

Ohjelmistokehitys Angular- ja Symfony-sovelluskehyksillä

Vesa Kivistö

Opinnäytetyö

Joulukuu 2017

Tekniikan ja liikenteen ala

Insinööri (AMK), ohjelmistotekniikan tutkinto-ohjelma

Tekijä(t) Kivistö, Vesa	Julkaisun laji Opinnäytetyö, AMK	Päivämäärä Joulukuu 2017
	Sivumäärä 62	Julkaisun kieli Suomi
		Verkojulkaisulupa myönnetty: x
Työn nimi Ohjelmistokehitys Angular- ja Symfony-sovelluskehyksillä		
Tutkinto-ohjelma Ohjelmistotekniikan tutkinto-ohjelma		
Työn ohjaaja(t) Ari Rantala		
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 raportointi-, autentikointi- ja PDF-järjestelmät. Raportointijärjestelmän tuli koostaa raportteja kutsujärjestelmän käyttöasteista. Autentikointijärjestelmä vaadittiin kutsujärjestelmän käyttäjien autentikointiin ja PDF-järjestelmää käytettiin luotujen kutsujen käsittelyyn.</p> <p>Työ toteutettiin toimeksiantajan tiloissa 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, 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 December 2017
		Language of publication: Finnish
	Number of pages 62	Permission for web publication: x
Title of publication Software development with Angular and Symfony frameworks		
Degree programme Software Engineering		
Supervisor(s) Ari Rantala		
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, be smooth to use and have a modern and simple layout. The focus was on the invitation system, though reporting, authentication and PDF service systems were also developed during the project. The reporting system had to generate reports from ticket usage rates, whereas the authentication system was needed for authenticating the invitation system's users, and the PDF service was used to handle invitations created in the invitation system.</p> <p>The work was carried out within the assigner's premises as a customer project. Different technologies were used to implement the 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, and the reporting system generates reports that meet the minimum requirements.</p>		
Keywords/tags (subjects) Angular, Symfony, TypeScript, PHP, software development, agile software development		
Miscellaneous (Confidential information)		

Sisältö

Termit.....	6
1 Työn lähtökohdat	9
1.1 Taustaa	9
1.2 Toimeksiantaja	9
2 Tehtävän kuvaus	10
2.1 Ongelma	10
2.2 Tavoitteet	10
2.3 Vaatimusmäärittely	11
2.3.1 Yleistä.....	11
2.3.2 Toiminnalliset vaatimukset.....	11
2.3.3 Ei-toiminnalliset vaatimukset	12
3 Ohjelmistokehitys.....	13
3.1 Yleistä	13
3.2 Menetelmät.....	13
3.2.1 Yleistä.....	13
3.2.2 Vesiputousmalli	14
3.2.3 Spiraalimalli	15
3.3 Ketterät menetelmät.....	15
3.3.1 Yleistä.....	15
3.3.2 Scrum	16
3.3.3 Kanban	17
4 Sovelluskehikset	17
4.1 Yleistä	17
4.2 Angular	18

	2
4.2.1 Arkkitehtuuri.....	18
4.2.2 Moduulit	19
4.2.3 Komponentit.....	20
4.2.4 Mallit.....	22
4.2.5 Palvelut	22
4.2.6 Direktiivit	23
4.3 Symfony.....	24
4.3.1 Reititys	24
4.3.2 Kontrollerit.....	25
4.3.3 Mallit.....	26
4.3.4 Palvelut	26
4.4 Doctrine	27
5 Muut teknologiat ja työkalut	29
5.1 Ohjelmointikielet.....	29
5.1.1 TypeScript	29
5.1.2 PHP.....	30
5.2 Ohjelmointiympäristöt	31
5.2.1 Yleistä.....	31
5.2.2 JetBrains PhpStorm.....	31
5.3 Versionhallinta.....	32
5.3.1 Yleistä.....	32
5.3.2 Phabricator	32
5.3.3 Git	32
5.4 Tietokannan hallintajärjestelmät	35
5.4.1 Yleistä.....	35
5.4.2 MariaDB	35

5.4.3	PostgreSQL.....	35
5.5	Jira	36
5.6	Slack.....	36
6	Toteutus.....	37
6.1	Kutsujärjestelmän selainsovellus	37
6.1.1	Angularin käyttöönotto	37
6.1.2	Käyttöliittymän toteutus	38
6.1.3	Tapahtuma ja tapahtumaan liittyvät toimijat	38
6.1.4	CSV-tuonti.....	40
6.1.5	Kutsujen lähettäminen	42
6.1.6	Kutsujen luku ja tarkistus.....	43
6.1.7	Raportointi.....	45
6.2	Kutsujärjestelmän palvelinsovellus	46
6.2.1	Symfonyn käyttöönotto.....	46
6.2.2	Käyttäjien autentikointi ja auktorisointi	47
6.2.3	Tapahtuman ja toimijoiden hallinta	48
6.2.4	Kutsujen lähetys ja tarkistus.....	51
6.2.5	Tietokannan hallinta	52
6.2.6	Tietokannan kokonaisrakenne	54
7	Tulokset	55
7.1	Lopullinen ohjelmisto	55
7.2	Testaaminen, viat ja puutteet	56
7.3	Jatkokehitys	56
8	Pohdinta.....	57
8.1	Kehitystyön toimivuus.....	57
8.2	Projektin parissa työskentely	57

8.3 Johtopäätökset	58
--------------------------	----

Lähteet	59
----------------------	-----------

Liitteet	61
-----------------------	-----------

Liite 1. Kuvakaappauksia selainsovelluksesta	61
--	----

Kuviot

Kuvio 1. Angularin kokonaisarkkitehtuuri	18
Kuvio 2. Esimerkki Angularin moduulista	19
Kuvio 3. Esimerkki Angularin komponentista	21
Kuvio 4. Esimerkki Angularin mallista	22
Kuvio 5. Esimerkki Angularin palvelusta	23
Kuvio 6. Esimerkki Angularin direktiivistä	24
Kuvio 7. Esimerkki Symfonyn kontrollerista	25
Kuvio 8. Esimerkki Symfonyn HTML-mallista	26
Kuvio 9. Palvelun käyttö Symfonyssä	27
Kuvio 10. Symfonyn ja Doctrinen entiteetti-luokka	28
Kuvio 11. Esimerkki yksinkertaisesta Gitin työkulusta	34
Kuvio 12. Esimerkki Gitin työkulusta branchien kanssa	34
Kuvio 13. Tapahtumajärjestäjien listaus	39
Kuvio 14. Tapahtumajärjestäjän luontilogiikka selainsovelluksessa	40
Kuvio 15. CSV-tuonnin virheilmoitus	41
Kuvio 16. CSV-tiedoston tuonti ja tarkistus	41
Kuvio 17. Näkymä kutsujen lähettämiseen sähköpostilla	42
Kuvio 18. Palvelinsovellukselle lähetyspyynnön lähettävä koodi	43
Kuvio 19. Kutsujen lukunäkymä	43
Kuvio 20. Kutsun tietojen hakeminen palvelinsovelluksesta resolvella	44
Kuvio 21. Raportointinäkymä	45
Kuvio 22. Näytteilleasettajan toimitettujen kutsujen päivitys	45
Kuvio 23. OrganizerControllerin traitit	49

Kuvio 24. Uudelleenkäytettävä luontimetodi	49
Kuvio 25. Entiteetin luonnin käsittelevä metodi	50
Kuvio 26. Sähköpostin lähettävä metodi	51
Kuvio 27. Esimerkki Doctrinen migraatitiedostosta.....	53
Kuvio 28. Migraatiokomennon suorittaminen	54
Kuvio 29. Tietokannan kokonaisrakenne	55

Taulukot

Taulukko 1. @NgModule-decoratorin määreet.....	20
Taulukko 2. @Component-decoratorin määreet.....	21

Termit

Annotaatio

Annotaatio on metadatan (ks. metadata) tyyppi, jolla dokumentoidaan koodia. Annotaatio voi vaikuttaa koodin ajamiseen.

Base64

Base64 on tekstin koodausmuoto, jossa binäärimuotoiseksi muutettu data esitetään tekstimuotoisena. Base64-koodausta käytetään usein tiedonsiirrossa varmistamaan tiedon muuttumattomuus.

CSS

Cascading Style Sheet. Käytetään web-kehityksessä tyylitiedostojen kirjoittamiseen.

CSV

Comma-separated values. CSV-tiedostossa jokainen rivi on oma tietueensa ja rivin eri arvot erotetaan toisistaan tavallisesti pilkulla. CSV-tiedostoja käytetään yleensä datan viennissä järjestelmästä toiseen.

Decorator

Decorator on Angularissa käytettävä funktio, joka vaikuttaa luotavan luokan ominaisuuksiin.

ECMAScript

JavaScriptin (ks. JavaScript) standardisointiin kehitetty kieli.

Entiteetti

Reaalimaailman asiaa ja sen ominaisuuksia mallintava olio.

HTML

Hypertext Markup Language. Verkkosivujen kirjoitukseen kehitetty mallintakieli.

JavaScript

ECMAScriptin (ks. ECMAScript) toteuttava ohjelmointikieli, jota käytetään erityisesti web-kehityksessä.

Jäsennin

Jäsennin tarkistaa, onko annettu syöte tietyn ohjelmointikielen syntaksin mukainen.

Metadata

Metadata määrittää ja kuvaa jotakin toista tietoa. Esimerkiksi tiedoston nimi on tiedoston metadataa.

MySQL

Relaatiotietokantojen (ks. Relatiotietokanta) hallintajärjestelmä.

Oliorelaatiotietokanta

Relaatiotietokannan (ks. Relatiotietokanta) kaltainen tietokanta, jossa on tuki olio-ohjelmoinnin olioiden tallentamiselle suoraan tietokantaan.

Relaatiotietokanta

Relaatiotietokannassa taulujen välille määritetään yhteyksiä, relaatioita. Taulujen tiedot liitetään toisiinsa taulujen avaimilla ja yhteyttä luovaa avainta kutsutaan vieras-avaimeksi (engl. foreign key).

Renderöinti

Renderöinti on kuvan tai web-kehityksessä verkkosivun elementin luomista datan ja mallin pohjalta.

Repository

Repository on versionhallintajärjestelmissä käytettävä koodin säilytyspaikka.

Semantiikka

Ohjelmointikielen semantiikan tarkoitus on helpottaa ohjelmointikielen ymmärtämistä sekä oikeaoppisen syntaksin (ks. Syntaksi) kirjoittamista.

Serialisointi

Serialisointi tarkoittaa datarakenteen tai olion muuttamista muotoon, jossa se on helpompi tallentaa tai käsitellä.

Syntaksi

Ohjelmointikielen syntaksi on joukko sääntöjä, jotka määrittelevät miltä kirjoitetun ohjelmointikielen oikeaoppinen muotoilu tulisi näyttää.

Tietokanta

Tietokanta on datan säilytyspaikka. Tietokannan dataa voidaan lisätä, muokata ja poistaa kyselyillä.

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 työnhallinta- 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

Opinnäytetyön toimeksiantajana toimi Protacon Solutions Oy (PTCS). PTSC on osa suurempaa Protacon konsernia, ja se perustettiin vuonna 1998, kun Protacon Engineering Oy osti Jiop Oy:n osake-enemmistön (Protacon historia 2017). PTSC:n toimialaan kuuluu ohjelmistoratkaisujen ja IT-palveluiden tuottaminen sisältäen ohjelmistojen suunnittelun, toteuttamisen ja dokumentoinnin sekä tuki- ja ylläpitopalveluiden tarjoamisen. Vuonna 2016 PTSC:n liikevaihto oli noin 7,9 miljoonaa euroa ja henkilöstömäärä 92. (Protacon Solutions Oy 2017.)

Protacon on vuonna 1990 perustettu suomalainen teknologia-alan palveluyritys, joka tuottaa asiakkailleen räätälöityjä ratkaisuja tuotannon, tuotekehityksen ja ylläpidon tarpeisiin käyttäen digitalisaation mukanaan tuomat hyödyt. Vuonna 2016 Protaconin liikevaihto oli noin 210 miljoonaa euroa ja henkilöstömäärä noin 250. (Me olemme Protacon 2017.)

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, kuitenkin keskittyen palvelun käytettävyyteen ja suorituskykyyn sisältäen myös vastaavien ohjelmistojen perustoiminnot, kuten tapahtumajärjestäjien, tapahtumien ja näytteilleasettajien luonnit.

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 opinnäytetyö on kirjoitettu, oli saada vaatimukset täyttävä tapahtumien järjestämiseen kehitetty ohjelmisto. Asiakkaalla on suunnitelmissa myydä toteutettua ohjelmistoa myöhemmin palveluna muille alan toimijoille.

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 määritelmänä. (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 suorituskvyssä tai puutteellisia tai täysin toteuttamatta jätettyjä toimintoja.

2.3.2 Toiminnalliset vaatimukset

Ohjelmiston toiminnalliset vaatimukset määrittävät, mihin toimintoihin ohjelmiston on kyettävä, kuinka ohjelmisto reagoi käyttäjältä tuleviin syötteisiin tai kuinka se reagoi toimintaympäristönmuutoksiin. Toiminnalliset vaatimukset tulisi lähtökohtaisesti kirjoittaa kehittäjälähtöisesti: toiminnallisesta vaatimuksesta tulisi siis käydä ilmi syötteet, tulosteet 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.)

Toteutettavaan ohjelmistoon oli vaatimuksena lisäys-, muokkaus- ja poistotoiminnot tapahtumajärjestäjille, tapahtumille, tapahtumien näytteilleasettajille sekä kutsuttaville. Luodulle tapahtumalle tuli olla mahdollista valita tapahtuman kutsuissa käytettävä ulkoasupohja, ja kunkin toimijan tietoja tarkasteltaessa tuli olla mahdollisuus kutsun esikatseluun. Kutsuttaville tuli olla mahdollista lähettää kutsut sähköpostitse, mikäli kutsuttavan sähköposti oli määritelty.

2.3.3 Ei-toiminnalliset vaatimukset

Ei-toiminnalliset vaatimukset puolestaan käsittelevät muun muassa ohjelmiston suorituskykyyn, 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.)

Ei-toiminnalliset vaatimukset keskittyivät toteutettavassa ohjelmistossa pääasiassa kehitystyöhön ja ohjelmiston käyttövaatimuksiin. Kirjoitetun koodin tuli jokaisen repositoryyn tehdyn muutoksen jälkeen kääntyä, ja ohjelmiston testaamiseen tarkoitettun ympäristön tuli päivittyä automaattisesti. Myöhemmän vaiheen vaatimuksissa oli myös tuotantoympäristön automaattinen päivitys, kunhan päivitysketjua on ensin hiottu. Ohjelmiston käytön kannalta vaatimuksena oli uusimpien selainten tuki, mikä johtui kehitykseen valituista teknologioista. Ohjelmiston käytön tuli olla sujuvaa ja ulkoasun moderni.

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 itsekkin voidaan lukea ohjelmistoiksi, ja videotuotannossa erityisesti editoinnissa ja mahdollisten 3D-mallien parissa käytetään ohjelmistoja.

Kehitykseen osallistuminenkaan ei vaadi mitään virallista ohjelmistokehittäjän ammattia. Ohjelmistoja voivat kehittää niin ammattilaiset, tutkijat kuin harrastelijat. Ammatillisesta ohjelmistokehityksestä käytetäänkin usein termiä ohjelmistotuotanto (engl. software engineering) viitaten ammatillisen ohjelmistokehityksen luonteeseen kehittää ohjelmistoja toisen tahon tarpeita varten. Ohjelmistotuotantoa varten on kehitetty sitä tukevia menetelmiä ohjelmistojen suunnitteluun ja määrittelyyn kehitystyön tehostamiseksi. (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 lukeutuvat 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 tuki kipilari 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 määritelyihin 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 ollut 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 toteuttaa haluttu ominaisuus toisenlaisella ratkaisulla.

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 säästäten 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 tehtäväksi otettavat ominaisuudet, jotka kerätään sprint backlogiin. Nämä ominaisuudet tulee valita niin, että niiden toteutus onnistuu sprintille annetussa ajanjaksossa. Sprinteissä projektiryhmän jäsenillä on päivittäisiä lyhyitä tapaamisia, 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 voivat 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 tulee olla kaikille projektiryhmän jäsenille katsottavissa. 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 Sovelluskehykset

4.1 Yleistä

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

Sovelluskehykset toki nopeuttavat ja tietyllä tavalla helpottavat ohjelmistokehittäjien työtä, mutta erityisesti sovelluskehysten käyttöönotossa voi mennä huomattaviakin aikamääriä. Sovelluskehysiin sisältyy usein huomattava määrä valmista koodia, jota ohjelmistokehittäjän on opiskeltava ymmärtääkseen sovelluskehysten toiminnan

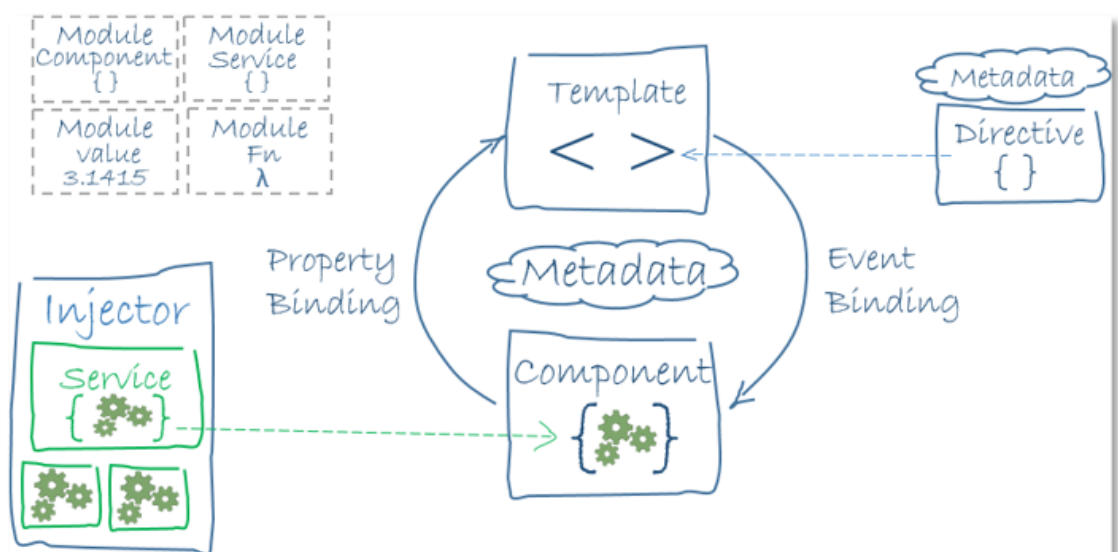
kunnolla ja osataksaan käyttää sen tarjoamia toiminnallisuuksia tehokkaasti. Käytettävään sovelluskehikseen onkin hyvä sitoutua pidemmäksi aikaa kehittäjän opittua sovelluskehiksen käyttöä paremmin.

4.2 Angular

Angular on web-kehittämiseen kohdistettu sovelluskehys, jonka tarkoitus on nopeuttaa ja helpottaa selainsovellusten kehittämistä. Angularia käytetään siis web-kehityksessä käyttäjälle näkyvien osien kehittämiseen. Yksi Angularin keskeisimmistä tavoitteista kehittämisen nopeuttamisen ja helpottamisen lisäksi on mahdollistaa kehitettyjen sovellusten käyttö niin verkossa kuin mobiili- ja työpöytälaitteilla. (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 käyvät ilmi 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, moduuleja. 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.) Kuviossa 2 on kuvattu Angularin moduuliluokan rakenne.

```
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 ja direktiivit
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.). Kuviossa 3 on kuvattu komponenttiluokan esimerkki.

```

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 Angularissa HTML-muotoinen mallitiedosto, 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 kuvion 4 alaosassa) 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. Palveluiden tavallisin rakenne on esitetty kuviossa 5.

```
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 sovelluskehys PHP:lle, jota käytetään web-kehityksessä palvelinsovellusten kehittämiseen. Angulariin eroten Symfonyllä kehitetyt ohjelmiston osat eivät siis ole suoraan loppukäyttäjälle näkyvissä. Symfonyssä kaikki osat koostuvat omista uudelleenkäytettävistä laajennoksistaan (Symfonyssä termi bundle), joita voi sisällyttää koodiin tarpeen mukaan. Laajennettavuutensa ansiosta Symfony sallii joustavuutta ohjelmointikehyksen 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 7 on esimerkki Symfonyn reitityksestä käyttäen *@Route*-määrettä.

4.3.2 Kontrollerit

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. Symfonyn mukana toimitetaan Twig-mallinnuskieli, jolla mallitiedostoista saadaan helpommin luettavia. (Symfony: templating n.d.)

```
<!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 (Symfony: templating n.d.)

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

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 erillisessä objektissa, 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 (Symfony: service container n.d.)

4.4 Doctrine

Symfonyssä itsessään ei ole tietokantojen käsittelyyn tarkoitettua komponenttia, vaan tähän tarkoitukseen on tarjolla useita eri laajennoksia ja kirjastoja, kuten Doctrine. 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. Symfony:n ja Doctrinen entiteettiluokka

Yksinkertaisen entiteettiluokan sisältö on kuvattu kuviossa 10. 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, tyyppin 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.)

Palvelinskriptaus

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

Komentoriviskriptaus

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 jäsennin. 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 tekemi-

nen 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 korjausehdotusten muodossa sekä koodin generointia. PhpStormin ominaisuuksia voi laajentaa sille kehitetyillä laajennusosilla, joita käyttäjät voivat kehittää itse omia tarpeitaan vastaaviksi. Esimerkki tällaisesta laajennuksista on tuki Symfony-ohjelmointikehykselle.

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 seurannassa 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 selaimiseen, virheiden seurantaan ja wikin ylläpitoon. Phabricator tukee Gitiä, Mercurialia ja Subversionia. (Phabricator 2017.)

Toimeksiantajalla Phabricatoria on käytetty projektien koodien säilytyspaikkana. Phabricatorissa koodin katselmointiin tarkoitetut työkalut ovat vastaavia palveluita paremmat ja projektien ryhmäoikeuksien hallitseminen on monipuolista.

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

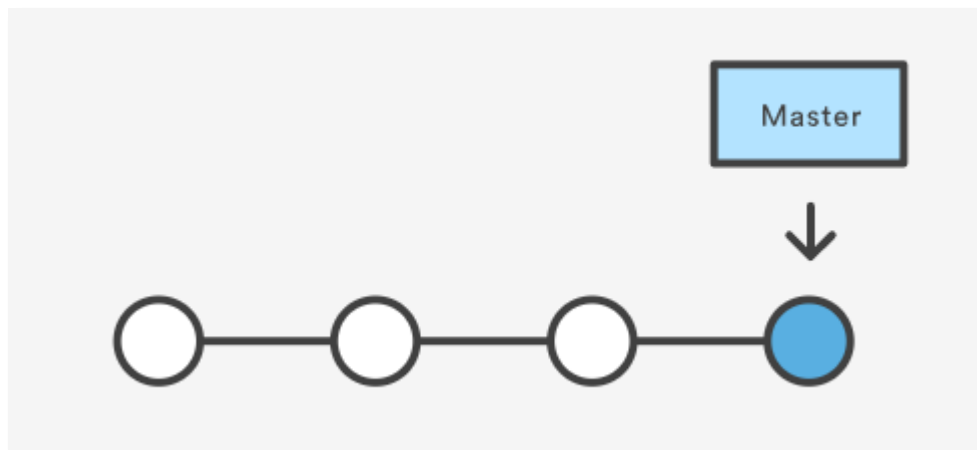
Gitin vahvuuksiin versionhallintajärjestelmänä kuuluu sen hajautettu arkkitehtuuri. Sen sijaan, että ohjelmiston koodi ja sen historia sijaitsisi vain yhdessä paikkaa, Gitiä käytettäessä jokainen kopio koodista sisältää kaikki koodiin tehdyt muutokset ja toimii omana repositorynään. Hajautettu arkkitehtuuri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

Branching-toiminto

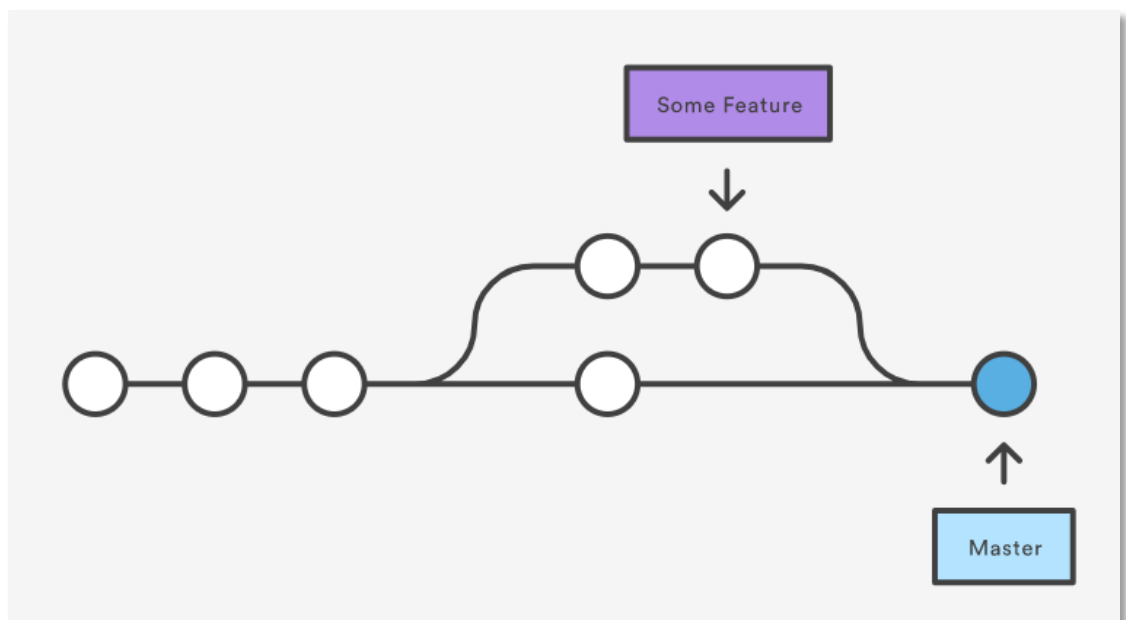
Toinen vahvuus on branching-toiminto (suom. haarautus). Branching-toiminnoissa ohjelmiston koodista luodaan oma itsenäinen kopio, branch (suom. haara). Branching-toiminto on erityisesti suurissa organisaatioissa ja suurten projektien kehityksessä hyödyllinen, sillä se mahdollistaa useamman ominaisuuden ja muutoksen kehittämisen samanaikaisesti: organisaatio voi esimerkiksi kehittää ohjelmiston uutta versiota yhdessä branchissa ja tehdä korjauksia vanhempaan versioon toisessa branchissa. 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 11 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 12). Brancheissa tehdyt muutokset saadaan master branchiin esimerkiksi merge requesting kautta. Merge request on pyyntö repositoryn hallitsijalle toisen branchin yhdistämisestä ja ennen merge requestin hyväksymistä yhdistettävään branchiin tehdyt muutokset katselmoidaan ja testataan toimiviksi. (What is Git n.d.)



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



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

Toimeksiantajalla Gitin nähdään muodostuneen ikään kuin standardiksi. Monipuolisten ominaisuuksien vuoksi Git on paljon käytetty versionhallintajärjestelmä toimeksiantajan projekteissa, ja toimeksiantajan työntekijöillä on hyvä osaaminen Gitin käytöstä omienkin projektien kautta.

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.)

Kutsujärjestelmässä MariaDB toimii pääasiallisena tiedon säilytyspaikkana. Tapahtumien ja niihin liittyvien toimijoiden välillä on havaittavissa selviä yhteyksiä, mikä itessään vei tietokantaratkaisun relaatiotietokantojen puolelle. MariaDB valittiin sen nopeuden ja edistyskellisyyden ansiosta, ja lisäksi se toimii hyvin Doctrinen kanssa.

5.4.3 PostgreSQL

PostgreSQL on avoimeen lähdekoodiin perustuva oliorelaatiotietokantojen 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.)

PostgreSQL:ää käytettiin kutsujärjestelmän rinnalle toteutetun PDF-järjestelmän tietovarastona. Luodut PDF-muotoiset kutsut tallennettiin PostgreSQL:ään myöhempää kutsujen tarkistusta ja raportointia varten.

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.)

Toimeksiantajalla Jira on pääasiallinen tehtävienhallintaohjelmisto projektien tehtävien hallintaan. Jira mahdollistaa useammalle projektille oman työtilan luomisen ja projektiin henkilöiden liittämisen. Erilaisten työnkulkujen tekeminen luominen on myös mahdollista, mikä antaa projektiryhmälle enemmän mahdollisuuksia oikeanlaisen työtilan luomiseen.

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) ja jokaisella ryhmällä voi olla useampi keskusteluhuone. (Slack (software) 2017.)

Slackia käytetään toimeksiantajalla sisäiseen viestintään. Slackin kanavat mahdollistavat projektiryhmille omien keskustelutilojen tekemisen ilman ulkopuolista häiriköintiä, ja tarvittaessa ulkopuolisia henkilöitä voidaan päästää toimeksiantajan Slack-ryhmään hetkellisesti.

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 korkealaatuisen 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 sähköpostien lähettämiseen.

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 tyylimäärittäyksiä. 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 tyylimäärittäyksiä 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.

The screenshot displays a web application for managing event organizers. On the left, a table titled 'Järjestäjät' (Organizers) lists existing entries. On the right, a form titled 'Järjestäjä 1' (Organizer 1) allows for creating a new entry.

Nimi	Yritystunnus	Osoite	Postinumero	Postitoimipaikka	Sähköposti
Järjestäjä 1	1234567-8	Katutiepolku 10	00000	Kaupunki	vesa.kivisto@protacon.com

Below the table, it indicates '1 valittu / 1 yhteensä' (1 selected / 1 total) and 'Rivejä sivulla: 10' (Rows per page: 10).

The 'Järjestäjä 1' form includes the following fields:

- Nimi** (Name): Järjestäjä 1
- Yritystunnus** (Company ID): 1234567-8
- Osoite** (Address): Katutiepolku 10
- Postinumero** (Postal Code): 00000
- Postitoimipaikka** (Post Office): Kaupunki
- Sähköposti** (Email): vesa.kivisto@protacon.com

At the bottom of the form, there are buttons for 'TALLENNUS' (Save), 'ESIKATSELE KUTSULIPPUI' (Preview Ticket), and 'POSTA' (Post).

Kuvio 13. Tapahtumajärjestäjien listaus

Tapahtumajärjestäjän luonti onnistuu lomakkeella (kuvio 13), 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 pyynnön tapahtumajärjestäjän luonnista palvelinsovelluksen kontrolleriin, joka varsinaisesti luo tapahtumajärjestäjän. Luonnin jälkeen palvelinsovellus palauttaa luodun järjestäjän tiedot vastauksena ja selainsovelluksessa listaus päivitetään. Tämä toiminnallisuus on kuvattu kuviossa 14.

```

public submit() {
    this.loading = true;

    let resource;

    if (this.id) {
        resource = this.organizerResourceService.update(this.organizerForm.value, {id: this.id});
    } else {
        resource = this.organizerResourceService.create(this.organizerForm.value);
    }

    resource
        .observable
        .finally(() => {
            this.loading = false;

            this.resourceListService.triggerUpdate();
        })
        .subscribe(
            next: (data: ResourceResult<Organizer>) => {
                this.messageService.simple(
                    this.translateService.instant( key: 'SAVED_SUCCESSFULLY')
                );

                // Update user access token
                this.refreshTokenService.updateAccessToken();

                if (!this.id) {
                    this.router
                        .navigate( commands: [data.id], extras: {relativeTo: this.activatedRoute})
                        .then( onfulfilled: () => { });
                }
            },
            error: (response: Response) => {
                const error = response.json() as ServerErrorInterface;

                this.messageService.simple(
                    this.translateService.instant( key: 'SAVE_FAILED', interpolateParams: {message: error.message})
                );
            }
        );
}

```

Kuvio 14. Tapahtumajärjestäjän luontilogiikka selainsovelluksessa

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 (kuvio 15). Kun tiedoston kaikki rivit on hyväksytty, tehdään jokaisesta tiedoston rivistä entiteetin luontipyyntö palvelinsovellukseen. Tiedoston tuonnin ja tarkistamisen hoitava logiikka on kuvattu kuviossa 16.

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 15. CSV-tuonnin virheilmoitus

```

.../**
... * Method to load a CSV file
... *
... * @param event
... */
... public loadFile(event) {
...     const reader = new FileReader();

...     this.csvFile = event.target.files[0];

...     reader.onload = () => {
...         const options = {
...             delimiter: ';',
...             trim: true,
...             skip_empty_lines: true,
...             skip_lines_with_empty_values: true,
...         };

...         parser(reader.result, options, (error, output: Array<Exhibitor>) => {
...             if (error) {
...                 this.invalidCsv = true;
...             } else {
...                 const header = output[0];

...                 // Check that CSV is in proper format
...                 if (header.toString() === this.header.toString()) {
...                     this.invalidCsv = false;

...                     this.parseRows(reader.result);
...                 } else {
...                     this.invalidCsv = true;
...                 }
...             }
...         });
...     };

...     reader.readAsText(this.csvFile, { encoding: 'UTF-8' });
... }

```

Kuvio 16. CSV-tiedoston tuonti ja tarkistus

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 yritettiin selvittää, mutta lopulliseksi ratkaisuksi päädyttiin ohjeistamaan käyttäjää 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 sähköpostitse tuli onnistua ohjelmistosta, mikäli kutsuttavalle oli määritelty 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.



Kutsujen toimitus sähköpostitse

Sähköpostin saateteksti valituille asiakkaille
Tässä olisi jotain järkevää tekstiä.

PERUUTA LÄHETÄ

Kuvio 17. Näkymä kutsujen lähettämiseen sähköpostilla

Kuviossa 17 on kuvattu kutsujen sähköpostitse toimittamiseen tarkoitettu dialogi. Dialogissa syötetään kutsujen sähköpostiin liitettävä teksti, jossa voidaan esimerkiksi kertoa kutsuvasta näytteilleasettajasta jotakin. Kuviossa 18 puolestaan on kuvattu palvelinsovellukselle sähköpostin lähetyspyynnön käsittelevä metodi.

```

/**
 * Method to submit form.
 */
public submit(): void {
  this.loading = true;

  let emails;

  if (this.query) {
    emails = this.participantResource.sendEmails(this.query, this.emailForm.value);
  } else if (this.entryCode) {
    emails = this.participantResource.sendSingleEmail(this.entryCode, this.emailForm.value);
  } else {
    emails = this.participantResource.sendEmail(this.participant.id, this.emailForm.value);
  }

  emails
    .finally(() => {
      this.loading = false;
    })
    .subscribe(
      (data: CountResponse) => {
        this.messageService.simple(
          this.translateService.instant( key: 'DIALOG_EMAIL_SUCCESSFULLY', data)
        );

        // This for user experience
        setTimeout( callback: () => {
          this.dialogRef.close( dialogResult: true);
        }, ms: 1000);
      },
      (response: any) => {
        const error = response.json() as ServerErrorInterface;


        this.messageService.simple(
          this.translateService.instant( key: 'DIALOG_EMAIL_FAILED', interpolateParams: {message: error.message})
        );
      }
    );
}

```

Kuvio 18. Palvelinsovellukselle lähetyspyynnön lähettävä koodi

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.


Tapahtuma 1, Järjestäjä 1
Lipuntarkastus

Sisäänkäynti
Etuovi

Lue tai syötä kutsun koodi

LUE KOODI

Kutsu hyväksytty

Kutsun tiedot
886AD4429F
Kutsuja
Kutsujayritys 1

Käyttötapahtumat

<small>Sisäänkäynti</small>	<small>Päivämäärä</small>
Etuovi	03.11.2017 08:59:27

Kuvio 19. Kutsujen lukunäkymä.

Kutsujen lukemiseen tarkoitetusta näkymästä tuli hyvinkin yksinkertainen (kuvio 19). 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.

Kutsun lukeminen palvelinsovelluksesta toteutettiin käyttämällä Angularin resolve-ominaisuutta, jolla datan haku tapahtuu ennen näkymän renderöintiä. Näkymän ”Lue koodi” -painikkeen painamisen jälkeen tehdään näkymän vaihto, jolloin URL-osoitteessa tapahtuu muutos ja kuviossa 20 kuvattu resolve-metodi ajetaan.

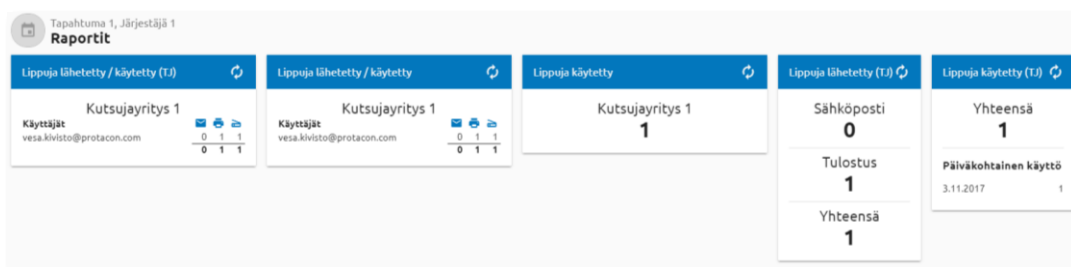
```
public resolve(  
  route: ActivatedRouteSnapshot,  
  state: RouterStateSnapshot  
) : Observable<Array<EntryCodeModel>> {  
  const query = {  
    where: {  
      code: route.paramMap.get('id'),  
      event: this.localStorage.retrieve( raw: 'eventId'),  
    },  
    populate: [  
      'EntryCode.code',  
      'EntryCode.participant',  
      'EntryCode.entryLogs',  
      'Participant.exhibitor',  
      'Exhibitor.name',  
      'EntryLog.createdAt',  
      'EntryLog.eventEntrance',  
      'EventEntrance.name'  
    ]  
  } as QueryInput;  
  
  return this.entryCodeResource  
    .query(query)  
    .$observable;  
}
```

Kuvio 20. Kutsun tietojen hakeminen palvelinsovelluksesta resolvella

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

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.



Kuvio 21. Raportointinäkymä.

Toteutettu raportointinäkymä on esitetty kuviossa 21. 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 21 lyhenne (TJ)), kuten kokonaisyhteenveto lippujen käytöstä päiväkohtaisesti.

Raportointinäkymän tiedot haetaan kutsujärjestelmän rinnalle toteutetusta raportointijärjestelmästä. Kuviossa 22 on kuvattu näytteilleasettajan toimitettujen kutsujen haku raportointijärjestelmästä. Metodissa tehdään GET-pyyntö ReportServicen *getExhibitorDeliveredData*-metodissa määritettyyn URL-osoitteeseen, ja mikäli pyyntö palauttaa dataa lajitellaan tämä data aakkosjärjestykseen. Raportointinäkymän muilla raporttietiedoilla on vastaavanlaiset metodit tietojen hakuun.

```
public updateExhibitorDelivered(): void {
    this.loadingExhibitorDeliveredData = true;

    this.reportService
        .getExhibitorDeliveredData(this.event.id)
        .finally(() => {
            this.loadingExhibitorDeliveredData = false;
        })
        .subscribe(
            next: (results) => {
                results.sort((a: any, b: any) => a.name < b.name ? -1 : ((a.name > b.name) ? 1 : 0));

                this.exhibitorDeliveredData = results;
            }
        );
}
```

Kuvio 22. Näytteilleasettajan toimitettujen kutsujen päivitys

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-sovel-
lus löytyy URL-osoitteesta *http://localhost:8000/*.

6.2.2 Käyttäjien autentikointi ja auktorisointi

Käyttäjien autentikoinnilla ja auktorisoinnilla on suuri merkitys ohjelmiston tietotur-
vassa. Käytännössä autentikointi tarkoittaa käyttäjän oikeellisuuden tarkistamista,
kun taas auktorisoinnilla 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 auktorisoinnin toteutukseen ohjelmistossa käytettiin toi-
meksiantajan 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 palvelinsovel-
lukselle, joka vertaa sitä konfiguroituun salaukseen. Mikäli tunniste täsmää konfigu-
roidun salauksen kanssa, käyttäjä auktorisoidaan ja hänelle annetaan kaksi uutta tun-
nistetta: refresh token (suom. päivitystunniste) ja access token (suom. pääsytun-
niste). Access token, jolla on erääntymisaika, kulkee käyttäjän palvelinsovellukselle
lähettämien kutsujen mukana. Refresh tokenia puolestaan käytetään access tokenin
uusimiseen sen vanhentuessa.

JWT-tunniste

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

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

Ensimmäinen osa

```
ew0KICAiYWxnIjogIkhTMjU2IiwNCiAgInR5cCI6ICJKV1QiDQp9
```

koostuu tunnisteen otsikkotiedoista. Otsikkotietoihin lukeutuu käytetty tunnisteen
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

1b5d3a636281978f7c766fb316218771d2c9b9db9f600f7d38a24fa008c49d61

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 RestController-luokan, jossa on määritelty yleiset kontrollerien vaatimat metodit.

Kuviossa 24 on nähtävissä uudelleenkäytettävä luontimetodi, joka on toteutettu traitina. Traitit ovat PHP:n 5.4-version mukanaan tuoma uudelleenkäytettävän koodin muoto. Traiteja voidaan sisällyttää luokkiin määrittelemällä käytettävät traitit luokan alussa, kuten kuviossa 23 on esitetty.

```

class OrganizerController extends RestController
{
    // Traits
    use RestAction\Exhibitor\Find;
    use RestAction\Organizer\FindOne;
    use RestAction\Organizer\Count;
    use RestAction\Organizer\Ids;
    use RestAction\Admin\Create;
    use RestAction\Organizer\Update;
    use RestAction\Admin\SoftDelete;
}

```

Kuvio 23. OrganizerControllerin traitit

OrganizerControllerin traitit ovat entiteettien luomiseen, muokkaamiseen ja poistamiseen määritellyjä metodeja. Koska toteutettavassa ohjelmistossa on usea eri käyttäjäryhmä omine oikeuksineen, tuli myös näillä metodeilla olla tieto niiden käyttöön oikeutettavasta käyttäjäryhmästä. Kuviossa 24 kuvatussa luontimetodin annotaatioissa on käytetty *@Security*-määrettä, jolle voidaan määrittää esimerkiksi käyttäjäryhmiin liittyviä vaatimuksia, kuten kuviossa on määritetty vaatimuksena ylläpitäjän käyttäjäryhmään kuuluminen.

```

trait Create
{
    use CreateMethod;

    /**
     * Create action for current resource.
     *
     * @Route("")
     * @Method({"POST"})
     * @Security("has_role('ROLE_ADMIN')")
     * @ApiDoc(resource=true)
     * @RestApiDoc
     * @throws \LogicException
     * @throws MethodNotAllowedHttpException
     * @throws HttpException
     * @param Request $request
     * @return Response
     */
    public function create(Request $request): Response
    {
        return $this->createMethod($request);
    }
}

```

Kuvio 24. Uudelleenkäytettävä luontimetodi

```

    public function createMethod(Request $request, array $allowedHttpMethods = null): Response
    {
        $allowedHttpMethods = $allowedHttpMethods ?? ['POST'];

        // Make sure that we have everything we need to make this work
        if (!$this instanceof RestController) {
            throw new \LogicException(
                'You cannot use App\Traits\Rest\Methods\Create trait within class that does not implement.' .
                'App\Controller\Interfaces\RestController interface.'
            );
        }

        if (!in_array($request->getMethod(), $allowedHttpMethods, strict: true)) {
            throw new MethodNotAllowedHttpException($allowedHttpMethods);
        }

        try {
            // Convert request content to DTO
            $dto = $this->getResponseService()
                ->getSerializer()
                ->deserialize(
                    $request->getContent(),
                    $this->getResourceService()->getDtoClass(),
                    format: 'json'
                );

            return $this->getResponseService()->createResponse(
                $request,
                $this->getResourceService()->create($dto),
                httpStatus: 201
            );
        } catch (\Exception $error) {
            if ($error instanceof HttpException) {
                throw $error;
            }

            if ($error instanceof OptimisticLockException || $error instanceof ORMInvalidArgumentException) {
                throw new HttpException(
                    statusCode: Response::HTTP_INTERNAL_SERVER_ERROR,
                    $error->getMessage(),
                    $error,
                    [],
                    code: Response::HTTP_INTERNAL_SERVER_ERROR
                );
            }

            throw new HttpException(
                statusCode: Response::HTTP_BAD_REQUEST,
                $error->getMessage(),
                $error,
                [],
                code: Response::HTTP_BAD_REQUEST
            );
        }
    }
}

```

Kuvio 25. Entiteetin luonnin käsittelevä metodi.

Luontimetodissa (kuvio 25) tarkastetaan aluksi, onko metodia kutsunut selainsovel-
lukselta tullut pyyntö käyttänyt sallittua HTTP-metodia sekä toteuttaako käytetty
kontrolleri RestController-luokan määrittymiset. Mikäli nämä molemmat täsmäävät, ei
poikkeuksia heitetä ja jatketaan try-catch -lauseeseen, jossa pyynnön sisältämä enti-
teetti serialisoidaan palvelinsovelluksen ymmärtämään DTO (data transfer object) -
muotoiseksi olioksi, luodaan ja tallennetaan entiteetti ja annetaan selainsovellukseen
vastaus onnistuneesta luonnista. Mikäli luonnin vaiheissa ilmenee jokin ongelmati-
lanne, heitetään siitä poikkeus ja entiteetin luonti lopetetaan.

6.2.4 Kutsujen lähetys ja tarkistus

Kutsujen lähetykseen sähköpostitse käytettiin Swift Mailer -nimistä kirjastoa, joka on sisäänrakennettu Symfonyyn SwiftmailerBundle-laajennoksena. Swift Mailerilla voidaan määrittää sähköpostin lähetykseen tarvittavat tiedot, kuten sähköpostin lähettäjä, vastaanottaja ja aihe sekä tarvittaessa liittää viestiin liitteitä.

```
/**
 * Method to process single participant ticket email.
 *
 * @param Participant $participant
 * @param string $emailText
 *
 * @return boolean
 *
 * @throws \RuntimeException
 * @throws \Doctrine\ORM\OptimisticLockException
 * @throws \Doctrine\ORM\ORMInvalidArgumentException
 */
public function processMail(Participant $participant, string $emailText): bool
{
    if ((string)$participant->getEmail() === '') {
        return false;
    }

    // Fetch common text for email
    $setting = $this->settingResource->findOne( ['id' => Setting::PDF_TICKET_EMAIL_CONTENT]);

    // Create body of email
    $body = $this->templating->render(
        [
            'name' => 'Emails/participant_email.twig',
            [
                'participant' => $participant,
                'emailText' => $emailText,
                'ticketText' => $setting !== null ? $setting->getValue() : '',
            ]
        ]
    );

    // Create new message
    $message = \Swift_Message::newInstance()
        ->setSubject($participant->getEvent()->getPdfEmailTitle())
        ->setFrom( ['addresses' => 'tuki@eventale.fi', $this->user->getFullName()] )
        ->setTo($participant->getEmail(), ['name' => $participant->getFirstName() . ' ' . $participant->getLastName()])
        ->setBody($body, ['contentType' => 'text/plain']);

    // Send message
    $this->mailer->send($message);

    // Create email send logs
    $this->participantResource->createEmailLog($participant);

    return true;
}
```

Kuvio 26. Sähköpostin lähetävä metodi

Kuviossa 26 kuvatussa metodissa tehdään sähköpostin lähetys. Lähetettävän sähköpostin tekstiosuuden luontiin käytetään Symfony'n templating-komponenttia, jonka *render*-metodilla voidaan luoda haluttu tekstiosuus antamalla metodille renderöitävä malli ja halutut muuttujat taulukkona. Tekstiosuuden luonnin jälkeen luodaan itse lähetettävä viesti *\Swift_Message::newInstance()*-metodilla, jolle voidaan määrittää tarvittavat tiedot sähköpostia varten. Metodin alussa oleva kenoviiva on tärkeä, sillä

se määrittää metodin kutsun globaalista nimiavaruudesta välttämällä sekaantumisen tiedostossa mahdollisesti määritetyn toisen nimiavaruuden samannimiseen metodiin.

6.2.5 Tietokannan hallinta

Palvelinsovelluksessa tietokannan hallinta toteutettiin Doctrinella. 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 27.

```

<?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 27. Esimerkki Doctrinen migraatitiedostosta.

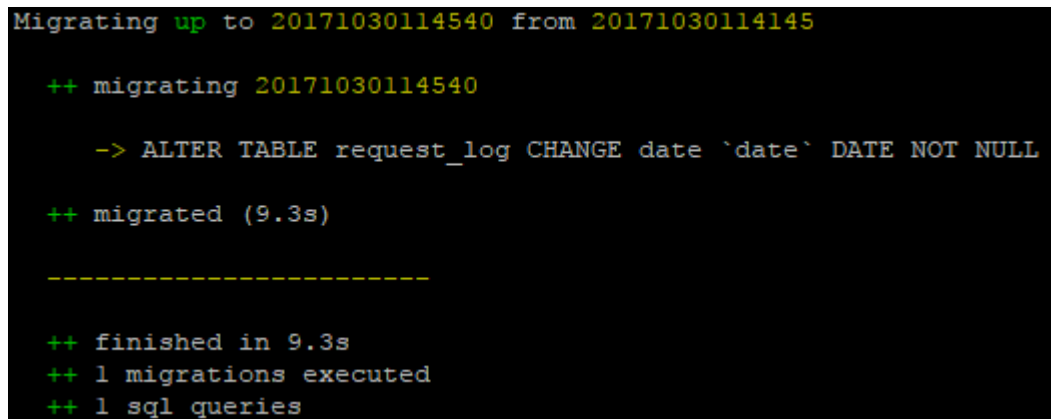
Varsinainen tietokannan rakenteen muutos tapahtuu komennolla

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

Komento suorittaa viimeisimmissä migraatitiedostoissa olevat *up()*-metodit vanhimmasta uusimpaan. Esimerkki tämän komennon suorittamisesta on kuviossa 28. Halutessaan komennossa voidaan käyttää esimerkiksi *prev*-lisämäärettä, joka suorittaa viimeksi ajatun migraatitiedoston *down()*-metodin. Vaihtoehtoisesti migraation suorittaminen onnistuu myös komennolla


```
php doctrine:migrations:exec migraatiotiedosto
```

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

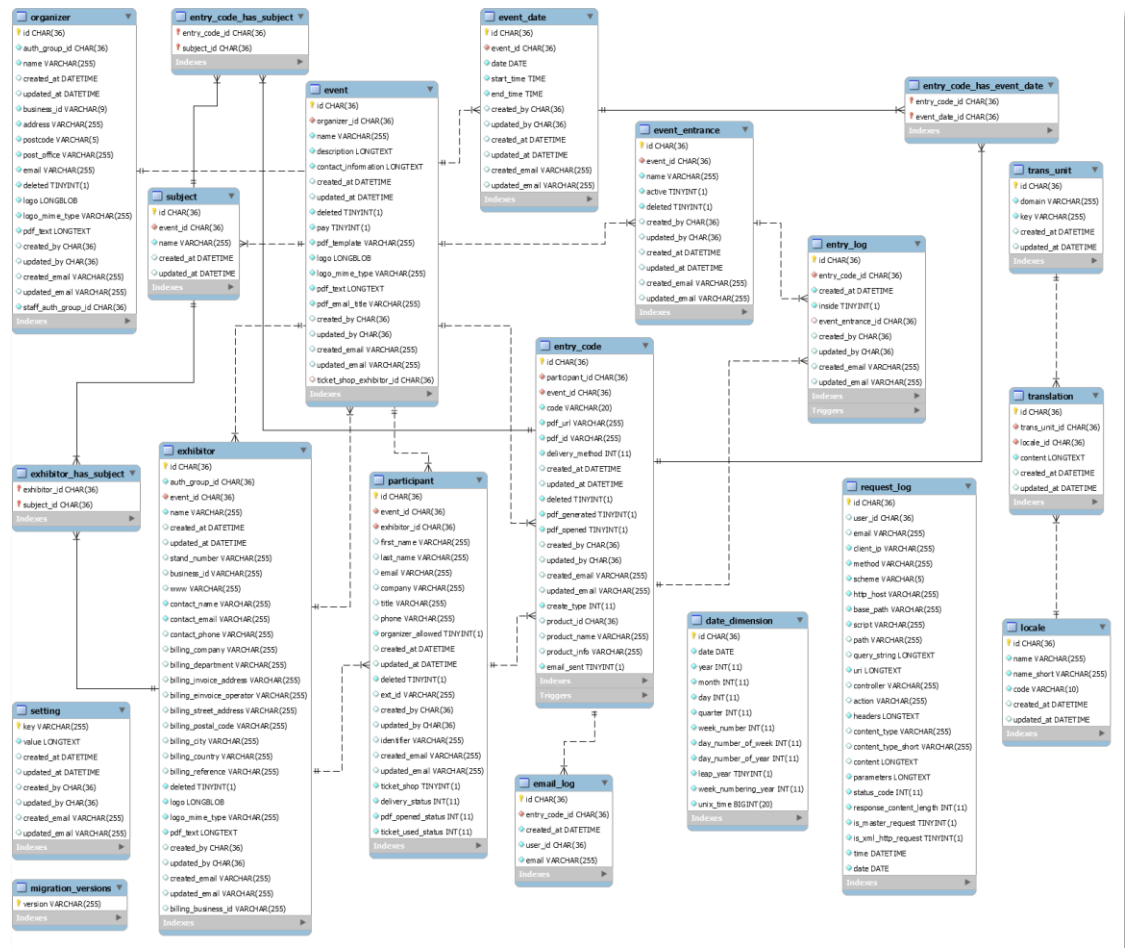
A terminal window with a black background and green text. The text shows the execution of a Doctrine migration command. It starts with 'Migrating up to 20171030114540 from 20171030114145', followed by '++ migrating 20171030114540'. Then it shows the SQL command '-> ALTER TABLE request_log CHANGE date `date` DATE NOT NULL'. This is followed by '++ migrated (9.3s)', a dashed line, 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 28. 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 29. Monimutkaisin taulu on tapahtumat tallentava *event*-taulu, jolla on suhteet kahdeksaan muuhun tauluun, kuten tapahtumajärjestäjän ja kutsukoodien tauluihin.



Kuvio 29. 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 kaksoiskappaletarkistukset 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 varsinainen 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. N.d. Esittely MariaDB:stä MariaDB:n kotisivuilla. Viitattu 28.10.2017.
<https://mariadb.org/about/>

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

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

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

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

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

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

IDE. 2017. Artikkel Wikipediassa ohjelmointiympäristöistä. Muokattu 25.10.2017. Viitattu 30.10.2017.
https://en.wikipedia.org/wiki/Integrated_development_environment

Jira (software). 2017. 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. N.d. Atlassianin artikkel Kanban-metodista. Viitattu 29.10.2017. <https://fi.atlassian.com/agile/kanban>

Me olemme Protacon. 2017. Protaconin yritys esittely. Viitattu 14.11.2017. <https://www.protacon.com/me-olemme-protacon/>

Phabricator. 2017. Artikkel Wikipediassa. Muokattu 8.10.2017. Viitattu 15.10.2017. <https://en.wikipedia.org/wiki/Phabricator>

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

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

PhpStorm. 2017. Wikipedia-artikkel JetBrains PhpStormista. Muokattu 28.9.2017. Viitattu 30.10.2017. <https://en.wikipedia.org/wiki/PhpStorm>

Protacon historia. 2017. Protaconin historiaa. Viitattu 14.11.2017. <https://www.protacon.com/historia/>

Protacon Solutions Oy. 2017. Kauppalehden yrityshaku. Viitattu 14.11.2017. <https://www.kauppalehti.fi/yritykset/yritys/protacon+solutions+oy/10023257>

Slack (software). 2017. Artikkelin 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. 2017. Artikkelin Wikipediassa sovelluskehiksestä. 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. N.d. Artikkelin Symfonyn kontrollereista Symfonyn kotisivuilla. Viitattu 7.10.2017. <https://symfony.com/doc/current/controller.html>

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

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

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

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

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

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

TypeScript. 2017. Wikipedia-artikkeli TypeScriptistä. Muokattu 20.9.2017. Viitattu 13.10.2017. <https://en.wikipedia.org/wiki/TypeScript>

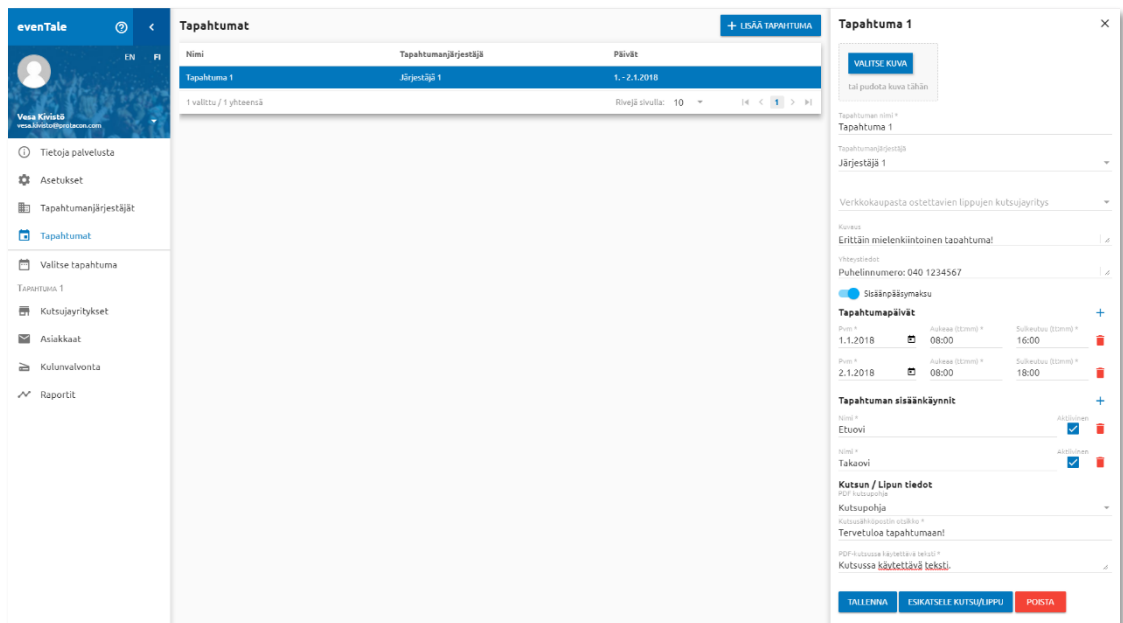
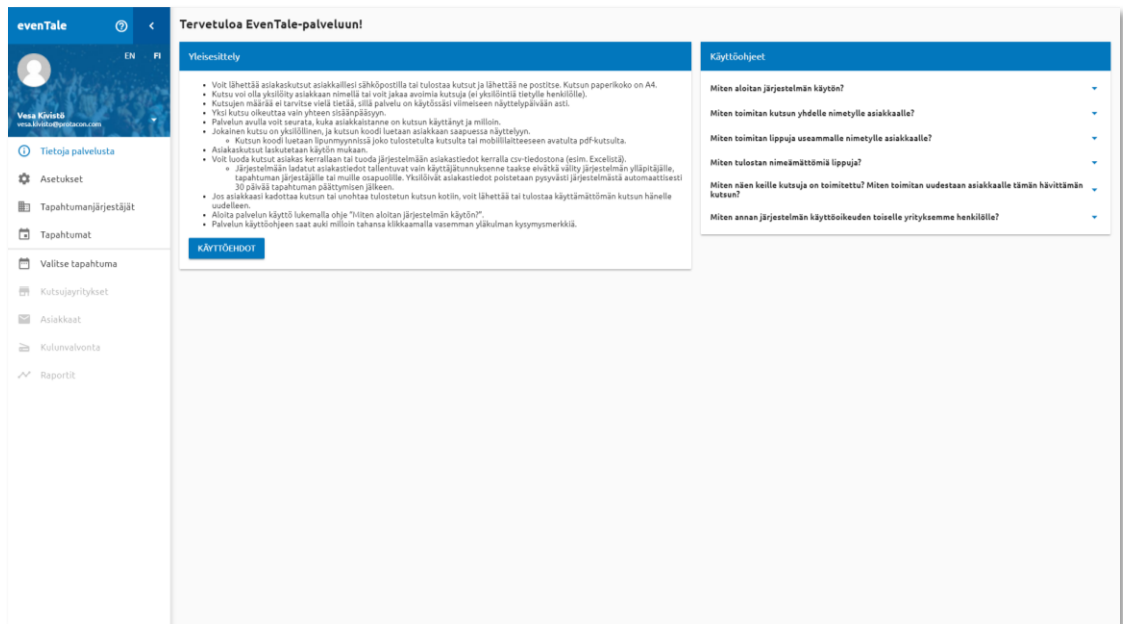
Using Branches. N.d. Artikkelin Gitinbranching-ominaisuuden käytöstä. Viitattu 4.11.2017. <https://www.atlassian.com/git/tutorials/using-branches>

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

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

Liitteet

Liite 1. Kuvakaappauksia selainsovelluksesta



evenTale

Tietoja palvelusta
Asetukset
Tapahtumanjärjestäjät
Tapahtumat
Valitse tapahtuma
TAPAHTUMA 1
Kutsujayritykset
Asiakkaat
Kulunvalvonta
Raportit

Tapahtuma 1, Järjestäjä 1

Kutsujayritykset

TUO CSV
+ LISÄÄ KUTSUJAYRITYS

Suodattimet

PALAUTA OLETUSSUODATTIMET

Hakusana

Nimi	Y-Tunnus	Osoitenumero	Lisätty
Kutsujayritys 1	1234567-8	A100	3.11.2017

1 valittu / 1 yhteensä

Rivejä sivulla: 10

Kutsujayritys 1

VALITSE KUVA

Lai pidetä kuva lähin

Yleiset tiedot

Nimi

Kutsujayritys 1

Y-tunnus

1234567-8

Osoitenumero

A100

WWW-osoite

www.osoite.com

Yritysosoite oma teksti kutsun

Kutsujayrityksen teksti kutsun

Yhteysthenkilö

Nimi

Vesa Kivistö

Sähköposti

vesa.kivisto@protacon.com

Puhelin

040 1234567

Laskutustiedot

Käyttäjät

TALLENNIA

ESIKATSELE KUTSU/LIPPU

POSTA

evenTale

Tietoja palvelusta
Asetukset
Tapahtumanjärjestäjät
Tapahtumat
Valitse tapahtuma
TAPAHTUMA 1
Kutsujayritykset
Asiakkaat
Kulunvalvonta
Raportit

Tapahtuma 1, Järjestäjä 1

Asiakkaat

TUO CSV
+ LISÄÄ KUTSUTTAJA

Suodattimet

PALAUTA OLETUSSUODATTIMET

Hakusana

Kutsujayritys

Sähköposti

Kutsun lähetystapa

Kutsujen lähetys

Kutsujen tulostus

Lippujen käyttö

Lipun luontitapa

Toiminnot kohdistuvat 4 asiakkaaseen

LÄHETÄ KUTSUA

TULOSTA KUTSUT

Etinimi	Sukunimi	Erätunniste	Kutsujayritys	Sähköposti	Puhelin	Yritys	Titteli
		Asiakaskutsut 1	Kutsujayritys 1				
Esa	Esimerkki		Kutsujayritys 1			Yritys	Edustaja
Vesa	Kivistö		Kutsujayritys 1	vesa.kivisto@protacon.com			
Matti	Meikäläinen		Kutsujayritys 1				

1 valittu / 4 yhteensä

Rivejä sivulla: 10

Matti Meikäläinen

Kutsujayritys

Kutsujayritys 1

Etinimi

Matti

Sukunimi

Meikäläinen

Erätunniste

Sähköposti

Puhelinnumero

Yritys

Titteli

Asiakas nro

Kutsut

Koodi

886AD4429F

Lähetetty

Tulostettu

Käytetty

3.11.2017

3.11.2017

TULOSTA KUTSUT

LISÄÄ KUTSU

POSTA