

Ohjelmistokehitys Angular- ja Symfony-kehyksillä

Vesa Kivistö

Opinnäytetyö

Marraskuu 2017

Tekniikan ja liikenteen ala

Insinööri (AMK), ohjelmistotekniikan tutkinto-ohjelma

Tekijä(t) Kivistö, Vesa	Julkaisun laji Opinnäytetyö, AMK	Päivämäärä marraskuu 2017
	Sivumäärä	Julkaisun kieli Suomi
		Verkojulkaisulupa myönnetty: x
Työn nimi Ohjelmistokehitys Angular- ja Symfony-kehyksillä		
Tutkinto-ohjelma Ohjelmistotekniikan tutkinto-ohjelma		
Työn ohjaaja(t) Rantala, Ari		
Toimeksiantaja(t) Protacon Solutions Oy		
<p>Tiivistelmä</p> <p>Opinnäytetyön tavoitteena oli toteuttaa toimeksiantajan asiakkaalle järjestelmä, jolla käyttäjien onnistui lähettämään asiakkailleen kutsuja erilaisiin tapahtumiin. Järjestelmässä tuli olla toiminnot järjestelmän ylläpitäjille, tapahtumajärjestäjille sekä tapahtuman näytteilleasettajille, ja järjestelmän tuli olla käytettävyyden osalta sujuvaa ja ulkoasultaan moderni ja yksinkertainen. Opinnäytetyön aiheena oli kutsujärjestelmän toteutus, mutta sen rinnalle toteutettiin myös verkkokauppa sekä raportointi- ja autentikointijärjestelmä. Verkkokaupasta tuli voida ostaa lippuja tapahtumiin ilman rekisteröintiä, raportointijärjestelmän tuli koostaa raportteja kutsujärjestelmän käyttöasteista, ja autentikointijärjestelmä vaadittiin kutsujärjestelmän käyttäjien autentikointiin.</p> <p>Työ toteutettiin toimeksiantajan sisällä asiakasprojektina. Järjestelmän eri osien toteuttamisessa käytettiin eri teknologioita: kutsujärjestelmän selainsovelluksissa hyödynnettiin Angular-kehystä ja TypeScript-ohjelmointikieltä, kun taas palvelinsovelluksessa hyödynnettiin Symfony-kehystä ja PHP-ohjelmointikieltä. Projektin kehitysmetodina käytettiin Scrumin ja Kanbanin ominaisuuksia yhdistelevää metodia.</p> <p>Työn tuloksena tuotettiin asiakkaan vaatimista järjestelmistä ensimmäiset julkaisukelpoiset versiot. Toteutetun kutsujärjestelmän selainsovellus kommunikoi palvelinsovelluksen kanssa saumattomasti halliten tapahtumajärjestäjien, tapahtumien ja näytteilleasettajien luonnin sekä kutsujen lähettämisen. Verkkokaupasta voi ostaa lippuja määritettyihin tapahtumiin ja raportointijärjestelmä tuottaa minimivaatimukset täyttäviä raportteja kutsujen lähetyksistä ja käytöistä</p>		
Avainsanat (asiasanat) Angular, Symfony, TypeScript, PHP, ohjelmistokehitys, ketterä ohjelmistokehitys		
Muut tiedot (salassa pidettävät liitteet)		

Author(s) Kivistö, Vesa	Type of publication Bachelor's thesis	Date October 2017
		Language of publication: Finnish
	Number of pages	Permission for web publication: x
Title of publication Software development with Angular and Symfony frameworks		
Degree programme Software Engineering		
Supervisor(s) Rantala, Ari		
Assigned by Protacon Solutions Oy		
<p>Abstract</p> <p>The objective was to develop a system to a customer where the customer could send invitations to their customers for various events. The system had to have functions for system administrators, event organizers and event exhibitors, and the system had to be smooth to use and have a modern and simple layout. The invitation system was the focus, though an online store and reporting and authentication systems were also developed during the project. From the online store customers needed to be able to purchase tickets for events without registration, the reporting system had to generate reports from ticket usage rates and the authentication system was needed for authenticating invitation system's users.</p> <p>The work was carried out within assigner's premises as a customer project. Different technologies were used to implement required systems: the front end for the invitation system was developed with Angular framework and TypeScript programming language, and the back end was developed with Symfony framework and PHP programming language. The method used during the development combined elements from Scrum and Kanban.</p> <p>As a result, first release-ready versions of the required systems were produced. The front end of the invitation system communicates with the back end to handle creating, modifying and deleting event organizers, events and event exhibitors and to send invitations. From the online store users can buy tickets for the event and the reporting system generates reports that meet the minimum requirements.</p>		
Keywords/tags (subjects) Angular, Symfony, TypeScript, wPHP, software development, agile software development		
Miscellaneous (Confidential information)		

Sisältö

Termit.....	5
1 Työn lähtökohdat	6
1.1 Taustaa	6
1.2 Toimeksiantaja	6
2 Tehtävän kuvaus	6
2.1 Ongelma	6
2.2 Tavoitteet	7
2.3 Vaatimusmäärittely	7
2.3.1 Yleistä.....	7
2.3.2 Toiminnalliset vaatimukset.....	8
2.3.3 Ei-toiminnalliset vaatimukset	8
3 Ohjelmistokehitys.....	9
3.1 Yleistä	9
3.2 Menetelmät.....	9
3.2.1 Yleistä.....	9
3.2.2 Vesiputousmalli	10
3.2.3 Spiraalimalli	11
3.3 Ketterät menetelmät.....	11
3.3.1 Yleistä.....	11
3.3.2 Scrum	12
3.3.3 Kanban	12
4 Ohjelmointikehykset	13
4.1 Yleistä	13
4.2 Angular	14

	2
4.2.1 Arkkitehtuuri.....	14
4.2.2 Moduulit	14
4.2.3 Komponentit.....	16
4.2.4 Mallit.....	18
4.2.5 Palvelut	18
4.2.6 Direktiivit	19
4.3 Symfony.....	20
4.3.1 Reititys	20
4.3.2 Kontrollerit.....	20
4.3.3 Mallit.....	21
4.3.4 Palvelut	22
4.4 Doctrine	23
5 Muut teknologiat ja työkalut	25
5.1 Ohjelmointikielet.....	25
5.1.1 TypeScript	25
5.1.2 PHP.....	26
5.2 Ohjelmointiympäristöt	27
5.2.1 Yleistä.....	27
5.2.2 JetBrains PhpStorm.....	27
5.3 Versionhallinta.....	27
5.3.1 Yleistä.....	27
5.3.2 Phabricator	28
5.3.3 Git	28
5.4 Tietokannan hallintajärjestelmät	30
5.4.1 Yleistä.....	30
5.4.2 MariaDB	30

5.4.3	PostgreSQL.....	31
5.5	JIRA	31
5.6	Slack.....	31
6	Toteutus.....	32
6.1	Kutsujärjestelmän selainsovellus	32
6.1.1	Angularin käyttöönotto	32
6.1.2	Käyttöliittymän toteutus	32
6.1.3	Tapahtuma ja tapahtumaan liittyvät toimijat	33
6.1.4	CSV-tuonti.....	34
6.1.5	Kutsujen lähettäminen	35
6.1.6	Kutsujen luku ja tarkistus.....	35
6.1.7	Raportointi.....	36
6.2	Kutsujärjestelmän palvelinsovellus	37
6.2.1	Symfonyn käyttöönotto.....	37
6.2.2	Käyttäjien autentikointi ja autorisointi.....	38
6.2.3	Tapahtuman ja toimijoiden hallinta	39
6.2.4	Kutsujen lähetys ja tarkistus.....	39
6.2.5	Tietokannan hallinta	40
6.2.6	Tietokannan kokonaisrakenne	42
7	Tulokset	43
7.1	Lopullinen ohjelmisto	43
7.2	Testaaminen, viat ja puutteet	44
7.3	Jatkokehitys	44
8	Pohdinta.....	45
8.1	Kehitystyön toimivuus.....	45
8.2	Projektin parissa työskentely	45

8.3 Johtopäätökset	46
--------------------------	----

Lähteet	48
----------------------	-----------

Liitteet	50
-----------------------	-----------

Liite 1. Kuvakaappauksia selainsovelluksesta	50
--	----

Kuviot

Kuvio 1. Angularin kokonaisarkkitehtuuri (Architecture Overview n.d.)	14
Kuvio 2. Esimerkki Angularin moduulista	15
Kuvio 3. Esimerkki Angularin komponentista.	17
Kuvio 4. Esimerkki Angularin mallista.	18
Kuvio 5. Esimerkki Angularin palvelusta.	19
Kuvio 6. Esimerkki Angularin direktiivistä.	19
Kuvio 7. Esimerkki Symfonyn kontrollerista	21
Kuvio 8. Esimerkki Symfonyn HTML-mallista. (Lähde)	22
Kuvio 9. Palvelun käyttö Symfonyssä. (Lähde)	23
Kuvio 10. Symfonyn ja Doctrinen entiteetti-luokka.	24
Kuvio 11. Esimerkki yksinkertaisesta Gitin työkulusta. (Using Branches n.d.)	29
Kuvio 12. Esimerkki Gitin työkulusta branchien kanssa. (Git Merge n.d.)	30
Kuvio 13. Tapahtumajärjestäjien listaus	34
Kuvio 14. CSV-tuonnin virheilmoitus	35
Kuvio 15. Kutsujen lukunäkymä.	36
Kuvio 16. Raportointinäkymä	36
Kuvio 17. Esimerkki Doctrinen migraatiotiedostosta	41
Kuvio 18. Migraatiokomennon suorittaminen	42
Kuvio 19. Tietokannan kokonaisrakenne.	43

Taulukot

Taulukko 1. @NgModule-decoratorin määreet.....	16
Taulukko 2. @Component-decoratorin määreet.....	17

Termit

1 Työn lähtökohdat

1.1 Taustaa

Erilaisten jokapäiväistä elämää helpottavien tietoteknisten laitteiden, kuten älypuhelimien, tablettien ja tietokoneiden, yleistyessä myös erilaiset ohjelmistot ovat yleistyneet. Ohjelmistoista haetaan niin tehokkuutta työtehtäviin työhallinta- ja palkanlaskentajärjestelmien avulla kuin viihdykettä vapaa-aikaan pelien ja erilaisten sosiaalisen median ohjelmistojen kautta. Useimmalla meistä ohjelmistoista on tullut huomattavan suuri osa jokapäiväistä elämäämme, välillä siihen pahemmin huomiota kiinnittämättä.

Saatavilla ja käytettävissä olevat ohjelmistot ovat nykyisin hyvin monipuolisia. Ohjelmistotyyppinä on lukuisia määriä ja niiden käyttötarkoitukset voivat olla toisistaan eroavia. Ohjelmistojen monipuolistuminen myös johtaa ohjelmistokehitystapojen monipuolistumiseen: suosituimpia ja mukautuvimpia ohjelmointikieliä on kymmeniä, erilaiset ohjelmointia helpottamaan pyrkivät ohjelmointikehykset nousevat enemmän esille ja tehokkaampia ja parempia menetelmiä pyritään kehittämään.

1.2 Toimeksiantaja

TODO: Toimeksiantajan esittely

2 Tehtävän kuvaus

2.1 Ongelma

Tapahtumien järjestämiseen suunniteltujen ohjelmistojen on tarkoitus helpottaa tapahtumien organisoimista niin järjestäjän kuin tapahtumissa käyvien näkökulmasta. Järjestelmässä tulisi voida määrittää tapahtumia ja niissä näytteilleasettajina toimivat yritykset. Kutsujen lähetys kutsuvieraille tulisi myös onnistua ongelmitta ja tapahtumaan lippujen oston tulisi olla mahdollista. Tällaisia ohjelmistoja on, mutta usein ne ovat kömpelöitä tai hankalia käyttää ja toiminnoiltaan puutteellisia.

Toimeksiantajan asiakkaalla on huomattu tarve yksinkertaisemmalle ja helppokäyttöisemmälle ohjelmistolle, joka palvelisi kaikkia tapahtuman organisointiin kuuluvia tahoja. Uuden tapahtumien järjestämiseen suunnitellun ohjelmiston tulisi olla aiempia monipuolisempi, mutta kuitenkin sisältää vastaavien ohjelmistojen perustoiminnot, kuten tapahtumajärjestäjien, tapahtumien ja näytteilleasettajien luonnit. Asiakkaalla oli myös tarve verkkokaupalle, josta tapahtumien lippujen oston tulisi onnistua nopeasti ja vaivattomasti.

2.2 Tavoitteet

Opinnäytetyön tavoitteena oli tutkia ohjelmistokehitystä opinnäytetyön tekijälle uusilla teknologioilla. Tutkimusosassa selvitettiin ohjelmistokehityksen eri menetelmiä ja tutustuttiin ohjelmistoalan pinnalla oleviin ohjelmointikehyksiin ja -kieliin, joita myös toimeksiantajan projekteissa käytetään. Tutkimuksessa opittuja asioita hyödynnettiin toteutusosiossa, joka toteutettiin toimeksiantajan asiakasprojektin pohjalta. Opinnäytetyön tavoitteena oli myös tutustua toimeksiantajan toimintamalleihin ja työskentelytapoihin, kehittyä työyhteisön ja projektiryhmän jäsenenä sekä kehittää opinnäytetyön tekijän taitoja ohjelmistoalan keskeisimpien perusteiden parissa.

Toimeksiantajan asiakkaalle tavoitteena asiakasprojektista, jonka pohjalta opinnäytetyön toteutusosa on kirjoitettu, oli saada vaatimukset täyttävä ohjelmisto tapahtumien järjestämiseen, jota voisi myydä palveluna.

2.3 Vaatimusmäärittely

2.3.1 Yleistä

Vaatimusmäärittely (engl. requirements engineering) on yksi ohjelmistokehityksen tärkeimmistä vaiheista. Vaatimusmäärittelyn tarkoituksena on selvittää asiakkaan tarpeet ja vaatimukset kehitettävän ohjelmiston suhteen: mihin ohjelmiston tulee kyetä ja kuinka sen pitää tehtävistään suoriutua. Vaatimusmäärittelystä tehdään tavallisesti oma dokumenttinsa, joka sisältää vaatimuksia niin yleisinä toteamuksina kuin hyvinkin yksityiskohtaisina määritelminä. (Sommerville 2016, 102.)

Vaatimukset voivat olla ohjelmiston loppukäyttäjän näkökulmasta kerrottuja käyttäjävaatimuksia (engl. user requirements), jolloin käsitellään ohjelmiston ominaisuuksia ja kuinka käyttäjä kommunikoi ja toimii ohjelmiston kanssa tai yleisesti järjestelmän näkökulmasta käsiteltyjä järjestelmävaatimuksia (engl. system requirements), jolloin perehdytään tarkemmin ohjelmiston teknilliseen toteutukseen ja kuinka eri toiminnot ja ominaisuudet tulisi implementoida. (Sommerville 2016, 102.)

Vaatimusmäärittely onkin tärkeä tehdä tiiviissä yhteistyössä asiakkaan kanssa. Hyvin toteutettu vaatimusmäärittely on yksi ohjelmistoprojektin onnistumisen takeista, sillä se takaa, että kehittäjä ymmärtää mitä ohjelmistolta vaaditaan ja mihin ja millä tavoin sen odotetaan kykenevän. Huonosti toteutettu vaatimusmäärittely puolestaan voi aiheuttaa ongelmia projektin eri vaiheissa, kuten puutteita suorituskyyvyssä tai puutteellisia tai täysin toteuttamatta jätettyjä toimintoja.

2.3.2 Toiminnalliset vaatimukset

Ohjelmiston toiminnalliset vaatimukset määrittävät, mihin toimintoon ohjelmiston on kyettävä, kuinka ohjelmisto reagoi käyttäjältä tuleviin syötteisiin tai kuinka se käyttäytyy toimintaympäristön tilassa tai kuinka toiminta muuttuu toimintaympäristön tilan muuttuessa. Toiminnalliset vaatimukset tulisi lähtökohtaisesti kirjoittaa kehittäjälähtöisesti: toiminnallisesta vaatimuksesta tulisi siis käydä ilmi syötteet, tulokset ja yksityiskohtainen toiminta. Asiakkaan ei tarvitse ymmärtää toiminnallisten vaatimusten tekstiä täysin, vaan toiminnallisten vaatimusten sisältö tulisi käydä ilmi myös selvemmin kirjoitetuista käyttäjävaatimuksista. (Sommerville 2016, 105).

TODO: Toteutettavan ohjelmiston toiminnallisia vaatimuksia.

2.3.3 Ei-toiminnalliset vaatimukset

Ei-toiminnalliset vaatimukset puolestaan käsittelevät muun muassa ohjelmiston suorituskyykyyn, käytettävyyteen, tietoturvaan, ohjelmiston kehitykseen ja etnisiin puoliin liittyviä vaatimuksia. Ohjelmistossa voidaan esimerkiksi vaatia tietty yhtäjaksoinen käynnissäoloaika tai tietyn tason ylittävä tietoturva. Vaikka ei-toiminnalliset vaatimukset eivät sinällään ota kantaa ohjelmiston toimintoihin, huomataan ei-toiminnallisia vaatimuksia selvittäessä useita toiminnallisiin vaatimuksiin viittaavia kohtia. (Sommerville 2016, 107-108.)

TODO: Toteutettavan ohjelmiston ei-toiminnallisia vaatimuksia.

3 Ohjelmistokehitys

3.1 Yleistä

Nykymaailmassa ohjelmistoja esiintyy kaikkialla. Hallitukset, yhteisöt ja kansalliset ja kansainväliset organisaatiot hyödyntävät ohjelmistoja päivittäisissä toiminnoissaan. Valtioiden infrastruktuurin hallinnoimisessa käytetään ohjelmistoja ja teollisissa tehtaissa käytetään erilaisia ohjelmistoja tuotannon seurannassa ja kontrolloinnissa. (Sommerville 2016, 18.)

Ohjelmistokehitys ei kuitenkaan rajoitu hallinnollisiin tai valtioiden käytössä olevien ohjelmistojen kehittämiseen. Ohjelmistokehitystä esiintyy esimerkiksi myös viihdealalla pelien ja erilaisen videotuotannon muodossa; pelien kehityksessä hyödynnetään ohjelmistoja ja pelit itsekin voidaan lukea ohjelmistoiksi, sekä videotuotannossa erityisesti editoinnissa ja mahdollisten 3D-mallien parissa käytetään ohjelmistoja.

Kehitykseen osallistuminenkaan ei vaadi mitään virallista ohjelmistokehittäjän ammattia. Ohjelmistoja voi kehittää niin ammattilaiset, tutkijat kuin harrastelijat. Ammatillisesta ohjelmistokehityksestä käytetäänkin usein termiä ohjelmistotuotanto (engl. software engineering), sillä ammatillisessa ohjelmistokehityksessä ohjelmistot kehitetään usein toisen tahon tarpeita varten, ja ohjelmistotuotantoa varten on kehitetty sitä tukevia menetelmiä ohjelmistojen määrittämistä ja suunnittelua varten. (Sommerville 2016, 19)

3.2 Menetelmät

3.2.1 Yleistä

Ohjelmistokehitystä varten on kehitetty useita erilaisia menetelmiä, jotka on tarkoitettu auttamaan ohjelmistojen kehitysprosessin kanssa. Näihin menetelmiin lukeutuu muun muassa vesiputousmalli, spiraalimalli sekä useat ketterät kehitysmallit, kuten scrum ja kanban. Ohjelmistoprojektissa valitun menetelmän tarkka seuraaminen voi

olla haastavaa, mutta edes tärkeimpien periaatteiden noudattaminen on hyvä tukipilariksi kehitysprosessille.

3.2.2 Vesiputousmalli

Vesiputousmalli on ensimmäisiä ohjelmistokehitystä varten suunniteltuja menetelmiä perustuen asevoimille kehitettyjen järjestelmien kehitysprosesseihin. Menetelmänä vesiputousmalli on hyvin lineaarinen, sillä kehitys etenee vaihe vaiheelta. Malliin kuuluvat vaiheet ovat osittain ohjelmistokohtaiset ja ne voivat olla esimerkiksi seuraavanlaiset: (Sommerville 2016, 47-48.)

- Vaatimusten määrittely. Selvitetään asiakkaan vaatimukset toteutettavan ohjelmiston suhteen
- Ohjelmiston suunnittelu. Suunnitellaan ohjelmiston tekninen toteutus.
- Toteutus. Varsinainen ohjelmiston toteutusvaihe.
- Integraatio. Integroidaan toteutettu ohjelmisto asiakkaan aiempiin ohjelmistoihin.
- Testaus. Testaan vaadittujen ominaisuuksien toiminta.
- Asennus. Asennetaan ohjelmisto asiakkaan tiloihin.
- Ylläpito. Ohjelmiston ylläpito ja mahdolliset päivitykset.

Vesiputousmallissa on tärkeää suunnitella jokainen vaihe ja niiden sisältö ennen ensimmäisen vaiheen aloittamista. Seuraavaan vaiheeseen ei saa vesiputousmallin määritelmän mukaan siirtyä ennen kuin aiempi vaihe on suoritettu kokonaan loppuun asti, mikä tekee vesiputousmallista suhteellisen huonon valinnan ohjelmistokehitykseen. (Sommerville 2016, 47-48.)

Vesiputousmalli ei salli palata vaiheissa taaksepäin. Vaatimusten määrittelyvaiheen suorittamisen jälkeen määriteltyihin vaatimuksiin ei saa lisätä uusia vaatimuksia, vaan ohjelmisto toteutetaan dokumentoitujen vaatimusten mukaan. Ohjelmistokehityksessä ei kuitenkaan ole ilmiselviä toisistaan eroavia vaiheita, vaan vaiheet ovat päällekkäisiä ja tietoa virtaa vaiheista molempiin suuntiin: toteutusvaiheessa voidaan huomata vaatimus, jota ei oltu huomattu vaatimusmäärittelyssä. Tässä tilanteessa ei kuitenkaan saisi muuttaa alkuperäistä vaatimusmäärittelyä, vaan vaatimus siirrettäisiin jatkokehitysideoihin. Mikäli kehitysvaiheessa taas huomattaisiin ongelma toteutettavan ominaisuuden kohdalla, tulisi ongelma sivuuttaa täysin tai kiertää erilaisella toteutuksella.

3.2.3 Spiraalimalli

Spiraalimallissa ohjelmistokehityksen kulku etenee nimensä mukaisesti spiraalin tavoin. Kehitystä tehdään kierroksittain, joiden sisältö voidaan jakaa karkeasti neljään osaan: (Boehm's Spiral Model n.d.)

- Tavoitteen asettaminen. Määritellään vaiheen tavoitteet, tunnistetaan riskit ja suunnitellaan projektin hallintasuunnitelma.
- Riskien analysointi. Tunnistetut riskit arvioidaan ja analysoidaan sekä tehdään riskienhallintasuunnitelma.
- Kehitys ja validointi. Riskien analysoinnin jälkeen valitaan kehitysmalli kierroksen ajaksi.
- Suunnittelu. Katselmoidaan kierroksen toteutus ja päätetään, jatketaanko kehitystä uudella kierroksella. Mikäli jatketaan, aloitetaan seuraavan kierroksen suunnittelu tavoitteen asettamisesta.

Spiraalimalli on hieman vesiputousmallia joustavampi, sillä se sallii kehityksen pienemmissä ja lyhyemmissä sykleissä. Tiedon kulku on eri vaiheiden aikana kuitenkin rajattua vesiputousmallin tapaan, joten ohjelmistokehitykseen spiraalimallikaan ei välttämättä ole hyvä valinta.

3.3 Ketterät menetelmät

3.3.1 Yleistä

Viime vuosina ohjelmistokehityksessä on huomattu tarve nopeammalle reagointiajalle muuttuvan ympäristön tai markkinoiden aiheuttamien vaatimusten suhteen. Ohjelmistojen kehityksen ja toimituksen halutaan tapahtuvan nopeammin ja erityisesti toimitusta halutaan automatisoida säästäen näin kehittäjiltä aikaa. Näitä tarpeita varten on kehitetty useita ketterän ohjelmistokehityksen menetelmiä.

Ketterä ohjelmistokehitys on iteratiivista, usein 2-4 viikon pituisina sykleinä tapahtuvaa kehitystä. Lyhyet iteraatioajat mahdollistavat nopeamman reagoinnin muuttuviin vaatimuksiin ilman, että koko ohjelmistoprojekti kärsii. Ketterässä ohjelmistokehityksessä usein kootaan kaikki ohjelmiston vaaditut ominaisuudet yhteen paikkaan ja näistä ominaisuuksista valitaan syklin aikana toteutettavat ominaisuudet. Syklin työ määrä tulee suunnitella niin, että valitut ominaisuudet varmasti saadaan toteutettua.

Ketterän ohjelmistokehityksen pyrkimyksenä on myös saada asiakas osallistumaan kehitykseen tiiviimmin. (Sommerville 2016, 73-74.)

3.3.2 Scrum

Scrum on kehitetty tehostamaan ketterän ohjelmistokehityksen resurssien käyttämistä ottamalla käyttöön projektiryhmän jäsenille annettavia rooleja. Näitä rooleja on muun muassa Scrum Master, joka vastaa osittain perinteisemmän projektipäällikön roolia. Kehityksen alussa ohjelmiston vaatimukset ja kehitettävät ominaisuudet kerätään niin sanottuun product backlogiin. Itse varsinainen kehitys tapahtuu parin kolmen viikon pituisina sykleinä, joista käytetään termiä sprint. (Sommerville 2016, 85-86.)

Jokainen sprint alkaa sprintin suunnitteluvaiheella, jossa product backlogista priorisoidaan ja valitaan työn alle otettavat ominaisuudet, jotka kerätään sprint backlogiin. Nämä ominaisuudet tulee valita niin, että niiden toteutus onnistuu sprintille annettussa ajanjaksossa. Sprinteissä projektiryhmän jäsenillä on päivittäisiä lyhyitä tapauksia, scrumeja, joiden aikana käydään läpi projektin tila ja tarvittaessa priorisoidaan tehtävät uudelleen. Scrumin aikana projektiryhmän jäsenten kesken käydään läpi mitä viime tapaamisen jälkeen on tapahtunut, mitä on tehnyt projektissa ja onko eteen tullut mahdollisia ongelmia. Mahdollisten ongelmien ratkaisuun voi esittää ideoita muut projektiryhmän jäsenet. Sprintin lopussa puolestaan pidetään katselmointipalaveri (engl. Sprint review), johon asiakas on hyvä saada osallistumaan. Katselmointipalaverissa asiakkaalle esitellään sprintin tuotokset ja kirjataan ylös asiakkaan palaute toteutetuista ominaisuuksista. Projektiryhmän kesken priorisoidaan product backlogista uusi sprint backlog ja käynnistetään jälleen uusi sprint. (Sommerville 2016, 86-87.)

3.3.3 Kanban

Kanban ei varsinaisesti ole prosessia ohjaava menetelmä, vaan enemmänkin runko ketterään ohjelmistokehitykseen. Kanban ei määrittele kehitykselle tiettyjä iteraatiojaksoja, vaan kehitys on jatkuvaa ja ohjelmiston toimitus voidaan tehdä esimerkiksi

päivittäin. (Kanban n.d.) Scrumiin verrattaessa Kanban on siis joustavampi ja vaatimusten muutoksiin voidaan reagoida heti sen sijaan, että projektiryhmä odottaisi meneillään olevan sprintin loppumista.

Kanbanin tärkein periaate on työn visualisointi. Työvaiheet ja meneillään olevat työt ovat tärkeää olla koko projektiryhmälle esillä. Visualisointiin käytetäänkin niin sanottua Kanban-taulua, jossa on sarakkeet toteutettavien ominaisuuksien eri vaiheille. Vaiheet tulee sopia projektiryhmän kesken, mutta vaiheiksi suositellaan vähintään To Do (Tekemättä), In Progress (Työn alla) ja Done (Toteutettu). Kanbanissa on tärkeää myös rajoittaa eri vaiheissa olevien töiden määrää, joten esimerkiksi In Progress -tilaan ei voida siirtää uutta työtä, jos sen sallittu maksimimäärä on täynnä. Työtä rajoittamalla huomataankin kehityksessä pullonkaulat: kehittäjä jumiutuu työssään johonkin vaiheeseen, jolloin kyseisen vaiheen työmäärä lisääntyy ja ongelma tulee heti ilmi. (Kanban n.d.)

4 Ohjelmointikehykset

4.1 Yleistä

Ohjelmistokehysten tarkoituksena on nopeuttaa uusien ohjelmistojen kehittämistä tarjoamalla kehittäjille valmis runko, joka tarjoaa työkalut ohjelmiston koonnille ja käyttöönotolle. Ohjelmistokehyksissä on myös tavallisesti joukko valmiita yleisimmin tarvittavia toiminnallisuuksia, joten ohjelmistokehittäjällä säästyy aikaa näiden toiminnallisuuksien uudelleen kirjoittamisen sijaan. Ohjelmistokehys voi koostua joukosta eri kirjastoja, apuohjelmia, työkaluja ja ohjelmointirajapintoja. (Software framework 2017.)

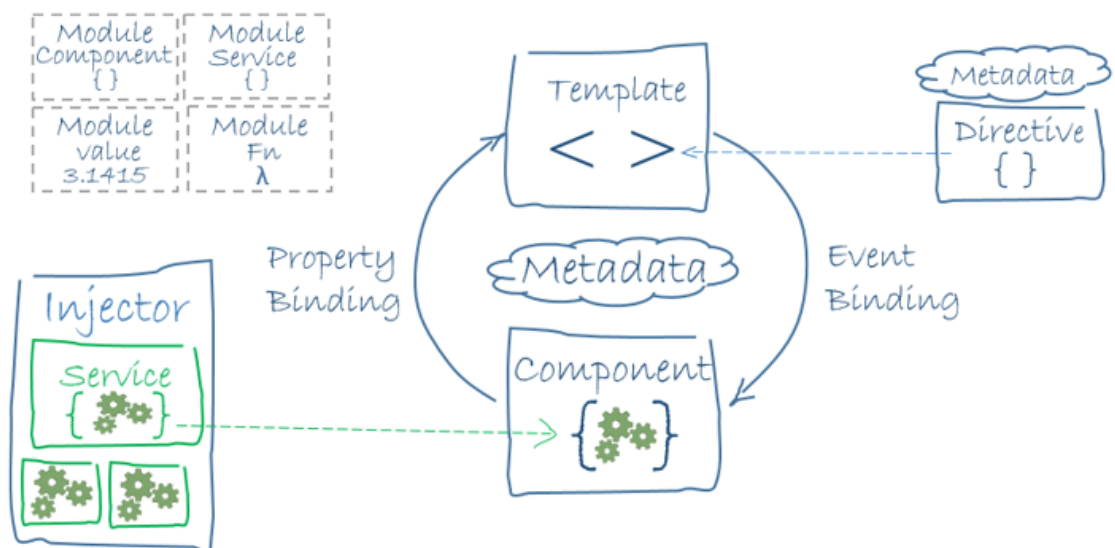
Ohjelmistokehykset toki nopeuttavat ja tietyllä tavalla helpottavat ohjelmistokehittäjien työtä, mutta erityisesti ohjelmistokehyksen käyttöönotossa voi mennä huomattaviakin aikamääriä. Ohjelmistokehyksiin sisältyy usein huomattava määrä valmista koodia, jota ohjelmistokehittäjän on opiskeltava ymmärtääkseen ohjelmistokehyksen toiminnan kunnolla ja osatakseen käyttää sen tarjoamia toiminnallisuuksia tehokkaasti. Käytettävään ohjelmistokehykseen onkin hyvä sitoutua pidemmäksi aikaa kehittäjän opittua ohjelmistokehyksen käyttöä paremmin.

4.2 Angular

Angular on verkkosovellusten kehittämiseen kohdistettu ohjelmointikehys, jonka tarkoitus on nopeuttaa ja helpottaa verkkosovellusten kehittämistä. Yksi Angularin keskeisimmistä tavoitteista kehittämisen nopeuttamisen ja helpottamisen lisäksi on mahdollistaa kehitettyjen sovellusten käyttö niin verkossa kuin mobiili- ja työpöytä-laitteilla. (What is Angular n.d.)

4.2.1 Arkkitehtuuri

Angularissa sovellusten rakenne voidaan jakaa karkeasti viiteen osaan: mallit (engl. template), komponentit (engl. component), palvelut (engl. services), direktiivit (engl. directives) ja moduulit (engl. module). Nämä osat keskustelevat keskenään sidosten (engl. binding) ja riippuvuussuhteiden (engl. dependency injection) välityksellä. (Architecture Overview n.d.) Kuviossa 1 on kuvattu Angularin kokonaisarkkitehtuuri, josta käy edellä mainittujen lisäksi komponenttien ja direktiivien toimintaa ja tarkoitusta täsmentävät metadatat.



Kuvio 1. Angularin kokonaisarkkitehtuuri (Architecture Overview n.d.)

4.2.2 Moduulit

Angularilla sovellukset pyritään kehittämään modulaarisina. Modulaarisuus tarkoittaa, että siihen voi lisätä ja siitä voi poistaa itsenäisiä osia ja kokonaisuuksia, moduu-

leja. Moduulit helpottavat organisoimaan sovelluksen toimintoja omiksi kokonaisuuksiksi ja näin helpottavat sovelluksen ylläpitoa. Angularilla kehitetyissä sovelluksissa on aina vähintään yksi moduuli, niin sanottu juurimoduuli (engl. root module), jonka nimi on tavanomaisesti AppModule. (Architecture Overview n.d.)

Angularissa on oma moduulijärjestelmä NgModules. Tätä ei kuitenkaan tule sekoittaa JavaScriptin moduulijärjestelmään, jota käytetään JavaScriptin moduulien hallinnointiin. Nämä kaksi eivät ole toisiinsa liittyviä, mutta ne täydentävät toisiaan. JavaScriptissä yksittäinen kooditiedosto on moduuli, johon voidaan tuoda muita moduuleja export-avainsanalla ja joka voidaan viedä muihin moduuleihin import-avainsanalla. Sama pätee Angularin moduuleihin, mutta Angularissa moduulin luokka kuitenkin tarvitsee @NgModule-decoratorin, joka varsinaisesti tekee kyseisestä luokasta ja tiedostosta moduulin. (Architecture Overview n.d.)

```
import { NgModule } from '@angular/core';
import { CommonModule } from '@angular/common';

import { ExampleComponent } from './example.component';
import { ExampleDirective } from './example.directive';
import { ExampleService } from './example.service';

@NgModule({
  declarations: [
    ExampleComponent,
    ExampleDirective
  ],
  exports: [
    CommonModule
  ],
  imports: [
    CommonModule
  ],
  providers: [
    ExampleService
  ]
})

export class ExampleModule { }
```

Kuvio 2. Esimerkki Angularin moduulista

@NgModule-decorator sisältää tavallisesti taulukon 1 kaltaiset määreet.

Taulukko 1. @NgModule-decoratorin määreet.

Määre	Merkitys
declarations	Moduuliin kuuluvat näkymäluokat (engl. view class). Näkymäluokkiin lukeutuu komponentit, direktiivit ja putkitukset
exports	Moduulissa määritellyt osat, jotka halutaan näkyviksi ja käytettäviksi muissa moduuleissa
imports	Muut moduulit, joiden ulosvietyjä osia moduuli tarvitsee
providers	Niin sanotut palveluiden rakentajat, providers-määreen sisälle määritellyt palvelut ovat koko sovelluksen käytettävissä
bootstrap	vain juurimoduulissa esiintyvä määre, jonka sisälle määritellään sovelluksen päänäkymäluokka

4.2.3 Komponentit

Sovelluksen eri näkymien hallinnointiin käytetään komponentteja, jotka ovat olio-ohjelmoinnista tuttuja luokkia. Näissä luokissa on määritelty näkymän vaatima toimintalogiikka. Luokasta tulee varsinainen Angularin komponentti vasta @Component-decoratorin käytön jälkeen, mikä liittää luokkaan vaadittuja metadatatietoja. Hyvin toteutetussa modulaarisessa sovelluksessa eri näkymät, kuten navigaatio, sivun alatunniste ja mahdolliset dialogit, on toteutettu omina komponentteinaan. Kehittäjien tulisi pyrkiä toteuttamaan komponentit suorittamaan vain tietty toiminnallisuus, jotta niitä voidaan käyttää uudelleen sovelluksen eri osissa. (Architecture Overview n.d.)

```

import { Component } from '@angular/core';

import { ExampleService } from "../example.service";

@Component({
  selector: 'app-example',
  templateUrl: './example.component.html',
  styleUrls: ['./example.component.scss']
})

export class ExampleComponent {
  public names = [
    'Matti Meikäläinen',
    'Essi Esimerkki',
    'Olli Osallistuja'
  ];

  private someVariable: string;

  /**
   * Component's constructor
   *
   * @param {ExampleService} exampleService
   */
  public constructor(private exampleService: ExampleService) {
    this.exampleService.someMethod();

    this.someVariable = 'Hello!';
  }
}

```

Kuvio 3. Esimerkki Angularin komponentista.

@Component-decorator sisältää tavallisesti taulun 2 kaltaiset määreet.

Taulukko 2. @Component-decoratorin määreet.

Määre	Merkitys
selector	Komponentille luotava yksilöivä valitsin, jota käyttämällä komponentti voidaan liittää jonkin toisen komponentin malliin
templateUrl	Polku komponentin mallitiedostoon
styleUrls	Kokoelma poluista komponentin tyylitiedostoihin
providers	Moduulin decoratorin mukainen palveluiden rakentaja, mutta käytössä vain kyseisessä komponentissa.

4.2.4 Mallit

Angularissa komponentit sisältävät näkymän toimintalogiikan, mutta niillä ei ole tietoa siitä, miten näkymä pitää renderöidä. Renderöinnin ohjeistamisesta vastaa malli, joka on Angularissa HTML-tiedosto, joka normaalin HTML-syntaksin lisäksi tukee Angularin omaa mallisyntaksia. (Architecture Overview n.d.). Angularin oma mallisyntaksi mahdollistaa tiedon sidonnan mallin ja komponentin välillä, toimintojen sidonnan HTML-tapahtumiin, direktiivit silmukoiden ja ehtolauseiden toteuttamiseen sekä muiden mallien liittämisen HTML:stä tutun merkintätavan mukaan (Template Syntax n.d.).

Parhaimmillaan mallisyntaksin tuomat muutokset vähentävät ylläpidettävän koodin määrää, sillä esimerkiksi suurten taulukkojen esitys näkymässä onnistuu muutamalla koodirivillä komponentin ja mallin puolella. Muiden mallien liittäminen (esimerkki kuviossa 4) puolestaan tukee Angularin modulaarisuuden tavoitetta. Toisaalta erityisesti tiedon sidonnan tehokas hyödyntäminen voi olla aloittelijoille hankalaa, vaikka Angular pyrkiikin olemaan helposti ymmärrettävä.

```
<h1>Example Component</h1>

<p>List of names:</p>
<ul>
  ...<li *ngFor="let name of names">
  ...  {{ name }}
  ...</li>
</ul>

<p appWarning>This is a warning!</p>

<app-second-example></app-second-example>
```

Kuvio 4. Esimerkki Angularin mallista.

4.2.5 Palvelut

Uudelleenkäytettävyys on tullut Angularissa esille jo komponenttien ja mallien kohdalla. Palveluita on näiden tapaan lähdetty kehittämään uudelleenkäytettävyys edellä: palvelun on tarkoitus olla uudelleen käytettävä toiminto, ominaisuus tai arvo, jota voidaan hyödyntää missä tahansa sovelluksen osassa. (Architecture Overview n.d.)

Palveluiden pääasiallisena tarkoituksena on hallita yleisimpiä sovelluksen tarvitsemia toimintoja, kuten tiedon hakua palvelimelta, käyttäjän syötteen validointia ja lokitiedostojen kirjoittamista. Näin komponenttien suurimmat toiminnot saadaan jaettua pienempiin kokonaisuuksiin ja komponentin tehtäväksi jää varmistaa sujuva käyttäjäkokemus näkymän ja kaiken logiikan välillä.

```
import { Injectable } from '@angular/core';

@Injectable()
export class ExampleService {

  /**
   * Method to do something.
   */
  public someMethod() {
    // Method implementation
  }
}
```

Kuvio 5. Esimerkki Angularin palvelusta.

4.2.6 Direktiivit

Angularissa on kahdenlaisia direktiivejä: rakennetta ja ominaisuutta muokkaavia direktiivejä. Rakennetta muokkaavat direktiivit, joihin lukeutuu Angularin mallisyntaksin **ngFor* ja **ngIf*, voivat lisätä, poistaa ja muokata näkymän elementtejä. Ominaisuutta muokkaavien direktiivien toiminnot puolestaan kohdistuvat elementin ulkoasuun tai käytökseen. (Architecture Overview n.d.) Esimerkki ominaisuutta muokkaavasta direktiivistä on kuviossa 6, jossa luodaan elementin taustavärin punaiseksi määrittävä direktiivi.

```
import { Directive, ElementRef } from '@angular/core';

@Directive({
  selector: '[appWarning]'
})
export class ExampleDirective {

  /**
   * Directive constructor.
   */
  @param (ElementRef) elementRef
  constructor(private elementRef: ElementRef) {
    this.elementRef.nativeElement.style.backgroundColor = 'red';
  }
}
```

Kuvio 6. Esimerkki Angularin direktiivistä.

4.3 Symfony

Symfony on SensioLabsin kehittämä ohjelmointikehys PHP:lle. Symfonyssä kaikki osat koostuvat omista uudelleenkäytettävistä laajennoksistaan (Symfonyssä termi bundle), joita voi sisällyttää koodiin tarpeen mukaan. Laajennettavuutensa ansiosta Symfony sallii joustavuutta ohjelmointikehityksen käytön suhteen: kehittäjä voi valita käyttävänsä pelkästään Symfonyn tarjoamia laajennoksia tai kirjoittaa suurempia koodikokonaisuuksia itse. (Symfony: six reasons n.d.)

4.3.1 Reititys

Web-kehityksessä selainsovelluksen tulee tietää mistä URL-osoitteesta palvelimelta saa mitään tietoa. Selainsovellus tekee kutsuja palvelimelle käyttäen kunkin kutsun kohdalla tiettyä URL-osoitetta, ja palvelimen tarkoitus on vastaanottaa nämä kutsut. Kutsujen reititystä varten Symfonyssä on oma reitittimensä. (Symfony: routing n.d.)

Reititys tarkoittaa URL-osoitteen kartoittamista tiettyyn kontrolleriin. Symfonyssä reittien luontiin on annettu melko luovat mahdollisuudet, joten reiteistä voi helposti tehdä mieluisensa näköiset. Kehittäjä voi esimerkiksi määrittää Symfonyssä reitin */blogi/teksti/{id}*, jossa *id* on blogitekstin tunniste blogitekstin hakemiseen sen sijaan, että osoite blogitekstiin olisi */blogi.php?teksti_id=1*. (Symfony: routing n.d.) Kuviossa 6 on esimerkki Symfonyn reitityksestä käyttäen *@Route*-määrettä.

4.3.2 Kontrollerit

<https://symfony.com/doc/current/controller.html>

Kontrolleri on funktio, joka vastaanottaa reitityksen perusteella kontrollerille ohjatun selainsovellukselta tulleen kutsun. Kontrolleri siis käsittelee kutsun pyynnön (engl. request) sille kirjoitettujen toimintojen perusteella ja palauttaa selainsovellukselle vastauksen (engl. response). Kontrollerin logiikassa itsessään voi olla tietokannan käsittelyä, sähköpostin lähettämistä tai mitä tahansa muuta mitä kutsun suorittamiseen vaaditaan. *@Route*-määreen, joka reitittää URL-osoitteen kontrolleriin, lisäksi on mahdollista määrittää muun muassa *@Method*-määre, joka määrittää kontrollerille sallitut kutsumetodit. (Symfony: controller n.d.). Esimerkki kontrollerista on kuvattu kuviossa 7.

```

<?php
declare(strict_types=1);

namespace App\Controller;

use Sensio\Bundle\FrameworkExtraBundle\Configuration\Method;
use Sensio\Bundle\FrameworkExtraBundle\Configuration\Route;
use Symfony\Bundle\FrameworkBundle\Controller\Controller;
use Symfony\Component\HttpFoundation\Request;
use Symfony\Component\HttpFoundation\Response;

/**
 * Class ExampleController
 */
/**
 * @Route("/example")
 */
class ExampleController extends Controller
{
    /**
     * @Route("/action")
     * @Method({"GET", "POST"})
     * @param Request $request
     * @return Response
     * @throws \InvalidArgumentException
     */
    public function someAction(Request $request): Response
    {
        // Controller implementation

        return new Response();
    }
}

```

Kuvio 7. Esimerkki Symfony:n kontrollerista.

4.3.3 Mallit

Mikäli Symfonyä käytetään esimerkiksi sivujen generointiin tai sähköpostien lähettämiseen, on hyvä käyttää malleja. Mallit ovat Symfonyssä tekstipohjaisia tiedostoja asettelujen määrittelyyn. Symfony:n mukana toimitetaan Twig-mallinnuskieli, jolla mallitiedostoista saadaan helpommin luettavia. (Symfony: templating)


```

<!DOCTYPE html>
<html>
  <head>
    <title>Welcome to Symfony!</title>
  </head>
  <body>
    <h1>{{ page_title }}</h1>

    <ul id="navigation">
      {% for item in navigation %}
        <li><a href="{{ item.href }}">{{ item.caption }}</a></li>
      {% endfor %}
    </ul>
  </body>
</html>

```

Kuvio 8. Esimerkki Symfonyn HTML-mallista. (Lähde)

Kuviossa 8 on esimerkki HTML-mallitiedostosta hyödyntäen Twig-mallinnuskieltä. `{{ page_title }}` tulostaa `page_title`-muuttujan tiedon sivulle näkyviin. `{% ... %}`-merkintöjen sisään puolestaan voidaan kirjoittaa toimintalogiikkaa, joka suoritetaan sivua luotaessa. Esimerkin tapauksessa käydään läpi `navigation`-taulun sisältö *for*-silmukalla.

4.3.4 Palvelut

https://symfony.com/doc/current/service_container.html

Palvelu Symfonyssä on objekti, jolle on määritelty tiettyjä toiminnallisuuksia ja ominaisuuksia. Tällaisia palveluita on Symfonyssä valmiina monia niin tietokantojen käsittelyyn, sähköpostien lähettämiseen kuin lokitiedostojen kirjoittamiseen. Palvelut säilytetään erillisen objektin, palvelusäiliön (engl. service container) sisällä, josta palvelut ovat haettavissa niiden tunnistetta käyttäen. Esimerkiksi lokitiedostojen kirjoittamiseen tarkoitetun palvelun hakemiseen voi käyttää `$container->get('logger')`; (Symfony: service container n.d.) Kuviossa 9 on esitetty palvelun haku ja käyttö kontrollerin sisällä.

```
// src/AppBundle/Controller/ProductController.php
namespace AppBundle\Controller;

use Symfony\Bundle\FrameworkBundle\Controller\Controller;

class ProductController extends Controller
{
    /**
     * @Route("/products")
     */
    public function listAction()
    {
        $logger = $this->container->get('logger');
        $logger->info('Look! I just used a service');

        // ...
    }
}
```

Kuvio 9. Palvelun käyttö Symfonyssä. (Lähde)

4.4 Doctrine

<https://symfony.com/doc/current/doctrine.html>

Symfonyssä itsessään ei ole tietokantojen käsittelyyn tarkoitettua komponenttia, vaan siihen on sisäänrakennettuna tuki Doctrine-nimiseen kirjastoon. Doctrine on ORM (Object-relational mapping) -kirjasto, jonka tavoitteena on tarjota tehokkaat työkalut tietokantojen hallintaan ja helpottaa tietokantojen käsittelyä. Symfonyssä integraatio Doctrineen on toteutettu pääosin käyttäen entiteettiluokkia, jotka ovat olio-ohjelmoinnin luokkien muotoon kirjoitettu tietokannan taulu ja sen kentät. (Symfony: Doctrine n.d.)

```

<?php
declare(strict_types = 1);

namespace App\Entity;

use Doctrine\ORM\Mapping as ORM;

/**
 * @ORM\Entity
 * @ORM\Table(name="example_entity")
 */
class ExampleEntity
{
    /**
     * @ORM\Column(name="id", type="integer",)
     * @ORM\Id()
     * @ORM\GeneratedValue(strategy="AUTO")
     */
    private $id;

    /**
     * @ORM\Column(name="name", type="string", length=50)
     */
    private $name;

    /**
     * @return int
     */
    public function getId(): int
    {
        return $this->id;
    }

    /**
     * @return string
     */
    public function getName(): string
    {
        return $this->name;
    }

    /**
     * @param string $name
     */
    public function setName(string $name)
    {
        $this->name = $name;
    }
}

```

Kuvio 10. Symfonyn ja Doctrinen entiteettiluokka.

Yksinkertaisen entiteettiluokan sisältö on kuvattu kuviossa 9. Luokan nimeksi määritellään taulun nimi ja kommentteiksi laitetaan `@ORM\Entity`-määre, joka merkitsee luokan entiteettiluokaksi ja `@ORM\Table(name="taulun_nimi")`-määre, joka ilmoittaa taulun nimen tietokannassa. Taulun jokainen kenttä määritellään luokakohtaisina muuttujina, joille annetaan kommentteissa `@ORM\Column`-määre, joka määrittelee kentän nimen, tyypin ja tarvittaessa pituuden tai muun määreen. Tietokannassa yksilöivänä tunnisteena toimivalle kentälle annetaan lisäksi

@ORM\Id()-määre. Lopuksi määritellään *get*- ja *set* -komennot kentille arvojen hakemiseen ja asettamiseen. (Symfony: Doctrine n.d.)

5 Muut teknologiat ja työkalut

5.1 Ohjelmointikielet

5.1.1 TypeScript

TypeScript on Microsoftin kehittämä JavaScriptiin perustuva avoimen lähdekoodin ohjelmointikieli. TypeScript on erityisesti suurten ohjelmistojen kehitykseen suunniteltu ohjelmointikieli ja syntaksiltaan ja semantiikaltaan JavaScriptin kanssa identtinen. TypeScriptin kehityksessä on pyritty yhteensopivuuteen JavaScriptin kanssa ja tämän vuoksi se kääntyy JavaScriptiksi omalla kääntäjällä. TypeScriptillä kirjoitetut ohjelmistot ovat siis valideja JavaScript-ohjelmistoja ja ohjelmistoissa onkin mahdollista yhdistää kokonaan JavaScriptillä ja TypeScriptillä kirjoitettuja koodinpätkiä. (TypeScript 2017.)

Tyypitykset

TypeScriptin tärkeimpiin ominaisuuksiin lukeutuu vapaavalintainen mahdollisuus koodin staattiseen tyypitykseen. Staattinen tyypitys tarkoittaa, että ohjelmiston koodissa annetaan esimerkiksi muuttujalle tai metodin palautusarvolle jokin tietty tyyppi ja koodin koontivaiheessa tyypitykset tarkistetaan ja testataan. Staattinen tyypitys ei lähtökohtaisesti vaikuta itse ohjelmiston suorittamiseen, vaan se tekee koodista turvallisempaa ja parantaa koodin luettavuutta. TypeScriptin staattisesta tyypityksestä poiketen JavaScriptissä on dynaaminen tyypitys, jossa esimerkiksi muuttujien tyyppi päätetään vasta ohjelmistoa ajettaessa. Dynaamisessa tyypityksessä on mahdollisuus tyyppivirheisiin etenkin laajojen ohjelmistojen kohdalla. (TypeScript 2017.)

Luokat

JavaScript ei virallisesti tue luokkia, vaan luokkien määrittelyt sisältyvät vasta ECMAScript 2015:n mukanaan tuomiin uudistuksiin. TypeScript kuitenkin sisällyttää omiin määrittelyksiinsä ECMAScript 2015:n uudistukset mukaan lukien luokat. Luokat tuovat TypeScriptiin olio-ohjelmoinnin ominaisuuksia parantaen kirjoitetun koodin

uudelleenkäytettävyyttä ja luettavuutta, ja ne myös tukevat TypeScriptin staattista tyyppitystä. Luokat ovat lisäksi Angularissa tärkeitä, sillä Angularin modulaarisuus rakentuu luokkien ja niille määritettävien annotaatioiden ympärille. (TypeScript 2017.)

5.1.2 PHP

PHP on laajasti käytetty, erityisesti web-kehitykseen suunniteltu ohjelmointikieli. PHP:n pääasiallisena tavoitteena on ollut tarjota kehittäjille työkalut dynaamisten verkkosivujen nopeaan kehittämiseen. (PHP Intro n.d.)

PHP tukee suosionsa vuoksi yleisimpiä käyttöjärjestelmiä. Ohjelmistojen kehittäminen PHP:lla onnistuu siis Windowsilla, Macilla ja Linuxilla. Vaikka PHP on pääasiallisesti web-kehitykseen suunniteltu ohjelmointikieli, ei sen käyttö ole rajoitettu pelkästään tähän. PHP:n käyttökohteet voidaankin jakaa kolmeen eri ryhmään. (PHP Intro n.d.)

Palvelinskripta

Palvelinskripta on perinteisin tapa hyödyntää PHP:ta. Palvelinskriptauksessa keskittään palvelimen ja verkkoselaimen väliseen keskusteluun, ja itse verkkosivulla PHP-koodi näytetään tavallisesti HTML-muotoisena merkistönä. (PHP: What Can Do n.d.)

Komentoriviskripta

PHP:lla kirjoitettua koodia voidaan ajaa myös ilman varsinaista palvelinta. Tällöin vaatimuksena on PHP:ta tukeva komentorivi, jossa on PHP-tuen lisäksi asennettuna PHP:n jäsentäjä. Komentorivillä ajettavat PHP-skriptit ovat tavallisesti pienempiä varsinaisiin ohjelmistoihin verrattuina, ja niiden toimintoja voi olla esimerkiksi tietokannan päivitykset. (PHP: What Can Do n.d.)

Työpöytäsovellusten kehittäminen

PHP:n web-kehityksen painotteisuus ei estä työpöytäsovellusten kehittämistä. Erityisesti graafisen käyttöliittymän sisältävien työpöytäsovellusten kehittämiseen PHP ei kuitenkaan ole paras mahdollinen ohjelmointikieli, sillä graafisten ulkoasujen tekeminen PHP:lla on haastavaa eikä sitä ei ole suunniteltu näiden kehittämistä varten. Kuitenkin laajan käyttöliittymätuen vuoksi myös työpöytäsovelluksista on mahdollista saada kehitettyä järjestelmäriippumattomia. (PHP: What Can Do n.d.)

5.2 Ohjelmointiympäristöt

5.2.1 Yleistä

Ohjelmointiympäristö on ohjelmisto, joka tarjoaa ohjelmistokehittäjille tarpeellisia ominaisuuksia. Lähdekoodieditori ja koonti- ja debuggaustyökalut ovat ohjelmointiympäristöjen tavallisimpia ominaisuuksia. Useimpiin kehittyneimpiin ohjelmointiympäristöihin lukeutuu lisäksi tuki koodin generoinnille, integroidulle versionhallintajärjestelmätuelle, selainnäkyimiä esimerkiksi luokkien tarkasteluun ja tuki laajennoksille. (IDE 2017.)

5.2.2 JetBrains PhpStorm

PhpStorm on JetBrainsin kehittämä maksullinen ohjelmointiympäristö. PhpStorm on nimensä mukaisesti erityisesti PHP-kehitykseen ja tietokantojen käsittelyyn tarkoitettu ohjelmointiympäristö, mutta sen sisältäessä JetBrainsin toisen ohjelmointiympäristön, WebStormin, ominaisuudet se sopii hyvin myös HTML:n ja CSS:n kirjoittamiseen sekä JavaScriptillä ja TypeScriptillä kirjoitettavien ohjelmistojen kehitykseen. (PhpStorm 2017.)

PhpStorm tukee muun muassa koodin analysointia, virheiden korjausta korjausedustusten muodossa sekä koodin generointia. PhpStormin ominaisuuksia voi laajentaa sille kehitetyillä laajennusosilla, joita käyttäjät voivat kehittää itse omia tarpeitaan vastaaviksi. Esimerkki tällaisesta laajennuksista on esimerkiksi tuki Symfony-ohjelmointikehykselle.

TODO: Käyttö toimeksiantajalla

5.3 Versionhallinta

5.3.1 Yleistä

Versionhallinta on nykyaikaisen ohjelmistokehityksen yksi tärkeimmistä osista. Versionhallinnan avulla ohjelmiston koodin historiaa voidaan seurata ja siihen tehdyt muutokset ovat aina saatavilla. Tämä puolestaan auttaa virheiden ja ongelmien seu-

rannassa ja korjaamisessa. Versionhallinta myös mahdollistaa ohjelmistosta julkaistujen versioiden säilyttämisen ja niiden tarjoamisen, sekä projektiryhmän uusille työntekijöille se antaa mahdollisuuden tutustua koodiin tarkemmin.

5.3.2 Phabricator

Phabricator on Phacility-nimisen yrityksen kehittämä ilmainen palvelu ohjelmistokehitykseen. Phabricatoriin sisältyy työkalut koodin katselmointiin, repositoryjen selailmiseen, virheiden seurantaan ja wikin ylläpitoon. Phabricator tukee Gitiä, Mercurialia ja Subversionia. (Phabricator 2017.)

TODO: Käyttö toimeksiantajalla

5.3.3 Git

Nykyisin suosituin ja laajimmin käytössä oleva alustariippumaton versionhallintajärjestelmä, Gitin kehitti alkujaan Linus Torvalds vuonna 2005. Git on aktiivisen kehityksen alla ja se perustuu avoimeen lähdekoodiin, joten kuka tahansa voi käyttää Gitiä projekteissansa veloituksetta. (What is Git n.d.)

Hajautettu arkkitehtuuri

Gitin vahvuuksiin versionhallintajärjestelmänä kuuluu sen hajautettu arkkitehtuuri. Sen sijaan, että ohjelmiston koodi ja sen historia sijaitsisi vain yhdessä paikkaa, Gitiä käytettäessä jokainen kopio koodista sisältää kaikki koodiin tehdyt muutokset ja toimii omana repositorynään. Hajautettu arkkitehtuuri mahdollistaa organisaation sisällä saman koodin säilytyksen useammalla eri työpisteellä. (What is Git n.d.)

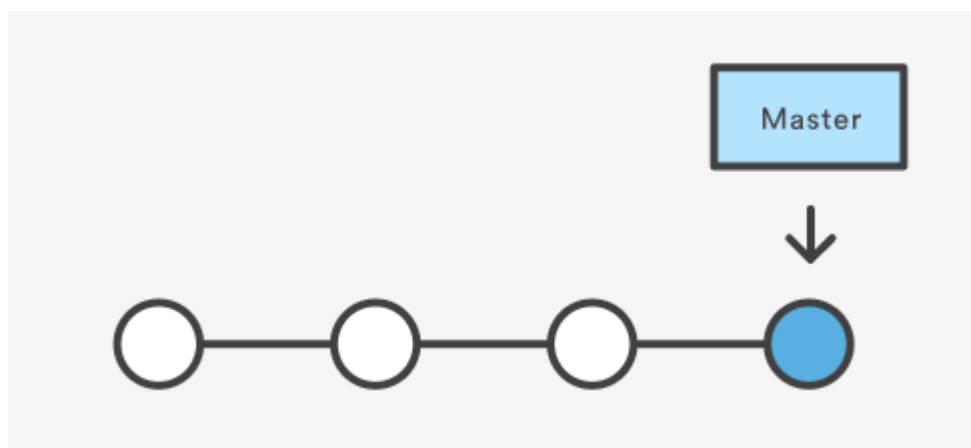
Branching-toiminto

Toinen vahvuus on branching-toiminto (suom. haaraus). Branching-toiminnossa ohjelmiston koodista luodaan oma itsenäinen kopio, branch (suom. haara). Branching-toiminto on erityisesti suurissa organisaatioissa ja suurten projektien kehityksessä hyödyllinen, sillä se mahdollistaa useamman ominaisuuden ja muutoksen kehittämisen samanaikaisesti: organisaatio voi esimerkiksi kehittää ohjelmiston uutta versiota yhdessä branchissa ja tehdä korjauksia vanhempaan versioon toisessa bran-

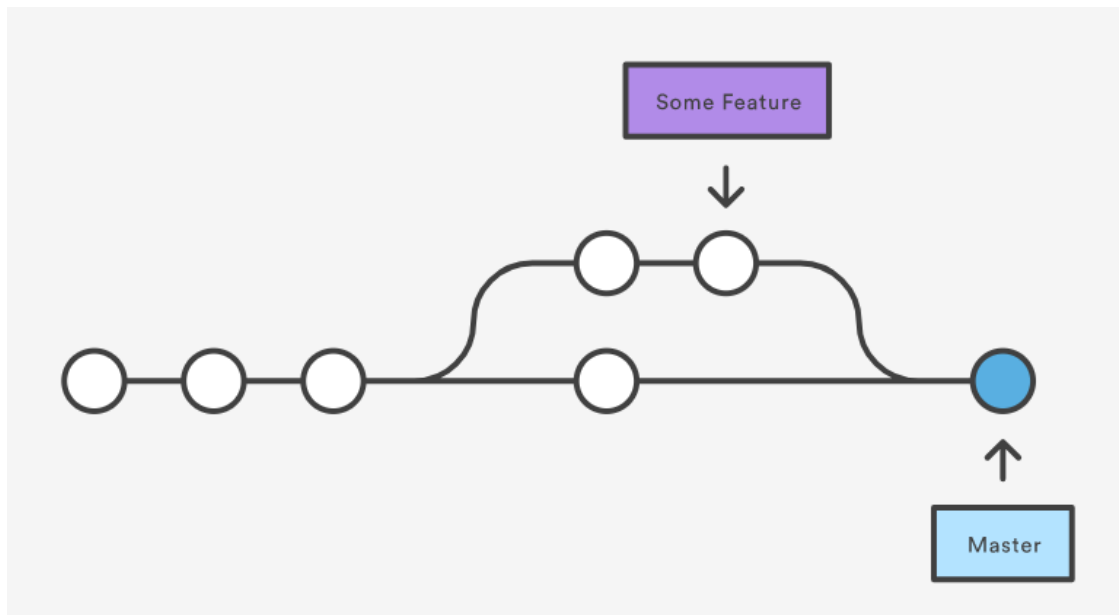
chissa. Tavallisesti useamman branchin projekteissa löytyy master branch (suom. alkuperäishaara), johon tehdään committeja muista brancheista vasta niiden katselmoinnin ja toimiviksi toteamisen jälkeen. (What is Git n.d.)

Joustavuus

Gitin saaman suosion vuoksi Git tukee useampaa olemassa olevaa käyttöjärjestelmää ja protokollaa. Git myös tarjoaa useamman mallin työnkululle; Gitiä työssään hyödyntävät eivät siis ole sidottuina tietyn malliseen työnkulkuun. Etenkin pienissä projekteissa työnkulku voi olla kuvion 1 tapaan yksinkertainen, jossa commitit tehdään suoraan master branchiin. Tämä malli ei kuitenkaan toimi useamman kehittäjän projekteissa, sillä kirjoitettujen koodien yhdistäminen voi olla haastavaa. Suuremmissa projekteissa branchien tekeminen onkin tyypillisempi ratkaisu (kuvio 2). Näistä brancheista tehdään tavallisesti merge request master branchiin, ja ennen merge requestin hyväksymistä ne katselmoidaan ja testataan toimiviksi. (What is Git n.d.)



Kuvio 11. Esimerkki yksinkertaisesta Gitin työnkulusta. (Using Branches n.d.)



Kuvio 12. Esimerkki Gitin työkalusta branchien kanssa. (Git Merge n.d.)

TODO: Käyttö toimeksiantajalla

5.4 Tietokannan hallintajärjestelmät

5.4.1 Yleistä

Tietokannan hallintajärjestelmä (engl. database management system, DBMS) on ohjelmisto, jota käytetään käyttäjän ja tietokannan väliseen kommunikointiin. Tietokannan hallintajärjestelmän tavoitteena on helpottaa ja nopeuttaa tietokannassa olevan datan hallintaa. Moderneissa tietokantojen hallintajärjestelmissä pyritään mahdollisimman realistiseen datan esitykseen hyödyntämällä reaaliaikailmaa: esimerkiksi yrityksen asiakasrekisteri voi koostua asiakas-nimisestä taulusta, jonka ominaisuuksina on yleiset osoitetiedot. (DBMS Overview n.d.)

5.4.2 MariaDB

MariaDB on yksi suosituimmista avoimeen lähdekoodiin perustuvista relaatiotietokantojen hallintajärjestelmistä, jonka kehittivät alun perin osa MySQL:n kehittäjistä. MariaDB:n tavoitteena on olla MySQL:n edistyneempi versio säilyttäen kuitenkin yhteensopivuuden MySQL:n kanssa. Nykyisin MariaDB:n kehitys on hyvin yhteisöpainotteista, mutta kehityksen tukemista varten on perustettu MariaDB Foundation. (About MariaDB n.d.)

TODO: Merkitys projektissa

5.4.3 PostgreSQL

PostgreSQL on avoimeen lähdekoodiin perustuva oliorelaatietietokantojen hallintajärjestelmä. PostgreSQL on erityisesti suurehkojen yritysten käyttöön suunnattu hallintajärjestelmä ja sen ominaisuuksiin kuuluu muun muassa kustomoitavien tietotyyppien luonti, mikä helpottaa olioiden muuttamista relaatiomuotoiseksi dataksi. PostgreSQL on myös hyvin skaalautuva niin käsiteltävän datan määrän kuin yhtäaikaisten käyttäjien suhteen. (About PostgreSQL n.d.)

TODO: Merkitys projektissa

5.5 JIRA

Jira on Atlassianin kehittämä maksullinen tehtävienhallintaohjelmisto, joka tarjoaa työkalut muun muassa virheiden seurantaan, tehtävien määrittelyyn ja projektinhallintaan. Jirassa tehtävistä käytetään termiä issue (suom. ongelma). Jira on erityisesti ketterän ohjelmistokehityksen parissa käytetty ohjelmisto. (Jira (software) 2017.)

TODO: Käyttö toimeksiantajalla

5.6 Slack

Slack on Slack Technologies -nimisen yrityksen kehittämä pilvipohjainen ryhmätyösovelluksia ja -palveluja sisältävä kokonaisuus. Slack tarjoaa pysyviä keskusteluhuoneita, joista Slackin sisällä käytetään termiä kanava (engl. channel). Slackin käyttäjä voi kuulua useampaan ryhmään (Slackissa team, suom. ryhmä) ja jokaisella ryhmällä voi olla useampi keskusteluhuone. (Slack (software) 2017.)

TODO: Käyttö toimeksiantajalla

6 Toteutus

6.1 Kutsujärjestelmän selainsovellus

6.1.1 Angularin käyttöönotto

Toimiakseen Angularin vaatii Node.js:n ja npm:n asennuksen. Nämä molemmat saa ladattua ja asennettua Node.js:n kotisivuilta löytyvien ohjeiden mukaisesti. Node.js:n ja npm:n asennuksen jälkeen tulee asentaa Angular CLI (command-line interface), jolla saadaan luotua ja ajettua Angular-projekteja. Angular CLI asennetaan npm:n avulla seuraavalla komennolla:

```
npm install -g @angular/cli
```

Tämä komento lataa ja asentaa Angular CLI:n tarvittavine paketteineen. Asennuksen jälkeen uuden Angular-projektin luonti onnistuu komennolla

```
ng new projekti-nimi
```

Komennon alussa *ng* on Angular CLI:n mukana asennetun komennon TODO, *new* viittaa uuden projektin luontiin ja kolmas määre on projektille haluttu nimi. Uuden projektin luonnissa kestää aikansa, sillä komennon aikana asennetaan kaikki Angular-projektin vakiopaketit.

Projektin luonnin jälkeen projekti voidaan ajaa komennolla

```
ng serve
```

Tällä komennolla projekti kootaan ja käynnistetään vakiona URL-osoitteessa <http://localhost:4200/>. *ng serve* jää koonnin ja käynnistämisen jälkeen seuraamaan projektin tiedostoihin tehtäviä muutoksia ja muutoksen huomattessa ajaa projektin koonnin uudestaan.

6.1.2 Käyttöliittymän toteutus

Selainsovelluksen käyttöliittymän toteutuksessa hyödynnettiin Angular Material2 -kirjastoa. Kirjasto rakentuu Googlen Material Design -määritysten ympärille sovittaen

niihin kuuluvat komponentin Angulariin sopiviksi ja pyrkien mahdollisimman korkealaatuiseen lopputulokseen. Kirjaston kehitys on vielä vaiheessa ja virallisten GitHub-sivujen mukaan kirjaston kehityksessä on meneillään beta-vaihe.

Käyttöliittymään toteutettiin omat näkymät tapahtumajärjestäjien, tapahtumien, näytteilleasettajien ja kutsuttavien listaukseen. Näiden lisäksi toteutettiin näkymä kutsujen sisään lukuun ja raportointiin sekä uudelleenkäytettäviä dialogeja CSV-tiedostojen tuontiin ja kutsujen ulkoasun esikatseluun.

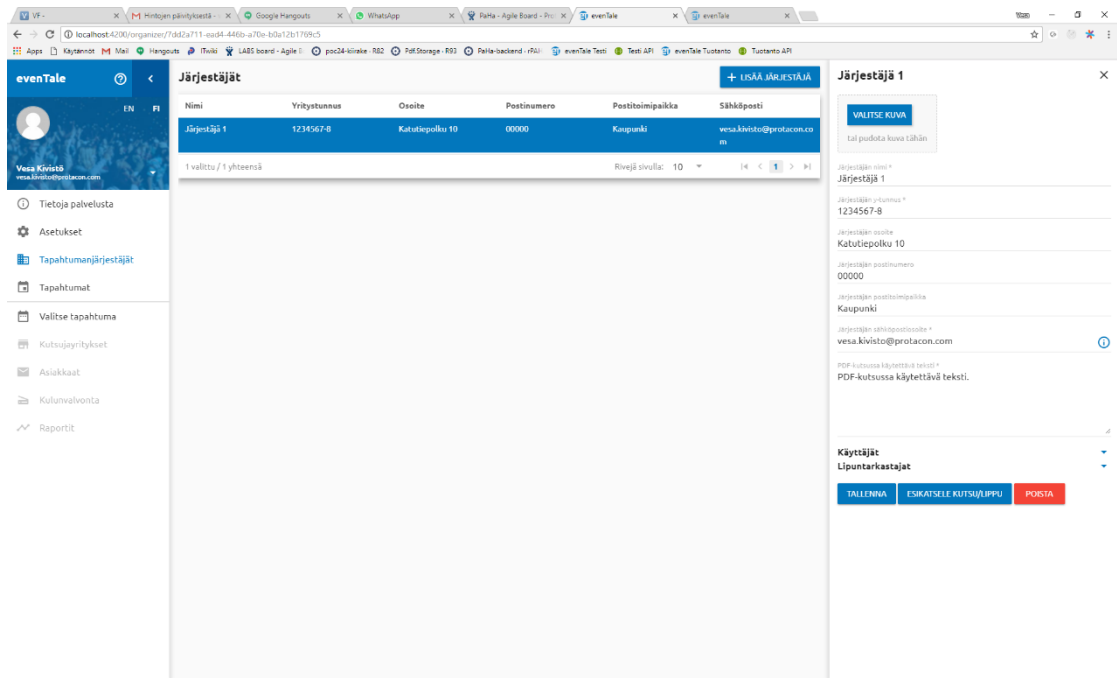
Koska käyttöliittymän ulkoasu rakennettiin vahvasti käytettävän kirjaston komponentteja hyödyntämällä, ei ulkoasusta tullut kovinkaan omaperäinen. Yleiset komponentit, kuten painikkeet, tehtiin lähestulkoon kokonaan kirjaston omilla elementeillä ja niiden ominaisuuksilla lisäten vain vähän omia tyylimääriä. Ohjelmistosta kehitettiin lisäksi vasta ensimmäistä julkaisukelpoista versiota, joten omaperäisyyttä ei varsinaisesti haettu vielä tässä vaiheessa.

Angular Material2:n ollessa vielä kehityksessä ei siihen sisälly kaikki mahdolliset komponentit. Käyttöliittymää toteutettaessa muun muassa taulukkokomponentti puuttui kokonaan, joten tämän korvaajaksi jouduttiin valita kolmannen osapuolen tarjoama komponentti. Valittu komponentti kuitenkin saatiin onnistuneesti liitettyä osaksi muita komponentteja käyttäen kirjaston tarjoamia tyylimääriä säästyen näin suuremmilta ulkonäöllisiltä eroavaisuuksilta muihin komponentteihin.

6.1.3 Tapahtuma ja tapahtumaan liittyvät toimijat

Tapahtumien järjestäminen ei onnistu ilman tapahtumajärjestäjää, eikä tapahtumiin tule ketään, ellei niissä ole näytteilleasettajia esillä. Yleiskäyttöiseen tapahtumien hallintaohjelmistoon kuuluu siis tapahtumajärjestäjien, tapahtumien ja näytteilleasettajien luontiin ja hallintaan liittyvät toiminnot. Ja koska kyseessä on kutsuohjelmisto, piti näytteilleasettajille toteuttaa toiminnot myös asiakkaiden kutsumiseen tapahtumiin.

Koska yllä mainituista toiminnosta tuli toimintalogiikaltaan ja käyttöliittymiltään hyvin samankaltaisia, käyn seuraavaksi läpi luontiprosessin vain tapahtumajärjestäjän kannalta.



Kuvio 13. Tapahtumajärjestäjien listaus

Tapahtumajärjestäjän luonti onnistuu lomakkeella, jossa käyttäjää pyydetään syöttämään tapahtumajärjestäjän tietoja. Osa näistä tiedoista, kuten nimi ja y-tunnus, ovat pakollisia eikä luonti onnistu ilman pakollisten tietojen syöttämistä. Tallenna-painikkeen painamisen jälkeen selainsovellus tekee kutsun palvelinsovelluksen kontrolleriin, joka varsinaisesti luo tapahtumajärjestäjän. Luonnin jälkeen selainsovellus palauttaa luodun järjestäjän tiedot vastauksena ja selainsovelluksessa listaus päivitetään.

TODO: Koodinpätkää, luontikutsun tekeminen selainsovellukseen ja listauksen päivittäminen

6.1.4 CSV-tuonti

Ohjelmistoon kehitettiin myös mahdollisuus luoda näytteilleasettajia ja kutsuttavia CSV-tiedoston tuonnilla. Tuontia varten luotiin oma dialogi, jossa voidaan valita tuotava tiedosto. Tiedoston valitsemisen ja ohjelmistoon lataamisen jälkeen tiedoston sisältämät rivit tarkistetaan ja virheelliset rivit näytetään käyttäjälle korjattaviksi.

Tuo asiakkaat CSV:nä

0 / 1

KÄSITTELE TIEDOSTO

Tiedoston sisällössä virheitä, ole ystävällinen ja korjaa nämä virheet.

Etunimi	Sukunimi	Sähköposti	Puhelinnumero	Yritys	Titteli	Asiakas nro	Kutsujen lukum...
Matti	Meikäläinen	osoite@domainfi	401234567	Yritys	Titteli	A123	1

Virheellinen sähköpostiosoite

PERUUTA

Kuvio 14. CSV-tuonnin virheilmoitus.

TODO: Kuvio CSV-tuonnista. Koodinpätkä?

Toiminnallisuutena CSV-tuonnista tuli suhteellisen yksinkertainen, mutta käyttäjän ohjeistamisen ja virheentarkistuksen vuoksi se on helppo käyttää. Toteuttamisen aikana ongelmaksi muodostui muun muassa tuotavien tiedostojen merkistökoodaus, sillä tietyillä merkistökoodauksilla ääkköset eivät toimineet oikein. Ongelmaa koitettiin selvittää, mutta lopulliseksi ratkaisuksi löydettiin vain käyttäjän ohjeistus käyttämään toimivaksi todettua merkistökoodausta. Ratkaisu ei ollut paras mahdollinen, sillä se vaatii käyttäjältä osaamista tiedoston tallentamiseen tietyllä merkistökoodauksella.

6.1.5 Kutsujen lähettäminen

Kutsujen lähettämisen tuli onnistua ohjelmistosta, mikäli kutsuttavalle oli määriteltä sähköpostiosoite. Kutsujen lähettämiseen toteutettiin oma dialoginsa, jossa käyttäjä syöttää lähetettävälle kutsuille saatetekstin. Selainsovelluksen puolella ei kutsujen lähettämisen yhteydessä tehdä kovin paljon, vaan lähettämiseen liittyvä toiminnallisuus tapahtuu enemmän palvelinsovelluksen puolella.

TODO: Kuvio näkymästä? Koodia?

6.1.6 Kutsujen luku ja tarkistus

Koska toteutettavasta ohjelmistosta pyrittiin saamaan kattamaan kaiken tapahtumien järjestämiseen ja seurantaan liittyvä, tuli myös kutsujen lähettämisen lisäksi niiden lukemisen onnistua samasta ohjelmistosta. Kutsujen lukemista varten ohjelmiin kehitettiin oma näkymä kutsujen lukemiseen.

Kutsujen lukemiseen tarkoitettu näkymästä tuli hyvinkin yksinkertainen. Näkymässä on kenttä kutsun koodin syöttöön sekä paremman seurannan mahdollistamiseksi myös pakollinen sisäänkäynnin valinta. Näkymän oikeaan laitaan tulee luetun kutsun tiedot, mikäli kutsun tarkistus onnistuu ja se löytyy järjestelmästä. Luetusta kutsusta myös luodaan lukutapahtuma järjestelmään, myös vaikka kyseinen lippu oltaisiin luettu jo kertaalleen.

Kuvio 15. Kutsujen lukunäkymä.

Itse kutsujen lukemista varten ohjelmistoon luotiin uusi käyttäjäryhmä, johon kuuluvilla käyttäjillä on pääsy ainoastaan tapahtuman valintaan ja kutsujen lukemiseen. Pelkästään kutsujen lukemiseen tarkoitettua käyttäjäryhmän myötä mahdollistettiin tapahtuman varsinaisella sisäänkäynnillä olevien vastaanottovirkailijoiden käyttäjien erottaminen muusta ohjelmistosta ilman suurempaa ajankäyttöä.

TODO: Jokin koodinpätkäkuva?

6.1.7 Raportointi

Käytettyjen ja lähetettyjen kutsujen käyttöasteista toteutettiin hyvin yksinkertainen raportointinäkymä käyttäen toimeksiantajan työntekijän kehittämää raportointirajapintaa. Raportointi on tärkeä osa tapahtumien kävijämäärän seuraamisen kannalta, joten yksinkertaisimmastakin raportoinnista on ainakin jotain hyötyä.

Tapahtuma 1, Järjestäjä 1

Raportit

Lippuja lähetetty / käytetty (TJ)

Käyttäjät

vesa.kivisto@protacon.com

0

1

1

Lippuja lähetetty / käytetty

Käyttäjät

vesa.kivisto@protacon.com

0

1

1

Lippuja käytetty

Kutsujayritys 1

1

Lippuja lähetetty (TJ)

Sähköposti

0

Tulostus

1

Yhteensä

1

Lippuja käytetty (TJ)

Yhteensä

1

Päiväkohtainen käyttö

3.11.2017

1

Kuvio 16. Raportointinäkymä.

Esimerkki toteutetusta raportointinäkymästä on kuviossa 16. Näkymässä listataan käytetyt ja lähetetyt liput käyttäjäkohtaisesti sekä yhteenveto käytetyistä lipuista näytteilleasettajien mukaan ja yhteenveto lähetetyistä lipuista lähetystavan mukaan. Osa raportointinäkymän listauksista näkyy vain tapahtumajärjestäjälle (kuviossa 16 lyhenne (TJ)), kuten kokonaisyhteenveto lippujen käytöstä päiväkohtaisesti.

TODO: Koodinpätkä raportointiin liittyen?

6.2 Kutsujärjestelmän palvelinsovellus

6.2.1 Symfonyn käyttöönotto

Symfonyn käyttöönottamista varten käytettävään tietokoneeseen tulee olla asennettuna PHP. Ohjeet PHP:n asentamiseen löytyy PHP:n kotisivuilta ja Windowsille PHP asentuu esimerkiksi WampServer-nimisen ohjelmiston mukana. PHP:n asennuksen lisäksi kannattaa varmistaa, että tietokoneessa on asennettuna Composer. Symfony käyttää Composeria laajennuspakettien hallintaan ja sen käyttö tulee ennemmin tai myöhemmin Symfonyllä kehittäessä vastaan. Composerin asennukseen Windowsille löytyy ohjeet Composerin kotisivuilta.

Windowsilla Symfonyn asennustiedosto tulee ladata aivan ensimmäiseksi. Asennustiedoston lataus onnistuu seuraavalla komennolla:

```
php -r "file_put_contents('symfony', file_get_contents('https://symfony.com/installer'));"
```

Yllä oleva komento lukee *https://symfony.com/installer* -osoitteesta löytyvän tiedoston sisällön ja kirjoittaa tämän sisällön *symfony*-nimiseen tiedostoon. Komennon suorittamisen jälkeen voidaan ajaa komento

```
php symfony new projekti-nimi
```

Tämä komento luo uuden Symfony-projektin sille määritetyllä nimellä. Komennon suorittamiseen menee hetki, sillä se asentaa Symfonyn ja luo projektille tarvittavat tiedostot. Uuden projektin luomisen jälkeen suositellaan asentamaan Symfonyn tarjoama palvelin, jotta kehitystyö on mahdollista paikallisesti. Palvelimen asennus onnistuu projektin hakemistossa komennolla

```
composer require symfony/web-server-bundle
```


Komento päivittää projektin *composer.json*-tiedostoa, jossa on määriteltynä muun muassa sovelluksen vaatimat riippuvuudet. Symfonyn tarjoaman palvelimen asennuksen jälkeen Symfony-sovellus voidaan käynnistää paikallisesti komennolla

```
php bin/console server:run
```

Komento käynnistää Symfonyn tarjoaman palvelimen ja käynnistetty Symfony-sovellus löytyy URL-osoitteesta *http://localhost:8000/*.

6.2.2 Käyttäjien autentikointi ja autorisointi

Käyttäjien autentikoinnilla ja autorisoinnilla on suuri merkitys ohjelmiston tietoturvassa. Käytännössä autentikointi tarkoittaa käyttäjän oikeellisuuden tarkistamista, kun taas autorisoinnilla tarkistetaan mihin toimintoihin käyttäjällä on oikeus. Näin käyttäjä todetaan aidoksi ja estetään käyttäjää pääsemästä käsiksi niihin ohjelmiston osiin, joihin hänellä ei tulekaan olla pääsyä.

Käyttäjien autentikoinnin ja autorisoinnin toteutukseen ohjelmistossa käytettiin toimeksiantajan työntekijän kehittämää järjestelmää, jossa hyödynnetään Auth0:aa. Auth0:ssa käyttäjän kirjautuessa palveluun käyttäjä saa JWT (JSON Web Token) -muotoisen tunnisteen. Tämä tunniste lähetetään kirjautumisen jälkeen palvelinsovellukselle, joka vertaa sitä konfiguroituun salaukseen. Mikäli tunniste täsmää konfiguroidun salauksen kanssa, käyttäjä autorisoidaan ja hänelle annetaan kaksi uutta tunnistetta: refresh token (suom. päivitystunniste) ja bearer token (suom. ylläpitotunniste). Bearer token, jolla on erääntymisaika, kulkee käyttäjän palvelinsovellukselle lähettämien kutsujen mukana. Refresh tokenia puolestaan käytetään bearer tokenin uusimiseen sen vanhentuessa.

JWT-muotoinen tunniste koostuu kolmesta base64-enkoodatusta osasta, esimerkiksi seuraava tunniste:

```
ew0KICAiYWxnIjogIkhTMjU2IiwNCiAgInR5cCI6ICJKV1QiDQp9.ew0KICAiaXNzIjogInRlc3RpLmZpIiwNCiAgImV4cCI6IDE1MTA0NDA1NzUgLA0KICAibmFtZSI6ICJWZXNhIEtpdmlzdMO2Ig0KfQ==.1b5d3a636281978f7c766fb316218771d2c9b9db9f600f7d38a24fa008c49d61
```

Ensimmäinen osa

```
ew0KICAiYWxnIjogIkhTMjU2IiwNCiAgInR5cCI6ICJKV1QiDQp9
```

koostuu tunnisteen otsikkotiedoista. Otsikkotietoihin lukeutuu käytetty tunniste

tyyppi sekä tunnisteessa käytettävän salauksen muoto. Tunnisteen toisessa osassa

```
ew0KICAiaXNzIjogInRlc3RpLmZpIiwNCiAgImV4cCI6IDE1MTA0NDA1NzUgLA0KICAibmFtZSI6ICJWZXNhIEtpdmlzdMO2Ig0KfQ==
```

puolestaan kuljetetaan tietoa käyttäjästä, kuten nimi ja roolit ohjelmistossa, ja ohjelmiston ominaisuuksista, joihin käyttäjällä on käyttöoikeus. Tunnisteen toisessa osassa kuljetetaan myös tavallisesti tieto tunnisteen voimassaoloajasta, jonka umpeutumisen jälkeen käyttäjä tarvitsee uuden tunnisteen.

Kolmas ja viimeinen osa koostuu kahden ensimmäisen osan sekä ohjelmistoon määritellyn salausavaimen salatusta muodosta. Salaus toteutetaan tunnisteen ensimmäisessä osassa esitellyllä salauksella.

6.2.3 Tapahtuman ja toimijoiden hallinta

Luotaessa tapahtumajärjestäjiä, tapahtumia tai muita tapahtumaan liittyviä toimijoita lähetetään palvelinsovellukselle tästä luontipyyntö. Tapahtumaa ja sen toimijoita varten palvelinsovellukseen luotiin jokaiselle toimijalle oma entiteettiluokkansa ja kontrollerinsa, joiden metodeja kutakin entiteettiä luotaessa käytetään.

Esimerkiksi tapahtumajärjestäjää luotaessa selainsovelluksesta lähetetään POST-pyyntö palvelinsovellukseen, jonka Symfony ohjaa määriteltyjen reititysten mukaan tapahtumajärjestäjän kontrollerille (OrganizerController). OrganizerController, kuten muutkin kontrollerit, toteuttavat kontrollereille kirjoitetun Base-kontrollerin, jossa on määritelty yleiset kontrollerien vaatimat metodit.

TODO: Lisää tekstiä. Koodinpätkää?

6.2.4 Kutsujen lähetys ja tarkistus

Kutsujen lähetykseen sähköpostitse käytettiin Swift Mailer -nimistä kirjastoa, joka on sisäänrakennettu Symfonyyn SwiftmailerBundle-laajennoksena.

Kutsujen lähetykseen sähköpostitse käytettiin Symfonyssä sisäänrakennettuna olevaa Swift Mailer -kirjastoa

6.2.5 Tietokannan hallinta

Palvelinsovelluksessa tietokannan hallinta toteutettiin Doctrinellä. Tapahtumasta ja siihen liittyvistä toimijoista luotiin entiteettiluokat käyttäen Doctrinen tarjoamia annotaatioita, joilla saatiin määriteltyä esimerkiksi käytettävän taulun nimi ja taulun suhteet muihin tauluihin. Varsinainen tietokannan hallinta kuitenkin tapahtuu Doctrinen komennoilla.

Entiteettiluokan toteuttamisen tai muuttamisen jälkeen entiteetti halutaan sisällyttää itse tietokantaan. Tätä varten Doctrineen sisältyy useampi komento, jolla saadaan luotua niin sanottu migraatitiedosto ja muutettua tietokannan rakennetta migraatitiedoston sisällön mukaan. Migraatitiedosto luodaan komennolla

```
php doctrine:migrations:diff
```

Komento käy läpi kaikki entiteettiluokat ja niihin tehdyt muutokset ja luo muutoksista migraatitiedoston. Migraatitiedostot toimivat ikään kuin tietokannan rakenteen versionhallintana, sillä migraatitiedostoilla tietokannan rakenteen muutoksia voidaan tehdä eteen- ja taaksepäin. Esimerkki migraatitiedostosta on kuviossa 17.

```

<?php
declare(strict_types=1);

namespace App\Migrations;

use Doctrine\DBAL\Migrations\AbstractMigration;
use Doctrine\DBAL\Schema\Schema;

/**
 * Auto-generated Migration: Please modify to your needs!
 */
class Version20171030114540 extends AbstractMigration
{
    /**
     * @param Schema $schema
     *
     * @throws \Doctrine\DBAL\Migrations\AbortMigrationException
     */
    public function up(Schema $schema)
    {
        // this up() migration is auto-generated, please modify it to your needs
        $this->abortIf(
            $this->connection->getDatabasePlatform()->getName() !== 'mysql',
            'Migration can only be executed safely on \'mysql\'.'
        );

        $this->addSql('ALTER TABLE request_log CHANGE date `date` DATE NOT NULL');
    }

    /**
     * @param Schema $schema
     *
     * @throws \Doctrine\DBAL\Migrations\AbortMigrationException
     */
    public function down(Schema $schema)
    {
        // this down() migration is auto-generated, please modify it to your needs
        $this->abortIf(
            $this->connection->getDatabasePlatform()->getName() !== 'mysql',
            'Migration can only be executed safely on \'mysql\'.'
        );

        $this->addSql('ALTER TABLE request_log CHANGE `date` date DATE DEFAULT NULL');
    }
}

```

Kuvio 17. Esimerkki Doctrinen migraatiotiedostosta.

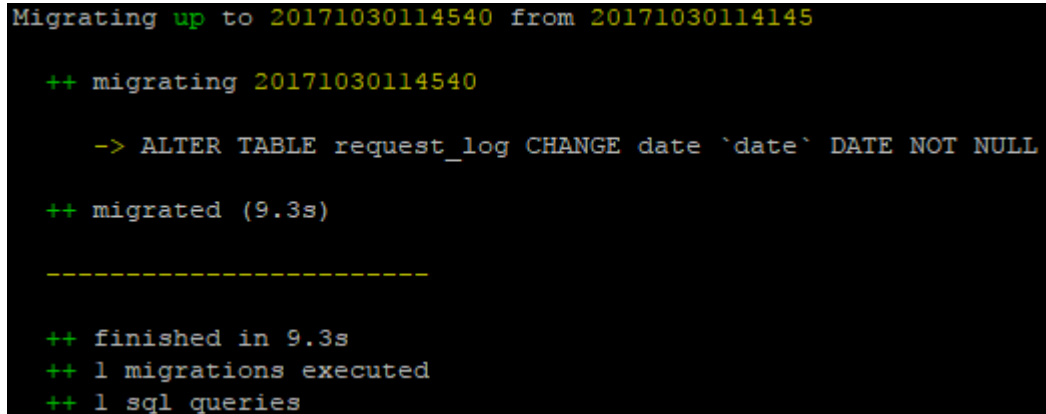
Varsinainen tietokannan rakenteen muutos tapahtuu komennolla

```
php doctrine:migrations:migrate
```

Komento suorittaa viimeisimmissä migraatiotiedostoissa olevat *up()*-metodit vanhimmasta uusimpaan. Esimerkki tämän komennon suorittamisesta on kuviossa 18. Halutessaan komennossa voidaan käyttää esimerkiksi *prev*-lisämäärettä, joka suorittaa viimeksi ajetun migraatiotiedoston *down()*-metodin. Vaihtoehtoisesti migraation suorittaminen onnistuu myös komennolla

```
php doctrine:migrations:exec migraatitiedosto
```

Tällä komennolla voidaan ajaa mikä tahansa migraatitiedosto. Komennolla voi kuitenkin tulla ongelmia, jos esimerkiksi yritetään perua muutosta taulun sarakkeeseen, jossa on suhde toiseen tauluun eikä kyseisen suhteen poisto sisälly ajettavaan migraatitiedostoon.

A terminal window with a black background and green text. The output shows a migration command being executed successfully. It starts with 'Migrating up to 20171030114540 from 20171030114145', followed by '++ migrating 20171030114540'. Then it shows the SQL command '-> ALTER TABLE request_log CHANGE date `date` DATE NOT NULL'. This is followed by '++ migrated (9.3s)', a dashed line separator, and finally '++ finished in 9.3s', '++ 1 migrations executed', and '++ 1 sql queries'.

```
Migrating up to 20171030114540 from 20171030114145

++ migrating 20171030114540

-> ALTER TABLE request_log CHANGE date `date` DATE NOT NULL

++ migrated (9.3s)

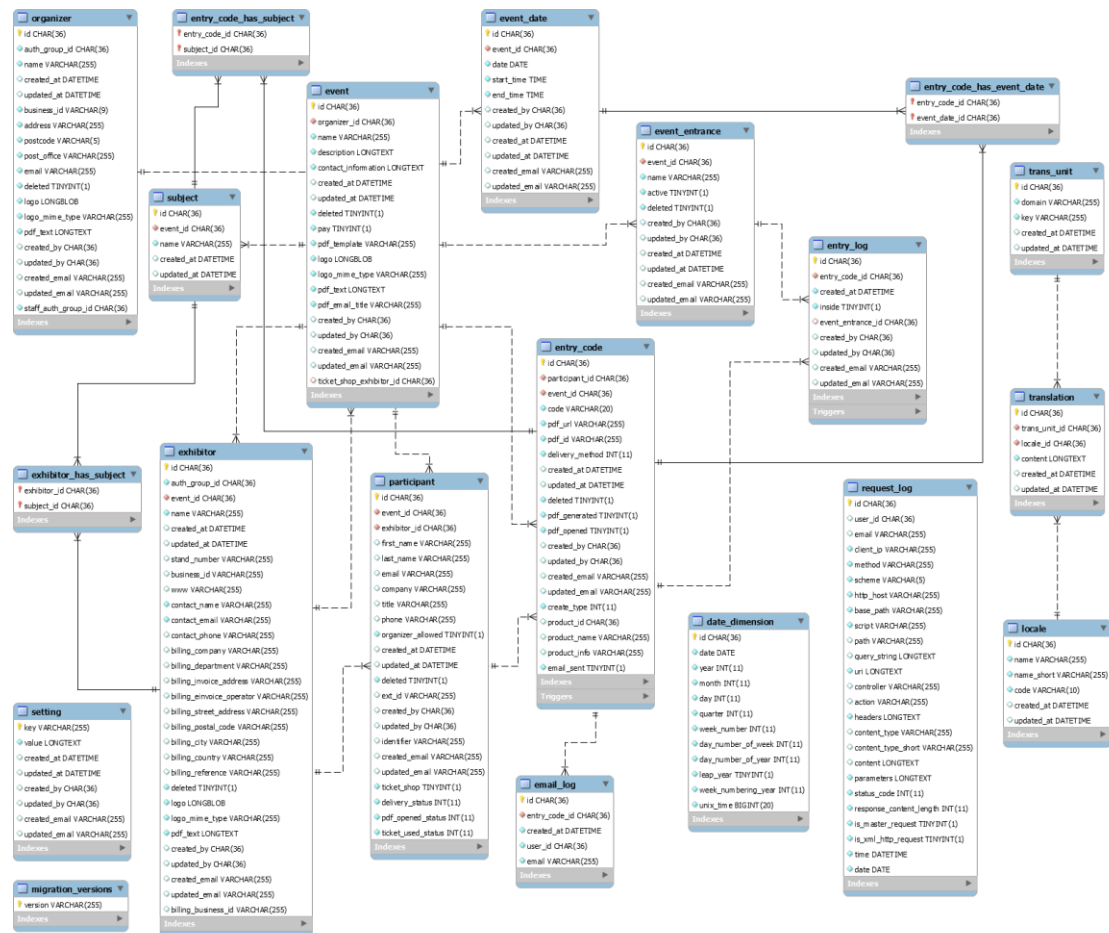
-----

++ finished in 9.3s
++ 1 migrations executed
++ 1 sql queries
```

Kuvio 18. Migraatiokomennon suorittaminen.

6.2.6 Tietokannan kokonaisrakenne

Koska toteutettava kutsujärjestelmä on itsessään hyvin laaja kokonaisuus, on myös sen käyttämä tietokanta monimutkainen. Toteutettu tietokanta koostuu 20 taulusta, joiden väliset suhteet voidaan huomata kuviossa 19. Monimutkaisin taulu on tapahtumat tallentava *event*-taulu, jolla on suhteet kahdeksaan muuhun tauluun, kuten tapahtumajärjestäjän ja kutsukoodien tauluhin.



Kuvio 19. Tietokannan kokonaisrakenne.

7 Tulokset

7.1 Lopullinen ohjelmisto

Opinnäytteen konkreettisena tuotoksena toteutettiin toimeksiantajan asiakkaalle osa asiakkaan vaatimuksista täyttävä ohjelmisto. Ohjelmistolla onnistuu tehdä kaikkea tapahtumien järjestämiseen ja seurantaan liittyvistä toiminnoista: tapahtumajärjestäjien, tapahtumien ja näytteilleasettajien luonti onnistuu ja ohjelmiston kautta voidaan tulostaa ja lähettää kutsuja kutsuttaville asiakkaille. Toteutetun ohjelmiston ollessa vasta ensimmäinen julkaisukelpoinen versio ei se ole täysin asiakkaan näkemyksen mukainen, vaan siitä puuttuu joitakin ominaisuuksia. Näiden ominaisuuksien toteuttamiseen pureudutaankin jatkokehityksessä ja projektin seuraavissa vaiheissa.

7.2 Testaaminen, viat ja puutteet

Varsinaista ohjelmiston testaamista ei aikataulun kiireellisyyden vuoksi ehditty toteuttamaan. Ohjelmiston käyttöä ja toimintoja testattiin pääosin sisäisesti projektiryhmän jäsenten kesken sekä asiakkaalle luodun testiympäristön avulla. Testauksessa erityisesti asiakas huomasi puutteita tietyissä ohjelmiston ominaisuuksissa, jotka ehdittiin nopeasti kirjaamisen jälkeen korjata.

Vähäinen testauksen määrä kuitenkin johti ongelmiin. Kehityksen aikana ominaisuuksia ja käytettävyyttä saatiin parannettua jonkin verran projektiryhmän ja asiakkaan tekemien testauksen pohjalta, mutta mitään tätä laajempaa testausta, kuten suorituskyytestausta, ei suoritettu. Puutteellisesti toteutetun testauksen seurauksena ohjelmistosta jäi huomaamatta suorituskyyteen liittyviä ongelmia, jotka tulivat esille vasta ohjelmiston oltua varsinaisessa käytössä. Esimerkiksi CSV-tuonnissa tiedoston käsittelyajat nousivat eksponentiaalisesti datamäärän kasvaessa ohjelmistossa. Ongelma johtui kutsukoodien generointiin liittyvistä tarkastuksista, ja ratkaisuksi ongelmiin kutsukoodista tehtiin muutaman merkin pidempiä ja otettiin varsinaiset kaksoiskappale tarkistukset pois. Uutta generointilogiikkaa testattiin, vaikkakin jälleen hyvin lyhyesti, eikä kaksoiskappaleita ilmentynyt. Logiikkamuutoksen myötä tiedostojen käsittelyajat saatiin takaisin hyväksyttävälle tasolle.

Toinen testauksen puutteellisuudesta johtuva suorituskyyongelma huomattiin vasta pari viikkoa ensimmäisen käyttöönottopahtuman jälkeen. Käyttäjä- ja datamäärän kasvaessa ohjelmiston käyttö hidastui yllättäen ja esimerkiksi näkymän vaihtoon meni useita sekunteja. Kaikeksi onneksi tämä ongelma ilmeni vasta, kun ohjelmiston kiireisin käyttöaika oli ohi, joten hitaudesta ei varsinaisesti ollut haittaa ohjelmiston käytön kannalta. Suorituskyyteen liittyvät ongelmat ovat kuitenkin kriittisiä ongelmia, joten korjaus otettiin heti työn alle ja saatiin ratkaistua melko nopeasti.

7.3 Jatkokehitys

Toteutetun ohjelmiston ollessa vasta ensimmäinen julkaisukelpoinen versio jatkuu sen kehitys toimeksiantajan asiakasprojektina. Toimeksiantajan asiakkaalla on ohjelmistolle useita jatkokehitysideoita niin käyttöliittymän ja yleisen käytettävyyden kuin

ominaisuuksien kannalta. Esimerkiksi tapahtuman luonnin yleistä työnkulkua selvennetään muuttamalla näkymiä selvemmiksi ja jakamalla toiminnallisuuksia pienempiin kokonaisuuksiin. Ulkoasussa ei myös ole kunnolla hyödynnetty responsiivisuutta, joten ohjelmistolle ei vielä ole mobiililaitetukea. Responsiivisuuden lisäksi ulkoasun räätälöinti on yksi jatkokehityksen aiheista.

8 Pohdinta

8.1 Kehitystyön toimivuus

Projektissa käytetyn Kanbania ja Scrumia yhdistävän metodin toimivuus oli paikoin toimivaa, mutta näiden seurannassa oli kuitenkin parantamisen varaa. Päivittäiset scrumit ja Kanban-taulun käytön perusteet toimivat hyvin ja viestintä niin projektiryhmän jäsenten kesken kuin asiakkaan suuntaan toimi hyvin. Metodin toteutuksessa oli kuitenkin joitakin heikkouksia, muun muassa Kanban-taulun työvaiheiden määrimäärät ylittyivät useasti ja tehdyt työt kasaantuivat erityisesti katselmointivaiheeseen.

Pienistä puutteista huolimatta kehitystyö oli sujuvaa. Opinnäytetyön tekijälle projekti oli ensimmäinen suuremman kokoluokan projekti, jossa valittua metodologiaa toteutettiin edes jokseenkin tarkasti. Kanbanin ja Scrumin yhdistelmä oli toimivaa ja päivittäiset scrumit olivatkin erittäin hyvä tukipilari Kanbanin tarkalle työn visualisoinnille. Kanbanista mainitsemisen arvoisena huomiona kuitenkin on, että metodina Kanban tuntuu jatkuvalta tekemiseltä. Scrumin sprinttien loppuissa olevaa katselmointipalaveria ei Kanbanissa ole, vaan yhden työn toteuttamisen jälkeen siirrytäänkin jo seuraavaan. Missään vaiheessa ei siis varsinaisesti tule vaihetta, jolloin voisi juhlistaa toteutettuja ominaisuuksia. Projektissa tätä kuitenkin tasapainotettiin säännöllisillä tapaamisilla asiakkaan kanssa.

8.2 Projektin parissa työskentely

Opinnäytetyön tekijälle opinnäytetyön aiheena toteutettu ohjelmisto oli ensimmäinen suuremman kokoluokan projekti. Ohjelmistotekniikan opintoihin kuuluvissa opinto-

jaksoissa toki oli myös asiakasprojekteihin keskittyviä opintojaksoja, mutta varsinaisen työelämän asiakasprojektit ovat kuitenkin täysin eri kokoluokkaa ja niissä on erilainen tekemisen tunne.

Projektin parissa työskenteleminen olikin loistava oppimistilaisuus. Jo projektin ensimmäisistä vaiheista lähtien uutta opittavaa tuli niin asiakastapaamisten ja eri projektivaiheiden kuin varsinaisen kehitystyön parissa.

8.3 Johtopäätökset

Opinnäytetyö oli hyvin opettava kokemus. Varsinaisen opinnäytetyön tekemisen lisäksi opinnäytetyön tekemisen aikana opinnäytetyön tekijä pääsi tutustumaan toimeksiantajan työyhteisöön ja perehtymään työskentelykäytäntöihin sekä aloittamaan oman työuransa. Kun työkavereina on alalla useita vuosia työskennelleitä kokeneempia henkilöitä, voi huoletta kysyä typerämmältäkin tuntuvia kysymyksiä. Asiakasprojektista sai kallisarvoista työkokemusta ja opinnäytetyön tekijän itseluottamus ja vastuunottokyky tuntuivat paranneen huomattavasti.

Haastavinta opinnäytetyössä oli uusien teknologioiden. Lähestulkoon kaikki projektissa käytetyt teknologiat olivat opinnäytetyön tekijälle uusia, joten jo pelkästään eri teknologioista opitun määrä oli huomattava. Oman haasteensa opinnäytetyön tekemiseen toi työssäkäynnin, vapaa-ajan ja opinnäytetyön kirjoittamisen tasapainottaminen. Normaalin kahdeksan tunnin työpäivän päätteeksi opinnäytetyön kirjoittaminen tuntui turhan raskaalta ensimmäisten viikkojen aikana ja varsinaisen vapaa-aika jäikin hyvin pieneksi, ellei jopa olemattomaksi. Vaikka opinnäytetyön kirjoittaminen oli paikka paikoin haastavaa, muuttui sen kirjoittaminen ja vapaa-ajan ja työskentelyn tasapainottelu helpommaksi.

Kokonaisuutena opinnäytetyö oli onnistunut. Asiakasprojektin aikana kehitetystä ohjelmistosta toimeksiantajan asiakas sai osan vaatimuksista täyttävän ohjelmiston. Taivoite ei kuitenkaan ollut kehittää ohjelmistoa kerralla loppuun asti, vaan kehittää aluksi ensimmäinen julkaisuvalmis versio, jota lähdetään myöhemmässä vaiheessa jatkokehittämään. Opinnäytetyön tekijä puolestaan tuntee kehittyneensä projektitiimin jäsenenä, vaikka opittavaa alalla toki riittää huomattavia määriä.

Lähteet

About MariaDB. Esittely MariaDB:stä MariaDB:n kotisivuilla. N.d. Viitattu 28.10.2017. <https://mariadb.org/about/>

About Mercurial. Artikkel Mercurialin esittelystä Mercurialin kotisivuilla. N.d. Viitattu 16.10.2017. <https://www.mercurial-scm.org/about>

About PostgreSQL. Esittely PostgreSQL:stä PostgreSQL:n sivuilla. N.d. Viitattu 28.10.2017. <https://www.postgresql.org/about/>

Architecture Overview. Artikkel Angularin kotisivuilla Angularin kokonaisarkkitehtuurista. N.d. Viitattu 6.10.2017. <https://angular.io/guide/architecture>

Boehm's Spiral Model. Ian Sommervillen artikkeli Boehmin spiraalimallista. N.d. Viitattu 29.10.2017. <http://iansommerville.com/software-engineering-book/web/spiral-model/>

DBMS Overview. Tutorialspoint-sivuston tietokannan hallintajärjestelmien opas. N.d. Viitattu 28.10.2017. https://www.tutorialspoint.com/dbms/dbms_overview.html

Git Merge. Artikkel Gitin merge-toiminnon käytöstä. N.d. Viitattu 4.11.2017. <https://www.atlassian.com/git/tutorials/git-merge>

IDE. Artikkel Wikipediassa ohjelmointiympäristöistä. Muokattu 25.10.2017. Viitattu 30.10.2017. https://en.wikipedia.org/wiki/Integrated_development_environment

Jira (software). Artikkel Wikipediassa. Muokattu 19.9.2017. Viitattu 15.10.2017. [https://en.wikipedia.org/wiki/Jira_\(software\)](https://en.wikipedia.org/wiki/Jira_(software))

Kanban. Atlassianin artikkeli Kanban-metodista. N.d. Viitattu 29.10.2017. <https://fi.atlassian.com/agile/kanban>

Phabricator. Artikkel Wikipediassa. Muokattu 8.10.2017. Viitattu 15.10.2017. <https://en.wikipedia.org/wiki/Phabricator>

PHP: Intro. PHP: esittely PHP:n kotisivuilla. N.d. Viitattu 13.10.2017. <http://php.net/manual/en/intro-what-is.php>

PHP: What Can Do. Artikkel PHP:n käyttökohteista PHP:n kotisivuilla. N.d. Viitattu 13.10.2017. <http://php.net/manual/en/intro-whatcando.php>

PhpStorm. Wikipedia-artikkeli JetBrains PhpStormista. Muokattu 28.9.2017. Viitattu 30.10.2017. <https://en.wikipedia.org/wiki/PhpStorm>

Slack (software). Artikkel Wikipediassa. Muokattu 28.9.2017. Viitattu 15.10.2017. [https://en.wikipedia.org/wiki/Slack_\(software\)](https://en.wikipedia.org/wiki/Slack_(software))

Software framework. Artikkel Wikipediassa ohjelmistokehyksistä. Muokattu 4.8.2017. Viitattu 22.10.2017. https://en.wikipedia.org/wiki/Software_framework

Sommerville, I. 2016. Software Engineering, Tenth Edition. Boston: Pearson Education.

Symfony: controller. Artikkelin Symfony:n kontrollereista Symfony:n kotisivuilla. N.d. Viitattu 7.10.2017. <https://symfony.com/doc/current/controller.html>

Symfony: Doctrine. Artikkelin Doctrinen käytöstä Symfony:n kotisivuilla. N.d. Viitattu 7.10.2017. <https://symfony.com/doc/current/doctrine.html>

Symfony: routing. Artikkelin Symfony:n reitityksestä Symfony:n kotisivuilla. N.d. Viitattu 7.10.2017. <https://symfony.com/doc/current/routing.html>

Symfony: service container. Artikkelin Symfony:n palveluista Symfony:n kotisivuilla. N.d. Viitattu 7.10.2017. https://symfony.com/doc/current/service_container.html

Symfony: six reasons. Artikkelin kuudesta Symfony:n tuomasta hyödystä Symfony:n kotisivuilla. N.d. Viitattu 7.10.2017. <https://symfony.com/six-good-technical-reasons>

Symfony: templating. Artikkelin Symfony:n malleista Symfony:n kotisivuilla. N.d. Viitattu 7.10.2017. <https://symfony.com/doc/current/templating.html>

Template Syntax. Artikkelin Angular-kehiksen kotisivuilla Angularin mallisyntaxista. N.d. Viitattu 6.10.2017. <https://angular.io/guide/template-syntax>

TypeScript. Wikipedia-artikkeli TypeScriptistä. Muokattu 20.9.2017. Viitattu 13.10.2017. <https://en.wikipedia.org/wiki/TypeScript>

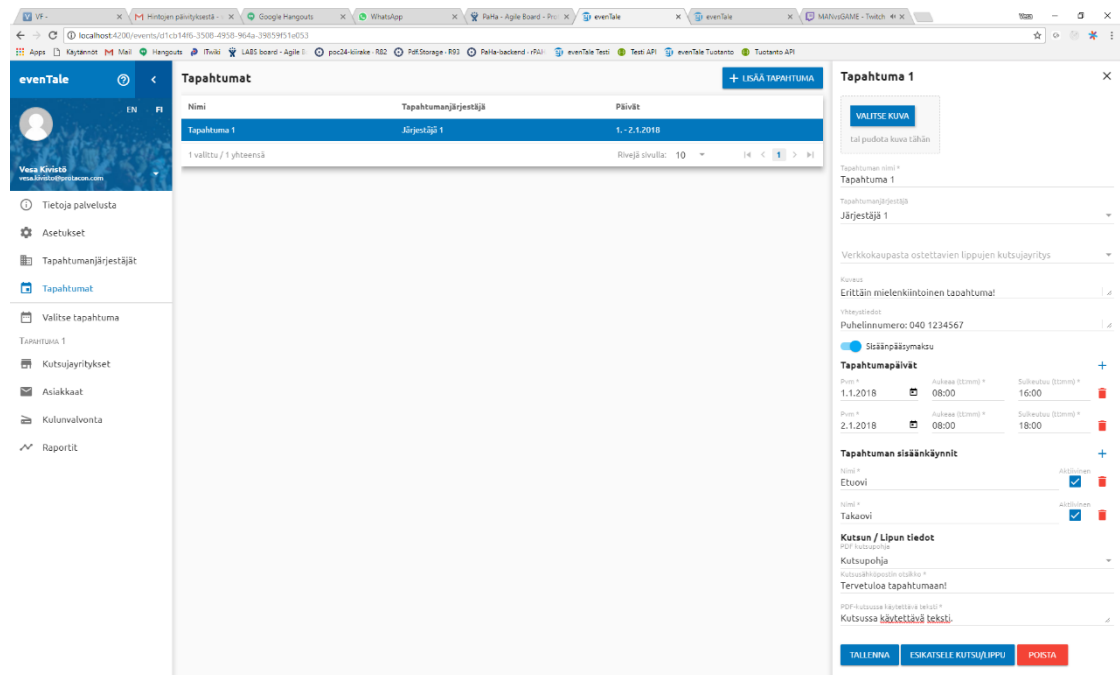
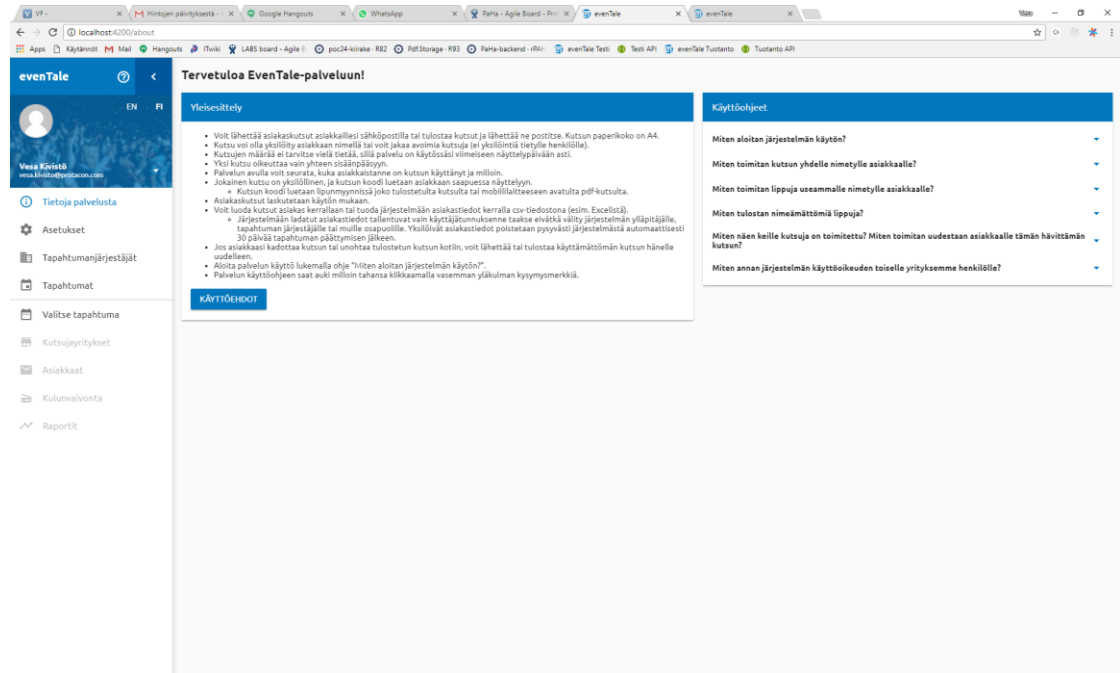
Using Branches. Artikkelin Gitinbranching-ominaisuuden käytöstä. N.d. Viitattu 4.11.2017. <https://www.atlassian.com/git/tutorials/using-branches>

What is Angular. Artikkelin Angularin kotisivuilla Angularin perusteista. N.d. Viitattu 6.10.2017 <https://angular.io/docs>

What is Git. Artikkelin Atlassianin kotisivuilla. N.d. Viitattu 16.10.2017 <https://www.atlassian.com/git/tutorials/what-is-git>

Liitteet

Liite 1. Kuvakaappauksia selainsovelluksesta



evenTale

Tapahtuma 1, Järjestäjä 1

Kutsujayritykset

Suodattimet

Hakusana

PALAUTA OLETUSSUODATTIMET

Nimi	Y-Tunnus	Osastonumero	Lisäty
Kutsujayritys 1	1234567-8	A100	3.11.2017

1 valittu / 1 yhteensä

Rivejä sivulla: 10

Kutsujayritys 1

Valitse kuva

Valitse kuva tähän

Yleiset tiedot

Nimi: Kutsujayritys 1

Y-tunnus: 1234567-8

Osastonumero: A100

WWW-sivusto: www.osoite.com

Yhteystietona oma teksti kutsuun: Kutsujayrityksen teksti kutsuun.

Yhteystiedot

Nimi: Vesa Kivistö

Sähköposti: vesa.kivistö@protacon.com

Puhelin: 040 1234567

Laskutustiedot

Käyttäjät

TALLENNNA ESIKATSELE KUTSUJAYRITYS POISTA

evenTale

Tapahtuma 1, Järjestäjä 1

Asiakkaat

Suodattimet

Hakusana

Kutsujayritys

Sähköposti

Kutsun lähetystapa

Kutsujen lähetytys

Kutsujen tulostus

Lippujen käyttö

Lipun luontitapa

Toiminnot kohdistuvat 4 asiakkaaseen

LÄHETÄ KUTSUJA TULOISTA KUTSUT

Etunimi	Sukunimi	Erätunniste	Kutsujayritys	Sähköposti	Puhelin	Yritys	Titled
			Asiakaskutsut 1				
Essi	Esimerkki		Kutsujayritys 1			Yritys	Edustaja
Vesa	Kivistö		Kutsujayritys 1	vesa.kivistö@protacon.com			
Matti	Meikäläinen		Kutsujayritys 1				

1 valittu / 4 yhteensä

Rivejä sivulla: 10

Matti Meikäläinen

Kutsujayritys: Kutsujayritys 1

Etunimi: Matti

Sukunimi: Meikäläinen

Erätunniste

Sähköposti

Puhelinnumero

Yritys

Titled

Asiakas nro

Kutsut: 886AD4429F

Lähetetty: 3.11.2017

Käytetty: 3.11.2017

TULOISTA KUTSUT LÄHETÄ KUTSU POISTA