimenetelmien hahmottuivat ja opinnäytetyön lopullinen rakenne selkeytyi, jonka pohjalta pystyin tekemään opinnäytetyön suoraviivaisesti loppuun. Kaiken kaikkiaan opinnäytetyö siis onnistui prosessina hyvin, sillä pysyin ennalta määritetyssä aikataulussa. Onnistumisesta kertoo myös tutkimuksen tulokset, jotka puhuvat produktin suunnittelun ja toteutuksen onnistumisen puolesta.

Opinnäytetyöprosessin eri vaiheiden kautta tuli myös opittua paljon erilaisia asioita. Yksi isoimmista opituista asioista oli suurikokoisen projektin toteutus hallittavasti, sillä siitä on varmasti hyötyä työelämässä. Toiseksi tärkeäksi opituksi asiaksi nousi sovelluksen

backend-osan toteutus ja siinä käytetyt ohjelmointityökalut, jotka ovat yhtä lailla työelämässä hyödynnettäviä asioita. Muita opittuja asioita olivat tietoperustassa selvitetyt työaikaan ja sovelluskehitykseen liittyvät tiedot. Tietoperustan asioiden lisäksi, opin tutkimuksen suunnittelua, tulosten analysointia ja raportointia, jotka ovat oleellisia taitoja myöhemmin käytäviä jatko-opintojani varten.

Lähteet

Dauzon, S., Bendoraitis, A. & Ravindran, A. 2016. Django: Web Development with Python. Packt Publishing. Birmingham.

Dayley, B., Dayley, B. & Dayley, C. 2017. Node.js, MongoDB and Angular Web Development, 2nd Edition. Addison-Wesley Professional. Boston.

DB-Engines 2018a. DB-Engines Ranking. Luettavissa: https://db-engines.com/en/ranking. Luettu: 5.2.2018.

DB-Engines 2018b. Method of calculating the scores of DB-Engines Ranking. Luettavissa: https://db-engines.com/en/ranking_definition. Luettu: 5.2.2018.

DB-Engines 2018c. DB-Engines Ranking – Trend Popularity. Luettavissa: https://db-engines.com/en/ranking_trend. Luettu: 5.2.2018.

DeBergalis, M. 1.12.2011. First Preview. Luettavissa: https://blog.meteor.com/first-preview-8d4675d7fe35. Luettu: 25.1.2018.

DeBergalis, M. 20.1.2012. Skybreak is now Meteor. Luettavissa: https://blog.meteor.com/skybreak-is-now-meteor-e680d1dbaf84. Luettu: 25.1.2018.

Forta, B. 2016. Sams Teach Yourself Microsoft® SQL Server T-SQL in 10 Minutes, Second Edition. Sams. Indianapolis.

García, B. 2017. Mastering Software Testing with Junit 5. Packt Publishing. Birmingham.

Giamas, A. 2017. Mastering MongoDB 3.x. Packt Publishing. Birmingham.

GitHub 2018a. Web application frameworks. Luettavissa: https://github.com/showcases/web-application-frameworks. Luettu: 26.1.2018.

GitHub 2018b. About stars. Luettavissa: https://help.github.com/articles/about-stars/. Luettu: 26.1.2018.

Gleeson, P. 2017. Working with Coders: A Guide to Software Development for the Perplexed Non-Techie. Apress. New York.

Goodrich, G. & Lenz, P. 2016. Rails: Novice to Ninja, 3rd Edition. SitePoint. Melbourne.

Gore, A. 2017. Full-Stack Vue.js 2 and Laravel 5. Packt Publishing. Birmingham.

Harrington, J. 2016. Relational Database Design and Implementation, 4th Edition. Morgan Kaufmann. Burlington.

Harrison, G. 2016. Next Generation Databases: NoSQL, NewSQL, and Big Data. Apress. New York.

Haviv, A., Mejia, A. & Onodi, R. 2016. Web Application Development with MEAN. Packt Publishing. Birmingham.

IEEE Spectrum 2017a. Interactive: The Top Programming Languages 2017. Luettavissa: https://spectrum.ieee.org/static/interactive-the-top-programming-languages-2017. Luettu: 18.1.2018.

IEEE Spectrum 2017b. IEEE Spectrum -selvityksen menetelmät ja lähteet. Luettavissa: https://spectrum.ieee.org/ns/IEEE_TPL_2017/methods.html. Luettu: 18.1.2018.

Kananen, J. 2014. Laadullinen tutkimus opinnäytetyönä: Miten kirjoitan kvalitatiivisen opinnäytetyön vaihe vaiheelta. Jyväskylän ammattikorkeakoulu. Jyväskylä.

Ketterän kehityksen manifesti 2001. Luettavissa: http://agilemanifesto.org/iso/fi/manifesto.html. Luettu: 12.3.2018.

Knaster, R. & Leffingwell, D. 2017. SAFe® 4.0 Distilled: Applying the Scaled Agile Framework® for Lean Software and Systems Engineering, First Edition. Addison-Wesley Professional. Boston.

PostgreSQL 2018a. About. Luettavissa: https://www.postgresql.org/about/. Luettu: 6.2.2018.

PostgreSQL 2018b. Advantages. Luettavissa: https://www.postgresql.org/about/advantages/. Luettu: 17.2.2018.

Robinson, J., Gray, A. & Titarenco, D. 2016. Introducing Meteor. Apress. New York.

Romano, F., Phillips, D. & van Hattem, R. 2016. Python: Journey from Novice to Expert. Packt Publishing. Birmingham.

Southekal, P. 2017. Data for Business Performance: The Goal-Question-Metric (GQM) Model to Transform Business Data into an Enterprise Asset. Technics Publications. Bradley Beach.

Stack Overflow 2018. Stack Overflow Trends. Luettavissa: https://insights.stackoverflow.com/trends?tags=ruby-on-rails%2Cdjango%2Claravel%2Csymfony%2Cmeteor%2Csails.js. Luettu: 26.1.2018.

Stauffer, M. 2016. Laravel: Up and Running. O'Reilly Media, Inc. Kalifornia.

Tutkimuseettinen neuvottelukunta 2012. Hyvä tieteellinen käytäntö ja sen loukkausepäilyjen käsitteleminen Suomessa. Luettavissa: http://www.tenk.fi/sites/tenk.fi/files/HTK_ohje_2012.pdf. Luettu: 4.2.2018.

Työaikalaki 9.8.1996/605. Luettavissa: https://www.finlex.fi/fi/laki/ajantasa/1996/19960605. Luettu: 29.1.2018.

Työsuojelu 2017. Työaika. Luettavissa: http://www.tyosuojelu.fi/tyosuhde/tyoaika. Luettu: 30.1.2018.

Työterveyslaitos 2016. Joustava työaika. Luettavissa: https://www.ttl.fi/tyontekija/tyoaika/joustava-tyoaika/. Luettu: 29.1.2018.

Welling, L. & Thomson, L. 2016. PHP and MySQL® Web Development, Fifth Edition. Addison-Wesley Professional. Boston.

Yhteiskuntatieteellinen tietoarkisto 2015. Kvantitatiivisen datatiedoston käsittely. Luettavissa: http://www.fsd.uta.fi/aineistonhallinta/fi/kvantitatiivisen-datan-kasittely.html. Luettu: 24.3.2018.

Yhteiskuntatieteellinen tietoarkisto 2017. Kvalitatiivisen datatiedoston käsittely. Luettavissa: http://www.fsd.uta.fi/aineistonhallinta/fi/kvalitatiivisen-datan-kasittely.html. Luettu: 24.3.2018.

Zwerman, S. & Okun, J. 2017. The VES Handbook of Visual Effects, 2nd Edition. Focal Press. Waltham.

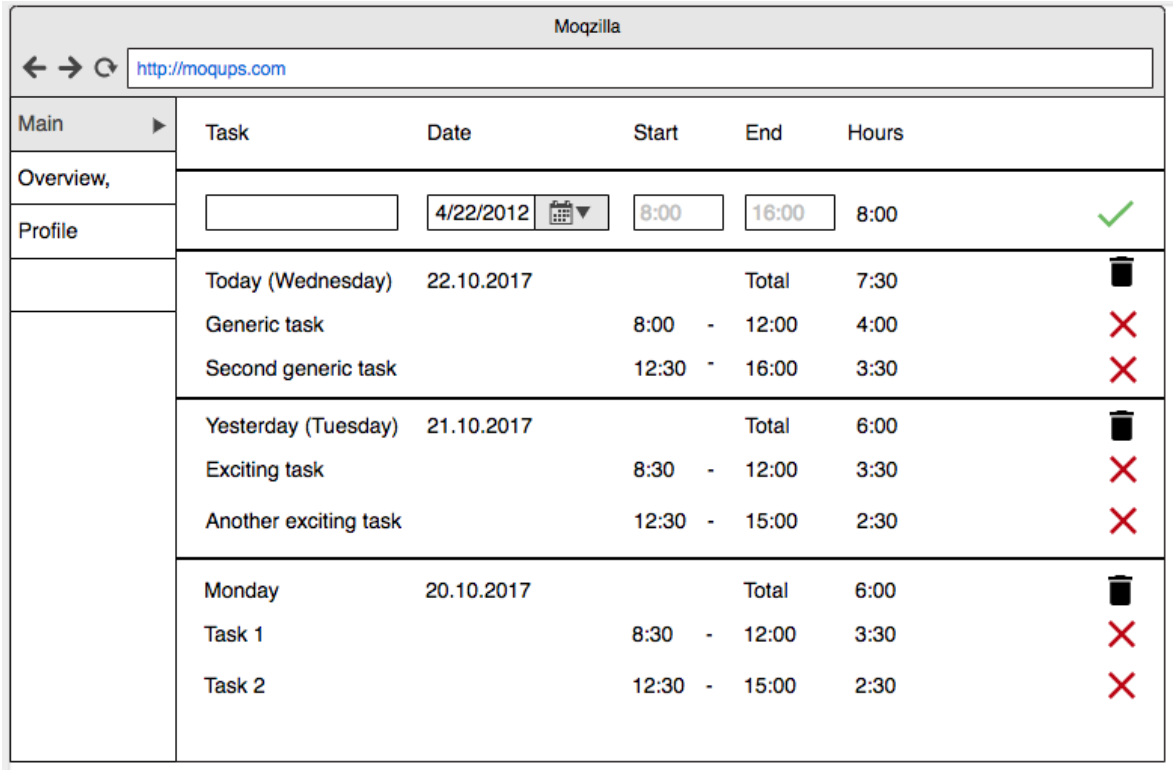
Liitteet

Liite 1. Projektin riskikartoitus

* (P = pieni, K = kohtuullinen, S = suuri)

Riski	Todennäköisyys*	Seurausvaikutus	Syyt	Ennaltaehkäisy
Uupumus / burnout	K	Projekti seisahtuu / hidastuu	Liikaa töiden ja projektin tekoa	Enemmän vapaa-aikaa
Sairastuminen tai loukkaantuminen	S	Projekti seisahtuu	Sääolosuhteet, ruokavalio, treenin puute, alkoholin holtiton nauttiminen, onnettomuudet	Ruokavalion parantaminen, treenin lisääminen, alkoholin kohtuullinen nauttiminen, säännöllinen ja riittävä nukkuminen
Läheisen sairastuminen	S	Projekti hidastuu	-||-	-||-
Motivaation puute	K	Projekti seisahtuu / hidastuu	Projektin tai toiminnallisuuden toteuttamisen haasteellisuus, työparin motivaation puute	Kokonaisuuksien pilkkominen pienempiin osiin, hauskan pitäminen
Osaamisen puute	K	Projekti hidastuu	Heikko ja huolimaton perehtyminen aiheeseen	Perusteellinen aiheeseen perehtyminen ja opettelu
Epäselvä toiminnallisuuden määritelmä	P	Projekti hidastuu	Huono suunnittelu	Huolellinen suunnittelu
Koodin tuhoutuminen	P	Projektin uudelleensuunnittelu ja käynnistys	Olematon versionhallinta ja varmuuskopiointi	Hyvin suunniteltu versionhallinta ja toteutus
Laitteen rikkoutuminen	P	Projekti seisahtuu, kunnes laite korvataan	Laitteen valmistajan huolimattomuus, käyttäjän aiheuttama laitteen ylikuormitus tai pudottaminen	Laitteen huolellinen käyttäminen
Ajanpuute / Työkiireet	K	Projekti hidastuu	Huono suunnittelu	Ajan varaaminen projektille
Luonnonilmiöt / Sähkökatkos	P	Projekti seisahtuu / hidastuu	Vaarallinen ympäristö	Työskentely turvallisessa ympäristössä
Virhe ohjelmiston riippuvuuskirjastossa	P	Projekti hidastuu	Huolimaton ohjelmointi kirjaston tekijältä	Toimivien versioiden käyttäminen, huolellinen kirjaston valitseminen sen käyttöönottaessa

Liite 2. Työaikamerkintänäkymän mallinnus



Liite 3. Flexer-sovelluksen tietokanta ja taulujen sisältö

Tietokantataulut

```
 Schema |             Name              | Type  |   Owner
--------+-------------------------------+-------+------------
 public | auth_group                    | table | flexeruser
 public | auth_group_permissions        | table | flexeruser
 public | auth_permission               | table | flexeruser
 public | auth_user                     | table | flexeruser
 public | auth_user_groups              | table | flexeruser
 public | auth_user_user_permissions    | table | flexeruser
 public | django_admin_log              | table | flexeruser
 public | django_content_type           | table | flexeruser
 public | django_migrations             | table | flexeruser
 public | django_session                | table | flexeruser
 public | flexer_client                 | table | flexeruser
 public | flexer_project                | table | flexeruser
 public | flexer_task                   | table | flexeruser
```

Asiakas-taulun esimerkki tietorivejä

```
flexer=# select * from flexer_client;
                  id                  |     name      |        email        |     phone     |     address      | zip_code |   city   | business_id
--------------------------------------+---------------+---------------------+---------------+------------------+----------+----------+-------------
 16fc865e-e75c-4316-8aa6-8afc47a84445 | flying wheels | contact@fwheels.com | +358 452 2320 | Wheelstreet 23 A |    11430 | Vantaa   | 02349102
 7eb224a2-8da4-46bf-8144-995deb82d86e | jack          | contact@jsand.com   | +358 002 3210 | Sandstreet 1 B   |    55500 | Helsinki | 9218310
(2 rows)
```

Projekti-taulun esimerkki tietorivejä

```
flexer=# select * from flexer_project;
                  id                  |    name    |          description           | total_hours |              client_id
--------------------------------------+------------+--------------------------------+-------------+--------------------------------------
 2912e204-70ac-4aac-bba3-209014ace936 | sandcastle | jack's sandcastle project      |             | 7eb224a2-8da4-46bf-8144-995deb82d86e
 c0a8138a-689c-4e25-bbb3-69fa683653bb | supercar   | create a toy car which is fast |             | 16fc865e-e75c-4316-8aa6-8afc47a84445
(2 rows)
```

Työaikamerkintä-taulun esimerkki tietorivejä

```
flexer=# select * from flexer_task;
               task_id               |       name        |         date          |        start          |          end          | break_time | total_hours |            project
_id_id
--------------------------------------+-------------------+-----------------------+-----------------------+-----------------------+------------+-------------+-----------------
------------------
 7da75966-6980-49bd-b7fb-407a7eca997d | install wheels    | 2018-01-10 16:45:11+02 | 2018-02-25 10:00:00+02 | 2018-02-25 18:00:00+02 | 00:30:00   | 07:30:00    | c0a8138a-689c-4e25
-bbb3-69fa683653bb
 ac70286a-5fb6-4005-b2d1-a477deeb2b4a | change left mirror | 2018-01-11 16:45:52+02 | 2018-02-25 12:30:00+02 | 2018-02-25 13:30:00+02 | 00:00:00   | 01:00:00    | c0a8138a-689c-4e25
-bbb3-69fa683653bb
(2 rows)
```

Liite 4. Suostumuslomake tutkimukseen osallistumisesta

SUOSTUMUS TUTKIMUKSEEN OSALLISTUMISESTA

Tutkimus on osa opinnäytetyötä, joka teetetään Haaga-Helian ammattikorkeakoululle. Opinnäytetyön tarkoituksena on selvittää, miten verkkosovellus kannattaa suunnitella ja ohjelmoida. Opinnäytetyöstä hyötyvät kaikki, jotka haluavat selvittää ohjelmointiprosessin vaiheita sekä mitkä ovat hyviä ja suosituimpia työkaluja prosessin kehitykseen. Tutkimuksesta ei synny rahallista hyötyä tutkimukseen osallistuvalle eikä tutkimuksen tekijälle.

Opinnäytetyössä on luotu tietoperustan selvityksen pohjalta työaikakirjanpitosovellus, jota verrataan vastaavanlaisen tuotteistetun sovelluksen kanssa. Vertailu tapahtuu mittaamalla molempien sovelluksien vastaavien toiminnallisuuksien suorittamiseen kuluvaa aikaa. Mittaustulosten pohjalta on tarkoitus lopulta selvittää, kuinka hyvin opinnäytetyön ohella luotu sovellus on onnistunut.

Mittaus tapahtuu nauhoittamalla osallistujan tutkimuksen yhteydessä käyttämää kannettavan tietokoneen näytön kuvaa sekä osallistujan ääntä. Nauhoitteista syntyvää aineistoa käytetään ainoastaan opinnäytetyön tutkimustarkoituksiin. Opinnäytetyössä raportoidaan nauhoitteiden tuloksia niin, että tutkimukseen osallistunutta henkilöä ei voida tunnistaa.

Tutkimuksesta raportoidaan sovellusten toiminnallisuuksien suorittamiseen kulunutta aikaa, joka lasketaan näytön kuvan nauhoitteesta. Tutkimuksesta raportoidaan myös mahdolliset häiriötilanteet, jotka ovat voineet vaikuttaa mittaustuloksiin, kuten esimerkiksi internetyhteyden katkeaminen tutkimuksen aikana tai osallistujan keskittymiseen vaikuttavat äänet.

Tutkija säilyttää nauhoitteista syntyneen aineiston niin, että siihen ei pääse kukaan muu kuin tutkija käsiksi. Opinnäytetyöprosessin päätyttyä, nauhoitteet hävitetään asianmukaisesti.

Osallistuja

Allekirjoittamalla osallistuja on ymmärtänyt edellä mainitun selvityksen tutkimuksen tarkoituksesta ja luonteesta, sekä osallistuu vapaaehtoisesti tutkimukseen. Halutessaan, osallistuja voi keskeyttää tutkimuksen minä hetkenä hyvänsä.

_____ _____

Allekirjoitus Nimenselvennys

_____ Päiväys __/__ 2018

Paikka

Tutkimuksen tekijä

Vakuutan selvittäneeni tutkimukseen osallistuvalle tutkimuksen tarkoituksen, luonteen ja osallistumisen vapaaehtoisuuden.

_____

Allekirjoitus

Matias Ranta
Haaga-Helia ammattikorkeakoulu, Pasila
Ratapihantie 13, 00520 Helsinki
Tietojenkäsittelyn koulutusohjelma
Puhelin **********
Sähköposti **********

Tästä dokumentista on luotu kaksi versiota: yksi tutkimukseen osallistuvalle ja toinen tutkimuksen tekijälle.

Liite 5. Mittaustilaisuudessa osallistujille annetut tehtäväselosteet

Ohjeet osallistujalle

Tehtävänäsi on suorittaa lisäys-, muokkaus- ja poistotoiminnallisuudet Flexer ja Toggl -työaikakirjanpitosovelluksissa. Molemmat sovellukset on avattu selaimessa kahteen paneeliin, jotka löytyvät selaimen osoitepalkin yläpuolelta. Toiminnallisuuksien suorittaminen on jaettu yhdeksään (9) tehtävään, jotka on tarkoitus tehdä ensiksi Flexerissä ja sitten Togglessa. Toteuta tehtävät yksi kerrallaan aloittaen tehtävästä 1 ja päättyen tehtävään 9.

Kun olet suorittanut tehtävät Flexerissä, siirry Toggleen klikkaamalla selaimen oikeanpuoleista paneelia. Koko prosessi päättyy, kun tehtävät 1-9 on suoritettu molemmissa sovelluksissa.

Tehtävä 1: Asiakkaan lisäys

Lisää sovellukseen yksi (1) asiakas nimeltä ”Maija Meikäläinen”.

Tehtävä 2: Projektin lisäys

Lisää sovellukseen yksi (1) projekti. Anna projektille nimeksi ”Auton kunnostus” ja projektin asiakkaaksi ”Maija Meikäläinen”.

Tehtävä 3: Työaikamerkintöjen lisäys

Lisää sovellukseen kaksi (2) työaikamerkintää seuraavilla tiedoilla:

Työaikamerkintä 1	
Syötekenttä	**Syötettävä tieto**
Tehtävän nimi	Eturenkaan vaihto
Päiväys	10.3.2018
Projekti	Auton kunnostus
Tehtävän aloitusaika	12:30
Tehtävän lopetusaika	15:30

Työaikamerkintä 2	
Syötekenttä	**Syötettävä tieto**
Tehtävän nimi	Etupyyhkijöiden vaihto
Päiväys	10.3.2018
Projekti	Auton kunnostus
Tehtävän aloitusaika	16:30
Tehtävän lopetusaika	17:00

Tehtävä 4: Työaikamerkinnän muokkaus

Vaihda ensimmäisen työaikamerkinnän nimeksi ”Takarenkaan vaihto”.

Tehtävä 5: Projektin muokkaus

Vaihda luomasi projektin nimeksi ”Opelin kunnostus”.

Tehtävä 6: Asiakkaan muokkaus

Vaihda luomasi asiakkaan nimeksi ”Matti Meikäläinen”.

Tehtävä 7: Työaikamerkintöjen poisto

Poista molemmat luomasi työaikamerkinnät.

Tehtävä 8: Projektin poisto

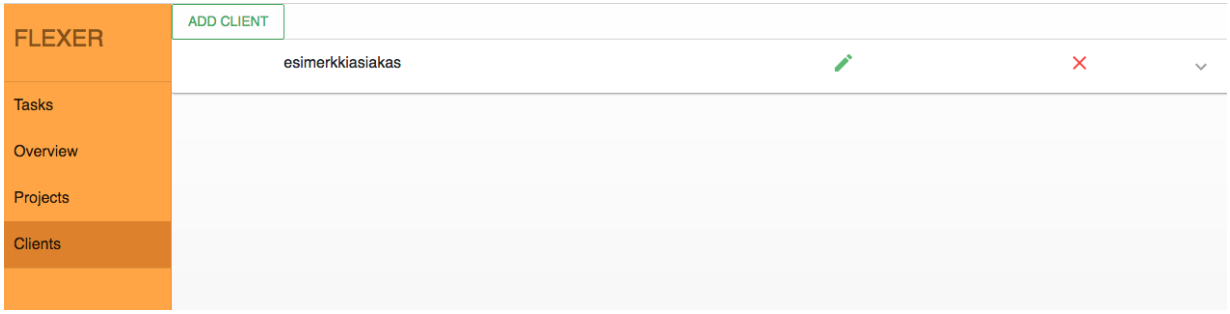
Poista luomasi projekti.

Tehtävä 9: Asiakkaan poisto

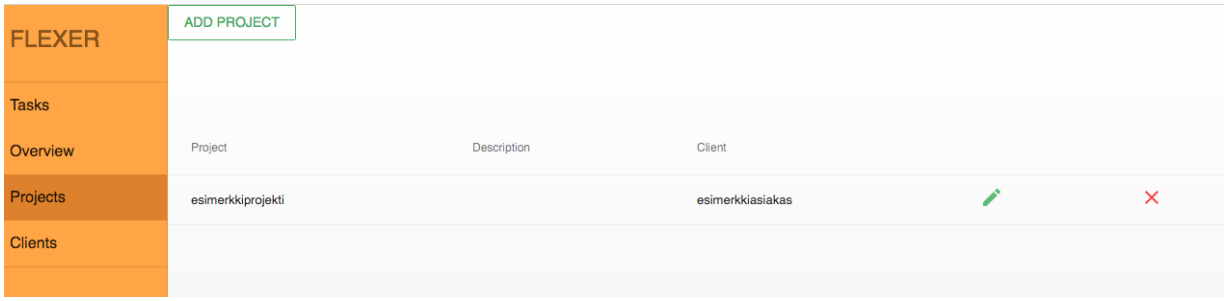
Poista luomasi asiakas.

Liite 6. Kuvakaappauksia Flexer-sovelluksen käyttöliittymästä

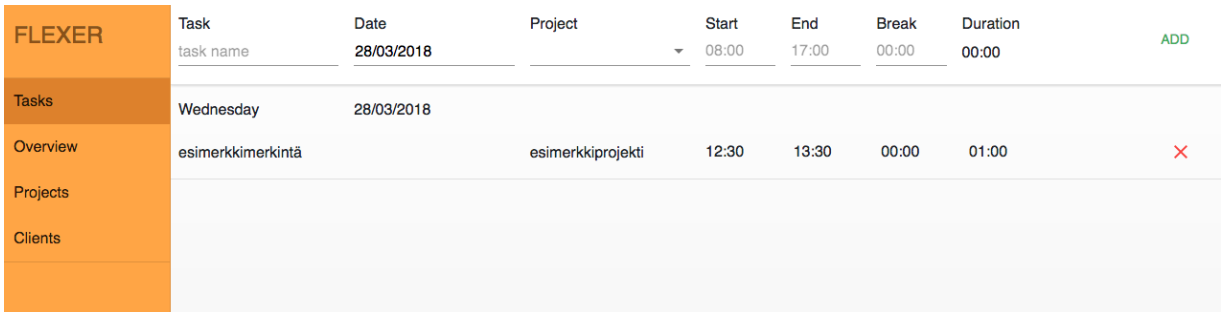
Asiakas-näkymä



Projekti-näkymä

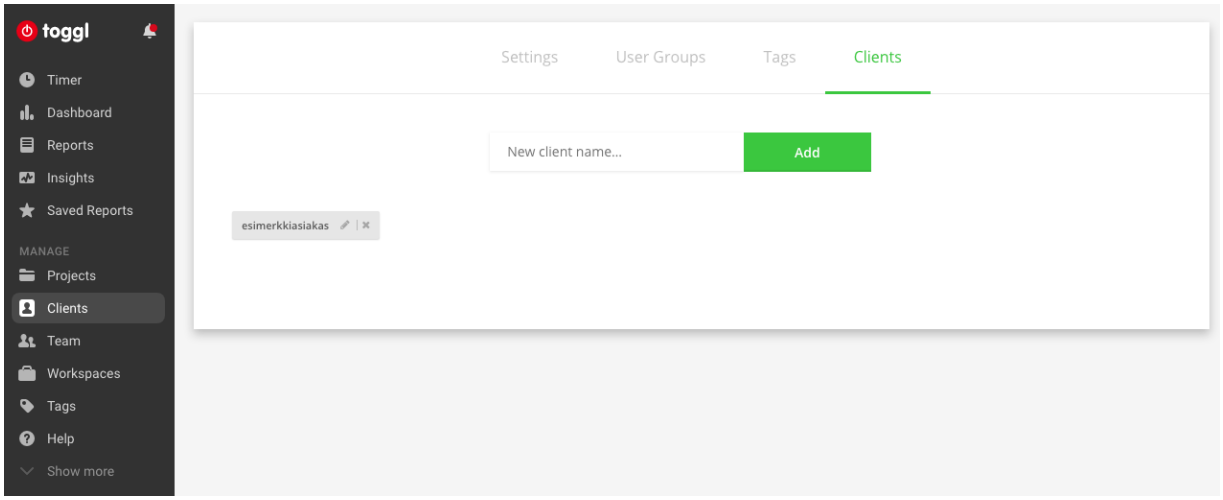


Työaikamerkintä-näkymä

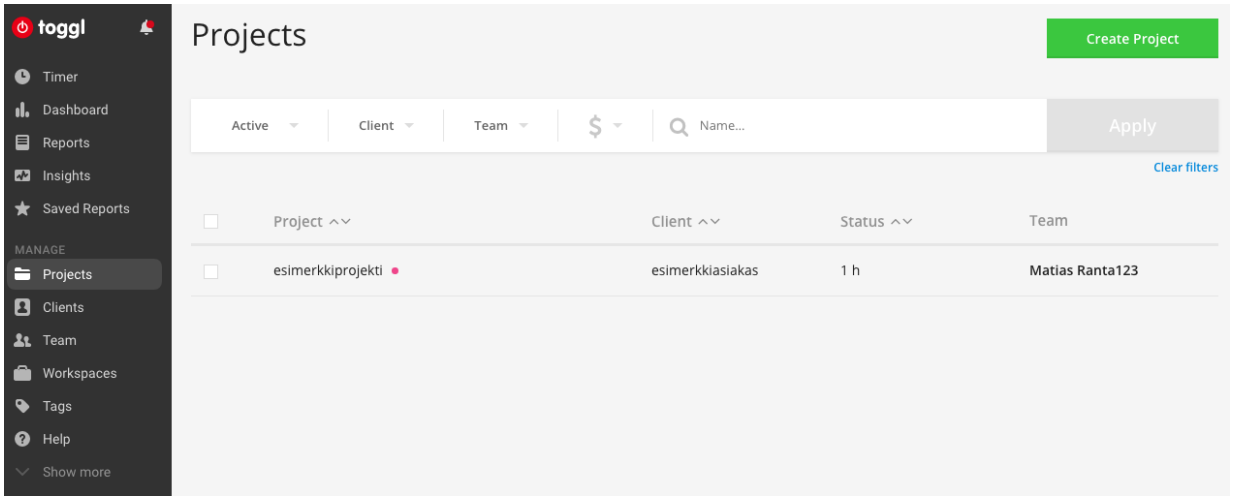


Liite 7. Kuvakaappauksia Toggl-sovelluksen käyttöliittymästä

Asiakas-näkymä



Projekti-näkymä



Työaikamerkintä-näkymä