LAPIN AMK
Lapland University of Applied Sciences

OHJELMISTON ELINKAARI JA WEB-SIVUSTON KEHITTÄMINEN

Playful Lapland -hanke

Tuomas Tuokkola

Opinnäytetyö
Teollisuus ja luonnonvarat
Tieto- ja viestintätekniikka
Insinööri (AMK)

2018

LAPIN AMK
Lapland University of Applied Sciences

Opinnäytetyön tiivistelmä

Teollisuus ja luonnonvarat
Tieto- ja viestintätekniikka
Insinööri (AMK)

Tekijä Tuomas Tuokkola Vuosi 2018
Ohjaaja Aku Kesti
Toimeksiantaja Lapin yliopisto, Mediapedagogiikkakeskus
Työn nimi Ohjelmiston elinkaari ja web-sivuston kehittäminen
Sivumäärä 65

Opinnäytetyön tavoitteena on tarjota “Playful Lapland” -hankkeen henkilöstölle ja muille aiheesta kiinnostuneille tietoa ohjelmistotuotannon ja -kehityksen vaiheista ja menetelmistä ohjelmiston elinkaaren eri vaiheissa. Tämä on opinnäytetyön teoriapainotteinen osuus.

Opinnäytetyön toisena tavoitteena on käydä web-sivuston elinkaari läpi käytännössä eli lähtökohtaisesti määritellä, suunnitella ja toteuttaa järjestelmä ja dokumentoida käytetyt menetelmät ja tekniikat. Kehittämisvaiheen määrittely- ja suunnitteluluvuissa katsotaan käytännön tasolla toteutetut määrittelyt ja sivuston tekninen suunnittelu. Suunnitteluvaiheessa esittelen useiden yleisesti käytössä olevien kaaviotekniikoiden periaatteita, miten ja miksi kaavioita käytetään.

Tutkimuksen toteutusvaiheessa käydään läpi, miten sivusto rakennetaan Laravel -ohjelmistokehyksellä, mitä muita työkaluja prosessissa käytettiin ja miksi ohjelmistokehyksiä tulee käyttää web-sivustojen toteutuksessa. Opinnäytetyössä pureudutaan myös testauksessa, ylläpidossa ja jatkokehityksessä oleviin haasteisiin, mutta niistä ei ehditä dokumentoida tarvittavaa määrää tietoa opinnäytetyön toteuttamisen aikana. Tästä syystä testaus, ylläpito- ja jatkokehitysvaiheet sisältävät opinnäytetyön näkökulmasta tietoa tulevaisuudessa toteutettavista askelista sivuston kehityksessä.

Ideoita, mitä olisi voinut ja mitä olisi pitänyt tehdä toisin, käydään läpi pohdintaosiossa. Opinnäytetyössä toteutettu web-sivusto on lähtökohtana sivuston jatkokehityksessä, kun hanke etenee seuraavaan vaiheeseen.

Avainsanat 4+1-malli, Laravel, MySQL, ohjelmiston elinkaari, UML, vesiputousmalli, V-malli

LAPIN AMK
Lapland University of Applied Sciences

Abstract of Thesis

Technology, Communication and Transport
Degree Programme in Information and Communication Technology
Bachelor of Engineering

Author Tuomas Tuokkola Year 2018
Supervisor Aku Kesti
Commissioned by Lapland University, Center of Media Pedagogy
Subject of thesis Software Lifecycle and Development of Websites
Number of pages 65

The aim of this Bachelor's thesis was to provide information on the tools and methods of software production and development to the staff of the Playful Lapland -project and others interested in the various phases of the software life cycle. The second goal of the Bachelor's thesis was to discuss the life cycle of a web site in practice, namely to define, design and implement the system and document the methods and techniques used.

In requirements specification and technical design phases, the specifications and the diagrammes used were studied in practice. During the design phase, many commonly used diagram techniques were introduced. It was also discussed how and why the diagrams were used. In the development phase, it was looked upon how the site should be built using Laravel framework, what other software tools were used in the implementation of the site and why software frameworks should be used in the implementation of a website. The thesis also deals with the challenges of testing, maintenance and further development, but there was not enough time to document the required amount of information during the implementation of the thesis, so the testing, maintenance and further development phases include actions that will take place in the near future considering the development of the site.

Ideas of what might have been and what should have been done differently were discussed in the discussion section. The web site implemented in the thesis will be used for further development as the project progresses to the next stage.

Key words 4+1 Model, Laravel, MySQL, Software Development Lifecycle, UML, V-Model, Waterfall model

SISÄLLYSLUETTELO

1 JOHDANTO 10

2 TIETOJÄRJESTELMIEN KEHITTÄMISEN ELINKAARI 12

2.1 Ohjelmistojen elinkaarimallit 12

2.1.1 Vesiputousmalli 12

2.1.2 Ketterät menetelmät 14

2.2 Esitutkimusvaihe 15

2.3 Määrittelyvaihe 17

2.4 Suunnitteluvaihe 18

2.4.1 Moduulisuunnittelu 19

2.4.2 Oliosuunnittelu 20

2.4.3 Arkkitehtuurin suunnittelu 20

2.5 Toteutusvaihe 21

2.6 Testausvaihe 22

2.6.1 Testausmenetelmät kehitysvaiheessa 23

2.6.2 Testausmenetelmät ennen julkaisua 25

2.7 Käyttöönottovaihe 26

2.8 Ylläpitovaihe 26

3 TUTKIMUKSEN TOTEUTUS 28

3.1 Tutkimuksen määrittelyvaihe 28

3.1.1 Sivuston toimeksianto, tarkoitus ja tavoitteet 28

3.1.2 Organisaatiossa vallitseva nykytilanne 28

3.1.3 Kohderyhmän määrittely 29

3.1.4 Toiminnalliset vaatimukset 29

3.1.5 Ei-toiminnalliset vaatimukset 29

3.1.6 Rajoitteet ja reunaehdot 29

3.2 Tutkimuksen suunnitteluvaihe 30

3.2.1 Käyttötapauskaaviot 30

3.2.2 Järjestelmän toiminnallisuudet 33

3.2.3 Sivuston prosessinäkymä 38

3.2.4 Sivuston toteutusnäkymä 42

3.3 Toteutuksessa käytetyt tekniikat 43

3.3.1 Composer 43

3.3.2 MySQL 43
3.3.3 Laravel 44
3.3.4 PuTTY 58
3.4 Tutkimuksen testausvaihe 58
3.4.1 Testaus suunnitteluvaiheessa 58
3.4.2 Testaus kehitysvaiheessa 59
3.4.3 Testaus ennen julkaisua....59
3.4.4 Testauksessa käytettävät työkalut....59
3.5 Käyttöönotto-, ylläpito- ja jatkokehitys-vaihe 60
4 MUUT KÄYTETYT OHJELMISTOT, TEKNOLOGIAT JA TEKNIIKAT....62
5 POHDINTA 63
LÄHTEET....65

ALKUSANAT

Haluan kiittää Sinikkaa loputtomasta tukemisesta ja ymmärryksestä. Huomasin, että vastaus vaikeimpiin kysymyksiin löytyi usein irtautumisen kautta, mihin opinnäytetyötä tekevä oli henkilökohtaisesti kykenemätön. Vastaus löytyi kumminkin terävästä naukaisusta, joka herätti huomion ruokakipon pohjan näyttäytyessä.

Haluan kiittää Lapin ammattikorkeakoulun tieto- ja viestintätekniikan koulutus- ja hankehenkilöstöä ammatillisesta valmentamisesta, lämpimästä vastaanotosta ja ystävällisestä otteesta opiskelujeni aikana. Te olitte avain mieluiseen ja kehittävään kokemukseen.

Haluan kiittää Lapin Yliopistoa mahdollisuudesta työskennellä Playful Lapland –hankkeessa ja mahdollisuudesta toteuttaa opinnäytetyö hankkeessa toteutetun projektin pohjalta. Erityisesti haluan kiittää mediapedagogiikkakeskuksen hanke- ja tutkimushenkilöstöä.

KÄYTETYT TERMIT JA LYHENTEET

Back-end	Back-end -ohjelmointi on ohjelmointityötä, joka keskittyy sivuston toiminnallisuuksiin ja arkkitehtuuriin, joka ei ole loppukäyttäjälle nähtävissä.
Bootstrap	HTML ja CSS-pohjainen avoimen lähdekoodin sivustojen graafisen käyttöliittymän suunnittelutyökalu.
CSS	Yhdessä HTML:n kanssa Cascading Style Sheet -ohjelmointikieltä käytetään web-sivustojen ulkoasujen kehittämiseen.
Ctype	Ctype on C kielen standardikirjasto, josta löytyy useita funktioita merkkien testaamiseen ja kartoittamiseen.
FK	Foreign Key, viiteavain on taulun kenttä, jolla referoidaan toisen taulun pääavainta relaatiotietokannassa.
Front-end	Front-end -ohjelmoinnissa keskitytään sivuston visuaaliseen suunnitteluun ja toteuttamiseen.
GPL	General Public Licence on ilmaisten ohjelmistojen julkaisuun tarkoitettu lisenssi.
HTML	Hypertext Markup Language -ohjelmointikielellä suunnitellaan ja kehitetään web-sivustojen ulkoasuja.
HTTP	Hypertext Transfer Protocol -pyyntö on protokolla käyttäjän ja palvelimen välillä, jonka toiminta pohjautuu pyyntöihin ja vastauksiin.

IDE	Integrated Development Service tai ohjelmointiympäristö, on yksittäinen tai joukko ohjelmointia helpottavia ohjelmia.
JavaScript	JavaScript on ohjelmointikieli, jota yhdessä CSS:n ja HTML:n kanssa käytetään yleisesti web-sivustojen toteuttamiseen.
JSON	JavaScript Object Notation on standardi ohjelmointikieli tiedon siirtämiselle.
Mbstring	Multi-byte string functions on PHP:n laajennus, jolla luetaan ei-ASCII merkkijonoja.
MVC-malli	Model-View-Controller on ohjelmistoarkkitehtuuri malli.
Ohjelmistokehys	Framework, eli ohjelmistokehys on ohjelmointityötä vähentävä ohjelmisto, jonka avulla luodaan alustava järjestelmä, jonka päälle sovellus luodaan.
OpenSSL	OpenSSL on avoin ohjelmistokirjasto, joka toteuttaa SSL ja TLS protokollia.
PDO	PHP Data Objects on tietokantatyökalu, joka helpottaa useiden tietokantojen käyttöä.
PHP	Hypertext Preprocessor on yleisohjelmointikieli.
PK	Primary Key (pääavain) on taulun avain relaatiotietokannassa, joka erottaa taulun objektit toisistaan.
SDLC	Software Development LifeCyclellä tarkoitetaan ohjelmistokehityksen elinkaarta.

Slack	Slack on pilvipohjainen ohjelmisto, joka sisältää tiimityöskentelyssä käytettäviä työkaluja ja palveluja.
SSL	Secure Sockets Layer on salausprotokolla, jolla luodaan turvallinen yhteys käyttäjän ja palvelimen välille.
SQL	Structured Query Language on ohjelmointikieli, jota käytetään relatiivisten tietokantajärjestelmien ohjelmoinnissa ja suunnittelussa.
UML	Unified Modeling Language on eniten käytetty ohjelmistojen graafinen mallinnustekniikka.
URI	Uniform Resource Identifier on internet protokolla. URI on käytännössä merkkijono, jolla kuvataan resurssin tiedostopolku.
XML	Extensible Markup Language on merkintäkieli, jota käytetään dokumenttien koodaamiseen sekä ihmiselle, että tietokoneelle luettavaan muotoon.

1 JOHDANTO

Lapin Yliopiston Kasvatustieteiden tiedekunnan mediapedagogiikkakeskuksen “Playful Lapland”-hanke on palkannut opinnäytetyön tekijän kehittämään web-sivuston. Web-sivusto tukee hankkeen tavoitteita kouluttaa “Playful Learning”-metodia lähtökohtaisesti koulutustapahtumiin osallistuville vierailijoille. Web-sivuston tarkoituksena on mahdollistaa vierailijoiden tutustuttaminen opetusmetodiin ennen kuin he ovat tulleet koulutustapahtumaan Suomeen.

Aihe on hyvin moniulotteinen kysymys, sillä hankeen viime testaustilaisuudessa, jossa testattiin pelillistä sovellusta, edustettuna oli useita eri kansallisuuksia. Yhdeksi isoksi kysymykseksi nousi, miten voimme maksimoida tuotteen arvon tehokkaasti. Vierailijat tarkastelivat eri lähtökohdista tuotetta, sillä eri kulttuureissa on painotettu hyvin erilaisia lähestymistapoja sekä koulutukseen, että digitaalisten laitteiden käyttöön.

Valitsin tämän aiheen, sillä olen kiinnostunut ohjelmistosuunnittelusta. Olen myös huomannut opiskeluaikani aikana, että jokainen oma-aloitteinen projekti, oli se sitten mobiilipeli tai hallintatyökalu, joka ei ole ottanut tuulta alleen, on pohjimmiltaan ollut seurausta huonosta tai olemattomasta määrittelystä ja suunnittelusta.

Opinnäytetyössä käsittelen tietojärjestelmien kehittämisen elinkaaren eri vaiheita ja toteutan web-sivuston elinkaarimallin avulla. Opinnäytetyössä käydään läpi suunnittelutyökaluja ja –menetelmiä, joiden avulla sovellus suunnitellaan.

Olen myös kiinnostunut yrittäjyyden mahdollisuudesta, joten koin tärkeäksi ymmärtää ohjelmiston suunnittelun kannalta oleelliset seikat ja työmenetelmät. Opinnäytetyössäni keskityn ohjelmistojen elinkaaren eri vaiheiden dokumentointiin teoriassa ja käytännössä. Aion myös kertoa asioista, jotka aiheuttivat eniten ongelmia tässä projektissa, joten tämän opinnäytetyön voi tässä mielessä ajatella oppimiskokemuksena.

Opinnäytetyössä esitellään myös käytettyjä web-kehityksen työkaluja. Toteutusvaiheessa tutustutaan käytettyihin työkaluihin ja käytännön tasolla käydään läpi sivustolle toteutettuja toimintoja.

Opiskeluaikana opiskelimme ohjelmistotekniikan, web-teknologioiden ja olio-ohjelmoinnin perusteita, joista oli huomattavaa hyötyä opinnäytetyön toteutusvaiheessa. Olen ollut harjoittelussa opiskeluaikanani pLab-ohjelmistolaboratoriossa, joka on Lapin Ammattikorkeakoulun ohjelmistotekniikan laboratorio. Harjoittelujen aikana harjoittelin CakePHP-ohjelmistokehyksellä web-sivustojen kehittämistä, joka oli merkittävin yksittäinen oppimiskokemus ennen hankkeen starttaamista.

2 TIETOJÄRJESTELMIEN KEHITTÄMISEN ELINKAARI

IT-organisaatiolla tulee olla huolellisesti suunniteltu toiminta-ajatus, johon voidaan asettaa prosesseja, toimintaperiaatteita ja toiminnan suuntauksia. Ohjelmistokehityksen elinkaari (SDLC, Software Development LifeCycle) määrittää tietojärjestelmien rakentamiselle toistettavan mallin. Elinkaari on kehitetty vastaamaan liiketoiminnan asettamiin tarpeisiin ohjelmistoteollisuudessa. Elinkaari oikein käytettynä minimoi järjestelmän kehityksessä esiintyviä riskejä, vähentää turhaa toimintaa ja lisää tehokkuutta. (Murch 2002, 13.)

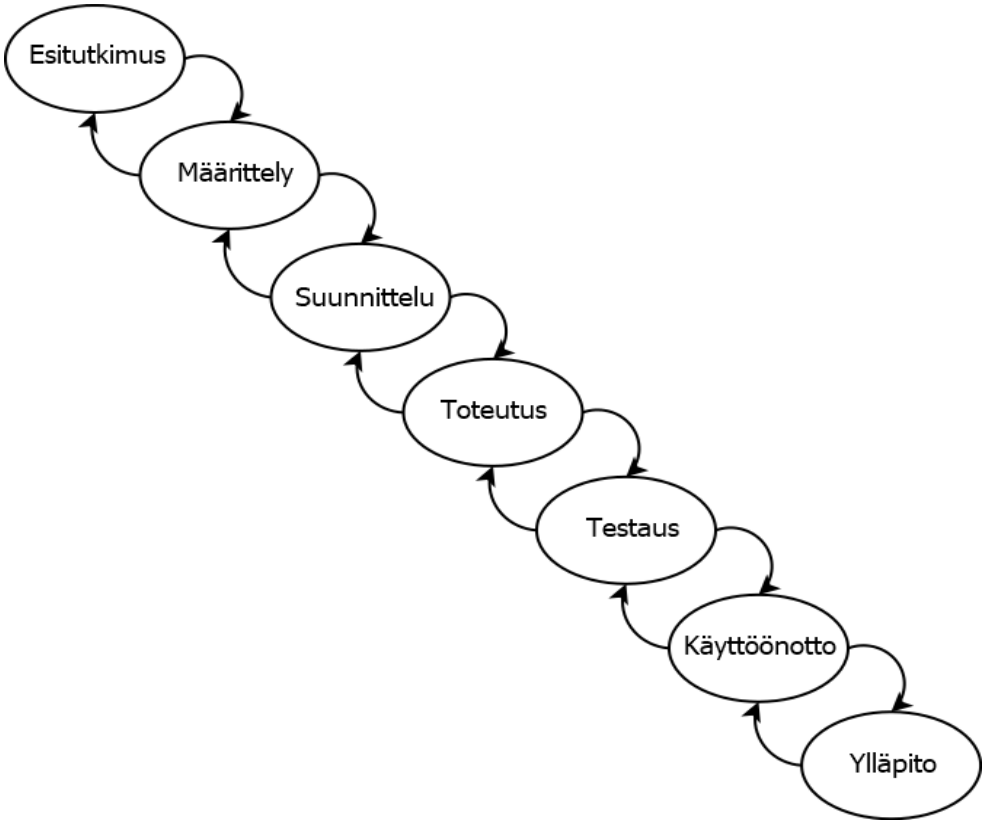
Opinnäytetyössä esitetty ohjelmistoprosessin vaihejako on akateeminen malli ohjelmistoprosessin läpiviennille. Malli ei ota kantaa esimerkiksi siihen, että usein vaiheet menevät ainakin osittain päällekkäin ja joskus vaiheet voivat toteutua samanaikaisesti. Tietojärjestelmien kehittäminen pitää sisällään useita elinkaaren läpi jatkuvia aktiviteetteja, joihin lukeutuvat muun muassa vaatimustenhallinta, laadunvarmistus, dokumentaatio ja riskien hallinta. (Pohjonen 2002, 39 – 40.)

2.1 Ohjelmistojen elinkaarimallit

Ohjelmistojen elinkaarimalleja on useita, joista esittelen vesiputousmallin ja ketterät menetelmät, sillä käytin vesiputousmallia ja ketteriä menetelmiä tutkimuksen toteuttamisessa. Toinen syy vesiputousmallin esittelylle on sen asema käytetyimpänä elinkaarimallina ohjelmistojen kehittämisessä. Useimmat ohjelmistokehitysmallit ovat vesiputousmallin johdannaisia.

2.1.1 Vesiputousmalli

Vesiputousmalli (Kuvio 1.) julkaistiin 1970-luvun alussa. Se suunniteltiin perinteisten prosessimallien pohjalta. Vesiputousmallissa esitetään kehittämisen vaiheet aloittamalla tuotteen elinkaari esitutkimuksesta ja saattamalla tuote lopulta ylläpitovaiheeseen. (Pohjonen 2002, 40.)



Kuvio 1. Vesiputousmallin vaiheet (Haikala & Mikkonen 2011, 37)

Vesiputousmalliin kuuluu iterointi taaksepäin mallin avainominaisuutena, mikä on tärkeää muistaa, sillä vesiputousmalli on usein väärinymmärretty ja yksinkertaistettu malliksi, jossa ei iteroida. Tarkastellessa tuoreempia projektimalleja, löytää niistä yleensä vesiputousmallin alkuperäisen idean tavalla tai toisella toteutettuna. (Haikala & Mikkonen 2011, 37.)

Olen valinnut edellisessä kappaleessa mainitusta syystä vesiputousmallin lähestymistavan opinnäytetyön pohjaksi. Vesiputousmalli myös soveltuu hyvin hankkeeseen, sillä toteutettava web-palvelu sisältää paljon työtä melko pitkällä aikavälillä ja vesiputousmalli on hyvin hallittu kokonaisuus, jota käytetään paljon teollisuudessa ja isommissa hankkeissa. Toteutuksessa käytän vesiputousmallia pohjana, mutta tuen ketteristä menetelmistä tulleilla menetelmillä projektin etenemistä, sillä toteuttava tiimi on pienikokoinen ja liiallisella määrittelyllä ja suunnittelulla ei tässä tapauksessa saavuteta tavoitteita.

Vesiputousmallin yksi keskeisimmistä ongelmista on se, että mallissa tarkastelujaksot ja dokumenttien katselmoinnit asetellaan vaiheiden rajapinnoille, jolloin prosessissa oletetaan automaattisesti aiemman vaiheen dokumentaation olevan syöte seuraavalle vaiheelle. Toinen mallin ongelma on se, että asiakkaalle pystytään esittämään tuote varsin myöhäisessä vaiheessa suhteessa esimerkiksi verrattaessa ketteriin menetelmiin tai prototyyppipohjaisiin malleihin. (Pohjonen 2002, 40.)

2.1.2 Ketterät menetelmät

Yleisin käyttöön otettu ketterä menetelmä on **Scrum**. Scrumin merkittävä etu on sen yksinkertaisuus. Periaatteet on helppo selittää nopeasti ja se on hyvin skaalautuva malli. Mallissa ei kuitenkaan oteta kantaa käytettyihin kehitysmenetelmiin ja työkaluihin. Scrum ei käytännössä ole projektinhallintamenetelmä, vaan enemmänkin tapa toteuttaa projekti iteraatioiden avulla. (Haikala & Mikkonen, 46 – 51.)

Scrumissa projektin eri osapuolet on jaettu kolmeen osaan. **Tuotteen omistajaan**, **Scrum-mestariin** ja **tiimiin**. Tuotteen omistaja vastaa taloudellisesta puolesta. Scrum-mestari vaatii projektilta prosessin noudattamista, huolehtii projektin asiakirjojen päivittämisen ja vastaa työtä haittaavien esteiden poistamisesta. Tiimin optimaalinen koko on seitsemän kokopäiväistä työntekijää, joilla kaikilla on omat tehtävänsä projektista riippuen. Tiimi toimii tehtävätaulun avulla, johon päivitetään tehtävät ja niiden tilat. Scrumissa projekti etenee pyrähdyksinä (sprint), joiden pituus on 30 kalenteripäivää. (Haikala & Mikkonen, 46 – 51.)

Pyrähdys aloitetaan suunnittelukokouksella. Kokoukseen osallistuvat kaikki osapuolet. Kokouksessa ensimmäisenä toteutetaan tuotteen työlista (product backlog) ja sen jälkeen tuotteen tehtävälistan alkiot (Product backlog item). Tehtävälistan alkiot puretaan tehtäviin (task), joiden kestoksi koitetaan saada 4-16 tuntia. Tehtävälistaa ei enää tämän jälkeen muuteta. Suunnitellut tehtävät toteutetaan seuraavan pyrähdyksen aikana. Tilannetta seurataan pyrähdyksen etenemiskäyrän (burndown chart) avulla. (Haikala & Mikkonen, 46 – 51.)

Pyrähdyksen aikana päivittäin pidetään Scrum -kokous (daily scrum), jonka kesto on noin 15 minuuttia. Pyrähdyksen päätyttyä pidetään katselmointikokous, johon kutsutaan myös mahdollisia asiakkaita ja demotaan tulokset tiimin puolesta, ja

sen jälkeen pidetään pyrähdyksen **arviointipalaveri** (retroperspective), johon osallistuvat tuotteen omistaja, Scrum-mestari ja tiimi. Arviointipalaverissa mietitään, mikä meni hyvin ja millä projektin osa-alueilla olisi tapahduttava parannusta tulevaisuudessa. (Haikala & Mikkonen, 46 – 51.)

2.2 Esitutkimusvaihe

Esitutkimuksessa selvitetään, onko tietojärjestelmän rakentaminen mahdollista yritykselle. Esitutkimuksessa ei ohjelmoida mitään eikä tehdä teknisiä ratkaisuja esimerkiksi laitteiston tai teknologioiden suhteen, vaan selvitetään, miksi järjestelmä tulisi rakentaa ja mitkä ovat järjestelmän tavoitteet. (Pohjonen 2002, 27.)

Esitutkimuksen tulisi sisältää ainakin seuraavat asiat:

- organisaation tietojenkäsittelyn nykytilanteen kuvaaminen siltä osin kuin se liittyy käsillä olevaan kehityshankkeeseen
- niiden ongelmien kuvaukset, joihin järjestelmän tuo ratkaisut
- kuvaukset niistä viite- ja sidosryhmistä, joita hanke koskee
- alustavien järjestelmän tavoitteiden ja rajausten määritykset
- uuden järjestelmän kehittämistavoitteiden määritykset
- eri toimintavaihtoehtojen kuvaukset perusteluineen ja
- alustava toimintasuunnitelma hankkeen läpiviemiseksi.

Esitutkimus yleensä tuottaa runsaasti erilaista aineistoa, joka kannattaa tallentaa myöhempää käyttöä varten. (Pohjonen 2002, 27.)

Haikala kirjoittaa teoksessaan ohjelmistotuotannon käytännöt seuraavasti:

“Monet tutkimukset osoittavat, että ohjelmistoprojektien epäonnistumisen syyt juontavat juurensa yli 60-prosenttisesti huonoon vaatimustenkäsittelyyn. Kunnolla hoidettu vaatimustenkäsittely on yksi onnistuneen ohjelmistoprojektin perusedellytyksiä. “(Haikala & Mikkonen 2011, 61.)

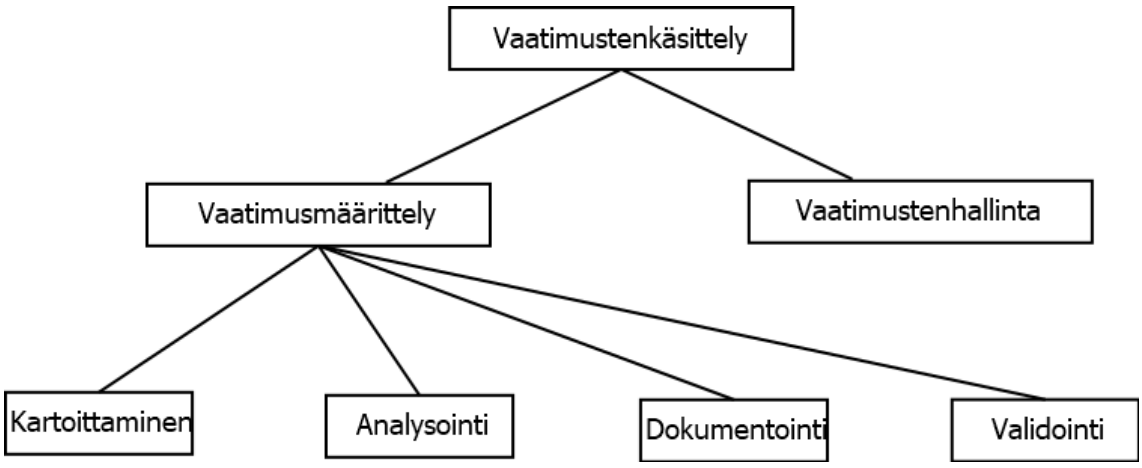
Vaatimustenkäsittelyssä käsitellään vaatimuksia, jotka kuvaavat, mitä tuotteella pystyy tekemään, tai ovat ominaisuuksia, joita tuotteella tulee olla. Vaatimukset luokitellaan toiminnallisiin ja ei-toiminnallisiin vaatimuksiin ja reunaehtoihin. Toiminnallinen vaatimus on esimerkiksi, että sivustolle on pystyttävä lähettämään

multimediasisältöä. Ei-toiminnallinen vaatimus puolestaan voi olla esimerkiksi, että käyttöliittymä on määritelmän mukainen. Reunaehto on esimerkiksi, että ohjelmisto on toteutettava C#-ohjelmointikielellä. (Haikala & Mikkonen 2011, 61.)

Vaatimustenhallinnassa tekstistä voidaan yleensä päätellä, että verbit, joita sivulla on, ovat toiminnallisia vaatimuksia. Substantiivit tekstissä puolestaan mielletään yleensä ei-toiminnallisiksi vaatimuksiksi. Vaatimukset lähtevät liiketoiminnan tarpeista, ja yleensä niiden tärkein lähde ovat asiakas ja ohjelmiston tulevat käyttäjät.

Perusominaisuuksia hyvälle vaatimukselle ovat virheettömyys ja selkeys. Muita hyvän vaatimuksen ominaisuuksia ovat tarkkuus ja ymmärrettävyys, jotta vaatimuksen täyttyminen voidaan mitata, sitä on pystyttävä mittaamaan. Muita kysymyksiä, kuten onko vaatimus täytetty, tai onko se jäljitettävissä taaksepäin, tulee myös kysyä säännöllisesti vaatimustenkäsittelyn aikana. Myöhemmin projektissa on voitava selvittää, mistä vaatimus on peräisin ja on voitava jäljittää, mikä on vaatimuksen tekninen toteutus ja mitkä testitapaukset testaavat sen täyttymisen. (Haikala & Mikkonen 2011, 64.)

Vaatimustenkäsittely voidaan jakaa kahteen eri osa-alueeseen, vaatimusmäärittelyyn ja vaatimustenhallintaan (Kuvio 2.). Vaatimusmäärittely voidaan edelleen jakaa neljään osaan, joita ovat kartoittaminen, analysointi, dokumentointi ja validointi. (Haikala & Mikkonen 2011, 65.)



Kuvio 2. Vaatimustenkäsittelyn osa-alueet (Haikala & Mikkonen 2011, 65)

Vaatimusmäärittely on dokumentti, johon on koottu hankeen toimeksiannon kuvaus ja kehitettävän järjestelmän eri sidosryhmien järjestelmälle asettamat vaatimukset, jotka on kategorisoitava toiminnallisiin ja ei-toiminnallisiin vaatimuksiin ja reunaehtoihin. Vaatimukset määrittelevät eri sidosryhmien tarpeet järjestelmän suhteen, mutta tässä vaiheessa ei huomioida teknistä toteutusta. Perinteinen tapa kerätä vaatimuksia on eri sidosryhmien edustajien haastattelu, sillä ei ole olemassa yhtä yksittäistä vaatimusten keräysmenetelmää, joka takaisi joka tilanteessa riittävän kattavan lopputuloksen. (Pohjonen 2002, 28.)

Vaatimustenhallinnan katsotaan sisältävän kirjanpidon vaatimusten tilasta, vaatimusten välisten riippuvuuksien seurannan ja niistä raportoinnin. Pienissä projekteissa vaatimustenhallinta hoidetaan usein hyvin kevyesti. Vasta projektin päättyessä siirrytään muodollisempaan muutostenhallintaan ylläpitovaiheessa. (Haikala & Mikkonen 2011, 66 – 67.)

2.3 Määrittelyvaihe

Määrittelyllä tarkoitetaan hankkeen vaatimusten dokumentointia asiakkaan näkökulmasta. (Haikala & Mikkonen 2011, 30).

Määrittelyvaiheessa tärkein määrittelytapa on luonnollinen kieli. Jos tarkkuutta ei saada tarpeeksi luonnollisen kielen avulla, tarvittaessa erilaisten kaavioiden käyttö saattaa tehostaa kommunikointia ja yksinkertaistaa yhtenäistä näkemystä järjestelmästä. (Haikala & Mikkonen 2011, 69 – 70.)

Luonnollista kieltä käytettäessä määrittely aloitetaan karkean tason määrittelystä. Asiakkaita on erilaisia, ja määrittelyistä ovat yleensä kiinnostuneita järjestelmää käyttävät työntekijät, tietohallintopäällikkö ja pääkäyttäjä. (Haikala & Mikkonen 2011, 70 – 71.)

Määrittelyn pohjimmainen tarkoitus on muodostaa ensimmäinen sopimus asiakkaan ja toteuttajan välille, joten tässä vaiheessa tulisi dokumentoida mitä tarkkaan ottaen ollaan tekemässä, mutta dokumentaation ei tule sisältää toteutuksessa käytettäviä tekniikoita, paitsi niiltä osin kuin tekniikat asettavat reunaehtoja toteutukselle. (Haikala & Mikkonen 2011, 70 – 71.)

Määrittelydokumentaatio vanhenee nopeasti, eikä kaikkea voi kustannustehokkaasti ylläpitää. Määrittelydokumentaatiosta saatava hyöty lakkaa ylläpitovaiheessa, jolloin järjestelmästä on saatavilla paljon täsmällisempää teknistä kuvausta. Määrittelydokumentaatiota ei tule ylläpitää ensimmäisen version toteutusta pidemmälle ainakaan täydellisenä. (Haikala & Mikkonen 2011, 70 – 71.)

2.4 Suunnitteluvaihe

Haikala tiivistää kirjassaan Ohjelmistotuotannon käytännöt suunnitteluvaiheen seuraavasti:

“Ohjelmistosuunnittelulla tarkoitetaan määrittelyn mukaisen järjestelmän teknistä suunnittelua.”

Keskeisimpänä suunnitteluvaiheen ongelmana hän pitää järjestelmän vaatimusten ja käytettävissä olevan toteutusteknologian välisen kuilun ylittämistä. (Haikala & Mikkonen 2011, 30.)

Ohjelmistosuunnittelu on kehitetty täyttämään tämä kuilu teknisen suunnittelun avulla, joka sisältää erilaisia diagrammeja kuvaamaan määrittelyjen pohjalta luotavaa teknistä toteutusta. Olennainen osa teknistä suunnittelua on myös sivuston arkkitehtuurin jakaminen pienempiin osiin. (Haikala & Mikkonen 2011, 177 – 178.)

Suunnittelun tärkein tehtävä on määritellä ohjelmistossa tarvittavat luokat tai oliot sekä niiden väliset yhteydet. Teknisen suunnittelun dokumentaatiossa tulee olla seuraavista osa-alueista kuvaukset:

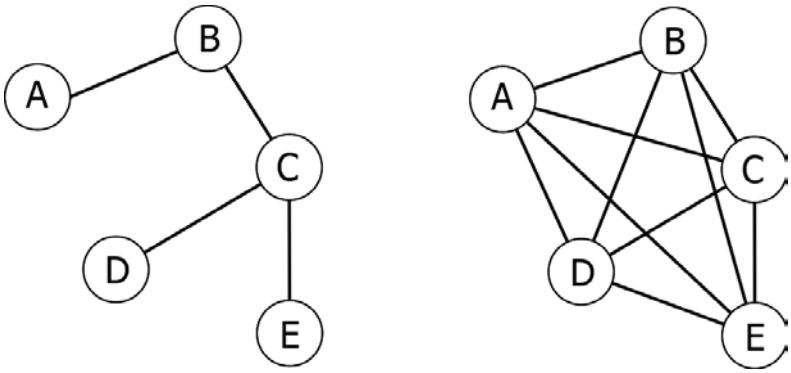
- Järjestelmän tarkoitus
- Järjestelmän sovellusalue ja järjestelmän osuus siinä
- Järjestelmän laitteisto ja ohjelmistoympäristö
- Järjestelmän toteutuksen keskeiset reunaehdot
- Järjestelmä ja sen ympäristön välinen vuorovaikutus
- Järjestelmän arkkitehtuuri
- Kaikki järjestelmän moduulit ja alijärjestelmät
- Toteutuksessa olevat rajoitteet

- Mahdolliset vaihtoehdot tai hylätyt ratkaisut ja
- Mahdolliset muut toteutukseen vaikuttavat seikat. (Pohjonen 2002, 33 – 34.)

2.4.1 Moduulisuunnittelu

Osia järjestelmästä kutsutaan moduuleiksi. Moduuli on kokonaisuus, johon kuuluvat tietyn järjestelmän toiminnot ja rajapinta, jonka avulla se on yhteydessä muihin moduuleihin. (Pohjonen 2002, 32.)

Suunnittelussa tavoitteena on rakentaa järjestelmä niin, että sen komponentit ovat mahdollisimman vähän riippuvaisia toisistaan (Kuvio 3.), jolloin komponenttiin tehdyt muutokset eivät vaikuta muihin komponentteihin. (Pohjonen 2002, 32.)



Kuvio 3. Esimerkki järjestelmän monimutkaistumisesta (Pohjonen 2002, 32)

Toiseksi suunnittelussa tulee kiinnittää huomio moduulin ja alimoduulien väliseen kommunikointiin. Moduulilla tulee olla maksimissaan muutama alimoduuli ja jos moduuliin vaikutetaan toisen moduulin kautta, tulisi sen olla aina alimoduuli vaikuttavalle moduulille. (Pohjonen 2002, 32 – 33.)

Moduulisuunnittelussa suunnitellaan jokaisen moduulin sisäinen rakenne. Avaintavoite suunnittelussa on pyrkiä mahdollisimman pieneen moduulikokoon, sillä suuret moduulit ovat vaikeita toteuttaa ja monimutkaisia ylläpitää. (Pohjonen 2002, 32 – 33.)

2.4.2 Oliosuunnittelu

Oliosuunnittelussa pätevät samat perusperiaatteet kuin moduulisuunnittelussa, mutta terminologiassa on eroja. Oliosuunnittelun pääperiaatteena on hahmottaa rakennettava järjestelmä joukkona olioita, jotka ovat vuorovaikutuksessa keskenään.

Olio on ohjelman perusyksikkö, joka voi pitää tietoa sisällään ja se voi olla toiminnallinen. Olion tietokenttiä kutsutaan attribuuteiksi, jotka voivat olla julkisia, suojattuja tai yksityisiä. Olion toiminnallisuuksia kutsutaan metodeiksi, jotka ovat olion tai muiden olioiden julkisten attribuuttien tietoja käsitteleviä operaatioita.

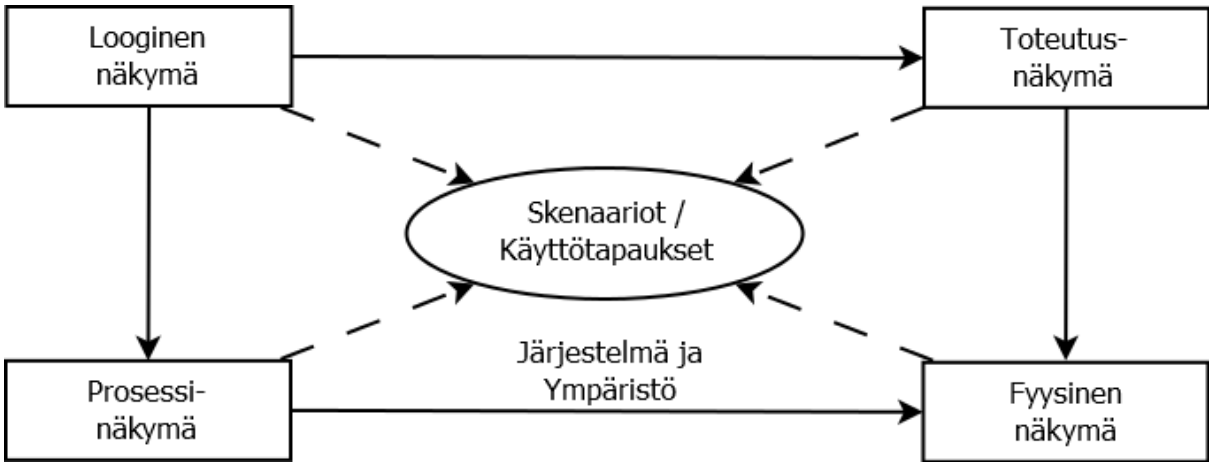
Oliot kuuluvat luokkiin, jotka puolestaan määrittelevät, mitä attribuutteja ja metodeja luokkaan kuuluvalla oliolla on. Vain attribuuttien arvot vaihtelevat.

Oliomallinnuksessa keskitytään järjestelmän staattisten rakenteiden ja käyttäytymisen mallintamiseen, jossa erilaiset kaaviot, kuten käyttö-, luokka- ja sekvenssikaaviot, ovat yleisiä kuvaustapoja.

UML on yhtenäisin ja eniten käytössä oleva oliomenetelmä, joka on saavuttanut standardin aseman mallinnuskielien joukossa. (Pohjonen 2002, 144 – 150)

2.4.3 Arkkitehtuurin suunnittelu

Arkkitehtuurin suunnittelussa käytetään usein erilaisia kaavioita. Keskeisin ja yleisin suositus on Philippe Kruchtenin esittämä 4+1 malli (Kuvio 4.).



Kuvio 4. Philippe Krichtenin 4+1 Malli (Haikala & Mikkonen 2011, 179)

Mallissa aloitetaan loogisesta näkymästä ja edetään toteutus- ja prosessinäkymien kautta fyysiseen näkymään. Näkymät tulee suunnitella suunniteltujen käyttötapausten kautta.

Mallissa kuvataan järjestelmän käyttäjien näkemys järjestelmästä ohjelmiston toimintojen avulla. Mallissa on 4+1 näkymää, jotka ovat

- looginen näkymä, jossa ohjelman toiminnallisuuden kuvaus esitetään toteutuksen kannalta, eli loppukäyttäjän näkökulmasta. Tyypillisiä kaavioita ovat luokka-, tapahtumasekvenssi- ja tilakaaviot.
- toteutusnäkymä, näkymässä on ohjelmiston toteutuksen rakenne, jossa ohjelmisto on jaoteltu tiedostoiksi, eli näkymä kuvataan ohjelmoijan perspektiivistä. Yleisiä käytettyjä kaavioita ovat komponentti- ja pakettikaaviot.
- prosessinäkymä, jossa selitetään systeemin prosessit ja niiden kommunikaatio ja perehdytään järjestelmän ajonaikaisiin suorituksiin. Tarkastelun kohteena ovat suoritustehoon ja skaalautuvuuteen liittyvät asiat. Näkymän visualisoinnissa voidaan käyttää aktiviteettikaaviota.
- sijoittelu- tai fyysinen näkymä, jossa ohjelmiston osat sijoitetaan fyysisille laitteille. Tyypillinen kaavio on käyttöönotto- ja komponentti- ja sijoittelukaavio.
- skenaariot tai käyttötapaukset, joissa arkkitehtuuri esitetään käyttötapauksilla, joista syntyy viides näkymä. Käyttötapaukset kuvaavat tapahtumia käyttäjien, mahdollisesti laitteiden ja prosessien välillä käyttämällä ohjelmistossa olevia toiminnallisuuksia. Käyttötapaukset kuvataan käyttötapauskaaviolla. (Haikala & Mikkonen 2011, 179 – 180.)

2.5 Toteutusvaihe

Toteutusvaiheessa ohjelmisto tai sen osa toteutetaan esimerkiksi jollain ohjelmointikielellä. Mikäli aiemmat toimenpiteet on suoritettu oikein, pitäisi toteutuksen olla melko suoraviivainen toimenpide, sillä kaikki ohjelmiston rakennetta ja toiminnallisuutta koskevat ratkaisut on tehty.

Toteutuksessa käytettäviin välineisiin ja tekniikoihin vaikuttavat mm. seuraavat seikat:

- sovellusalue, toiset toteutusvälineet sopivat tietyille sovellusalueille paremmin kuin toiset
- käytetyt menetelmät ja ohjelmistotuotannon mallit, tietty menetelmä vaatii tietynlaisen toteutusvälineen käytön
- tehokkuusvaatimukset, tehokkuusvaatimukset voivat rajata käytettävissä olevien toteutusvälineiden joukkoa ja
- toteutus- ja käyttöympäristö, jotkin ympäristöt tukevat tai edellyttävät tietyn toteutusvälineen käyttöä. (Pohjonen 2002, 34.)

Erityisen hankalia kysymyksiä toteutuksen kannalta ovat siirrettävyys ja ylläpidettävyys, sillä järjestelmiä tai niiden osia käytetään usein erilaisissa laite- ja käyttöjärjestelmäympäristöissä. Ratkaisu kumpaankin ongelmaan löytyy asianmukaisesta suunnittelusta ja toteutuksesta. Suunnitteluvaiheessa voidaan eristää laiteriippuvainen koodi erilliseksi moduuliksi, jolloin siirrettävyys saadaan kontrolliin. Ylläpidettävyydessä avainasemassa on ohjelmointityyli. Ohjelmakoodia kirjoitettaessa on huolehdittava, että muutkin kuin koodin alkuperäinen kirjoittaja pystyy seuraamaan ja ymmärtämään koodin funktionaalisuuden. Yksinkertainen ja selkeä ohjelmointityyli on suositeltavaa. (Pohjonen 2002, 35.)

2.6 Testausvaihe

Testausvaihe itsessään voidaan jakaa neljään osaan, joita ovat testauksen suunnittelu, testiympäristön luominen, testien suorittaminen ja testitulosten tarkastelu.

Haikala kirjoittaa kirjassaan olennaisista osista testausta:

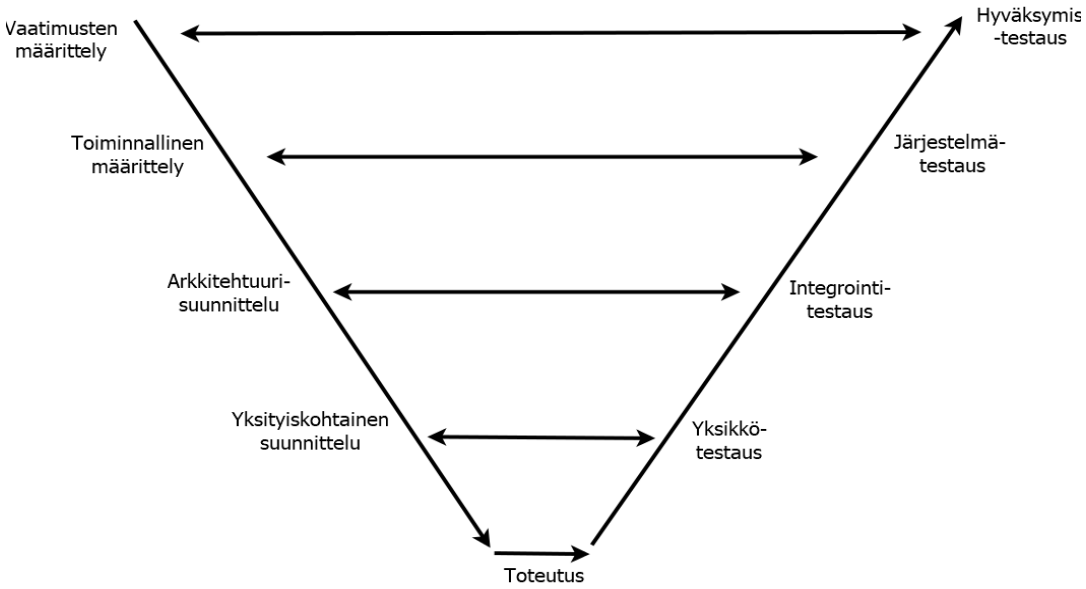
“Näihin työvaiheisiin ja niihin läheisesti liittyvään virheiden jäljitykseen ja korjaamiseen kuluu tyypillisesti yli puolet ohjelmistoprojektin resursseista, joten testauksen läpivientiin kannattaa kiinnittää erityisen paljon huomiota.”

Testaustavat voidaan jakaa kolmeen tasoon:

- yksikkötestaus, jossa testataan yksittäisiä luokkia, joita voivat olla esimerkiksi käyttäjän lisääminen tietokantaan

- integrointitestaus, jossa yhdistetään luokkia, jotka muodostavat kokonaisia osajärjestelmiä. Testausmuodossa painotetaan rajapintojen toimivuuden tutkimista. Integrointitestaus etenee usein rinnan yksikkötestauksen kanssa, eikä sitä tarvitse tarkastella erillään ja
- järjestelmätestaus, jossa testataan koko järjestelmä. Järjestelmätestauksessa testataan myös järjestelmän ei-toiminnalliset ominaisuudet. (Haikala & Mikkonen 2011, 205 – 207.)

Ohjelman kehittämisen ja testaamisen suhdetta havainnollistetaan usein V-mallin avulla (Kuvio 5.). Mallissa kuvataan ohjelmiston suunnittelun ja testaustason suunnittelun suhdetta toisiinsa. Testausprosessissa hyödynnetään suunnittelun aikana tehtyjä päätöksiä ja dokumentteja, joiden pohjalta luodaan kriteerejä testauksen suorittamiselle, testitapauksille ja niiden hyväksymiselle. (Kasurinen 2013, 14.)



Kuvio 5. Testauksen V-malli (Kasurinen 2013, 14)

2.6.1 Testausmenetelmät kehitysvaiheessa

Kehitysvaiheessa testaus noudattaa V-mallin mukaista lähestymistapaa, jossa ensin tehdään yksikkötestausta komponenteille, tämän jälkeen komponentit yhdistetään toisiinsa integrointitestauksessa ja lopuksi koko järjestelmää testataan.

Testauksessa käytettävät menetelmät voidaan jakaa staattiseen ja dynaamiseen testaamiseen. (Kasurinen 2013, 64 – 65)

Staattisessa testaamisessa järjestelmää ei käytetä, vaan järjestelmää tutkitaan esimerkiksi koodianalysaattorien, koodiarviointien tai arkkitehtuurisuunnittelun näkökulmasta. Näillä menetelmillä katetaan perustason tarkastukset ja poistamaan ilmiselvät ongelmat järjestelmästä ennen tarkemman testaamisen aloittamista. Yleisesti staattisessa testauksessa poistetaan syntaksin virheitä ja järjestelmän sisäisiä virheitä. Staattinen testaaminen voidaan aloittaa jo pelkän arkkitehtuurisuunnitelman pohjalta. (Kasurinen 2013, 64 – 65)

Dynaaminen testaaminen on staattisen testaamisen vastakohta. Dynaamisessa testaamisessa testattavaa järjestelmää käytetään ja järjestelmän reaktioita annettuihin syötteisiin seurataan. Useimmat testausmenetelmät, kuten yksikkötestaus, integrointitestaus ja kuormitustestaus ovat dynaamista testaamista. Dynaaminen testaaminen voidaan aloittaa, kun ohjelmistosta on tuotettu ensimmäinen prototyyppi, tai kun järjestelmä on osittain toteutettu. (Kasurinen 2013, 70 – 72)

Testauksen menetelmiä kehitysvaiheessa ovat:

- musta laatikko -testaus on perinteisin testauksen muoto. Ohjelmalle annetaan syötteitä ja katsotaan, mitä ohjelma tekee. Testauksessa ei tarkastella, mitä ohjelman sisällä tapahtuu. Tavallisia testitapauksia ovat esimerkiksi tiedostojen tallentaminen järjestelmään, lomakkeiden syöttäminen järjestelmään, painikkeista tapahtuvat toiminnot, järjestelmän reagointi virheellisiin syötteisiin ja SQL-upotuksiin tekstissä.
- lasilaatikkotestaus, jossa järjestelmää testataan tarkastelemalla sen sisäistä toimintaa testauksen aikana. Lasilaatikkotestaajien on ymmärrettävä järjestelmän logiikka, ymmärrettävä ohjelmointityöstä ja tunnettava testattava järjestelmä niin hyvin, että he pystyvät tarkastelemaan järjestelmää lähdekooditasolta alkaen.
- harmaa laatikko -testaus, jonka ideana on yhdistää musta- ja lasilaatikkotestauksen elementtejä. Testauksessa pyritään testaamaan mallin- ja koodin kattavuus, eli että kaikki vaatimukset täytetään ja että lähdekoodi on tarkastettu. Harmaa laatikko -testaus sopii tilanteisiin, jossa koko järjestelmään ei päästä käsiksi, mutta paikallisiin komponentteihin

päästään käsiksi. Esimerkiksi lähes kaikki verkkopalvelut. (Kasurinen 2013, 65 – 67.)

Regressiotestaus on termi, jolla tarkoitetaan uudelleentestaamista. Regressiotestauksesta puhutaan, kun ohjelmiston osaa muutetaan ja halutaan varmistaa systeemin toiminta. Testausmuodon ajatuksena on se, että yleensä järjestelmissä esiintyvät virheet johtuvat joko uusista komponenteista tai toiminnoista, jotka ovat yhteydessä uuteen komponenttiin. (Kasurinen 2013, 65 – 67.)

2.6.2 Testausmenetelmät ennen julkaisua

Kaikkea testaamista ei ole mielekästä toteuttaa kuin valmiille tai lähes valmiille tuotteelle. Testaukset, jotka tulisi tehdä ennen julkaisua ovat:

- käytettävyystestaus, jossa testataan järjestelmän käyttöliittymän toimivuus ja intuitiivisuus
- kuormitustestaus, jossa tehdään sivustolla useita eri toimintoja esimerkiksi luomalla virtuaalikäyttäjiä. Testauksen pääpainona on tunnistaa järjestelmän pullonkaulat, selvittää systeemin kapasiteetti ja järjestelmän selviytyminen käyttöolosuhteista, jossa tehdään tavallisia toiminnallisuuksia. Jos kuormitustestaus viedään pidemmälle, puhutaan rasitustestauksesta. Tämä toteutetaan esimerkiksi lisäämällä käyttäjämääriä yli suunnitellun määrän
- suorituskykytestaus on rasitustestauksen kevennetty muoto, jossa testataan, että järjestelmä suoriutuu sille annetuista tehtävistä määrittelyjen mukaisesti ja
- savutestaus on testauksen muoto, jolla testataan perusasioiden toimivuus. Käytännössä tämä toteutetaan tekemällä tarkastuslista, johon johdetaan määrittelystä tarvittavat tarkastukset. Esimerkiksi “toimivatko valikon toiminnallisuudet” tai “toimiiko asennusohjelma määritellyllä tavalla”.
- alfa- ja betatestaus ovat testauksen vaiheita, joita sovelletaan erityisesti itsenäisiin ohjelmistoihin, kuten peleihin. Alfatestaus on vuorossa

integrointi- ja järjestelmätestauksen jälkeen ja siinä testauksen tavoitteena on tutkia miten järjestelmä toimii kokonaisuutena. Alfatestausta voi ajatella hyväksymistestauksena, joka toteutetaan sisäisesti. Betatestaus on hyväksymistestaus ohjelmistolla, jolla ei ole varsinaista yhtä tai hyvin rajattua käyttäjäkuntaa. Betatestauksessa tuotetta voidaan muokata, sillä tuote ei ole vielä julkaistu. (Kasurinen 2013, 70 – 73.)

2.7 Käyttöönottovaihe

Kun järjestelmä on testattu, voidaan se ottaa käyttöön. Tietojärjestelmien käyttöönotossa on merkittäviä huomioitavia tekijöitä, jotka on huomioitava ja valmisteltava huolellisesti ajoissa. (Pohjonen 2002, 37).

Resurssit tulisi suunnata muunnossuunnitelman, käyttöohjeiden, käyttöönottosuunnitelman ja käyttäjädokumentaation tuottamiseen. Tavoitteita ovat muokatun tai uuden järjestelmän muunnokset ja ympäristön luominen, jossa järjestelmä toimii jatkuvasti hyvin. (Murch 2002, 125 – 130.)

2.8 Ylläpitovaihe

Ohjelmiston elinkaari ei pääty käyttöönottoon. Ohjelmiston elinkaaren pisin yksittäinen vaihe on ylläpito. (Pohjonen 2002, 37.)

Ylläpito sisältää neljä eri perustapausta

- korjaava ylläpito on toimenpide, jossa korjataan käyttöönoton jälkeen havaittuja virheitä.
- sopeuttava ylläpito sisältää järjestelmän siirtämisen uuteen ympäristöön.
- täydentävä ylläpito on uusien ominaisuuksien toteuttamista järjestelmään.
- ennakoiva ylläpito on järjestelmän tai sen dokumentaation tason parantamista tulevia ylläpitotilanteita silmällä pitäen. (Pohjonen 2002, 37.)

Yleisin ylläpitoa vaikeuttava tekijä on puutteellinen dokumentaatio. Ylläpitovaiheessa järjestelmän taustalla vaikuttavia ratkaisuja on vaikea ymmärtää ilman asianmukaista dokumentaatiota. (Pohjonen 2002, 37.)

Ylläpitovaiheen yhteydessä voidaan toteuttaa myös ohjelmiston jatkokehitystä. Jatkokehityksessä ohjelmistolle voidaan tuoda lisää toiminnollisuuksia ja sitä voidaan muokata vastaamaan paremmin tarpeita, kun sivustolta on kerätty tarpeeksi palautetta sivuston eri käyttäjä- ja sidosryhmiltä.

3 TUTKIMUKSEN TOTEUTUS

Tässä osiossa käydään läpi opinnäytetyön määrittely, suunnittelu, toteutus, testaus, käyttöönotto, ylläpito ja jatkokehitys -vaiheet käytännössä. Minut palkattiin Lapin Yliopistolle suunnittelijaksi toteuttamaan web-sivusto, joten päätös web-sivuston hankinnasta oli jo tehty puolestani. En tästä syystä toteuta esitutkimusvaihetta opinnäytetyössä. Pääsin testausvaiheeseen asti tutkimuksen toteutuksessa, joten käyttöönotto-, ylläpito ja jatkokehitysvaiheet on käsitelty tehtyjen suunnitelmien pohjalta, mutta informaatio näissä vaiheissa on rajattu.

3.1 Tutkimuksen määrittelyvaihe

Tutkimuksen määrittelyvaiheessa keräämme tietoa järjestelmän tarkoituksesta ja tavoitteesta, organisaatiossa vallitsevasta nykytilanteesta, sivustoa käyttävistä sidosryhmistä ja vaatimuksista ja rajoitteista. Määrittelyvaiheen tiedonkeräys tapoja olivat sidosryhmien haastattelut ja yhteiset palaverit. Olennaisia työskentelykohteita olivat sovelluksen rajaukset ja haluttujen toimintojen kokoaminen yhteen ja toimintojen erittely ei-toiminnallisiin ja toiminnallisiin vaatimuksiin.

3.1.1 Sivuston toimeksianto, tarkoitus ja tavoitteet

Sivustolla on useita tarkoituksia. Tarkoitukset voidaan jakaa opiskelun ohjaamiseen, potentiaalisten asiakkaiden tavoittamiseen ja tulevan yrityksen myynnin ja vision laajentamiseen.

3.1.2 Organisaatiossa vallitseva nykytilanne

Nykyisellään organisaatiolla ei ole sivustoa, jolla voitaisiin toteuttaa tarpeeksi dynaaminen palvelu vastaamaan projektin tarpeisiin. Lapin Yliopisto käyttää Infoweb -verkkopalvelualustaa, joka on helppokäyttöinen ja tehokas sisällönhallintajärjestelmä, mutta helppokäyttöisyys kuljettaa mukanaan myös toisen ulottuvuuden. Web-palveluissa helppokäyttöisyys rajoittaa sivustolla toteutettavia toiminnallisuuksia huomattavasti. Infoweb sopii hyvin bloggauksen ja uutisten julkaisualustaksi, mutta prosessin opetustyökaluna se on hankeen kannalta liian staattinen.

3.1.3 Kohderyhmän määrittely

Sivuston kohderyhmään kuuluvat asiakkaat, jotka käyttävät palvelua osana koulutustapahtumaa, työntekijät, jotka käyttävät palvelua kouluttajina ja leikillisen pedagogiikan asiantuntijoina ja sivuston pääkäyttäjä, joka vastaa sivuston päivittämisestä ja kehittämisestä.

Tuotteen kohderyhmiä ovat opettajat, kouluttajat ja opettajaksi opiskelevat. Kohderyhmän tarpeita ovat sivuston loogisuus ja helppokäyttöisyys, sillä opetettavan asian tulee olla pääroolissa ja kohderyhmän sisällä on eroja tietoteknisten laitteiden käytön hallinnassa. Myös visuaalinen näyttävyys on huomioitava ja sivustolle on luotava teema, joka tukee projektin ilmettä.

3.1.4 Toiminnalliset vaatimukset

Sivuston ensimmäisessä versiossa tavoitellaan seuraavia asioita:

- kohderyhmien jäsenet pystyvät kirjautumaan palveluun
- käyttäjät pystyvät tutustumaan sivustolla esiteltävää opetustavan mallia
- käyttäjät pystyvät testaamaan opetustapaa luomalla mallin
- käyttäjät pystyvät hakemaan sivustolta tietoa tulevista tapahtumista ja katsomaan uutisia koskien leikillistä pedagogiikkaa
- kouluttajat pystyvät lisäämään sivustolle esimerkkimalleja opetuksesta
- kouluttajat pystyvät päivittämään ja lisäämään sivustolle olevia uutisia ja
- sivustolla on oltava kontaktisivu, jossa on näkyvillä hankkeen yhteystiedot.

3.1.5 Ei-toiminnalliset vaatimukset

Sivuston on tuettava yleisimpiä selaimia. Sivusto ei saa hajota ja sivuston käyttöliittymän on oltava selkeä ja yksinkertainen. Sivuston on oltava dynaaminen ja responsiivinen.

3.1.6 Rajoitteet ja reunaehdot

Sivuston käyttöliittymän toteuttamiseen palkataan hankkeeseen suunnittelija. Uuden suunnittelijan vastuulla ovat teeman toteuttaminen ja sivuston ulkoasun

suunnittelu, joten minulta vapautuu resursseja sivuston toiminnallisuuksien kehittämiseen ja dokumentointiin.

3.2 Tutkimuksen suunnitteluvaihe

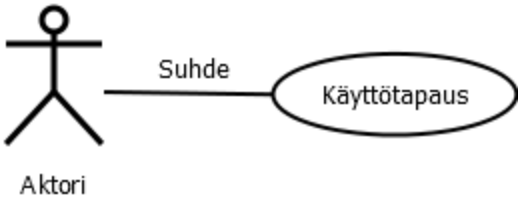
Web-sivuston suunnitteluvaihe toteutettiin pääosin Philippe Kruchtenin 4+1 mallin mukaisesti. Yksinkertaistin mallista kaksi vaihetta. Yksinkertaisten web-sivustojen, jotka sijaitsevat yhdellä palvelimella, suunnittelussa toteutus- ja sijoittelunäkymästä saatava tieto voidaan kuvata varsin suoraviivaisesti luonnollisella kielellä.

Käytännön tasolla suunnitteluvaihe nitoutui yhteen määrittelyvaiheen kanssa, sillä mielestäni määrittelyä helpotti huomattavasti teknillinen peruskuvaus kirjautumisvaiheesta tai tietokantarakenteesta. Pelkkä suullinen analyysi ei ollut riittävä, sillä projektin henkilöstöllä ei ole huomattavaa kokemusta IT-alan suunnittelun terminologiasta ja työkaluista, joten kaavioiden käyttö lisäsi yhteisymmärrystä toteutettavasta sovelluksesta. Etenkin peruskuvaus sivuston rakenteesta ja peruskuvaus käyttäjien toiminnoista lisäsivät huomattavasti yhteisymmärrystä toteutettavasta järjestelmästä.

3.2.1 Käyttötapauskaaviot

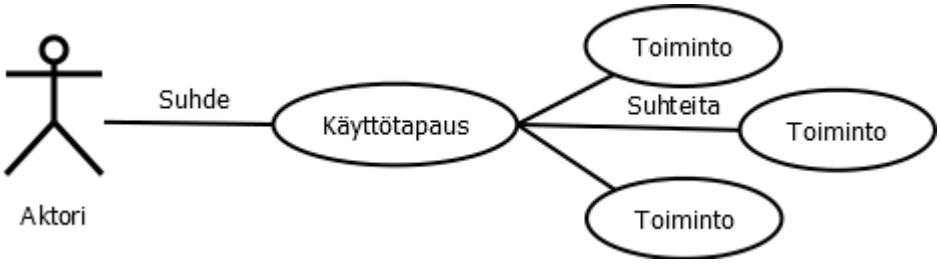
Käyttötapaukset ovat Kruchtenin mallissa nimetty skenaarioiksi tai käyttötapauksiksi. Käyttötapauskaaviot koostuvat aktoreista, käyttötapausten kuvauksista ja niiden välisistä suhteista (Kuvio 6.). Käyttötapauskaaviot ovat yleinen tapa kuvata järjestelmälle asetetut vaatimukset käyttäjän näkökulmasta. (Pohjonen, 2002, 152.)

Käyttötapauskaavioiden suosio perustuu niiden intuitiiviseen rakenteeseen. Käyttötapauskaavioita on vaikea ymmärtää väärin ja ne ovat nopea tapa tutustua järjestelmältä vaadittuihin toimintoihin eri käyttäjien näkökulmista.



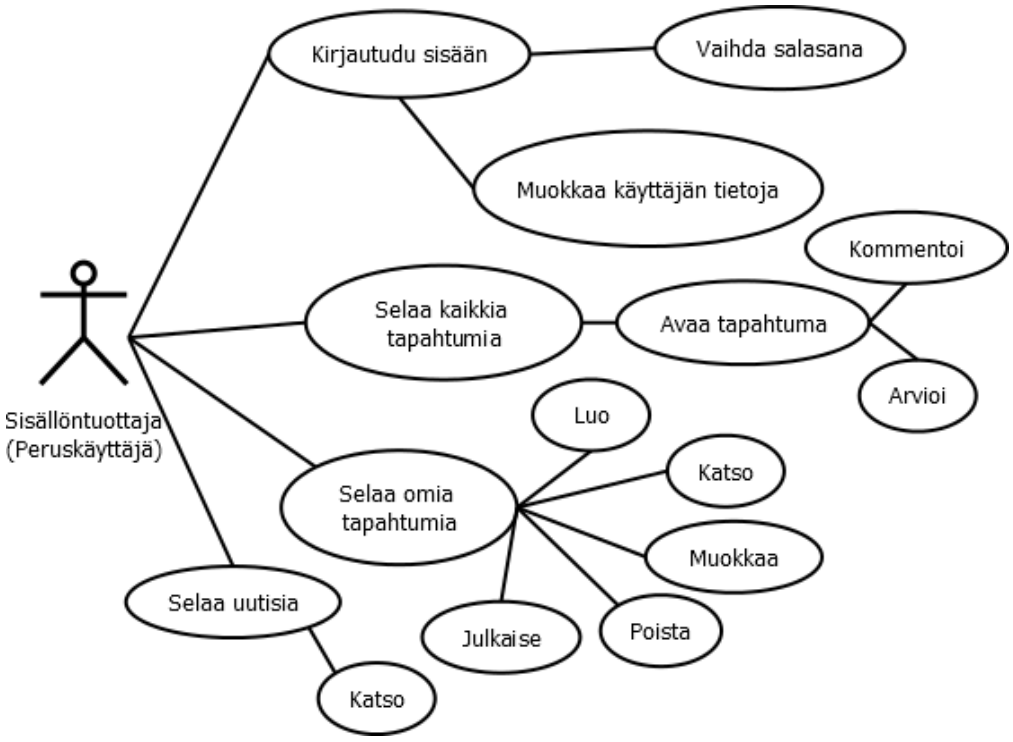
Kuvio 6. Käyttötapauskaavion terminologia

Käyttötapaus voidaan jakaa edelleen toimintoihin. Olen 7. kuviossa jakanut käyttötapauksen kolmeen eri toimintoon.



Kuvio 7. Käyttötapauskaavion jatketut toiminnot

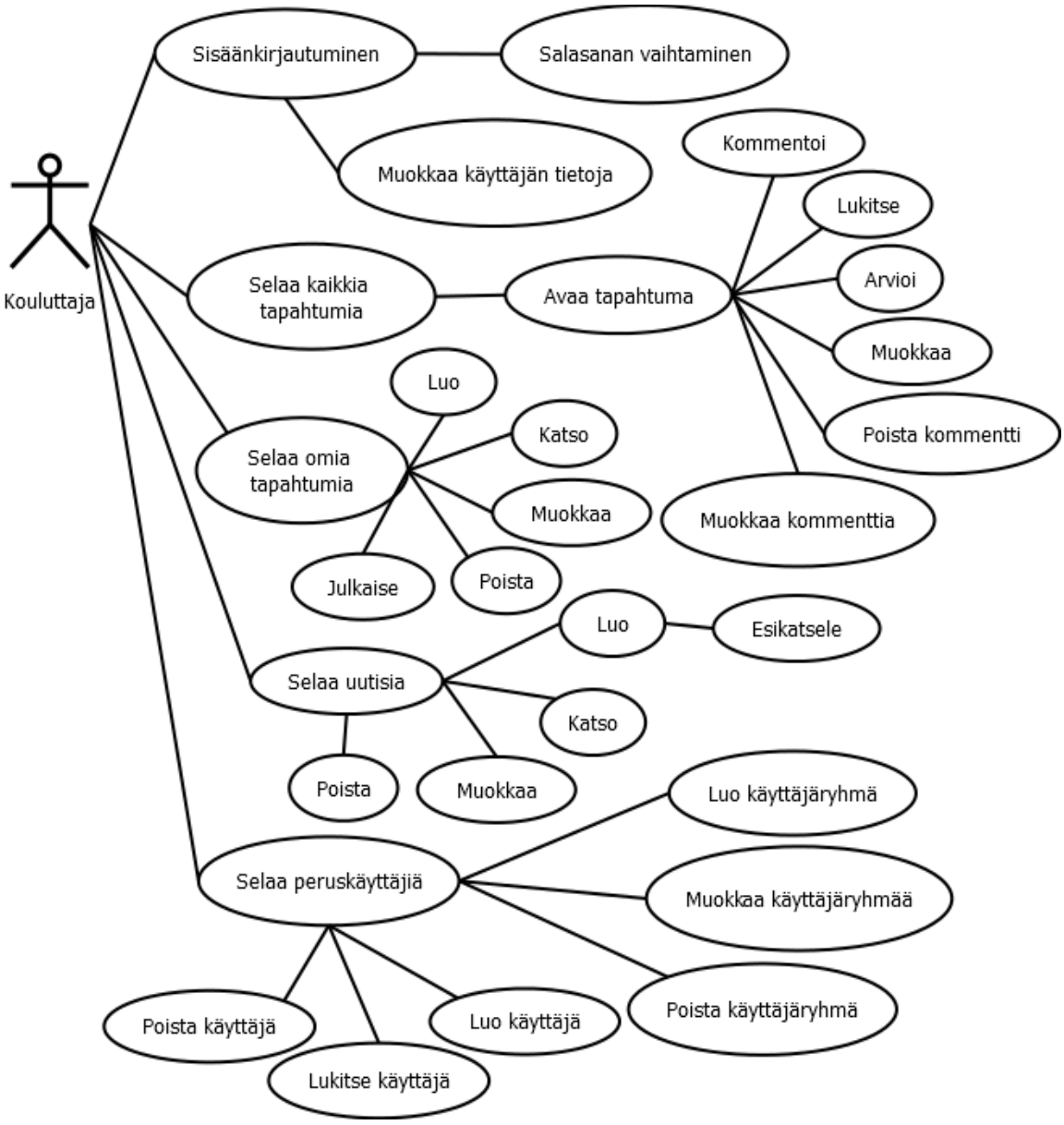
Käyttötapauskaaviot on jaettu eri käyttäjiin. Sivustolla olevia käyttäjäkuntia ovat peruskäyttäjät (Kuvio 8.), kouluttajat (Kuvio 9.) ja pääkäyttäjä (Kuvio 10.).



Kuvio 8. Peruskäyttäjän käyttötapaukset sivustolla

Peruskäyttäjän on pystyttävä kirjautumaan sisään ja ulos palvelusta. Hänen on myös pystyttävä muokkaamaan omia tietojaan ja salasanaa.

Käyttäjän tulee pystyä tarkastelemaan palvelussa kouluttajien ja muiden käyttäjien luomia opetustapahtumia ja kommentoimaan niitä. Käyttäjä pystyy luomaan ja poistamaan oman opetustapahtuman, sekä katsomaan ja muokkaamaan omaa opetustapahtumaa. Käyttäjä voi julkaista oman opetustapahtuman, jonka jälkeen se näkyy muille käyttäjille. Käyttäjän on pystyttävä selaamaan uutisia, sekä hänen on pystyttävä katsomaan uutista, jolloin uutinen avataan uuteen näkymään, jossa näytetään koko uutinen.



Kuvio 9. Kouluttajan käyttötapaukset sivustolla

Kouluttajalla on kaikki peruskäyttäjän ominaisuudet. Lisäksi kouluttaja pystyy lukitsemaan ja muokkaamaan peruskäyttäjien julkisia opetustapahtumia. Kouluttaja pystyy myös muokkaamaan ja poistamaan kommentteja.

Kouluttajan on lisäksi pystyttävä selaamaan, katsomaan, muokkaamaan, poistamaan ja luomaan uutisia. Luonnin yhteydessä on oltava esikatselu -tila, jossa uutinen näytetään samanlaisena kuin se sivustolla on.

Kouluttaja pystyy myös hallinnoimaan peruskäyttäjiä. Hänellä tulee olla mahdollisuus poistaa, lukita tai luoda käyttäjä.

Kuvio 10. Pääkäyttäjän käyttötapaukset sivustolla

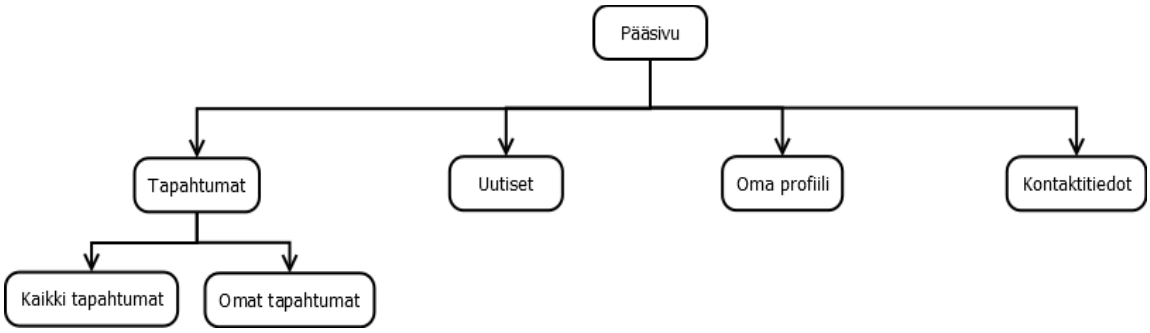
Pääkäyttäjä on ainoa, joka pystyy asettamaan käyttäjän luokan kouluttajaksi. Lisäksi pääkäyttäjä pystyy poistamaan ja lukitsemaan minkä tahansa käyttäjän. Pääkäyttäjän on voitava asettaa sivustolle käyttökatko- tai muu ilmoitus.

3.2.2 Järjestelmän toiminnallisuudet

Järjestelmän toiminnallisuudet ovat Kruchtenin mallissa nimetty loogiseksi näkymäksi. **Luokkakaavio** kuvaa sovelluksen luokat ja niiden sisällön, ja luokkien vä-

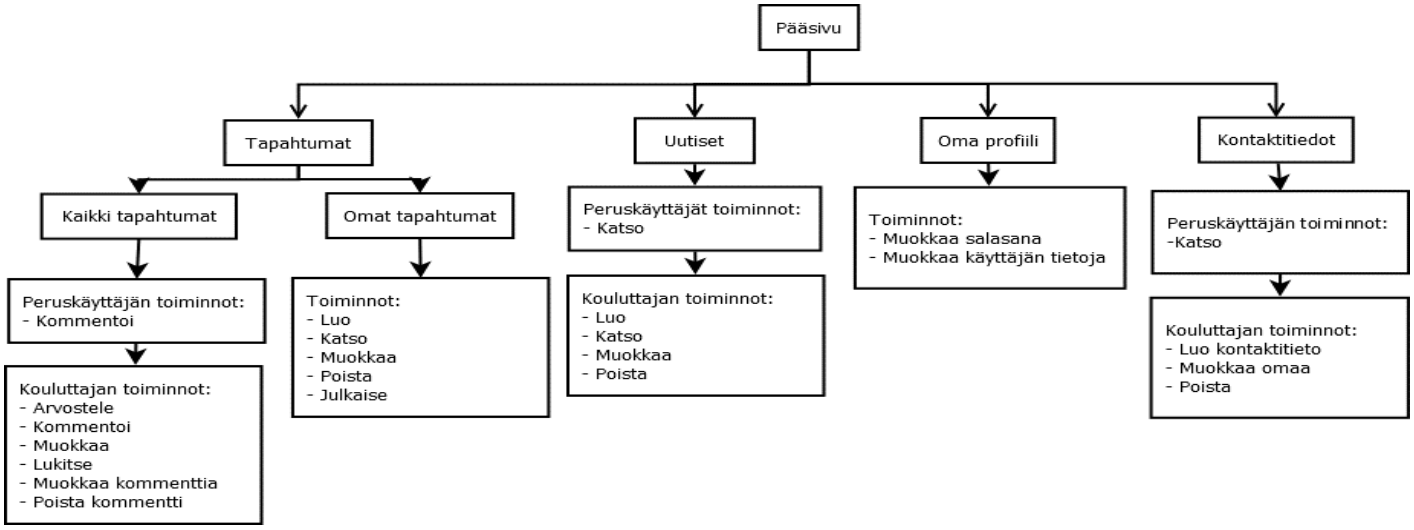
liset suhteet. Luokkakaavio mallintaa sovelluksen staattista rakennetta. Luokkakaaviolla voidaan kuvata tietokannat, niiden sisällöt ja suhteet. Sekvenssikaavio kuvaa tietyn toiminnon tai tapahtumasarjan siihen osallistuvien olioiden vuorovaikutusten kannalta. Tilakaavio kuvaa olioiden välistä toimintaa. Se ei ota kantaa olion sisäiseen dynamiikkaan, mutta käsittää olion tilan, jossa aktiviteetti tapahtuu. (Pohjonen 2002, 153 – 157.)

Käytän mallissa rakennekaaviota (Kuvio 11.) ja navigaatiokarttaa aiemmin esiteltyjen kaavioiden sijaan, sillä rakennekaavio on nopeampi toteuttaa ja siitä saatava informaatio on riittävä hankkeen tarpeisiin.



Kuvio 11. Sivuston peruskäyttäjän rakennekaavio

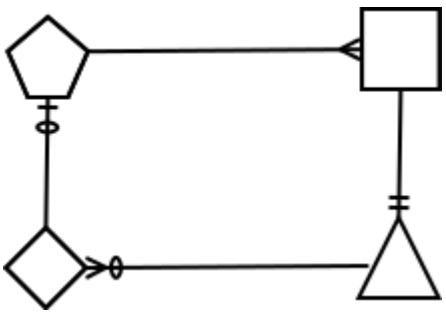
Rakennekaavio kuvaa sivuston navigoinnin yleisellä tasolla. Tähän rakennekaavioon on kuvattu peruskäyttäjän näkemät sivuston toiminnallisuudet. Rakennekaavioista voidaan johtaa kaavio, jossa näkyvät myös sivuilla tapahtuvat toiminnallisuudet. Navigaatiokartasta (Kuvio 12.) saa nopeasti ja kattavasti kuvan sivuston toiminnallisuuksista ja navigaatiosta.



Kuvio 12. Sivuston navigaatiokartta

Navigointikartassa sivuston toiminnallisuudet ovat esitettyinä tekstinä. Sivuston toiminnallisuudet on johdettu käyttötapauskaavioiden toiminnallisuuksista. Navigaatiokartta on yksinkertainen tapa nähdä kokonaisuus, sillä kaavioissa tulisi pyrkiä mahdollisimman yksiselitteiseen ja -kertaiseen esitysmuotoon. Käyttötapauskaaviot sisältävät *<<extend>>* ja *<<include>>* toiminnallisuudet, mutta niiden käyttö johtaa nopeasti kaavioiden monimutkaistumiseen.

Crow’s Foot Notation on tietokantojen suhteiden merkintätapa. Kuvio 13. on esimerkki kaaviossa käytettävistä suhteista. Tauluilla tällaisia suhteita ei kumminkaan voi olla, sillä kummallakin tietokannalla on oltava suhde toisiinsa (Kuvio 14.).

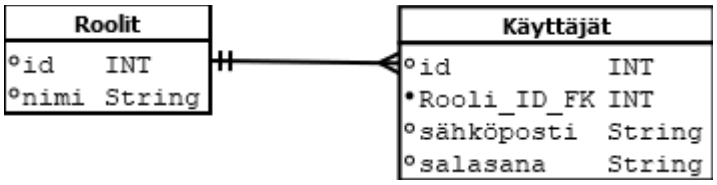


Kuvio 13. Crow’s Foot Notation terminologia

Yllä olevassa kuviossa on neljä erilaista tietokantasuhdetta. Ne ovat ainoat suhteet, joita Crow’s Foot Notation tukee. Viisikulmio, nelikulmio, kolmio ja timantti esittävät tietokannassa olevia tauluja.

Viisikulmiosta kulkee suhde nelikulmioon. Tämä suhde tarkoittaa, että viisikulmiolla on yksi tai useampi nelikulmio. Nelikulmiosta menee suhde kolmioon. Tämä suhde tarkoittaa, että nelikulmiolla on yksi ja vain yksi kolmio. Kolmiosta menee suhde timanttiin. Tämä suhde tarkoittaa, että kolmiolla on nolla tai enemmän timanttia. Timantista kulkee suhde viisikulmioon. Tämä suhde indikoi, että timantilla on yksi tai ei yhtään viisikulmiota.

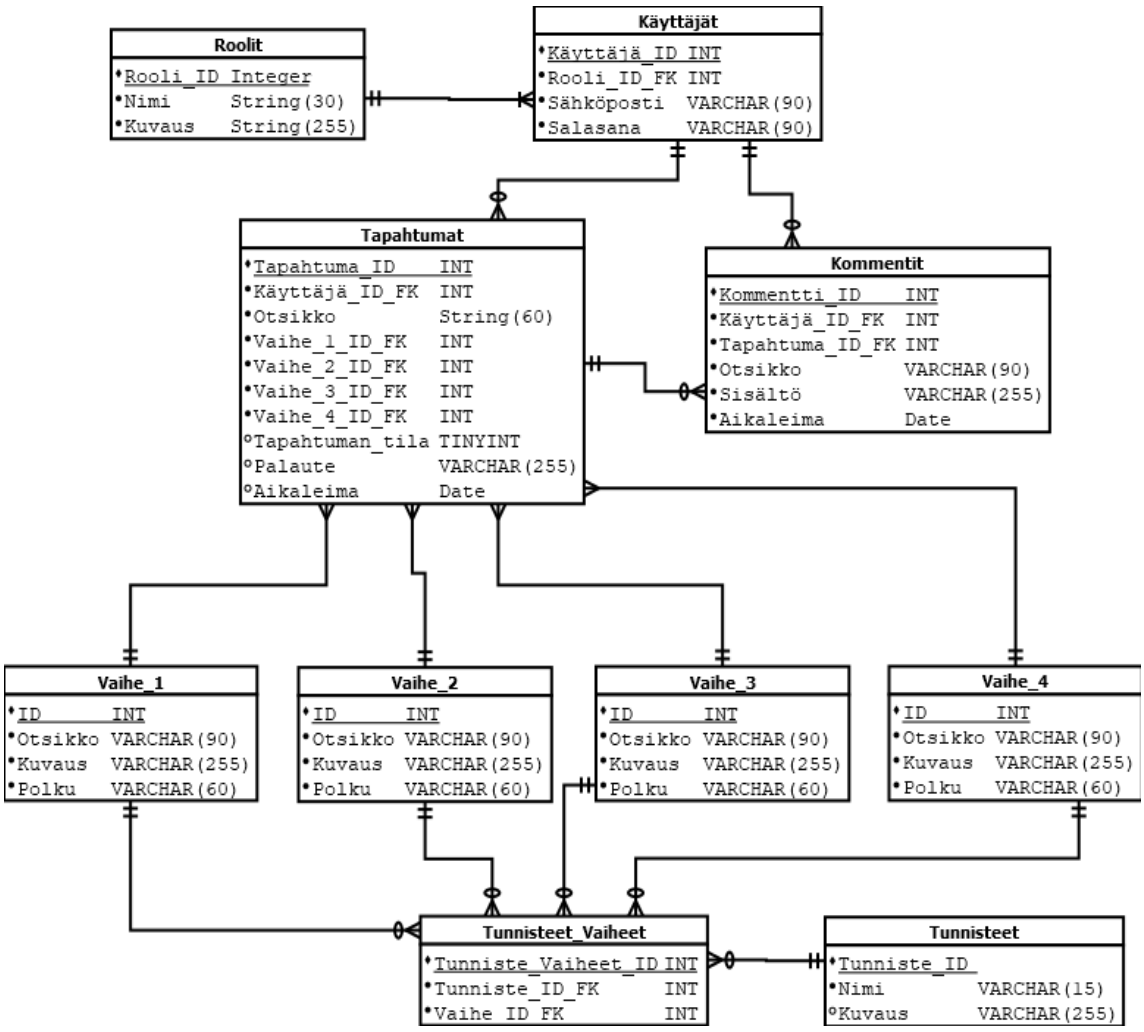
Kuviossa 14 on esitetty kaksi tietokantataulua. Roolit- ja käyttäjät-taulut ovat suhteessa toisiinsa erilaisilla suhteilla. Käyttäjällä on yksi ja vain yksi rooli, mutta sama rooli voi kuulua usealle eri käyttäjälle.



Kuvio 14. Esimerkki taulujen välisestä suhteesta

Käyttäjät -taulun sisällä on kenttä *Rooli_ID_FK*, johon tallennetaan käyttäjän roolin tunniste, koska tieto yhteydestä on tallennettava tauluun. Esimerkiksi jos käyttäjän rooli on “pääkäyttäjä” ja pääkäyttäjän id Roolit -taulussa on 3, tallennetaan kokonaisluku 3 Käyttäjät -taulun *Rooli_ID_FK*-kenttään.

Kuvaan tietokannat *Crow’s Foot Notation*-kaaviolla (Kuvio 15.). Se on kaaviotyyppi, joka on suunniteltu nimenomaan tietokantojen suunnitteluun.



Kuvio 15. Sivuston tietokantarakenne

Tietokantaan tehdään taulut rooleille, käyttäjille, tapahtumille, kommenteille, tapahtuman vaiheille, tunnisteille ja lisäksi tunnisteille tehdään välitaulu. Roolit -tauluun tallennetaan roolin nimi ja kuvaus. Roolit -taulu on yhteydessä käyttäjät tauluun. Yhdellä roolilla voi olla monta käyttäjää.

Käyttäjät -tauluun tallennetaan käyttäjän sähköposti, salasana ja viiteavain käyttäjän roolista. Käyttäjällä voi olla vain yksi rooli. Käyttäjällä voi olla monta tapahtumaa, mutta käyttäjän ei ole pakko luoda tapahtumia. Käyttäjällä voi olla myös monta kommenttia, mutta käyttäjällä ei ole pakko olla yhtään kommenttia.

Kommentit -tauluun tallennetaan kommentin tehneen käyttäjän viiteavain, tapahtuman, jota on kommentoitu, viiteavain, kommentin otsikko, sisältö ja aikaleima. Kommentin on pakko kuulua yhteen opetustapahtumaan ja kommentilla on myös pakko olla yksi käyttäjä.

Tapahtumat -tauluun tallennetaan viiteavain tapahtuman tehneestä käyttäjästä, tapahtuman otsikko, tapahtuman vaiheiden viiteavaimet, tapahtuman tila, palaute ja aikaleima. Taulu tarvitsee tiedon tapahtuman tehneestä käyttäjästä, sillä käyttäjän tiedot asetetaan osaksi tapahtumaa. Tapahtuman tila on totuusarvo, joka kuvastaa, onko tapahtumaa julkaistu. Jos tapahtumaa ei ole julkaistu, se ei näy muille käyttäjille. Palaute on kouluttajan antama kirjallinen palaute tapahtumasta. Tapahtuma koostuu neljästä vaiheesta, joita tapahtumalla voi olla vain yksi, jonka seurauksena niistä tallennetaan tieto tauluun.

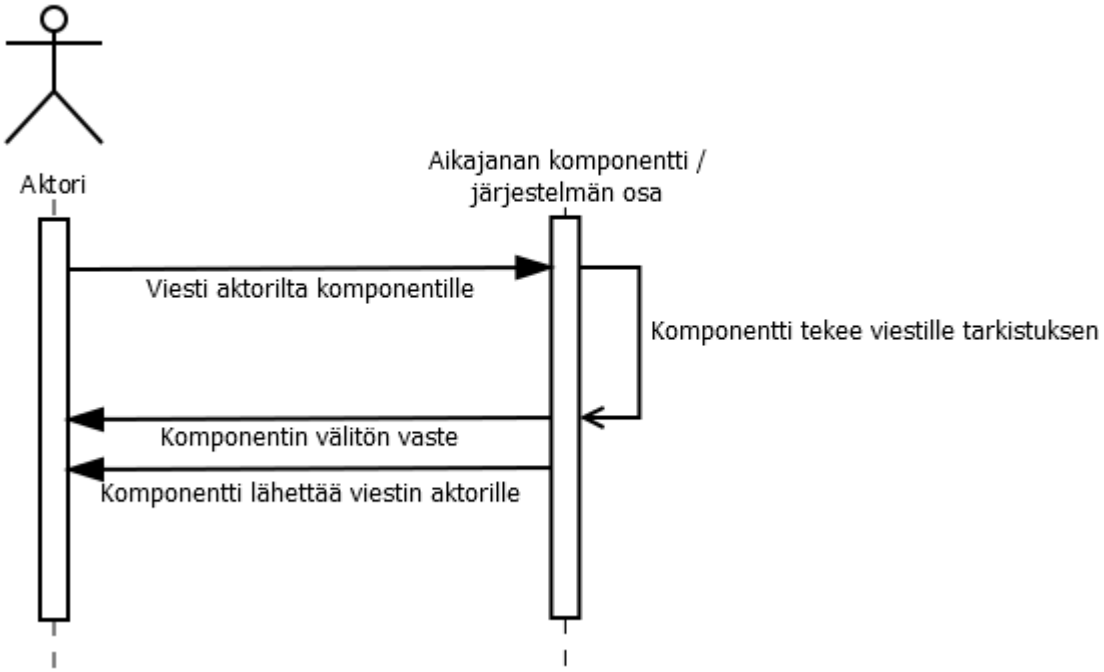
Vaiheiden taulut sisältävät vaiheiden otsikot, kuvaukset ja polut. Polkuihin tallennetaan järjestelmään tallennettavien kuvien järjestelmäpolku. Vaiheilla voi olla monta tunnistetta, mutta ei ole pakko olla yhtään tunnistetta. Vaihe voi kuulua myös moneen opetustapahtumaan ja sen on pakko kuulua ainakin yhteen opetustapahtumaan, sillä jos opetustapahtuma poistetaan, poistuu myös opetustapahtuman vaiheet, mikäli niillä ei ole yhteyksiä muihin opetustapahtumiin.

Tunnisteet -taulu sisältää tunnisteen nimen ja kuvauksen. Tunniste- ja vaiheiden taulujen välillä on välitaulu, sillä taulut sisältävät moni moneen yhteyden kummankin päin. Tunniste voi kuulua moneen vaiheeseen ja moni vaihe voi sisältää useita tunnisteita. Välitaulu sisältää kummankin taulun viiteavaimen. Siihen luodaan aina uusi instanssi, kun luodaan uusi yhteys tunnisteen ja vaiheen välille.

3.2.3 Sivuston prosessinäkymä

Prosessinäkymässä huomioidaan järjestelmää ajonaikaisten suoritusten perusteella. Näkymässä kuvataan sivuston prosessit ja niiden kommunikaatio käyttämällä aktiviteettikaaviota.

Kuviossa 16. kuvataan aktiviteettikaavio. Aktiviteettikaavio koostuu aktorista, järjestelmän osista ja viesteistä. Pystysuorat palkit kuvaavat niiden yläpuolella olevan aktorin tai komponentin viestejä vastaanottavaa komponenttia järjestelmässä. Viestit voivat olla eri tahojen välisiä tai ne voivat olla tarkistuksia.

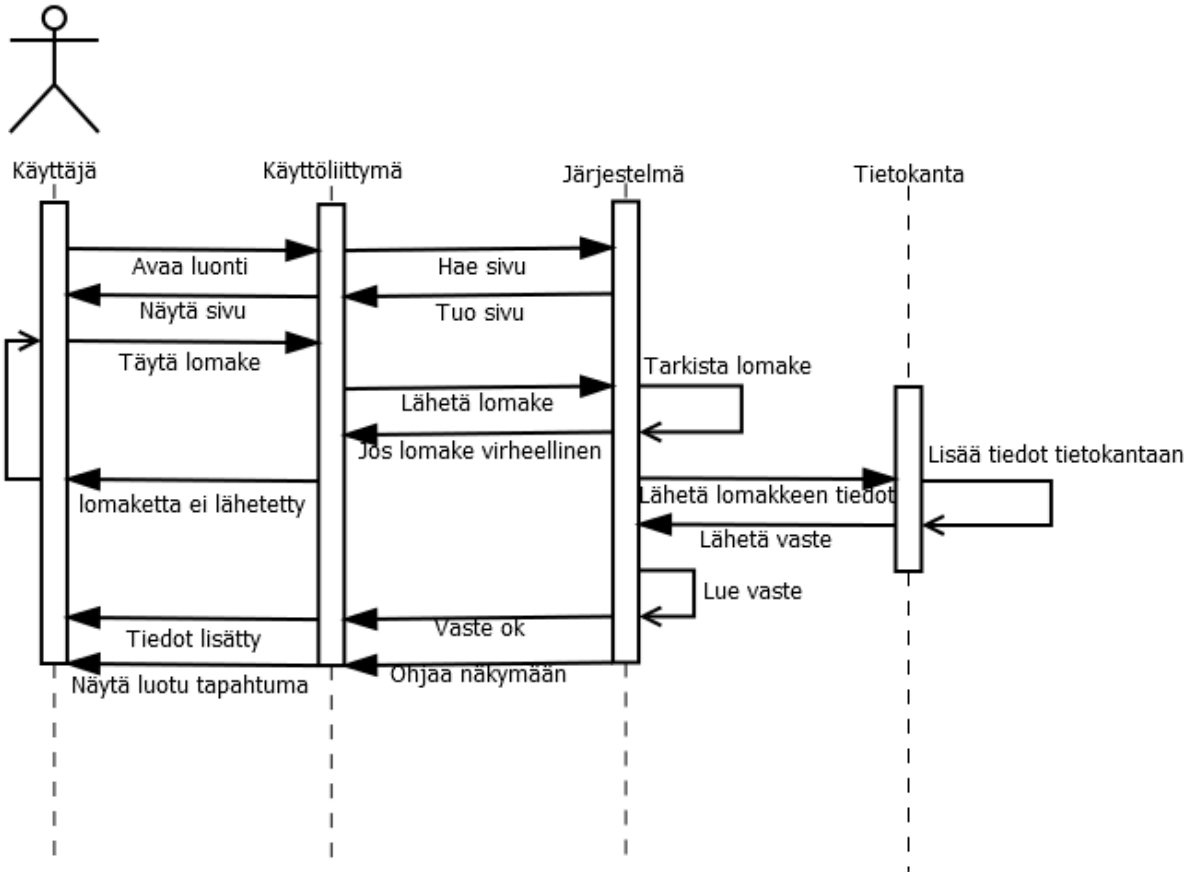


Kuvio 16. Aktiviteettikaavion terminologia

Aktiviteettikaavioilla kuvataan yleensä käyttötapauksista johdettuja toiminnallisuuksia. Ne kuvaavat tarkasti tiettyä järjestelmän toimintoa. Aktiviteettikaavio koostuu yksinkertaisista toiminnoista, joita toteutetaan esimerkiksi käyttäjän, käyttöliittymän, järjestelmän ja tietokannan välillä.

Aktiviteettikaavio voidaan kuvata myös vaakatasossa, jolloin yleisesti käytettäviä kaavion komponentteja ovat alkutila, aktiviteetit, looginen operaattori, synkronointipalkki ja lopetustila.

Kuvio 17. aktiviteettikaaviossa käyttäjä luo uuden tapahtuman sivustolle. Käyttäjä aloittaa prosessin painamalla käyttöliittymästä painiketta, joka ohjaa kaavion alkutilanteeseen.



Kuvio 17. Tapahtuman lisäämisen kuvaus aktiviteettikaaviolla

Kun käyttäjä on painanut painiketta, haetaan järjestelmästä sivu, joka tuodaan käyttöliittymään ja näytetään käyttäjälle.

Käyttöliittymässä olevalla sivulla on lomake, joka sisältää tapahtumalle oleelliset tiedot. Käyttäjä täyttää lomakkeen järjestelmään painamalla lähetä -painiketta, jolloin käyttöliittymä lähettää lomakkeen tiedot järjestelmään. Järjestelmässä tehdään lomakkeen kenttien validointi, josta järjestelmä palauttaa totuusarvon tosi tai epätosi.

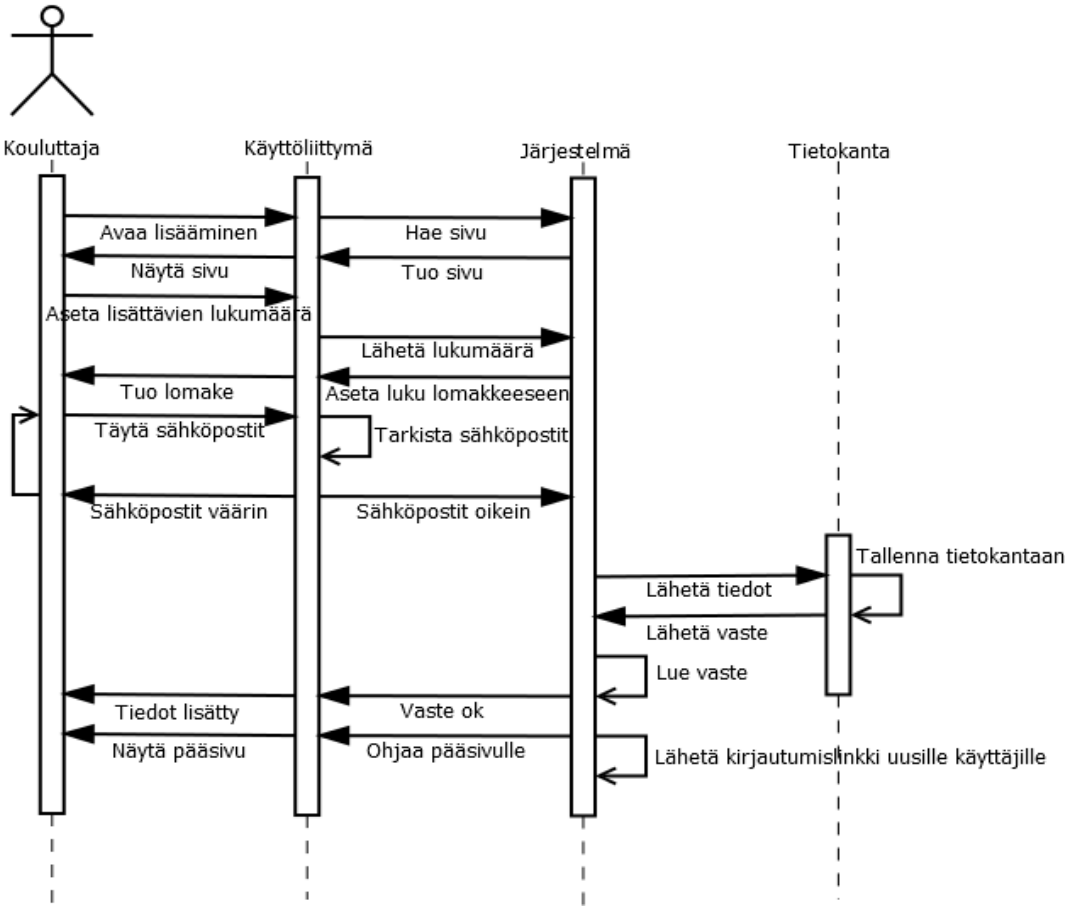
Jos lomake on virheellinen, lähetetään se takaisin käyttäjälle ja käyttäjälle näytetään tiedot, jotka hän on täyttänyt. Käyttäjälle näytetään myös virheviesti, joka sisältää tiedon, miksi järjestelmä ei hyväksynyt lomaketta.

Jos lomake lähetetään onnistuneesti, sen tiedot lisätään tietokannan tauluun. Tietokannasta lähetetään viesti, jonka järjestelmä lukee. Viestistä käy ilmi, lisättiinkö tietoja tietokantaan. Tietojen pitäisi tässä vaiheessa mennä tietokantaan

ongelmitta, sillä lomakkeen validointi tehtiin ennen tietojen lähettämistä tietokantaan.

Jos tiedot on lisätty onnistuneesti tietokantaan, lähetetään käyttöliittymään vaste, joka avaa viestin. Viestissä käy ilmi tietojen lisäyksen onnistuneen tietokantaan. Järjestelmä ohjaa käyttäjän luotuun tapahtumaan, jossa käyttäjä voi tarkastella tapahtumaa.

Kuvio 18. aktiviteettikaaviossa kouluttaja luo uutta käyttäjäryhmää. Kouluttaja aloittaa prosessin painamalla käyttöliittymässä sijaitsevaa painiketta.



Kuvio 18. Käyttäjän tai käyttäjäryhmän lisäämisen kuvaus aktiviteettikaaviolla

Käyttöliittymästä lähetetään järjestelmään viesti, jossa käsketään järjestelmää lähettämään sivu, jossa kysytään kouluttajalta, montako käyttäjää tämä haluaa lisätä. Kouluttaja antaa haluamansa luvun käyttöliittymään ja lähettää lukumäärän järjestelmään. Järjestelmä puolestaan lähettää käyttöliittymään muuttujan, jonka

avulla kouluttajalle tuodaan lomake, jossa on niin monta sähköpostikenttää, kuin kouluttaja oli aiemmin määritellyt.

Kouluttaja täyttää jokaisen sähköpostikentän. Tässä vaiheessa käyttöliittymä tarkistaa ajonaikaisesti, että sähköpostikentät noudattavat yleisiä sähköpostille ominaisia ominaisuuksia, kuten @-merkin olemassaolon kouluttajan syötteissä.

Kun kouluttaja on täyttänyt kaikki kentät, hän lähettää tiedot järjestelmään, joka tallentaa ne tietokantaan. Tietokannasta lähetetään järjestelmälle vaste, jossa käy ilmi, onnistuiko tallennus. Jos tallennus on onnistunut, kouluttajalle lähetetään viesti käyttöliittymään käyttäjien lisäyksestä tietokantaan. Viestin lukemisen jälkeen käyttäjä ohjataan pääsivulle. Lopuksi järjestelmä lähettää aktivointilinkin jokaiseen tallennettuun sähköpostiin.

3.2.4 Sivuston toteutusnäkymä

Toteutusnäkymä on tiedostonäkymä projektista. Näkymän visualisoinnissa käytetään yleensä komponentti- tai pakettikaaviota.

Komponenttikaavio kuvaa komponentit ja niiden väliset suhteet. Komponentti voi olla rajapinnan toteuttava olio tai luokka. Myös esimerkiksi html-sivu, dokumentti, kirjasto tai jokin muu osa. Kyseisiä kaavioita käytetään hyvin suurten systeemien riippuvuussuhteiden määrittelyssä. Projektin tavoitteiden kannalta resurssien käyttäminen sivuston komponenttien kuvaamiseen ei anna työpanokseen nähtävää lisäarvoa, joten en toteuta sivuston komponenttikuvausta tässä projektissa.

Käyttöönottokaavio on kuvaus sivuston jokaisesta rajapinnasta käyttäjän näytöltä selaimeen ja selaimesta palvelimelle. Siinä näytetään jokainen järjestelmään kuuluva laitteisto. Sijoittelukaavioita käytetään kuvaamaan aktiiviset laiteyksiköt, niiden yhteydet ja ohjelmistojen osien sijoittumisen niihin.

En käytä web-sivuston sijoittamisen kuvaamiseen kaavioita, sillä sijoittaminen on hyvin suoraviivainen prosessi, jossa muutan ympäristön paikallisesta virtuaaliympäristöstä palveluntarjoajan palvelinympäristöön. Laitteistolle ei ole merkittäviä rajoitteita.

3.3 Toteutuksessa käytetyt tekniikat

Toteutuksessa on käytetty Composer-paketinhallintaohjelmaa, joka hallinnoi sivuston sisäisiä riippuvuuksia, MySQL -relaatiotietokannan hallintajärjestelmää ja Laravel-ohjelmistokehystä sivuston rakentamisessa. Toteutuksessa käytettiin myös PuTTY-emulaattoria, jolla otin yhteyden palvelimelle.

3.3.1 Composer

Composer on projektipohjainen PHP:n riippuvuuksienhallintatyökalu, jonka avulla ohjelmoijan ei tarvitse huolehtia projektin riippuvuuksista yksilötasolla. Composer ei oletuksena asenna mitään globaalisti tietokoneelle.

Composeriin ilmoitetaan kirjastot, joista projekti on riippuvainen ja ohjelma asentaa paketit, jotka se määrittelee tarpeellisiksi asentaa. (Adermann & Boggiano 2018.) Laravel -projekti voidaan asentaa paikalliseen ympäristöön käyttämällä Composeria.

3.3.2 MySQL

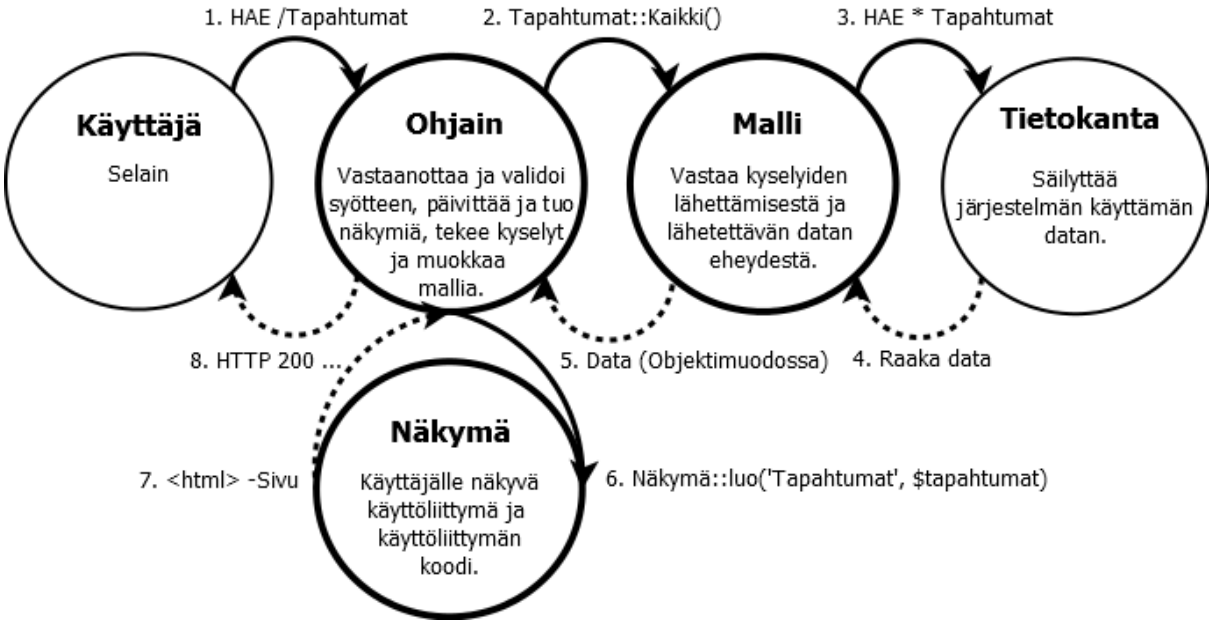
MySQL ei itsessään ole tietokanta. Se on ohjelmisto, joka mahdollistaa matalan budjetin relaatiotietokantojen toteutuksen, hallinnan ja ylläpidon.

Tämän tyyliset sovellukset tunnetaan tietokantojen hallintajärjestelminä. MySQL on relatiivinen tietokantojen hallintajärjestelmä. Relaatiotietokanta on tietokanta, joka organisoi datan tauluihin ja esittää suhteita taulujen välillä. Suhteet taulujen välillä mahdollistavat datan yhdistämisen useista eri tauluista mahdollistaen useita erilaisia näkymiä datasta.

MySQL alkoi matalan tason vaihtoehtona voimakkaammille tietokannoille, mutta se on ajan myötä kasvanut kattamaan myös suurempien yritysten tarpeet. (Reese, Yarger, King & Williams 2002, 3-4.)

3.3.3 Laravel

Sivusto on toteutettu Laravel 5.6 frameworkilla. Web-sivustolla käytettyjä ohjelmointikieliä ovat PHP, HTML, CSS, Bootstrap, JavaScript ja SQL. Laravel pohjautuu MVC-malliin, jonka ominaisia osia ovat Malli (Model), Näkymä (View) ja Ohjain (Controller), joista lyhenne muodostuu. MVC-mallissa HTTP-kutsu käsitellään käyttäjän, ohjaimen, mallin, tietokannan ja näkymän välillä (Kuvio 19.). Jokaisella komponentilla on oma spesifi tehtävänsä kutsun käsittelyssä.



Kuvio 19. HTTP-kutsun käsittely Laravelissa

Mallit ovat vastuussa tietokantojen välisten suhteiden ja yhteyksien luomisesta, näkymät luovat käyttöliittymän sivustolle ja ohjaimet sisältävät tietokantoihin tehtävien pyyntöjen käsittelylogiikan. Lisäksi mallille olennainen osa on Reitit (Routes), joka toimii muiden komponenttien yhdyskäytävänä.

Laravel toiminnot pohjautuvat oletuksena nimeämiskäytäntöihin, joten projektin onnistuminen on myös paljolti kiinni näiden käytäntöjen noudattamisesta. Nimeämiskäytäntöjen noudattaminen luo sivuston toteutukselle ennustettavuutta, jolloin koodin lukeminen helpottuu huomattavasti.

Laravelilla on joitain vaatimuksia järjestelmälle. Järjestelmässä on oltava PHP versio >= 7.1.3 ja ympäristön on tuettava OpenSSL, PDO, Mbstring, Tokenizer, XML, Ctype ja JSON PHP-laajennuksia. Ainakin, mikäli kyseinen toiminto halutaan käyttöön. Tulevissa kappaleissa käyn läpi Laravelin toiminnan kannalta keskeiset työkalut ja tekniikat, miten tekniikat toimivat ja miksi näin tehdään.

Artisan on Laravelin komentorivityökalu, johon on rakennettu käskyjä, joilla pystytään luomaan pohjia esimerkiksi kontrollereista, malleista ja migraatioista. Pohjiin voi komennolla asentaa valmiita ennakkotietoja, jolloin minimoidaan ohjelmoijan tekemä toistuva ja aikaa vievä työ. (Otwell 2018a.)

Artisanilla voidaan käynnistää paikallinen kehityspalvelin (Kuvio 20.) antamalla käynnistyskomento Laravel –projektitiedoston sisällä. 127.0.0.1:8000 on tietokoneen localhost portissa 8000. Portin voi tarvittaessa määritellä erikseen.

```
C:\>cd xampp/htdocs/Laravel/blog

C:\xampp\htdocs\Laravel\blog>php artisan serve
Laravel development server started: <http://127.0.0.1:8000>
```

Kuvio 20. Laravel kehityspalvelimen käynnistäminen.

Kehityspalvelin on tarkoitettu nopeaan kehitykseen tilapäisvälineenä, sillä Laravel Homestead -virtuaalikone on laajempi kehitysympäristö, mutta sen asentaminen vie huomattavasti aikaa.

Reittejä (Routes) käytetään näkymien, suojauksen ja kontrollerien yhteenlinkittämisessä. Jos näkymää ei ole määritelty reittitiedostoihin, siihen ei saa yhteyttä sivustolla.

Kaikki reitit Laravelissa määritellään reittitiedostoissa, jotka ovat Routes kansion sisällä. Routes kansiossa sijaitseva web.php tiedosto määrittää kaikki reitit, jotka on tarkoitettu web-käyttöliittymälle. Näille reiteille määritetään automaattisesti web-väliohjelmaryhmä, joka tarjoaa sisäänrakennettuja ominaisuuksia, kuten istuntotilamuuttujan ja CSRF-suojauksen. Web.php tiedostoon määritellyt reitteihin käyttäjä voi ottaa yhteyden syöttämällä määritellyn URL:n selaimeen. (Otwell 2018b.) Kuvio 21. esitellään tapa asettaa reitti *web.php*–tiedostoon silloin, kun reitteihin ei ole määritelty pääsyvaatimuksia.

```
Route::get('/welcome', 'HomeController@index');
Route::Resource('/contacts', 'ContactsController');
Route::Resource('/news', 'NewsController');
```

Kuvio 21. Sivuston julkiset reitit

Route::get() komennolla, luodaan yksittäinen polku ja *Route::Resource()* -komennolla luodaan polut ennalta määriteltyjen *CRUD*-ominaisuuksien (create, read, update ja delete) avulla. *Route::Resource()* komentoa käytetään, kun halutaan toteuttaa koodista yksinkertaisempi malli, sillä jokaista toiminnallisuutta ei tällöin tarvitse listata erikseen.

Ensimmäinen parametri määrittää sivuston domainin jatko-reitin, eli sivulle welcome päästään osoitteesta http://playfullearning.fi/welcome. Sivustolle, jolle on määritelty CRUD -ominaisuudet, luodaan sivustoreitti, joka sisältää index, create, store, show, edit, update ja destroy -ominaisuudet.

Ominaisuuksista index, show, create ja edit ovat käyttäjälle suunnattuja sivuja, joissa käyttäjä katsoo kaikkia tai yhtä instanssia, luo instanssin tai muokkaa instanssia HTML-lomakkeiden välityksellä. Store, update ja destroy -ominaisuudet käsittelevät lähetetyn tietokantapyynnön ja palauttavat lopuksi ennalta määritellyn sivun, eli näillä sivuilla ei ole visuaalista ulkomuotoa. Toinen parametri määrittelee sivuston ohjaimen.

Kuvio 22. Reittimäärittely on *web.php* tiedostossa oleva toiminnallisuus, jossa käytetään Route::group() metodia, jolla jaetaan usealle reitille kerralla middleware tai nimiavaruus. Tässä tapauksessa reiteille jaetaan Laravelin sisäänrakennettu middleware Auth, jolloin kirjautumaton käyttäjä ei pääse sivuille.

```
Route::group(['middleware'=>'auth'], function() {
    Route::Resource('/events', 'EventsController');
    Route::get('/events/index/user', 'EventsController@userIndex');

    Route::Resource('/tags', 'TagsController');
    Route::Resource('/users', 'UsersController');
    Route::Resource('/photos', 'PhotosController');
    Route::Resource('/account', 'AccountsController');
});
```

Kuvio 22. Kirjautuneille käyttäjille tarkoitetut reitit

api.php -tiedosto sisältää ohjelmointirajapinnoille tarkoitetut polut. Tähän tiedostoon voidaan rekisteröidä uusia ohjelmointirajapintoja.

Ohjaimia (Controllers) käytetään yleisesti käsittelemään sivustolta palvelimelle menevät pyynnöt esimerkiksi tallennettaessa tai poistettaessa tietoja tietokannasta näkymän välityksellä. Ohjaimet voivat ryhmitellä tietyn mallin kaikkien pyyntöjen käsittelylogiikan yhdeksi luokaksi. Tämä on valtava etu koodin luettavuuden kannalta, sillä kaikki mallin käsittelylogiikka löytyy samasta paikasta. Ohjain jakautuu kahteen osaan, joita ovat määrittely- (Kuvio 23.) ja toiminnallinen osa (Kuvio 24.). (Otwell 2018c.)

```
namespace App\Http\Controllers;

use Illuminate\Http\Request;
use App\News;
```

Kuvio 23. Ohjaimen määrittelyosa

Määrittelyosaan tyypillisesti määritellään käytettävät mallit ja nimiavaruus. Nimiavaruus *App\Http\Controllers* on se kansio, jossa ohjaimet sijaitsevat. *Illuminate\Http\Request* luokka vastaa HTTP-pyynnön saavuttamisesta. Luokka toimii tyyppi-vihjeiden avulla, joilla määritellään funktiolle annettavan pyynnön formaatti. Use App\News tarkoittaa käytännössä sitä, ettei ohjelmoijan tarvitse jokaisella luokan käsittelykerralla syöttää koko polkua *App\News*, vaan hän voi syöttää pelkästään News hakiessaan tietoa *App\News* luokasta.

```
class NewsController extends Controller
{

    public function index()
    {
        $news = News::all();

        return view('news.index', compact('news'));
    }
}
```

Kuvio 24. Ohjaimen toiminnallinen osa

Toiminnallinen osa alkaa luokan määrittelyllä. Uutisohjain täydentää ohjain -luokkaa, joka on Laraveliin asetettu ohjainmalli. Ohjainmalli tarjoaa useita ohjelmointia helpottavia metodeja, esimerkiksi show-metodin (Kuvio 25.). Myös middlewaren voi helposti sisällyttää ohjaimen toimintoihin, jos luokka täydentää ohjainmallia.

```
/**
 * Display the specified resource.
 *
 * @param  int  $id
 * @return \Illuminate\Http\Response
 */
public function show($id)
{
    $news = News::findOrFail($id);

    return view('news.show', compact('news'));
}
```

Kuvio 25. Esimerkki ohjaimen toiminnallisuudesta

Ennen metodia show, asetetaan sille **tyyppi-vihjeenä** parametri, joka pitää sisällään kokonaislukumuuttujan *$id*. Samalla tyyppi-vihjeeksi määritellään palautusarvona **Response** -objekti, joka pitää sisällään tiedot esimerkiksi vasteen statuksesta HTTP -statuskoodina ja vasteen alkuperäisen sisällön.

Funktion tulee sisältää tyyppi-vihje *$id* ja käytännössä tämä tarkoittaa sivustopolun olevan http://playfullearning.fi/news/show/{id}, eli *$id* -muuttujalla määritellään, mitä uutista halutaan katsoa. *$news* muuttujaan etsitään uutinen tietokannasta, jonka tunniste on sama kuin *$id*. Funktiossa ladataan määritelty näkymä ja ‘compact’ metodi lähettää muuttujan $news näkymään, jossa se voidaan visuaalisesti esittää loppukäyttäjälle.

Näkymät (Views) ovat *HTML* ja *CSS* -koodista koostuvia tiedostoja, jotka luovat sivustolle visuaalisen ilmeen ja esittävät käyttäjälle sivuston. **Blade** on Laravelin käyttämä yksinkertainen mallinnusmoottori, joka helpottaa muuttujien ja ehtolauseiden rakentamisen näkymän sisälle. (Otwell 2018d.)

Sivustolla on päänäkymä, joka pitää sisällään jokaisen sivuston jokaisella sivulla esiintyvän elementin. Päänäkymään on määritelty navigointipalkki, alatunniste,

sivuston sisältö, joka tuodaan toisista näkymistä päänäkymään, sivuston yhteinen CSS-koodi ja sivuston otsikko. Kuvio 26. pitää sisällään esimerkin päänäkymän sisällöstä ja alatunnisteesta.

```
<div class="content">
    @yield('content')
</div>


<div class="footer">
@yield('footer')
</div>
```

Kuvio 26. Päänäkymän sisältö ja alatunniste

Päänäkymä käyttää alinäkymiä, jotka sisältävät sivuston sisällön ja sivun käyttämät elementit (Kuvio 27.). Alinäkymiä käytetään lohkottamaan koodi pienemmiksi pätkiksi, mikä tekee koodista helpommin luettavaa.

```
@extends('layouts.app')

@section('sidebar')
    @extends('teaching_events/partials/sidebar')
@endsection

@section('content')

    <h1><a href="{{route('teaching_events.edit', $teaching_event->id)}}">{{$teaching_event->header}}</a></h
@endsection
```

Kuvio 27. Alasivulla määritelty sivuston sisältö

Alinäkymässä määritellään komennolla *@extends('layouts.app')*, että se käyttää päänäkymää, joka sijaitsee kansiossa *layouts* tiedostona *app.blade.php*. Komento *@section('content)* määrittelee, mitä näytetään päänäkymän *@yield('content')* -komennon määrittelemässä kohdassa. Komento *@endsection* sulkee sisältöalueen, eli mitään sen jälkeen tulevaa ei näytetä *@yield* komennossa.

Migraatiot ovat taulun luontia helpottavia tiedostoja, joihin yleisesti määritellään taulun, tai sen osan, sisältö lisäys- (Kuvio 28.) ja poistovaiheessa (Kuvio 29.). Migraatiot voivat sisältää taulun luonnin, päivityksen ja poistamisen. Laravel pitää kirjaa migraatioista, joka mahdollistaa erilaisten komentojen käytön. Esimerkiksi viimeksi tehtyjen muutosten poistaminen tapahtuu *rollback*-komennolla. (Otwell 2018e.)

```php
public function up()
{
    Schema::defaultStringLength(200);
    Schema::create('users', function (Blueprint $table) {
        $table->increments('id');
        $table->string('name');
        $table->integer("age")->nullable();
        $table->string('nationality')->nullable();
        $table->integer('role_id')->default(1);
        $table->string('email')->unique();
        $table->string('password');
        $table->rememberToken();
        $table->timestamps();
    });
}
```

Kuvio 28. Käyttäjät –taulun migraation ylös ajo-funktio

Kaavio (Schema) on Laravelissa oleva luokka, joka tarjoaa tavan tietokannan taulujen käsittelyyn. Kaavio toimii hyvin kaikkien Laravelin tukemien tietokantojen kanssa. Määrittelen kaavion alustavan merkkijonojen pituuden. Sen jälkeen luon kaavion, jonka ensimmäinen parametri on tietokantaan tulevan taulun nimi ja funktion muuttujaksi asetetaan Blueprint -objekti $table. Blueprint objektia käytetään lisäämään kenttiä tietokannan tauluihin niiden luomisen yhteydessä.

Funktion sisällä määrittelen kaikki taulun kentät $table muuttujaan. $table -muuttujaan syötetään jokainen tauluun lisättävä kenttä. Blueprint -objektiin on määritelty käytettävät muuttujat, jotka ottavat parametrina kentän nimen ja jatkofunktioina lisämääritelmiä luotavalle kentälle. Esimerkiksi määritelmä ”*nullable*” mahdollistaa kentän tyhjäksi jättämisen lisäystoimenpiteen yhteydessä ja default asettaa automaattisesti arvon, vaikka käyttäjä ei anna arvoa kentälle luonnin yhteydessä.

```php
/**
 * Reverse the migrations.
 *
 * @return void
 */
public function down()
{
    Schema::dropIfExists('users');
}
```

Kuvio 29. Migraation poistofunktio

Poistovaiheessa tieto poistetaan tietokannasta. Tietokannat määritellään .env (environment) -tiedostossa, johon on luotu valmiiksi useita eri vaihtoehtoja tietokannoille ja valmiit muuttujat jokaiselle tuetulle tietokantatyypille. Kehittäjä ainoastaan päivittää tiedot oikeiksi.

.env tiedostoa ei tule koskaan jakaa julkisesti, sillä example.env tiedosto sisältää pohjan *.env*-tiedostolle. *.env*-tiedosto sisältää arkaluontoista tietoa, kuten tietokantojen salasanoja. *.env*-tiedosto sisältää nimensä mukaisesti ympäristömuuttujat tietokantayhteyksissä.

Mallit (Eloquent Models) ovat tietokantatyöskentelyä helpottavia luokkia. Mallit sallivat tiedon hakemisen taulusta ja tiedon lisäämisen tauluun. Esimerkiksi roolin malli sisältää kaiken tarvittavan tiedon roolin relaatioista (Kuvio 30.) muihin tauluihin ja tietokannassa sijaitsevan taulun täytettävät ja piilotetut kentät.

Modeliin voidaan sijoittaa myös funktioita, jotka muokkaavat haluttua dataa joko ennen tiedon hakemista tai tiedon hakemisen jälkeen. **Accessorit** muokkaavat haettua dataa ennen hakua ja **Mutatorit** haun jälkeen. Käytännössä Accessorit ovat get-datan muokkaimia ja Mutatorit set-datan muokkaimia. (Otwell 2018f.)

```php
<?php

namespace App;

use Illuminate\Database\Eloquent\Model;

class Role extends Model
{
    protected $fillable = [
        'name',
        'description'
    ];

    public function user() {
        return $this->hasMany('App\User');
    }

}
```

Kuvio 30. Esimerkki mallista

Rooli -mallin määrittelyssä käytetään yleistä mallien määrittelymenetelmää, jossa luokka jatkaa pohjamallia.

Pohjamalliin on asetettu ennalta määriteltyjä tietoja, joita voidaan muokata mallissa, jos kehittäjä ei noudata yleisiä nimeämiskäytäntöjä. Esimerkiksi pääavain on määritelty muuttujaan *$primaryKey*, jossa kentän nimi on oletuksena ‘id’ ja muuttuja *$keyType* sisältää id-kentän tyypin, joka on vakiona kokonaislukumuuttuja.

Pohjamallia ei saa muokata, sillä se periytyy tyypillisesti useille eri malleille. Mallissa muokattu muuttuja *$fillable* pitää sisällään taulukon, johon määritellään taulun kentät, joita käyttäjä voi muokata.

Funktio *user* sisältää tietokannan relaation toiseen tauluun. Roolitaulu on yhteydessä käyttäjätauluun, jossa käyttäjällä on kenttä role_id, joka pitää sisällään käyttäjän roolin.

```php
namespace App;

use Illuminate\Notifications\Notifiable;
use Illuminate\Foundation\Auth\User as Authenticatable;

class User extends Authenticatable
{
    use Notifiable;

    // These fields are mass-assignable
    protected $fillable = [
        'name', 'email', 'password',
    ];


    // These fields are excluded from JSON representations.
    protected $hidden = [
        'password', 'remember_token',
    ];

    public function role() {
        return $this->belongsTo('App\Role');
    }
}
```

Kuvio 31. Käyttäjän malli

Käyttäjämalli käyttää Notifiable luokkaa, joka sisältää tuen sähköpostien, SMS ja Slack -viestien lähettämiselle.

$hidden -muuttuja pitää sisällään kentät, jotka leikataan pois mallin taulukko ja JSON representaatioista. Toinen vaihtoehto on käyttää $visible kenttää, johon määritellään kaikki kentät, joiden halutaan näkyvän mallin lähettämissä taulukoissa ja muut kentät ovat automaattisesti piilotettuja.

Käyttäjätaulusta on yhteys roolitauluun samalla tapaa, kuin roolitaululla on yhteys käyttäjätauluun.

Yleensä toinen yhteyksistä on käänteinen. Esimerkiksi jos videolla voi olla monta kommenttia, on yhteys hasMany -tyyppinen. Kommentilla yhteys puolestaan on belongsTo -tyyppinen, sillä kommentilla ei voi olla montaa videota.

Tinker on Laravelissa oleva työkalu, jolla käsitellään tietokantayhteyksiä käyttämällä malleja. Tinker on komentorivityökalu ja toimii samalla periaatteella kuin kontrolleriin tehtävä logiikka tiedon syöttöön, mutta on usein nopeampi ja turvallisempi käyttää kuin ohjelman muokkaaminen tai tietokantaan manuaalisesti tietojen lisääminen.

HTML-lomakkeet (HTML-forms) (Kuvio 32.) sisältävät erilaisia syöte-elementtejä, kuten syötekenttiä (tekstikentät ja valintakentät) ja erilaisia painikkeita. Lomakkeet ovat tapa kuljettaa tietoa käyttäjän ja palvelimen välillä. Lomakkeiden validointi voidaan toteuttaa metodina, luokkametodina tai kontrollerin osana. (Otwell 2017.) Mediatiedostojen lataaminen palvelimelle toteutetaan osana lomaketta.

```
{!! Form::open(['method'=>'POST', 'action'=>'EventsController@store', 'files'=>'true']) !!}

<div class="form-group">

    {!! Form::label('header', 'Header:') !!} <br>
    {!! Form::text('header', null, ['class'=>'form->control']) !!}
    {!! Form::file('step_1_file', ['class'=>'form->control']) !!}

</div>

<div class="form-group">
    {!! Form::submit('Create an Event', ['class'=>'btn btn-primary']) !!}
    {{ csrf_field() }}
</div>
{!! Form::close() !!}
```

Kuvio 32. Esimerkki HTML-lomakkeesta

Lomake on yhdistelmä lomake-elementtejä ja asetteluelementtejä. Kun lomake avataan, on sille määriteltävä lähetysmetodi. Lähetysmetodin määrittelee ohjaimen metodi, johon se lähetetään (Kuvio 33.).

Verb	URI	Action	Route Name
GET	/photos	index	photos.index
GET	/photos/create	create	photos.create
POST	/photos	store	photos.store
GET	/photos/{photo}	show	photos.show
GET	/photos/{photo}/edit	edit	photos.edit
PUT/PATCH	/photos/{photo}	update	photos.update
DELETE	/photos/{photo}	destroy	photos.destroy

Kuvio 33. Laravelin tukemat lähetysmetodit (Otwell 2017g)

Verb kuvaa lähetysmetodia. *URI* on sivustopolku projektin sisällä ja *Action* kuvaa ohjaimen metodia. Toisena on määriteltävä ohjaimen metodi. Kolmantena on muuttuja, jonka avulla järjestelmä ottaa lomakkeesta vastaan tiedostoja ja tallentaa ne public/images -kansioon.

Lomakkeen ensimmäinen elementti on HTML Label, jonka ensimmäinen parametri on sen nimi näkymässä. Toinen parametri on siinä näkyvä teksti. Lomakkeen toinen elementti on HTML tekstikenttä, jonka ensimmäinen parametri on sen nimi, toinen parametri on sen esiasetettu arvo ja kolmas parametri on sen luokka näkymässä. Kolmas elementti on tiedostoelementti, jonka ensimmäinen parametri on sen nimi ja toinen parametri luokka. Tiedostoelementti luo painikkeen ja tekstikentän, jossa näkyy valitun tiedoston nimi. Painiketta painettaessa avautuu loppukäyttäjän tietokoneen tiedostonhallintaohjelma, jonka kautta voi valita lähetettävän kuvan. Neljäs elementti on lomakkeen lähetyspainike, jonka avulla lomake lähetetään ohjaimeen. Ensimmäinen parametri on painikkeen päällä oleva teksti ja toinen parametri on painikkeen luokka. (Otwell 2017g.)

Viides elementti on CSRF-merkki. Laravel käyttää CSRF-suojausta (Cross-Site Request Forgery protection) lomakkeissa, joka estää käyttäjän selainta toteuttamasta ei-haluttuja, toiselta sivulta tai ohjelmalta tulevia käskyjä. Käyttäjän kirjautuessa palveluun suojaus otetaan käyttöön. Lopulta lomake suljetaan. Sulkemiselementin jälkeen lomake ei huomioi muita elementtejä.

Lomakkeiden yhteydessä kulkee $errors-muuttuja, jonka avulla järjestelmästä pystytään esittämään loppukäyttäjälle virheilmoituksia lomakkeen lähetyksen yhteydessä (Kuvio 34.).

```
@if(count($errors) > 0)

    <div class="alert alert-danger">
        <ul>
            @foreach($errors->all() as $error)
                <li>{{$error}}</li>
            @endforeach
        </ul>
    </div>
@endif
```

Kuvio 34. Virheilmoitukset näkymässä

$errors-muuttuja voidaan ottaa käyttöön, kun ohjaimen metodiin asetetaan validointitoiminto (Kuvio 35.). Toiminnolla tarkastellaan lomakkeen mukana tulleita arvoja.

```
/**
 * Store a newly created resource in storage.
 *
 * @param  \Illuminate\Http\Request  $request
 * @return \Illuminate\Http\Response
 */
public function store(Request $request)
{

//request class is super global, it is used to hold the form data.

    // Validate request data

    $this->validate($request, [
        'header' => 'required|max:35',
        'phase_1_comment' => 'required|max:200',
        'phase_2_comment' => 'required|max:200',
        'phase_3_comment' => 'required|max:200',
        'phase_4_comment' => 'required|max:200'
    ]);
```

Kuvio 35. Syötteen validointi

$request on muuttuja, joka pitää sisällään lomakkeen lähettämät tiedot ohjaimessa. Validointimääreiksi on listattu vaatimukset, että se on olemassa ja merkkijonon maksimipituus on 200 merkkiä. Muita mahdollisia määreitä ovat esimerkiksi "*bail*", joka lopettaa määreiden tarkastelun ensimmäiseen tarkasteluun, joka ei mene läpi ja "*min*", joka määrittelee pienimmän mahdollisen merkkijonon pituuden.

Artisaniin on rakennettu toiminto, jolla voidaan toteuttaa valmis pohja kirjautumiselle, joka suojaa salasanat *hashauksella*, sisältää istunto-tokenin ja sähköpostin linkityksen sivustoon, jonka avulla voidaan lähettää salasanan nollausviesti käyttäjän määrittelemään sähköpostiin.

Ajamalla Artisan-komento *"php artisan make:auth"* rakentuu projektiin järjestelmä kirjautumiselle. Järjestelmä sisältää tarvittavat näkymät ja ohjaimet valmiina. (Otwell 2018g.)

Middleware (Sivuston suojauselementti) sisältää suojausmekanismeja sivustolle tulevien HTTP-pyyntöjen filtteröinnille. Esimerkiksi Laravel sisältää Middlewaren, joka tarkastelee, onko käyttäjä kirjautunut sisään palveluun. Jos käyttäjä ei ole kirjautunut ja hän yrittää päästä käsiksi materiaaliin, joka on käytettävissä vain kirjautuneille käyttäjille, hänet ohjataan middlewaren toimesta login-sivulle. Middlewaren käyttöönotossa middleware määritellään kernel.php tiedostoon.

Middlewaret voivat toteuttaa useita eri toiminnallisuuksia. Esimerkiksi middleware voi asettaa tunnisteen järjestelmästä lähteviin rajapinta-kutsuihin tai kirjoittaa ylös kaikki pääkäyttäjän järjestelmään tekemät muutokset, jolloin ylläpitovaiheessa säästyy resursseja, jos muutoksia tehdään runsaasti tai pääkäyttäjiä on enemmän kuin yksi. (Otwell 2018h.) Middleware määritellään ohjaimen *__construct()*-funktioon (Kuvio 36.), jolloin se otetaan käyttöön, kun ohjaimessa ajetaan metodi.

```
class EventsController extends Controller
{

    public function __construct() {
        $this->middleware('IsAdmin');
    }
```

Kuvio 36. IsAdmin –middlewaren käyttöönotto

Funktio __contruct() sisältää ohjaimen logiikan, joka ajetaan aina ennen muita metodeja. Sen takia se on täydellinen paikka määritellä middleware. Olen rakentanut IsAdmin-middlewaren, joka tarkastaa, onko käyttäjä pääkäyttäjä sivustolla (Kuvio 37.).

```php
namespace App\Http\Middleware;

use Closure;
use Illuminate\Support\Facades\Auth;

class IsAdmin
{
    /**
     * Handle an incoming request.
     *
     * @param  \Illuminate\Http\Request  $request
     * @param  \Closure  $next
     * @return mixed
     */
    public function handle($request, Closure $next)
    {
        $user = Auth::user();
        if ($user == null || !$user->isAdmin()){
            return redirect()->intended('/login');
        }

        return $next($request);
    }
}
```

Kuvio 37. Middlewareen tulevien kutsujen käsittelijä

Middleware on luokka, joka sijaitsee tavallisesti nimiavaruudessa *App\Http\Middleware*. Closure on luokka, jota käytetään esittämään tuntemattomia funktioita. (The PHP Documentation Group 2018). Niitä käytetään kutsumismenetelminä ja parametreina funktioissa.

Illuminate\Support\Facades\Auth käsittelee käyttäjän tietoja järjestelmässä. Middlewaressa halutaan päästä käsiksi käyttäjän rooliin, joten joudumme hakemaan tiedon *Auth*-komponentin kautta. *Handle*-funktio käsittelee kaikki ohjaimesta tulevat pyynnöt. Ensimmäisenä funktiossa haetaan käyttäjän instanssi, joka mahdollistaa pääsyn käyttäjän tietoihin.

Jos käyttäjä ei ole kirjautunut sisään, palautetaan haussa *NULL*-arvo. ||-symboli tarkoittaa tässä yhteydessä loogista tai -lausetta. Jos edes toinen vertailu palauttaa totuusarvon tosi, ohjelma menee *if*-lohkon sisälle. Toisessa vertailussa ajetaan käyttäjä-mallin sisällä oleva funktio *isAdmin*. !-merkki tarkoittaa, että funktio palauttaa epätoden arvon.

Käyttäjämallissa oleva funktio *isAdmin()* (Kuvio 38.) sisältää yksinkertaisen vertailun. Jos käyttäjän roolin nimikentässä oleva arvo on Admin, palauttaa vertailu arvon tosi. Käyttäjän kautta voidaan hakea rooli, vaikka ne ovat eri tauluissa, sillä käyttäjä-mallilla on tietokantasuhde rooli-malliin.

```
public function isAdmin() {

    if($this->role->name == 'Admin') {
        return true;
    } else {
        return false;
    }
}
```

Kuvio 38. Käyttäjämallin pääkäyttäjävertailu

3.3.4 PuTTY

Käytin PuTTY pääte-emulaattoria ottaessani yhteyden palvelimeen. Jouduin luomaan yhteyden palvelimelle, sillä ajoin Laraveliin rakennettuja artisan-komentoja.

3.4 Tutkimuksen testausvaihe

Tutkimuksen testausvaiheessa perehdytään toteutettuihin ja suunniteltuihin testauksiin. Testaustuloksia en ehtinyt kerätä opinnäytetyöhön aikataulutuksen takia.

3.4.1 Testaus suunnitteluvaiheessa

Staattista testaamista tapahtui jonkin verran jo suunnitteluvaiheessa, sillä huomasin joitain virheitä suunnittelutason kaavioissa. Kaavioiden nopea vanhentuminen oli isoin yksittäinen ongelma, sillä monesti saimme uusia näkökulmia,

mutta emme päivittäneet kaavioita välittömästi, jonka seurauksena jouduin päivittämään tietokanta- ja prosessikaaviot ja suunnittelemaan osan sivuston toiminnoista uudelleen.

3.4.2 Testaus kehitysvaiheessa

Testaus integroitui pääosin osaksi toteutusta, sillä virheet hajottivat järjestelmän, eikä edistymistä tapahtunut ennen virheiden korjaamista. Ominaisuudet, jotka aiheuttivat virheitä, olivat määritelty pääosin järjestelmän määrittely- ja suunnitteluvaiheissa osaksi järjestelmää toimintoja. Testaus tapahtui tässä vaiheessa lasilaatikkotestauksena, sillä testasin tuottamani koodin kattavuuden toteutuksen yhteydessä.

3.4.3 Testaus ennen julkaisua

Opinnäytetyö kirjoitettiin vähän ennen julkaisua. Ennen julkaisua toteutettavan testauksen tulisi sisältää käytettävyystestaus, kourmitustestaus ja savutestaus. Käytettävyystestaus toteutetaan henkilöiden avulla, jotka eivät ole olleet mukana sivuston kehityksessä ja joilla on eritasoisia lähtökohtia sivuston käyttöön. Kuormitustestauksessa aion käyttää yhtä tai kahta tulevassa kappaleessa esiteltyä testauksessa käytettävää työkalua. Projektin yhteydessä tapahtuva kuormitus web-sivustolla ei sisällä kuin kaksikymmentä yhtäaikaista käyttäjää palvelimella, joten kevyet testaustyökalut käyvät hyvin tarkoitukseen. Savutestaus toteutetaan opetustapahtumassa, jonka yhteydessä sivusto otetaan käyttöön.

3.4.4 Testauksessa käytettävät työkalut

Olen suunnitellut käyttäväni testauksessa **Laravel Duskia** ja **Selenium** -ohjelmointiympäristöä. **Dusk** on Laraveliin sisäänrakennettu testaustyökalu. Duskin ominaisuuksia ovat helppokäyttöinen selainautomaatio ja testausrajapinta. **Laravel Dusk** käyttää Google Chromea oletus-selaimena testauksessa. Duskiin yhteyteen voidaan asentaa Selenium -palvelin, jolla voidaan ajaa testejä suosituimmilla selaimilla. (Otwell 2018i.)

Selenium on kokonaisuus erilaisia ohjelmistotyökaluja, joilla pyritään tukemaan testausautomaatiota eri näkökulmista. Yleinen lähestymistapa on opetella yksi tai

kaksi työkalua, jotka tukevat projektin tarpeita. Selenium on lisensoitu **Apache** lisenssinä, joka takaa ilmaisen käytön loppukäyttäjälle. Käytettäessä ohjelmistoa loppukäyttäjä on vastuussa toteutetuista skripteistä ja niiden tarkoitusperistä.

Seleniumin tuotevalikoimaan kuuluvat **Selenium 2, Selenium 1 Selenium IDE** ja **Selenium-Grid**. Yleisin aloitusvalinta on Selenium IDE, jota käytetään yksinkertaisiin testeihin. Selenium on suunnattu erityisesti kaikenlaisten web-applikaatioiden testaamiseen. (Selenium Project 2018.)

3.5 Käyttöönotto-, ylläpito- ja jatkokehitys-vaihe

Sivusto otetaan käyttöön smoke testissä, jossa järjestetään opetustapahtuma. Opetustapahtumaan osallistuu noin 20 koehenkilöä, joiden avulla järjestelmää testataan.

Käyttöönottovaiheessa ongelmia aiheutti tarjolla oleva ohjelmiston versio, joka oli vanhempi versio, kuin mikä alustavasti toteutettiin. Vian korjaamiseksi järjestelmää muokattiin vanhemman version kanssa yhteensopivaksi. Web-hotellissa tuettiin Laravelin versiota 5.1 ja sivusto rakennettiin versiolla 5.6. Tämä ei kuitenkaan aiheuttanut vakavia ongelmia, sillä ohjelmistoissa oli vain pieniä eroja funktioiden nimissä ja joitain funktioita ei ollut versiossa 5.1. Myös aikainen ongelman huomaaminen helpotti merkittävästi projektin palauttamista aikatauluun. Sivustolle ei ole tarkoitus toteuttaa käyttöönottovaiheessa manuaalia, vaan sen tulisi olla tarpeeksi intuitiivinen käyttää.

Olen sitoutunut järjestelmän pääkäyttäjäksi ja kehittäjäksi, kunnes työsopimukseni yliopiston kanssa loppuu. Käytössä tulevat virheilmoitukset ilmoitetaan minulle ja teen tarvittavat toimenpiteet ja päivitykset niiden korjaamiseksi. Teknisiä jatkokehitys-suunnitelmia ovat opettajien ja käyttäjien toiminnallisuuksien lisääminen, ehtojen lisääminen sivustolle ja erilaisten CRUD-elementtien lisääminen jokaisen mallin toiminnollisuuksiksi. Toiminnallisia jatkokehitysideoita ovat muun muassa opetustapahtumien näyttäminen listalla arvostelun perusteella ja sähköpostiin asetettava aktivointiaika, eli jos käyttäjä ei aktivoi tiliä tietyn aikamääreen sisällä, luotu käyttäjä poistetaan tietokannasta automaattisesti.

Sivustosta on tulevaisuudessa tarkoitus suunnitella myös mobiiliversio, jolla käyttäjä pystyy toteuttamaan samat asiat kuin sivustolla voidaan tällä hetkellä toteuttaa. Tavoitteena on myös lisätä tietokantaan taulu ryhmille ja tehtäville. Jos käyttäjille voidaan luoda ryhmiä, helpottaisi se suuresti käyttäjien hallintaa. Tehtävien tavoitteena on antaa käyttäjille palautetta suunnitelluista opetustapahtumista. Sivuston kasvaessa ja projektitiimin muuttuessa sivustolle tulee toteuttaa ehdot, jotka käyttäjän on hyväksyttävä ennen sivuston käyttöä.

4 MUUT KÄYTETYT OHJELMISTOT, TEKNOLOGIAT JA TEKNIIKAT

Kaikki sivuston diagrammit on toteutettu **Dia**-diagrammityökalulla. Dia on ilmainen ja avoimen lähdekoodin diagrammiohjelmisto. Dian lisenssityyppi on GPL ja se takaa loppukäyttäjille ilmaisen ohjelmiston suorittamisen, opiskelun, jakamisen ja muokkaamisen. Myös ohjelmistolla tuotettujen diagrammien käyttö on ilmaista.

Sublime Text 3 on maksullinen ja "*Nagware*" tekstieditori. Nagware tarkoittaa ohjelmistotyyppiä, joka on ilmainen käyttää, mutta ohjelmisto ehdottaa tietyin väliajoin pop-up viesteillä ohjelmiston ostamisesta. Kun ohjelmisto on ostettu, pop-up ilmoitukset poistuvat. Sublime Text 3 ei tukenut *.blade.php* -kooditiedostoja. Parempi editori tähän tarkoitukseen olisi ollut JetBrains PhpStorm, mutta en saanut lupaa yritykseltä käyttää ohjelmistoa opinnäytetyöhön, sillä opinnäytetyö oli yhteistyössä Yliopiston kanssa toteutettu projekti.

Käytin **Udemyssä** (https://www.udemy.com/) olevaa Edwin Diazin toteuttamaa verkkokurssia "*PHP with Laravel for beginners - Become a Master in Laravel*" referenssimateriaalina toteutusvaiheessa. Opetuspaketin käyttäminen nopeutti huomattavasti ymmärrystäni Laravel-ohjelmointiympäristön käyttömahdollisuuksista ja -tarkoituksista. Leikkasin koodista otetut kuvakaappaukset **MS Paintilla** pienemmiksi paloiksi.

5 POHDINTA

Tavoitteena opinnäytetyössäni oli tarjota kattava kuva ohjelmiston elinkaaren eri vaiheista vesiputousmallin avulla ja soveltaa mallista käytännön toteutus web-sivuston kehityksessä. Opinnäytetyön toteutusvaiheessa pääsin sivuston kehittämisvaiheen loppupuolelle. Sivuston back-end toimi täysin opinnäytetyön toteutuksen loppupuolella ja sivustolle oli tilattu ulkoasu graafiselta suunnittelijalta. Opinnäytetyössä tarkastelin myös toteutuksessa käytettyjä työkaluja ja esittelin testauksessa käytettäviä työkaluja.

Lähtökohdistani projekti oli kiinnostava ja haastava kokonaisuus. Web-sivustojen kehittäminen kokonaisuutena on monimutkainen ja haastava prosessi, vaikka projekti sisälsi pääosin yksinkertaisia elementtejä. Ongelma oli pikemminkin työtuntien minimoiminen, sillä elementtejä oli todella paljon ja vaihdettaessa elementistä toiseen jouduin lähes aina käymään läpi aikaisemmin tehtyjä samoja elementtejä tai katsomaan dokumentaatiosta, mitä ja millä tavalla elementteihin kehitetään toiminnallisuudet.

Vaikeinta projektissa oli kuitenkin projektin tarpeiden määrittely, sillä monesti tarpeet kuvattiin useista eri näkökulmista ja tarpeet muuttuivat kahdesti projektin toteutuksen aikana, joten osittain jouduin tekemään paikkaavaa työtä. Olen työtä tehdessäni oppinut paljon ohjelmistokehityksen teoriassa esitellyistä menetelmistä ja malleista. Kehityin huomattavasti suunnittelun ja määrittelyn toteuttamisessa. Aluksi yritin liian teoreettista lähestymistapaa, mikä ei toiminut halutulla tavalla. Kun yhdistin määrittely- ja suunnitteluvaiheet, sain tarpeelliset määrittelyt selville sivustosta.

Viimeinen huomioitava asia web-sivuston toteutuksessa oli Scrum –menetelmän käyttäminen osana toteutusta siitä hetkestä lähtien, kun webhotelli tilattiin palvelinpaikka hankkeelle. Vesiputousmallin suurin käytännön ongelma on projektin sidosryhmien ymmärtämättömyys projektin etenemisestä, sillä toteutusvaiheessa ei esitellä tuotetta ollenkaan.

Tämä oli isoin yksittäinen syy Scrumin käyttämiseen projektin toteutusvaiheessa. Toinen syy oli henkilökohtaisen motivaation kasvattaminen, sillä ajallisesti pieniksi pilkottuja kokonaisuuksia oli huomattavan helppo hallita ja tuloksia oli helpompi esitellä hankeen muulle henkilöstölle.

LÄHTEET

Adermann, N. & Boggiano, J. 2018. Introduction. Viitattu 22.3.2018 https://getcomposer.org/doc/00-intro.md.

Haikala, I. & Mikkonen, T. 2011. Ohjelmistotuotannon käytännöt. 12. Helsinki: Talentum Media Oy.

Kasurinen, J. 2013. Ohjelmistotestauksen käsikirja. Jyväskylä: Docendo Oy.

Murch, R. 2002. IT-projektinhallinta. Helsinki: Edita Publishing Oy.

Otwell, T. 2017. Forms & HTML. Viitattu 21.3.2018 https://laravelcollective.com/docs/5.4/html.

- 2018a. Artisan Console. Viitattu 20.3.2018 https://laravel.com/docs/5.6/artisan.

- 2018b. Routing. Viitattu 20.3.2018 https://laravel.com/docs/5.6/routing.

- 2018c. Controllers. Viitattu 20.3.2018 https://laravel.com/docs/5.6/controllers.

- 2018d. Views. Viitattu 20.3.2018 https://laravel.com/docs/5.6/views.

- 2018e. Migrations. Viitattu 20.3.2018 https://laravel.com/docs/5.6/migrations.

- 2018f. Eloquent Models. Viitattu 21.3.2018 https://laravel.com/docs/5.6/eloquent.

- 2018g. Authentication. Viitattu 21.3.2018 https://laravel.com/docs/5.6/authentication.

- 2018h. Middleware. Viitattu 21.3.2018 https://laravel.com/docs/5.6/middleware.

- 2018i. Laravel Dusk. Viitattu 22.3.2018 https://laravel.com/docs/5.6/dusk.

Pohjonen, R. 2002. Tietojärjestelmien kehittäminen. Jyväskylä: Docendo Finland Oy.

Reese, G., Yarger, R., King, T. & Williams, H. 2002. Managing & Using MySQL. 2. Sebastopol: O'Reilly Media.

Selenium Project. 2018. Introducing Selenium. Viitattu 22.3.2018 https://www.seleniumhq.org/docs/01_introducing_selenium.jsp.

The PHP Documentation Group. 2018. The Closure Class. Viitattu 22.3.2018 http://php.net/manual/en/class.closure.php.