



TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

JYRI LAINE

SOVELLUKSEN KÄYTTÖTIEDON KERÄÄMINEN JA ANALYSOINTI

Diplomityö

Tarkastaja: professori Kari Systä
Tarkastaja ja aihe hyväksytty
Tieto- ja sähkötekniikan
tiedekuntaneuvoston kokouksessa
06. helmikuuta 2013

TIIVISTELMÄ

TAMPEREEN TEKNILLINEN YLIOPISTO

Tietotekniikan koulutusohjelma

Laine, Jyri: Sovelluksen käyttötiedon kerääminen ja analysointi

Diplomityö, 50 sivua, 0 liitesivua

Toukokuu 2013

Pääaine: Ohjelmistotuotanto

Tarkastaja: professori Kari Systä

Avainsanat: Käyttötieto, MariaDB, PHP, tietokanta

PiceaHub on Piceasoft-yrityksen kehittämä sovellus, joka on toteutettu helpottamaan mobiililaitteen ja tietokoneen välillä tapahtuvaa tiedonsiirtoa. Sovellusta kehitetään edelleen ja kehittäjien kannalta on hyödyllistä kerätä käyttäjiltä käyttötietoa, jotta sovellusta voidaan parantaa, siinä olevat mahdolliset virheet saadaan korjattua, ja sekä tuotekehitys että markkinointi saadaan suunnattua oikeille alueille.

Tässä diplomityössä suunniteltiin ja toteutettiin järjestelmä, joka mahdollistaa PiceaHub-sovelluksesta käyttötiedon keräämisen, tallentamisen ja analysoidun tiedon esittämisen sovelluskehittäjille. Ennen järjestelmän toteuttamista työssä analysoitiin ja valittiin järjestelmään sopiva tietokannan hallintajärjestelmä. Valittuun tietokannan hallintajärjestelmään puolestaan suunniteltiin ja toteutettiin tietokanta, johon käyttötieto saatiin tallennettua. Tietokannan toteutuksen jälkeen työssä toteutettiin järjestelmän ydin, joka muodostuu käyttötietoa vastaanottavasta sovelluksesta, käyttötietoa tietokantaan jäsentävästä sovelluksesta ja WWW-sivustoista.

ABSTRACT

TAMPERE UNIVERSITY OF TECHNOLOGY

Master's Degree Programme in Information Technology

Laine, Jyri: Collecting and Analyzing Application usage data

Master of Science Thesis, 50 pages, 0 Appendix pages

May 2013

Major: Software Engineering

Examiner: Professor Kari Systä

Keywords: Usage data, MariaDB, PHP, database

PiceaHub is an application developed by a Finnish software company called Piceasoft to simplify data transfer between a mobile phone and a computer. The software development is still under process and to make the software even better, it is beneficial for the developers to collect usage data from the customers. The usage data helps developers to make the software better, fix possible software errors and to direct both the product development and the marketing to the right areas.

In this master thesis a system was designed and engineered to make it possible to collect data, to store it and to show analyzed data from PiceaHub application to the application developers. Before the system was developed a suitable database management system was analyzed and selected. To the selected database management system, a database was designed and engineered to store the usage data. After the database a system core was developed, it consists of an application that receives data, an application that parses data to the database and WWW-pages.

ALKUSANAT

Tämä diplomityö käsittelee PiceaHub-sovelluksen käyttötiedon keräämiseen, tallentamiseen ja analysointiin liittyvän järjestelmän suunnittelua ja toteutusta. Käyttötiedon keräämisen mahdollistavan järjestelmän toteutus aloitettiin vuoden 2013 tammikuussa yhdessä Piceasoft-yrityksen työntekijöiden kanssa. Piceasoftilta työssä mukana olivat erityisesti Jani Väänänen ja Samuli Kivinen.

Haluan kiittää Jani Väänästä, Samuli Kivistä, Kari Systää ja kaikkia muita, jotka ovat osallistuneet diplomityöni ohjaamiseen, lukemiseen ja parannusehdotusten tekemiseen.

Tampereella 12. toukokuuta 2013

Jyri Laine
jyri.laine@tut.fi
040-7590425

SISÄLLYS

Abstract	iii
Termit ja niiden määritelmät	vii
1 Johdanto	1
2 Tausta ja tarpeet	3
2.1 Yleiskuvaus	3
2.2 Toiminnalliset vaatimukset	5
2.3 Ei-toiminnalliset vaatimukset.....	6
2.3.1 Ylläpidettävyys	6
2.3.2 Kustannukset.....	7
2.3.3 Skaalautuvuus	7
2.3.4 Suorituskyky	8
3 Käytetyt tekniikat	9
3.1 WWW-tekniikat	9
3.1.1 PHP	9
3.1.2 HTML	10
3.1.3 CSS	11
3.1.4 Javascript	11
3.2 Relaatiotietokannat.....	11
3.3 Arkkitehtuurimallit.....	12
3.3.1 MVC -arkkitehtuuri	12
3.3.2 Asiakas-palvelin -arkkitehtuuri.....	12
3.4 Tietoliikennetekniikat	13
3.4.1 SSL, HTTP ja HTTPS	13
4 Tietokannan toteutus	14
4.1 Tietokannan hallintajärjestelmä vaihtoehdot	14
4.1.1 MySQL	14
4.1.2 MariaDB	15
4.1.3 PostgreSQL.....	16
4.1.4 NoSQL-tietokannan hallintajärjestelmät	16
4.2 Tietokannan hallintajärjestelmien vertailu	18
4.2.1 Relaatiotietokannat vs NoSQL-tietokannat	18
4.2.2 MySQL vs MariaDB.....	19
4.2.3 PostgreSQL vs MariaDB	20
4.2.4 Tietokannan hallintajärjestelmän valinta	21
4.3 Sovelluksen tietomalli.....	22
4.3.1 Asiakas.....	23
4.3.2 Istunto ja yleinen tapahtuma	24
4.3.3 Käyttöjärjestelmän tiedot.....	24
4.3.4 Laitetiedot	24

4.3.5	Tapahtuma	24
4.3.6	Ominaisuus	25
4.3.7	Metatieto	25
4.4	Tietokannan fyysinen suunnittelu	25
4.4.1	Tietoturvallisuus	26
5	PHP-sovellusten toteutus	28
5.1	Toteutusvaihtoehdot	28
5.2	XML-tiedon vastaanottaminen	29
5.3	XML-tiedostojen jäsentäminen ja tallennus	30
5.4	WWW-sivuston toteutus	31
5.4.1	Arkkitehtuuri	31
5.4.2	Käyttöliittymä	32
5.4.3	Sovelluslogiikka	34
5.4.4	Tietoturvallisuus	41
6	Arviointi	42
6.1	Tietokannan arviointi	42
6.1.1	Yleiskäyttöisyys	42
6.1.2	Suorituskyky	43
6.1.3	Skaalautuvuus	43
6.2	PHP-sovellusten arviointi	44
6.2.1	XML-tiedon vastaanottaja	45
6.2.2	XML-tiedostojen jäsentäjä	45
6.2.3	WWW-sivustot	46
7	Yhteenveto	47
	Lähteet	48

TERMIT JA NIIDEN MÄÄRITELMÄT

BSD	BSD eli Berkeley Software Distribution on yksi vanhimmista ja käytetyimmistä avoimen lähdekoodin lisensseistä.
CSS	Cascading Style Sheets on rakenteellisten dokumenttien, kuten HTML:n esitystavan määrittelevä tyylikieli. Se mahdollistaa sisällön erottamisen esitystavasta.
GPL	GPL eli General Public License on avoimen lähdekoodin ohjelmistojen julkaisemiseen tarkoitettu lisenssi.
GUID	GUID eli Globally Unique Identifier on yksilöllinen 128-bittinen numerosarja, jota käytetään ohjelmistoissa tunnistena.
HTML	Hypertext Markup Language on hypertekstidokumenttien kuvauskieli.
HTTP	HyperText Transfer Protocol on hypertekstin siirtoprotokolla, jota WWW-selaimet ja WWW-palvelimet käyttävät viestinnässä keskenään.
HTTPS	Hypertext Transfer Protocol Secure on HTTP-protokollan ja SSL/TLS-protokollan yhdistelmä. Sitä käytetään suojatussa tiedon siirrossa WWW:ssä.
NPS	NPS eli Net Promoter Score arvioi asiakkaan todennäköisyyttä ostaa yrityksestä uudelleen tai suositella sitä ystävälle.
NoSQL	NoSQL on ei-relaatiotietokannan hallintajärjestelmä, joka on suunniteltu isojen tietomäärien käsittelyyn.
PHP	Hypertext Preprocessor on WWW-ohjelmointikieli ja sitä voidaan kutsua myös HTML upotetuksi skriptikieleksi.
PIWIK	PIWIK on ohjelmisto, joka on suunniteltu WWW-sivustojen käyttäjämäärien seurantaan.
PostgreSQL	PostgreSQL on yksi suosituimmista avoimen lähdekoodin olio-relaatiotietokannan hallintajärjestelmistä.
MariaDB	MariaDB on MySQL -tietokannan hallintajärjestelmän kehityshaarauma, joka on suunniteltu korjaamaan MySQL:n puutteita ja lisäämään myös uutta toiminnallisuutta.

MySQL	MySQL on suosituin ja käytetyin avoimen lähdekoodin relaatiotietokannan hallintajärjestelmä.
SQL	Structured Query Language on rakenteinen kyselykieli, jonka avulla voidaan käsitellä tietokantaa.
SSL	Secure Socket Layer on turvaprotokolla, jolla salataan tietoliikennettä.
UML	Unified Modeling Language on graafinen ohjelmistojen mallinnuskieli.
URI	Uniform Resource Identifier on merkkijono, abstraktin tai fyysisen resurssin indentifioimiseksi.
URL	Uniform Resource Locator on URI, joka sisältää tavan hakea resurssi sekä tiedon, mistä resurssi haetaan. Sitä käytetään osoittamaan WWW-sivuja.
WWW	World Wide Web on Internetissä toimiva hypertekstijärjestelmä, jonka hypertekstiä luetaan WWW-selaimella.
XML	XML (eXtensible Markup Language) on tiedostoformaatti ja World Wide Web konsortio (W3C) määrittelemä standardi rakenteellisten dokumenttien kuvaamiseen.

1 JOHDANTO

Piceasoft on syksyllä 2012 perustettu suomalainen yritys, joka on erikoistunut sovelluksiin, jossa tietoa siirretään mobiililaitteen ja tietokoneen välillä. PiceaHub puolestaan on Piceasoftin kehittämä sovellus, joka on toteutettu yksinkertaistamaan mobiililaitteen ja tietokoneen välillä tapahtuvaa sisällön luontia ja hallintaa. PiceaHub -sovelluksen kehitys jatkuu edelleen ja sen kehittäjien kannalta on hyödyllistä kerätä loppukäyttäjiltä käyttötietoa.

Käyttötiedon keräämiseen on olemassa useita syitä ja ne vaihtelet tapauskohtaisesti. Eräässä nettipalaveripalvelun käyttötiedon keräämiseen liittyvässä tutkimuksessa käyttötietoa kerättiin palvelun käyttäjiltä, koska sen kehittäjät halusivat tietää, miten palvelua käytetään [1]. Tutkimuksen tarkoitus oli kerätä todellista tietoa palvelun käytettävyydestä ja käyttötapauksista. Varsinainen käyttötiedon kerääminen toteutettiin tutkimuksessa palvelun ominaisuuksien tasolla, jossa käyttäjien tieto kerättiin apuohjelmien ja lokitiedostojen avulla [1]. Tutkimuksessa palvelun ominaisuuksia olivat esimerkiksi tehtävälista tai sähköpostiviestin lähetys. Käyttötiedon esittämiseen analysoidussa muodossa käytettiin kuvaajia, jossa nähtiin muun muassa, kuinka paljon palvelun ominaisuuksia on käytetty tietyinä ajanhetkenä [1].

Eräässä toisessa käyttötietoon liittyvässä tutkimuksessa tutkittiin keinoja käyttötiedon keräämiseen keinoja käyttötiedon keräämiseen [2]. Tutkimuksessa todettiin käyttötietoa kerääville järjestelmille olevan yhteistä ja tärkeää, että käyttötieto on laadultaan merkityksellistä, yksinkertaista ja selkeästi selitettyä [2]. Tutkimuksessa korostettiin myös, että tiedon kerääminen sovelluksesta ei saa vaikuttaa sen käyttäjien käyttökokemukseen [2].

Näihin periaatteisiin liittyen tämän diplomityön aiheena on järjestelmän toteuttaminen, joka mahdollistaa PiceaHub-sovelluksen loppukäyttäjiltä saatavan käyttötiedon keräämisen, tallentamisen ja analysoinnin. Käyttötiedon tarkoitus on auttaa sovelluskehittäjiä parantamaan sovellusta, korjaamaan ongelmia, priorisoimaan testausta ja suuntamaan sekä tuotekehitys että markkinointi oikeille alueille. Työssä toteutettavaan järjestelmään liittyvät sopivan tietokannan hallintajärjestelmän valinta ja analyysi, johon suunnitellaan ja toteutetaan käyttötiedon tallentamista varten tietokanta. Tietokannan toteutuksen lisäksi työssä toteutetaan sovellukset, jotka vastaanottavat PiceaHub-sovelluksesta lähetettävää käyttötietoa ja jäsentävät sen tietokantaa. Jotta

tietokantaan tallennettu käyttötieto saadaan esitettyä käyttäjille, työssä toteutetaan myös WWW-sivustot, jossa käyttötieto esitetään analysoidussa ja hyödyllisessä muodossa.

Diplomityössä ensimmäiseksi käydään läpi työn toteutukseen liittyviä taustoja ja tarpeita sekä kuvataan toteutettua järjestelmää yleisellä tasolla. Luvussa 3 kerrotaan järjestelmän toteutuksessa käytettäviä tekniikoita, jonka jälkeen luvussa 4 käsitellään tietokannan suunnittelua ja toteutusta. Luvussa 5 puolestaan käsitellään käyttötietoa vastaanottavien ja tietokantaan jäsentävien sovellusten sekä WWW-sivustojen toteutusta. Luvussa 6 arvioidaan toteutettua tietokantaa, sovelluksia ja WWW-sivustoja, jonka jälkeen työssä pohditaan yleisellä tasolla toteutettua järjestelmää.

2 TAUSTA JA TARPEET

Tässä luvussa käsitellään diplomityön toteutukseen liittyviä taustoja ja tarpeita. Luvussa kuvataan ongelmia, tavoitteita ja vaatimuksia, jotka liittyvät työssä toteutettavan järjestelmän suunnitteluun ja toteuttamiseen.

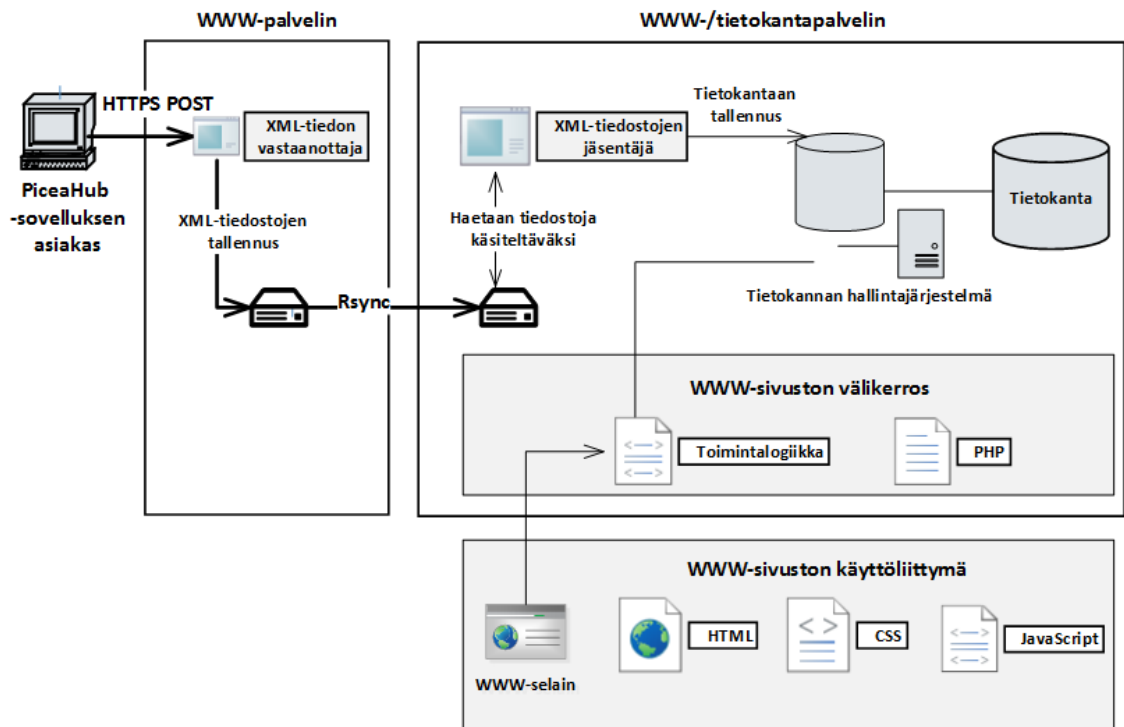
2.1 Yleiskuvaus

Tutkimukset ovat viime aikoina osoittaneet, että sovellukset sisältävät paljon ominaisuuksia ja toiminnallisuutta, joita loppukäyttäjät eivät käytä ja ovat siten arvottomia [3]. Turhat ominaisuudet vievät myös turhaan ohjelman suorituskykyä ja hidastavat sen toimintaa [2]. Osittain tämä johtuu ohjelmistokehityksen trendeistä, joissa tuotteet pyritään saamaan nopeasti markkinoille ja sovelluksen ominaisuuksien arvoja ei tyypillisesti mitata [3]. Asiaan vaikuttaa myös se, että sovellukset ovat moniulotteisia, joka puolestaan hankaloittaa sovellusten arvon mittaamista ja arviointia. Tähän ongelmaan liittyen yritykset ovat viime vuosina pyrkineet panostamaan arvoihin perustuvaan ohjelmistokehitykseen, jossa pyritään tunnistamaan sovellukselle tärkeitä ominaisuuksia ja poistamaan turhia ominaisuuksia [3]. Jotta sovellusten tärkeät ja turhat ominaisuudet saadaan tunnistettua, tarvitaan usein järjestelmä, joka mahdollistaa sovelluksen loppukäyttäjiltä saatavan käyttötiedon keräämisen, tallentamisen ja analysoinnin.

Tässä diplomityössä käsitellään PiceaHub-sovelluksen käyttötiedon keräämiseen liittyvää järjestelmää ja sen toteutusta. Käyttötiedon tarkoitus on auttaa sovelluksen kehittäjiä tunnistamaan arvokkaat ominaisuudet ja karsimaan sekä turhat että arvottomat ominaisuudet pois. Arvokkaiden ominaisuuksien tunnistamisen lisäksi käyttötieto mahdollistaa tuotekehityksen ja ominaisuuksien suuntaamisen oikeille alueille. Lisäksi käyttötieto helpottaa sovelluksen virheiden ja ongelmakohtien paikantamista.

Järjestelmän toteutukseen liittyy oleellisesti sopivan tietokannan hallintajärjestelmän valinta, joka mahdollistaa käyttötiedon tallentamisen. Mahdollisia tietokannan hallintajärjestelmiä on useita, joten tässä työssä pyritään analysoimaan ja valitsemaan toteutuksen kannalta sopivin vaihtoehto. Valittavaan tietokannan hallintajärjestelmään työssä suunnitellaan ja toteutetaan tietokanta, johon varsinainen käyttötieto tallennetaan. Tietokannan lisäksi työssä toteutetaan kaksi sovellusta, jotka vastaanottavat PiceaHub-sovelluksesta WWW-palvelimelle lähetettävän käyttötiedon ja tallentavat sen tietokantaan. Näiden kahden sovelluksen lisäksi työssä toteutetaan WWW-sivut, jotka

mahdollistavat tietokannassa olevan käyttötiedon esittämisen PiceaHub-sovelluksen kehittäjille hyödyllisessä muodossa. Kuvassa 2.1 on kuvattu toteutettava järjestelmä.



Kuva 2.1. Yleiskuva järjestelmästä

Yllä esitetty järjestelmän yleiskuva kuvaa, miten PiceaHub-sovelluksesta lähetettävä käyttötieto kulkee toteutettavassa järjestelmässä ja mitä eri vaiheita tiedon kulkuun liittyy ennen kuin tieto varsinaisesti tallennetaan tietokantaan. Lisäksi kuvassa on havainnollistettu WWW-sivustojen toteutusta yleisellä tasolla.

Jotta käyttötieto saadaan sovelluksesta kerättyä, tarvitaan menetelmä sovelluksen tapahtumien tunnistamiseen ja lähettämiseen. Sovelluksen tapahtumien tunnistamiseen on tyypillisesti kaksi tunnettua menetelmää. Toisessa menetelmästä olemassa olevaa ja tarkasteltavaa sovellusta laajennetaan, jotta sen tapahtumia voidaan tarkastella ja lähettää [4]. Toisessa menetelmästä puolestaan tarvitaan erillinen ohjelma, joka tarkkailee olemassa olevaa sovellusta ja sieppaa sovelluksen tapahtumat lähetyksistä varten [4]. Näistä menetelmistä jälkimmäinen on vaikeampi toteuttaa ja sitä käytetään tyypillisesti WWW-sivustoissa, sillä niissä tapahtumia, kuten hiiren painalluksia on helpompi siepata. Olemassa olevaa ohjelmaa ei muokata, vaan sen toiminta perustuu sen kykyyn tunnistaa, mitä sovelluksia on ajossa käyttöjärjestelmässä tai selaimessa. Esimerkiksi Google Analytics mahdollistaa WWW-sivujen käytön seuraamisen asettamalla seurattavalle sivulle pätkän Javascript-koodia [4]. Tässä toteutuksessa PiceaHub-sovellusta laajennettiin tapahtumien tunnistamista ja lähettämistä varten, sillä se oli toteutusvaihtoehdoista helpompi ratkaisu.

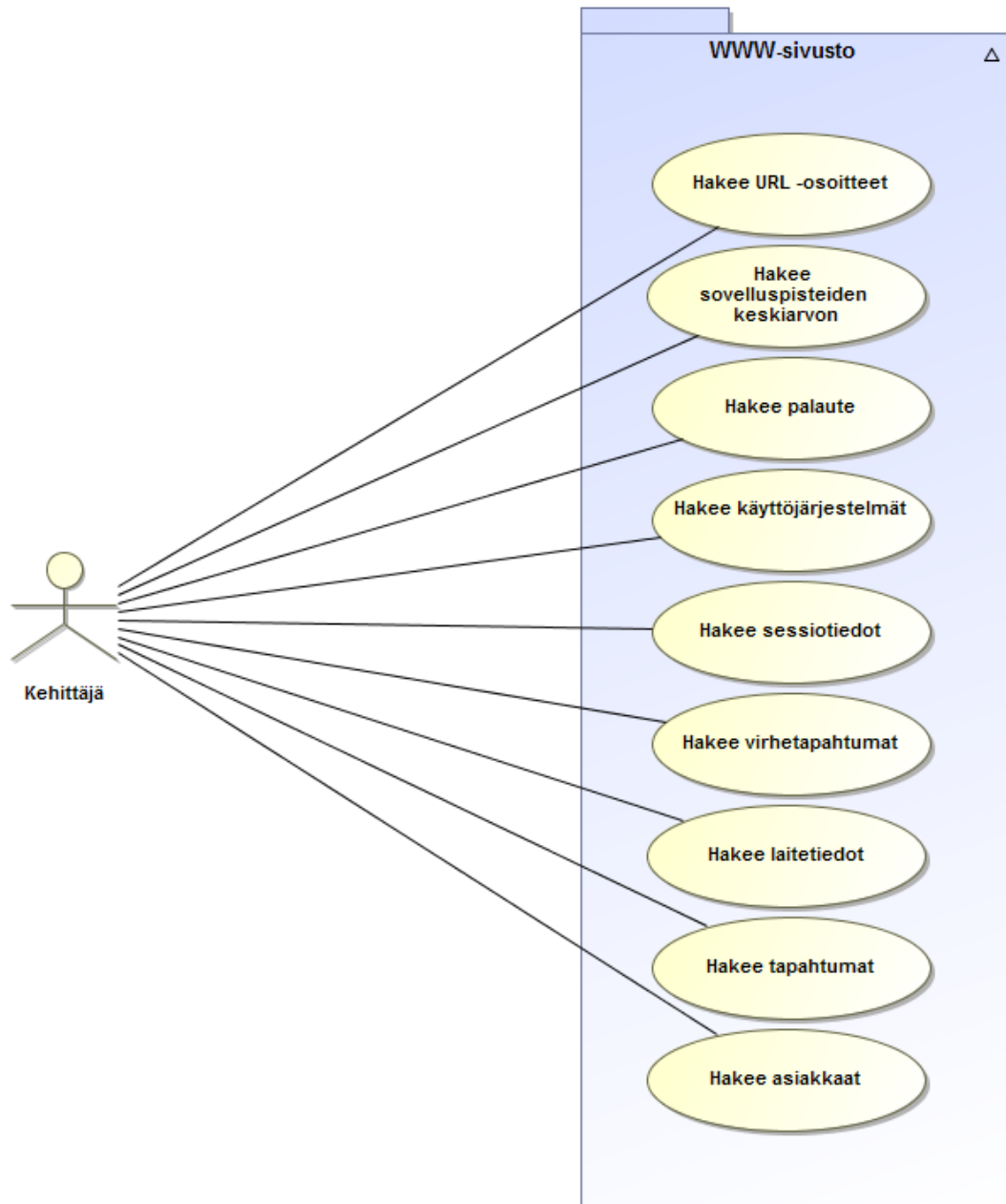
Käyttötiedon lähettäminen PiceaHub-sovelluksesta WWW-palvelimelle toteutettiin HTTPS POST-pyyntöön sisällytetyn XML-tiedon mukana. Pyyntöjen vastaanottamista ja tiedon jäsentämistä varten palvelimelle toteutettiin kaksi sovellusta. Toinen toteutettavista sovelluksesta on XML-tiedon vastaanottaja ja toinen XML-tiedostojen jäsentäjä. Näiden kahden sovelluksen lisäksi käyttötiedon esittämistä varten tarvittiin tietokantapalvelimelle WWW-sovellus, jonka avulla tietokantaan tallennettava käyttötieto voitiin analysoida ja näyttää sovelluskehittäjien kannalta hyödyllisessä muodossa.

XML-tiedon vastaanottaja, XML-tiedostojen jäsentäjän ja WWW-sivustojen toimintalogiikka toteutettiin PHP-ohjelmointikieltä käyttämällä. Lisäksi WWW-sivustojen käyttöliittymän toteutuksessa käytettiin HTML-kuvailukieltä, CSS-tyylikieltä ja JavaScriptiä. Näitä tekniikoita on kuvattu tarkemmin luvussa 3.

2.2 Toiminnalliset vaatimukset

Projektin alussa oli selvää, että toteutettavaan tietokantaan tultaisiin tallentamaan vähintään kolmenlaista tietoa: asiakkaita, istuntoja ja tapahtumia. Asiakkailta tallennetaan esimerkiksi asiakkaan yksilöivä tunnus, luontiaika ja maatieto. Istuntotiedot sisältävät yleistä tietoa sovelluksen käyttöympäristöstä, laitteista ja käyttäjästä. Tapahtumat puolestaan sisältävät yksityiskohtaisempaa sovelluksen käyttötietoa, esimerkiksi mitä ominaisuuksia sovelluksesta käytetään.

WWW-sivustoille asetetut toiminnalliset vaatimukset on esitetty UML-käyttötapauskaaviossa kuvassa 2.2. Sivustolla näkyvän käyttötiedon avulla saadaan selville miten, milloin ja mistä PiceaHub-sovellusta käytetään. Lisäksi sivujen avulla haluttiin saada selville sovelluksen käytössä tapahtuvat poikkeavuudet ja virhetilanteet. Käyttäjiltä saadun palautteen esittäminen todettiin myös tärkeäksi.



Kuva 2.2. Käyttötapauskaavio

2.3 Ei-toiminnalliset vaatimukset

Tietokannan hallintajärjestelmälle ja tietokannalle asetettiin ei-toiminnallisia vaatimuksia, kuten järjestelmän ylläpidettävyys, kohtuulliset kustannukset, skaalautuvuus ja suorituskyky. Nämä vaatimukset toimivat pohjana sopivan tietokannan hallintajärjestelmän valinnalle, jota käsitellään tarkemmin luvussa 4.

2.3.1 Ylläpidettävyys

Tietokannan hallintajärjestelmän eräänä ei-toiminnallisena vaatimuksena oli ylläpidettävyys. Ylläpidettävyyden kannalta on tärkeää, että tietokantaa voidaan hallita

komentorivirajapinnan lisäksi myös graafisen käyttöliittymän avulla. Tämä mahdollistaa sen, että tietokantaa voivat käyttää myös henkilöt, joilla ei ole tietämystä SQL-kielen käytöstä. Graafisen käyttöliittymän avulla voidaan siis ottaa yhteys tietokantapalvelimeen ja hallita tietokantoja, tauluja, kenttiä, hakemistoja, käyttäjiä ja käyttöoikeuksia.

2.3.2 Kustannukset

Kustannukset ovat merkittävä tekijä tietokannan hallintajärjestelmää valittaessa. Kustannukset koostuvat muun muassa hankittavan ohjelmiston hinnasta ja järjestelmän ylläpitoon liittyvistä kustannuksista. Sopivaa järjestelmää valittaessa tulee myös ottaa huomioon tietokannan hallintajärjestelmän asettamat rajoitukset laitteistoille, jotka saattavat aiheuttaa yritykselle lisäkustannuksia. Muita kustannuksia voi tulla henkilöstön kouluttamisesta, jotta järjestelmää voidaan käyttää toivotulla tavalla. Kustannusten minimoimiseksi on tärkeää, että tietokannan hallintajärjestelmä on myös hyvin dokumentoitu.

Monet suosituimmista tietokannan hallintajärjestelmistä ovat ilmaisia ja avointa lähdekoodia, joten avoin lähdekoodi asetettiin myös valittavan tietokannan hallintajärjestelmän vaatimukseksi. Avoimen lähdekoodin lisäksi tietokannan hallintajärjestelmän käyttö tulee olla täysin ilmaista, jolloin niiden tulee olla saatavissa joko GPL- tai BSD-lisenssillä. GPL-lisenssissä ohjelmaa tai sen lähdekoodia saa käyttää, miten haluaa, mutta sitä muokattaessa täytyy lähdekoodi julkaista. BSD-lisenssi eroaa GPL-lisenssistä siinä, että se ei vaadi lähdekoodin julkaisemista, jos muokattua ja lisensoitua ohjelmaa levitetään eteenpäin.

Avoimen lähdekoodin ehtona on kuitenkin, että valittavan tietokannan hallintajärjestelmän kehitysyhteisön tulee olla aktiivinen, jotta voidaan olla varmoja jatkuvasta kehityksestä ja siitä, että tietokannan hallintajärjestelmän mahdolliset ongelmat korjataan nopeasti. Tietokannan hallintajärjestelmän tietyt osa-alueet saattavat ajan myötä vanheta, joten aktiivinen kehitysyhteisö mahdollistaa ajan tasalla pysymisen myös tulevaisuudessa ja teknologian kehittyessä.

2.3.3 Skaalautuvuus

Valittavan tietokannan hallintajärjestelmän tulee olla skaalautuva. Skaalautuvuudella tarkoitetaan tietokannan kykyä suoriutua sen koon ja käyttäjämäärien kasvaessa. Jotta valittava tietokannan hallintajärjestelmä olisi mahdollisimman skaalautuva, tulee sen osata suorittaa replikointia ja klusterointia. Replikointi on tyypillinen skaalautuvuustekniikka, jossa sama tieto on useassa paikassa. Klusteroinnilla puolestaan tarkoitetaan joukkoa kytkettyjä tietokoneita, jotka työskentelevät yhdessä yhtenä laskentaresurssina ja luovat kuvitelman, että ne ovat yksi tietokone. Klusteroinnin avulla

tietokanta voidaan siis hajauttaa useaan tietokantaan, jotka näkyvät ulospäin yhtenä tietokantana.

2.3.4 Suorituskyky

Valittavan tietokannan hallintajärjestelmän tulee olla myös suorituskykyinen, jotta se pystyy suoriutumaan tehokkaasti suurista tietomääristä ja tietokantaoperaatioiden vasteajat olisivat mahdollisimman alhaisia. Suorituskyky vaikuttaa myös oleellisesti toteutettavan järjestelmän ja erityisesti WWW-sivujen käytettävyyteen. Järjestelmän suorituskyvyn merkitys kasvaa tiedon määrän kasvaessa ja tulevaisuudessa tietoa tulee odotetusti olemaan paljon.

3 KÄYTETYT TEKNIIKAT

Tässä luvussa käsitellään diplomityön toteutuksessa käytettyjä tekniikoita. Luvussa kerrotaan sovellusten toteutuksessa käytetystä ohjelmointikielistä, tietokantatekniikoista sekä arkkitehtuureista. Viimeiseksi luvussa kerrotaan lyhyesti tietoliikennetekniikoista, joita käytetään kerätyn käyttötiedon luottamuksellisuuden ja eheyden turvaamisessa.

3.1 WWW-tekniikat

3.1.1 PHP

Yleisesti sanottuna PHP on WWW-ohjelmointikieli, mutta sillä on myös komentorivirajapinta ja sitä voidaan käyttää sulautettuihin sovelluksiin. PHP:ta voidaan kuvailla myös yleiskäyttöiseksi ja tulkattavaksi ohjelmointikieleksi. Kielen perusideana on mahdollistaa dynaamisten WWW -sivujen luominen nopeasti. [5]

PHP:ta käytettäessä on myös huomioitavaa, että se on palvelimen puolella käytettävä teknologia. PHP:ta ei voi siis käyttää asiakassovelluksiin (WWW-selaimiin) kohdistuvien ominaisuuksien toteuttamisessa. Asiakassovelluksiin liittyviä ominaisuuksia ovat esimerkiksi uuden ikkunan luominen selaimen ja sen koon muuttaminen. Toisaalta PHP:n avulla voidaan generoida yleiskäyttöistä JavaScriptia, jota käytetään asiakas-sovelluksissa dynaamisen sisällön luomiseen WWW-sivuille. [5]

PHP:ta voidaan käyttää sekä proseduraaliseen että olioperustaiseen ohjelmointiin ja lisäksi näitä kahta ohjelmointiparadigmaa voidaan yhdistellä. Ohjelmointiparadigmalla tarkoitetaan ohjelmien keskeisten toteutusperiaatteiden kuvaamista. Proseduraalisella ohjelmoinnilla tarkoitetaan esimerkiksi irrallisista funktioista, muuttujista ja tietorakenteista koostuvaa joukkoa. Olio-ohjelmoinnissa irralliset osat ovat yhdistettynä yhteen olioon, jota käsitellään yhtenä kokonaisuutena. Toisin sanoen proseduraalisessa ohjelmoinnissa ohjelman rakenne jäsennetään toiminnasta käsin, kun taas olio-ohjelmoinnissa se jäsennetään datasta käsin. Ohjelmointikielenä PHP on hyvin monipuolinen ja helposti opittava, mistä johtuen se sopii sekä aloittelevalle että kokeneemmalle ohjelmoijalle. [5]

PHP on käytettävissä usealla eri käyttöjärjestelmällä, kuten Linux, Unix ja Windows. Lisäksi PHP:ssa on myös tuki useimmille WWW-palvelimille, joka tekee kielestä hyvin monipuolisen. Dynaamisten palveluiden luomisessa tietokantojen hyödyntäminen on

tärkeää ja PHP tarjoaa suoraan tukea relaatiotietokannoille, kuten PostgreSQL, MySQL ja Mariadb. [5]

PHP:n syntaksista löytyy paljon samoja piirteitä, kuten C-, Java- ja Perl-ohjelmointikielistä. Näille kielille on yhteistä muun muassa koodissa olevien lauseiden erottaminen puolipisteellä. PHP:n syntaksille on myös ominaista muuttujien heikko tyyppitys, jolloin ne voivat saada minkä tyyppisen arvon tahansa ja tietotyyppiä voi vaihtua ohjelman suorituksen aikana. Heikosti tyyplitettyjen muuttujien lisäksi eräs mielenkiintoinen ominaisuus PHP:ssa on heterogeeninen taulukko, jonka alkiot voivat olla eri tietotyyppiä. [5]

Tämän projektin toteutuksessa olisi voitu hyödyntää myös muita palvelimen puolella suoritettavia ohjelmointikieliä, kuten Javaa, Perliä tai ColdFusionia. Näitä ohjelmointikieliä käytetään tyypillisesti WWW-sivustoissa, joissa tietoliikennettä on paljon. Tässä toteutuksessa WWW-sivustojen tietoliikenne todettiin olevan myös tulevaisuudessa pientä, johon PHP-ohjelmointikieli erityisesti sopii. PHP-ohjelmointikielillä voitiin myös toteuttaa kaikki järjestelmän toteutukseen liittyvät sovellukset, joka oli merkittävä syy sen valintaan toteutukseen. Lisäksi muihin ohjelmointikieliin verrattuna se on helppo asentaa ja ylläpitää, koska palvelimissa on tyypillisesti valmiina Apachen HTTP-palvelinohjelma, johon PHP:n saa asennettuna laajennoksena vaivattomasti, ellei se löydy jo valmiina.

Toinen PHP:n valintaan vaikuttanut syy oli se, että PHP:ta pidetään luotettavampana ja suorituskykyisempänä ohjelmointikielenä kuin ColdFusionia. ColdFusionin vahvuus on sen hyvässä tiedonhakumoottorissa, mutta tässä työssä sitä ominaisuutta ei koettu merkittäväksi. Perl puolestaan on PHP:n verrattuna monimutkaisempi ohjelmointikieli, joka johtuu siitä, että PHP suunniteltiin käytettäväksi WWW:ssä, kun taas Perl suunniteltiin myös muihin käyttötarkoituksiin. PHP:n vahvuus Perliin verrattuna on se, että sitä on helpompi upottaa HTML-kieleen, joka oli toteutuksen kannalta merkittävä ominaisuus. Java olisi ollut hyvä vaihtoehto toteutukseen, mutta PHP:n todettiin olevan kuitenkin ketterämpi vaihtoehto ja sen käytöstä oli enemmän käytännön kokemusta, tämän vuoksi se myös valittiin toteutukseen.

3.1.2 HTML

HTML (Hypertext Markup Language) on World Wide Webissä käytetty kuvauskieli. Sen avulla voidaan kertoa WWW-dokumenttien eli WWW-sivujen tekstisisältö ja tekstinosien eli elementtien ohjeet. Ohjeiden avulla määritellään elementtien looginen rakenne ja ulkoasu. Dokumentit voivat sisältää tekstin lisäksi esimerkiksi kuvia, äänitiedostoja ja grafiikkaa. [5]

HTML on tarkoitettu ainoastaan dokumentin rakenteen kuvaamiseen. Varsinaisen dokumentin ulkoasun määrittelyyn Word Wide Web Consortium (W3C) suosittelee CSS-tyylikieltä. [5]

3.1.3 CSS

Cascading Style Sheets (CSS) on tyylikieli, joka suunniteltu HTML-dokumenttien ulkoasun kuvaamiseen. Sen avulla voidaan siis erottaa HTML-dokumentin sisältö ja ulkoasu toisistaan. WWW-dokumentit voidaan esittää usealla eri tavalla ja tyylikieli kuvaa, miten HTML-elementit tulisi esittää. [5]

Suosittelua tapa käyttää CSS-tyylikieltä on ulkoisen tyylitiedoston avulla, jolloin HTML-dokumentteja on tehokkaampaa ja selkeämpää ylläpitää. Muita CSS-tyylikielen käytön hyötyjä on WWW-sivujen saavutettavuuden paraneminen. CSS-tyylikielen avulla HTML-dokumenttien sisältö voidaan esittää useille erille kohderyhmille, kuten näkövammaisille tai mobiililaitteiden käyttäjille. [5]

3.1.4 Javascript

JavaScript on WWW-ohjelmoinnissa yleisesti käytettävä skriptikieli, jota voidaan upottaa HTML-sivuihin. Sitä voidaan suorittaa kaikilla tunnetuimmilla selaimilla ja se on asiakkaan puolella eli WWW-selaimessa suoritettava kieli, jonka syntaksi perustuu C-ohjelmointikielen [6]. JavaScriptia voidaan kuvailla dynaamisesti tyyppitetyksi, tulkittavaksi ja oliopohjaiseksi kieleksi [6].

JavaScriptia voidaan käyttää HTML-dokumentissa usealla tavalla, kuten tapahtumamäärittelyillä, erillisillä tiedostoilla tai suoraan upottamalla HTML-dokumenttiin [7]. Tapahtumamäärittelyllä tarkoitetaan attribuutteja, jotka tarkentavat elementin merkitystä. HTML-kielen elementeillä on useasti joukko mahdollisia määrittelyitä, joita siihen voidaan liittää. JavaScriptia voidaan siis upottaa esimerkiksi yksittäiseen elementteihin liittyviin tapahtumiin [7].

3.2 Relaatiotietokannat

Relaatiotietokannat perustuvat relaatiomalliin, joka on looginen tietomalli, jolla voidaan kuvata täsmällisesti tietojen organisointi sekä tietoaisteihin tehtävät kyselyt, poistot ja päivitykset [8]. Tietomallilla puolestaan tarkoitetaan abstraktia mallia siitä, kuinka tieto esitetään ja kuinka sitä käsitellään [9]. Relaatiotietokannoissa tietojen varsinaisen tallennusrakenne näkyy rakenteen puolesta käyttäjälle samalla tavalla eli relaatioina, joista tietokanta koostuu.

Relaatiot ovat yksilöllisesti nimettyjä ja rakenteeltaan samanlaisia monikkoja. Relaation rakenne määritellään joukolla ominaisuuksia, joilla on yksilöllinen nimi ja arvoalue eli

tietotyyppi. Relaation monikot ovat jokainen erillisiä ja jokaisen relaation solu sisältää tasan yhden atomisen arvon [9]. Yleinen tapa relaatioiden ilmentymien esittämiseen on esittää ne tauluina [9]. Tauluissa ominaisuutta vastaa sarake ja jokaista monikkoa rivi.

Relaatiotietokantojen hyvänä puolena on, että relaatioista voidaan koostaa uusia erilaisia joukkoa niitä yhdistelemällä. Relaation monikoille voidaan tehdä myös joukko-operaatioita, jolloin esimerkiksi erilaisten koosteiden tekeminen on helppoa ja tehokasta.

Relaatiotietokannan hallintajärjestelmiä voidaan käsitellä rakenteisen SQL-kielen avulla. SQL on rakenteinen tietokannan kyselykieli, joka mahdollistaa tiedon varastoimisen ja hakemisen tietokannoista. Kielelle on olemassa ANSI-standardi, jota monet tietokannan hallintajärjestelmät, kuten PostgreSQL ja MySQL tukevat [10]. Standardin ja tietokannan hallintajärjestelmien välisten SQL-kielten versioiden välillä voi kuitenkin olla pieniä eroja.

3.3 Arkkitehtuurimallit

3.3.1 MVC -arkkitehtuuri

MVC (Model-View-Controller) on suunnittelumalli, jolla sovelluslogiikka saadaan erotettuna käyttöliittymästä. MVC-arkkitehtuuria tukevissa ohjelmissa toiminta on jaettu kolmeen loogiseen kokonaisuuteen: malliin, näkymään ja ohjaimeen. Sitä käytetään tyypillisesti sovelluksessa, jossa halutaan tehdä useita erilaisia näkymiä ja helpottaa käyttöliittymän muuttamista.

Tyypillisissä MVC -ratkaisuissa malli huolehtii tiedon varastoinnista ja käsittelystä. Näkymä määrittää käyttöliittymän ulkoasun ja huolehtii tietojen esittämisestä käyttäjälle. Ohjain puolestaan käsittelee mallia ja ottaa vastaan näkymän vaatimat toiminnot. [12]

Tässä työssä MVC-arkkitehtuuri mahdollistaa WWW-sivujen helpon ylläpidettävyyden, sillä käyttöliittymää on helppo muuttaa ja sama tieto voidaan esittää usealla tavalla käyttämällä eri näkymiä. Lisäksi MVC mahdollistaa WWW-sivujen yleiskäyttöisyyden, esimerkiksi ohjaimet voivat käyttää samaa mallia tietokantaan kohdistuvien operaatioiden käsittelyyn.

3.3.2 Asiakas-palvelin -arkkitehtuuri

Asiakas-palvelin -arkkitehtuurissa järjestelmä koostuu resurssia kontrolloivista ja siihen liittyviä palveluja tarjoavista palvelimista ja palveluja tarvitsevista asiakkaista [13].

Tyypillisessä asiakas-palvelin arkkitehtuurissa asiakkaat ovat sovelluksia, jotka eivät tunne toisiaan. Myöskään palvelimet eivät tunne asiakkaita ja ideaalitapauksessa asiakkaiden ei tarvitse tietää yksikohtaisia tietoja palvelun tarjoamasta palvelimesta.

Palvelimet tarjoavat rajapinnan, jonka avulla asiakas voi pyytää palvelimen tarjoamia palveluita ja näyttää palvelimen palauttamien tulokset [13]. Palvelimet odottavat asiakkailta tulevia pyyntöjä ja vastaavat niihin. Asiakas-palvelin -arkkitehtuuria käytetään tyypillisesti tietovarastopalvelimissa, jossa hallitaan yhteisiä resursseja. Asiakas-palvelin arkkitehtuuri helpottaa myös tietovarastojen ylläpitämistä ja muunneltavuutta [13].

3.4 Tietoliikennetekniikat

3.4.1 SSL, HTTP ja HTTPS

SSL eli Secure Socket Layer on yksi Internetin tunnetuimmista ja käytetyimmistä salausprotokollista. Sen avulla voidaan suojata tietoliikenne IP-verkkojen yli, jotta ulkopuolinen ei pääse käsiksi luottamuksellisiin tietoihin [14]. Luottamuksellisuuden lisäksi SSL:ssä varmistetaan tiivistefunktiolla siirrettävän tiedon eheys ja varmenteiden eli sertifikaattien avulla voidaan varmistua siitä, että ollaan käyttämässä oikeata palvelua ja toimijaa. SSL:ää käytetään tyypillisesti WWW-sovelluksien autentikoinneissa, joissa selaimesta siirretään palvelimelle salasana turvallisesti.

SSL toimii TCP-protokollan päällä ja muiden sovellustason protokollien, kuten FTP:n tai HTTP:n alla [14]. SSL:n ansiosta tietoliikenne voidaan suojata URL-osoitetta ja HTTP-protokollan otsikoita myöden [14].

HTTP (HyperText Transfer Protocol) tarkoittaa hypertekstin siirtoprotokollaa, jota WWW-selaimet ja WWW-palvelimet käyttävät viestinnässä keskenään. Se on yleinen protokolla ja sillä voidaan siirtää kaikenlaisia tietoja. HTTP-protokolla toimii pyyntö-vastaus -periaatteella eli WWW-palvelimelle voidaan lähettää pyyntöjä, joihin se käsittelyn jälkeen vastaa. Asiakasohjelman ja WWW-palvelimen välillä oleva yhteys on voimassa vain siirtotapahtuman ajan.

HTTPS eli Hypertext Transfer Protocol Secure on puolestaan HTTP-protokollan ja SSL-protokollan yhdistelmä, jossa ensin luodaan SSL-yhteys asiakasohjelman ja palvelimen välille. Kättelyvaiheessa sovitaan yhteyskohtaisista ja kertakäyttöisistä salausavaimista, joita käytetään istunnon ajan tiedon salaamiseen ja tulkintaan [15]. Asiakasohjelman ja palvelimen välille luodun SSL-yhteyden sisällä puolestaan käytetään HTTP-protokollaa tiedon siirtämiseen.

4 TIETOKANNAN TOTEUTUS

Tässä luvussa tarkastellaan ja vertaillaan toteutukseen soveltuvia tietokannan hallintajärjestelmiä, jonka jälkeen niistä valitaan toteutukseen paras vaihtoehto. Valinnan jälkeen tässä luvussa kerrotaan varsinaisesta tietokannan toteutuksesta, johon liittyvät käsitteellinen mallintaminen ja fyysinen toteutus.

4.1 Tietokannan hallintajärjestelmä vaihtoehdot

Tietokannanhallintajärjestelmän merkittävimpiä valintakriteerejä todettiin olevan skaalautuvuus, ylläpidettävyys, vähäiset kustannukset sekä suorituskyky. Tässä luvussa kerrotaan varteenotettavimmista relaatiotietokannan hallintajärjestelmien vaihtoehdoista, jotka ovat sekä suosittuja että vastaavat luvussa 2.3 esitettyjä vaatimuksia. Relaatiotietokantojen lisäksi luvussa tarkastellaan myös yleisellä tasolla NoSQL tietokannan hallintajärjestelmiä, jotka ovat tunnettuja suurten tietomäärien käsittelystä.

4.1.1 MySQL

MySQL on yksi maailman suosituimmista relaatiotietokantajärjestelmistä, joka on tunnetusti laadukas, luotettava ja nopea [16]. Se soveltuu monelle alustalle ja on osana tunnettua LAMP (Linux, Apache, MySQL, PHP) -alustaa. MySQL soveltuu monenlaiseen tiedonhallintaan, jota käyttävät sekä pienet että suuret yritykset, kuten Wikipedia, Twitter ja Facebook.

MySQL on joustava tietokannan hallintajärjestelmä ja sitä voidaan käyttää myös vaativissa ympäristöissä, kuten WWW-sovelluksissa [17]. Se soveltuu myös esimerkiksi sulautettuihin sovelluksiin, tietovarastoihin, sisällön indeksointiin ja korkeasti redundanteihin järjestelmiin [17].

MySQL:n ehkä tärkein ominaisuus on sen tietokantamoottorien arkkitehtuuri, jossa erotetaan kyselyiden käsittely sekä muut palvelimen tehtävät tiedon varastoinnista ja hauista [17]. Arkkitehtuuri mahdollistaa sen, että voidaan valita, miten tietoa varastoidaan ja millaisia muita ominaisuuksia esimerkiksi suorituskykyyn liittyen halutaan käyttää.

MySQL:ssä on sisäänrakennettu komentorivirajapinta, jonka avulla voidaan suorittaa tietokantaan liittyviä kyselyitä, kuten luoda tauluja tai suorittaa muita tietokantaan

liittyviä perustoimenpiteitä. MySQL:ssä ei ole valmiina graafista käyttöliittymää tietokannan hallintaa varten, mutta siihen tarkoitukseen on olemassa ilmaisia ohjelmia, kuten MySQL Workbench ja phpMyAdmin.

MySQL:n heikkoja puolia on se, että se ei tue kaikkia muiden tietokannan hallintajärjestelmien toimintoja ja ominaisuuksia [16]. Esimerkiksi siinä ei voi määritellä omia tietotyyppejä, kuten PostgreSQL:ssä. Lisäksi MySQL:stä on olemassa kaksi lisenssiä, joista toinen on avoimen lähdekoodin versio ja toinen on kaupallinen versio. Kaupallisessa versiossa on ominaisuuksia, jotka tekevät siitä paremman vaihtoehdon verrattuna avoimen lähdekoodin versioon. Hyvänä esimerkkinä on ainoastaan kaupalliseen versioon saatava säieallas, jonka avulla palvelimen suorituskyky saadaan säilymään myös suurella määrällä yhteyksiä.

4.1.2 MariaDB

MariaDB on avoimen lähdekoodin relaatiotietokannan hallintajärjestelmä, joka perustuu MySQL:ään ja on sen yksi kehityshaarauma. MariaDB:ssä ei ole kaupallista lisenssiä, kuten MySQL:ssä ja se on saatavissa ainoastaan GNU GPL -lisenssillä, joka mahdollistaa sen ilmaisen käytön ja muuntamisen. Lisenssin ehtoihin kuuluu, että muunnettu lähdekoodi joudutaan julkaisemaan, jos sitä levitetään eteenpäin.

MariaDB:n eräs hyvistä puolista on sen binäärinen sopivuus MySQL:n kanssa. Tällä tarkoitetaan, että näitä kahta tietokannan hallintajärjestelmää voidaan vaihtaa vaivattomasti toiseen ilman, että tietoa häviää tai olemassa olevia tietokantoja ja niitä käyttäviä sovelluksia tarvitsee muuttaa. Binäärisellä yhteensopivuudella tarkoitetaan myös molemmissa tietokannan hallintajärjestelmissä asiakas rajapinnat, protokollat ja rakenteet ovat samanlaisia. Myös yhdistäjät ovat samanlaisia, joiden avulla tietokantaan luodaan yhteys ja niitä voidaan käsitellä eri ohjelmointikielillä, kuten PHP:ta käyttämällä.

MariaDB sisältää saman toiminnallisuuden kuin MySQL, mutta korjaa samalla MySQL:n ongelmia. MariaDB:ssä koodi on laadukkaampaa ja sitä testataan paremmin, jonka takia sitä kutsutaan MySQL:ää vakaammaksi tietokannan hallintajärjestelmäksi [18]. Lisäksi MariaDB:ssä on todettu olevan vähemmän ohjelmointivirheitä ja käänkösvaiheessa tapahtuvia varoituksia [18].

Skaalautuvuuteen liittyen MariaDB tarjoaa tärkeitä skaalautuvuustekniikoita, kuten klusteroinnin ja replikoinnin. MySQL:ään verrattuna nämä tekniikat ovat molemmissa samanlaisia ja niissä ei suuria eroja ole.

MySQL:n toiminnallisuuden lisäksi MariaDB sisältää myös täysin uutta toiminnallisuutta, jota MySQL:stä ei löydy. Näistä ominaisuuksista ehkä mielenkiintoisimpia ovat dynaamiset taulun sarakkeet, jonka avulla tietokannassa taulun

riveillä voi olla erilaisia sarakkeita [19]. Dynaamisten sarakkeiden toiminta perustuu blob-tietorakenteeseen, jonka käsittelemiseen MariaDB tarjoaa valmiita funktioita. Toinen mielenkiintoinen MariaDB:n lisäominaisuus on niiden normaalia tarkemmissa aikaleimoissa, joita voidaan käsitellä mikrosekuntien tarkkuudella.

Tietokannan hallintajärjestelmän kannalta merkittävimpiä ominaisuuksia ovat tiedontallennusmoottorit, joita MariaDB:ssä on laaja valikoima. Tiedontallennusmoottorit sopivat erilaisiin käyttötarkoituksiin ja antavat mahdollisuuden optimoida tietokantaa myös niitä hyödyntämällä.

4.1.3 PostgreSQL

PostgreSQL on yleiskäyttöinen relaatiotietokannan hallintajärjestelmä, joka tunnetaan luotettavuudesta ja ominaisuuksien suuresta määrästä. Se on avointa lähdekoodia ja soveltuu kaikille yleisimmille alustoille. Lisäksi PostgreSQL noudattaa hyvin SQL:n standardeja [20].

PostgreSQL on suunniteltu tukemaan tehokkaasti rinnakkaisuutta käyttämällä MVCC (Multi-Version Concurrency Control) -järjestelmää. MVCC-järjestelmän ansiosta transaktioissa tapahtuvat kirjoitus- ja lukuoperaatiot eivät estä toisiaan suoriutumasta. MVCC-järjestelmä vähentää poissulkeutumisen mahdollisuutta ja mahdollistaa paremman suorituskyvyn, koska lukkoja ei käytetä [20]. Lisäksi MVCC yksinkertaistaa rinnakkaisuutta vaativien sovellusten suunnittelua ja hallintaa.

PostgreSQL on hyvin skaalautuva ja suorituskykyinen asiakas-palvelin -tietokanta. Se sisältää paljon ominaisuuksia, joita voidaan konfiguroida. Eräs PostgreSQL:n tärkeimmistä ominaisuuksista on sen laajennettavuus, joka mahdollistaa esimerkiksi omien tietotyyppien, operaattoreiden, hakemistotyyppien ja funktionaalisten kielten luomisen [20].

PostgreSQL noudattaa ACID-ominaisuuksia (atomisuus, eheys, eristyneisyys, pysyvyys), joiden avulla varmistetaan tietokantatapahtumien luotettavuudesta ja tiedon eheydestä. Tiedon luotettavuuden ja eheyden lisäksi PostgreSQL tarjoaa toimintatapoja, joiden avulla voidaan parantaa tietokantojen saavutettavuutta ja replikointia. Replikoinnin lisäksi PostgreSQL mahdollistaa klusteroinnin, joka mahdollistaa tehokkaan skaalautumisen.

4.1.4 NoSQL-tietokannan hallintajärjestelmät

NoSQL-tietokannan hallintajärjestelmät ovat varteenotettava vaihtoehto relaatiotietokannoille. NoSQL:ssä ei käytetä SQL-kieltä kyselyiden tekemiseen, mutta tietokannat voivat käyttää muita SQL:n tapaisia kyselykieliä. NoSQL-tietokannoille on

tyypillistä, että ne ovat avointa lähdekoodia, vaikka on olemassa myös kaupallisia versioita [21].

Relaatiotietokannoissa tietoa ei voida tallentaa ilman, että sille on määritelty skeema. Skeema määrittelee tietokannan rakenteen, joka kertoo, mitä tauluja tai mitä sarakkeita on olemassa, sekä minkä tyyppistä tietoa niihin voidaan tallentaa. NoSQL-tietokannat puolestaan ovat skeemattomia ja tiedon tallentaminen on yksinkertaisempaa ja joustavampaa, koska tallennettavaa tietoa ei tarvitse etukäteen määrittellä. Tämän vuoksi NoSQL-tietokantaa on helpompi muuttaa, jos tulee tarve lisätä uudentyyppistä tietoa. NoSQL-tietokannoista tietoa on myös helpompi poistaa, sillä tietyyntyyppisen tiedon tallentaminen voidaan helposti lopettaa. Relatiotietokannoissa tietoa poistettaessa joudutaan myös poistamaan sarakkeita, jolloin vaarana on vanhan tiedon katoaminen [21].

NoSQL-tietokannan hallintajärjestelmiä kutsutaan horisontaalisesti skaalautuviksi, jolla tarkoitetaan tietokannan kykyä jakaa tietoa ja suorituskuormaa usealle palvelimelle ilman, että muistia tai kovalevytilaa jaetaan palvelinten kesken [22]. Tämä mahdollistaa isojen tietomäärien käsittelyn sekä selviytymisen ruuhkaisista tietokantaan kohdistuvista kirjoitus- ja lukuoperaatioista. Klustereiden käyttö lisää kuitenkin huomattavasti tietokannan monimutkaisuutta, josta johtuen sen käyttö ilman pakkoa ei ole suositeltavaa [21].

NoSQL-tietokannat voidaan kategorioida niiden tietomallien mukaan. Tyypillisiä NoSQL-tietokantojen tietomalleja ovat avain-arvo -varastot, dokumentaatiovarastot, sarakevarastot sekä graafiset varastot. Näistä vaihtoehdoista avain-arvo varastot ovat helppokäyttöisimpiä ja suosituimpia. Avain-arvo -tietokannassa tietoa haetaan avaimella, asetetaan avaimelle arvo tai tuhoetaan avain tietovarastosta. NoSQL:n tietomallit ja niitä käyttäviä tietokantoja on kuvattu alla esitettyssä taulukossa.

Taulukko 4.1. *NoSQL:n yleisimmät tietomallit ja niitä käyttävät tietokannat [21]*

Tietomalli	Esimerkkejä tietokannoista
Avain-arvo	BerkeleyDB, LevelDB, Memcachedm Project Voldemort, Redis, Riak
Dokumentti	CouchDB, MongoDB, OrientDB, RavenDB, Terrastore
Sarake	Amazon SimbleDB, Cassandra, HBase, Hypertable
Graafi	FlockDB, HyperGraphDB, Infinite Graph, Neo4j, OrientDB

4.2 Tietokannan hallintajärjestelmien vertailu

Tässä luvussa vertaillaan edellisessä luvussa kuvattuja tietokannan hallintajärjestelmiä ja tehdään päätelmiä niiden sekä hyvistä että huonoista puolista. Luvun päätteeksi vertailtavista tietokannan hallintajärjestelmistä valitaan toteutuksen kannalta paras vaihtoehto.

4.2.1 Relaatietietokannat vs NoSQL-tietokannat

Eräs merkittävä ero relaatiotietokantojen ja NoSQL-tietokantojen välillä on tietokantojen eheydessä ja niiden määritelmässä. Tyypillisesti NoSQL-tietokannat eivät tue transaktioille ACID -ominaisuuksia, kuten taas relaatiotietokannat tukevat. NoSQL-tietokannan eheyden yhteydessä puhutaan relaatiotietokannoille tyypillisten ACID-ominaisuuksien sijaan usein BASE-ominaisuuksista. BASE tulee sanoista Basically Available, Soft Stater ja Eventually consistent. ACID tulee puolestaan sanoista Atomicity, Consistency, Isolation ja Durability.

Base-ominaisuuksia tukevissa tietokannoissa päivitykset laitetaan lopulta eteenpäin koko järjestelmään, mutta lukuoperaatioiden eheydelle ei ole varmuutta [22]. Yleisesti katsottuna NoSQL-tietokannoissa on huonompi tiedon eheyden hallinta kuin relaatiotietokannoissa. CAP-teorian mukaan NoSQL-tietokannoissa on kolme ominaisuutta, jotka ovat eheys, saatavuus ja osituksen sietokyky. Teorian mukaan näistä ominaisuuksista vain kaksi voi olla voimassa samaan aikaan. Tyypillisesti NoSQL-tietokannat luopuvat eheys -ominaisuudesta saavuttaakseen paremman skaalautuvuuden ja suorituskyvyn [22]. Toisaalta tiedon eheys vaihtelee paljon eri NoSQL-tietokantojen välillä, tästä johtuen asiat eivät todellisuudessa ole niin yksiselitteisiä. PiceaHub-sovelluksen kannalta eheys on kuitenkin erityisen tärkeä ominaisuus, jonka vuoksi relaatiotietokannat ovat tässä mielessä parempi vaihtoehto toteutukseen.

Relaatiotietokannat ovat useasti suunniteltuja olemaan ajossa yhdellä tietokoneella, kun taas NoSQL-tietokannat ovat suunniteltu olemaan ajossa klustereilla, jotka soveltuvat paremmin isojen tietomäärien käsittelyyn. Näiden suunnitteluperiaatteiden ja tiedon eheydestä luopumisen takia NoSQL-tietokannat voivat saavuttaa relaatiotietokantoja paremman skaalautuvuuden sekä kyvyn laajentaa kapasiteettia tarpeen vaatiessa. Lisäksi NoSQL:n tietomallit voivat tarjota relaatiomallia yksinkertaisemman tiedon tallennustavan ja samalla ne voivat vähentää kirjoitettavan koodin määrää [21].

Relaatiotietokantojen eräs merkittävimmistä eduista NoSQL-tietokantoihin verrattuna on niiden käyttämässä SQL-kielessä. Kaikkia relaatiotietokantoja voidaan käyttää saman kyselykielen avulla, kun taas NoSQL-tietokannoissa vastaavaa yhtenäistä kieltä ei ole. Relatiotietokannoissa yhtenäistävät standardit mahdollistavat sen, että niitä voidaan helposti vaihtaa toiseen. NoSQL-tietokannoissa ei ole yhtenäistä kyselykieltä tai rajapintaa, koska kaikki NoSQL-tietokannat ovat erilaisia ja ne sisältävät omat

rajapinnat, kirjastot ja kielet niiden sisältämän tiedon käsittelyyn [23]. Relaatiotietokannoissa tietoa voidaan puolestaan hakea helposti melkein mitä tahansa ohjelmointikieltä käyttämällä. Tekniikoiden yhtenäisyyden ansiosta relaatiotietokantoja on helpompi käyttää ja hallita kuin NoSQL-tietokantoja.

PiceaHub-sovelluksen käyttötiedon luonteesta johtuen tarvitaan tietokanta, josta tietoa haettaessa joudutaan se useasti yhdistämään useasta osasta. Vaikka NoSQL-tietokannat skaalautuvat joissain tapauksissa paremmin ja ovat nopeampia yksittäisen tiedon hakemisessa kuin relaatiotietokannat, valittiin toteutukseen relaatiotietokanta. Relaatiotietokannoissa suurten tietomäärien yhdistäminen on huomattavasti helpompaa ja tehokkaampaa toteuttaa kuin NoSQL-tietokannoissa, joka puolestaan on tietokannan toteutuksen kannalta merkittävä ominaisuus. Tämä perustuu osittain siihen, että relaatiotietokannat ovat suunniteltuja hakemaan tietoa joukkoina, kun taas NoSQL-tietokannoissa tiedon hakeminen keskittyy yksittäisiin arvoihin tai dokumentteihin. Lisäksi valintaan vaikutti myös se, että NoSQL-tietokantoja on tulevaisuudessa vaikeampi ylläpitää, sillä niiden epäyhtenäiset standardit tekevät suurten muutosten tekemisestä vaikeaa [19].

4.2.2 MySQL vs MariaDB

Relaatiotietokannat osoittautuivat NoSQL-tietokantoja paremmaksi vaihtoehdoksi toteutettavaan järjestelmään, joten tässä luvussa vertaillaan kahta vartenotettavaa relaatiotietokannan hallintajärjestelmää MySQL:ää ja MariaDB:tä. Tarkasteltavat tietokannan hallintajärjestelmät ovat monella tapaa samanlaisia tietokannan hallintajärjestelmiä, koska MariaDB on MySQL:n kehityshaarauma. Molemmissa SQL-kieltä käytetään samalla tavalla ja asetustiedostot ovat samanlaisia. Lisäksi molemmat tarjoavat samoja asiakaspuolen rajapintoja, protokollia ja rakenteita. Myös tietokantayhteyksien luominen ja tietokannan kanssa keskustelu eri ohjelmointikielillä tapahtuu samalla tavalla.

Ehkä merkittävin ero tietokannan hallintajärjestelmien välillä on, että MySQL:stä on olemassa sekä kaupallinen että avoimen lähdekoodin versio. MariaDB:ssä on olemassa ainoastaan avoimen lähdekoodin versio, jossa kaikki on ilmaista ja siihen on sisällytetty myös MySQL:n kaupallisen version ominaisuuksia.

MySQL tunnetaan hyvästä suorituskyvystä, mutta MariaDB on osoittanut tutkimuksissa vielä suorituskykyisemmäksi [24]. Parempi suorituskyky perustuu osittain MariaDB:n optimizer engineen, joka toimii sekä MySQL:n ja MariaDB:n "ytimessä" [24]. Optimizerin tehtävänä vastaanottaa SQL-komentoja ja muuntaa ne tietokannan ymmärtämiksi ohjeiksi. MariaDB:n parannettu optimizer suoriutuu monimutkaisista kyselyistä nopeammin kuin MySQL [24].

MySQL:n ja MariaDB:n tärkeimpiä ominaisuuksia ovat niiden tiedontallennusmoottorit. Tiedontallennusmoottorilla tarkoitetaan komponenttia, joka käsittelee SQL-operaatioita tietokannan tauluille. MySQL:ssä ja MariaDB:ssä on useita vaihtoehtoisia tiedontallennusmoottoreita, jotka soveltuvat erilaisiin käyttötarkoituksiin. MariaDB:ssä tiedontallennusmoottorit ovat osoittautuneet monipuolisemmiksi ja suorituskykyisemmiksi kuin MySQL:ssä. MariaDB tosin sisältää samoja tiedontallennusmoottoreita kuin MySQL, mutta ne ovat uusittuja ja parannettuja versioita. MySQL:n tunnetuin ja vakiona käytetty tiedontallennusmoottori on InnoDB, joka sopii sekä transaktionaalisiin että ei-transaktionaalisiin käyttötarpeisiin. XtraDB puolestaan on MariaDB:n vakiona käytetty tiedontallennusmoottori, joka on InnoDB:n paranneltu ja suorituskykyisempi versio. XtraDB:n parempi suorituskyky perustuu sen kykyyn skaalautua paremmin useita ytimiä ja enemmän muistia sisältävillä tietokoneilla [24].

Eräs merkittävä asia tietokannan suorituskyvyn kannalta on asetus, jossa tietokannan käyttöön liittyviä säikeitä voidaan ajaa säiealtaassa. Säiealtaan avulla järjestelmän resursseja voidaan paremmin hallita, joka puolestaan lisää suorituskykyä [25]. MySQL:ssä säieallas on saatavilla kaupallisessa versiossa, kun taas MariaDB:ssä se on sisäänrakennettu. MariaDB:ssä säiealtaan toteutus on todettu olevan tehokkaampi, jonka vuoksi MariaDB on joillakin osa-alueilla suorituskyvyltään MySQL:ää parempi [25].

MariaDB:tä ja MySQL:ää verratessa on selvää, että kumpikin tietokannan hallintajärjestelmä on hyvä vaihtoehto toteutettavan järjestelmän kannalta. Skaalautuvuuden näkökulmasta ei näiden tietokannahallintajärjestelmien välillä suurta eroa ole, sillä molemmat tarjoavat samat skaalautuvuustekniikat, kuten klusteroinnin ja replikoinnin. Toisaalta MariaDB tarjoaa säiealtaan ilmaiseksi, kun taas MySQL:ssä se on saatavilla ainoastaan kaupallisessa versiossa. Tämän perusteella MariaDB:tä voidaan pitää paremmin skaalautuvana vaihtoehtona, sillä luvussa 2.3.2 esitettyjen vaatimusten perusteella MySQL:n kaupallinen versio on suljettu pois valittavista vaihtoehdoista. MariaDB on joidenkin tutkimusten mukaan myös suorituskyvyltään parempi vaihtoehto kuin MySQL ja sisältää myös uutta toiminnallisuutta, kuten dynaamisia sarakkeita ja tarkempia aikaleimoja [26]. Suurin syy MariaDB:n paremmuuteen on, että MySQL:ssä ominaisuudet, kuten tiedontallennusmoottorit voivat siirtyä avoimen lähdekoodin versiosta kaupalliseen versioon. MariaDB:ssä ei vastaavaa riskiä ole, joten se on myös tältä osin parempi vaihtoehto kuin MySQL.

4.2.3 PostgreSQL vs MariaDB

Edellä olevassa luvussa todettiin MariaDB:n olevan toteutuksen kannalta parempi vaihtoehto valittavaksi tietokannan hallintajärjestelmäksi kuin MySQL, joten viimeiseksi vertaillaan MariaDB- ja PostgreSQL -tietokannan hallintajärjestelmiä. Kummatkin näistä vaihtoehdoista sisältävät paljon ominaisuuksia ja ne toimivat useilla

alustoilla, kuten Linux, Unix, Mac OS X ja Windows. Molemmat ovat myös avointa lähdekoodia sekä toimivat hyvin pienissä ja suurissa järjestelmissä.

MariaDB:tä pidetään hyvin soveltuvana erityisesti WWW-sivustojen toteutukseen, sillä tietokannasta tapahtuvat lukuoperaatiot ja pienien kyselyiden suorittaminen on nopeaa. Toisaalta MariaDB sisältää vähemmän tiedon eheyden tarkistusta kuin PostgreSQL, jota pidetään paljon ominaisuuksia sisältävänä ja vakaana tietokannan hallintajärjestelmänä. PostgreSQL soveltuu hyvin sovelluksiin, jossa tarvitaan vahvat ACID-ominaisuudet ja tiedon eheyden tarkistukset.

Molemmat tietokannan hallintajärjestelmät ovat nopeita tietyissä tehtävissä. MariaDB:ssä voidaan vaihtaa tiedontallennusmoottoria, jotka toimivat eri tavalla eri tilanteissa. MariaDB:n Aria-tiedontallennusmoottori on suorituskykyisin vaihtoehto kaikista tiedonhallintamoottoreista, koska siinä suoritetaan vähemmän tiedon eheyden tarkistuksia [18]. Aria soveltuu erityisen hyvin sovelluksiin, jossa suoritetaan lukuisia lukuoperaatioita. Aria ei kuitenkaan sovellu sovelluksiin, joissa sekä kirjoitetaan että tallennetaan arkaluontoista tietoa. Toisaalta MariaDB sisältää XtraDB-tiedontallennusmoottorin, joka puolestaan tukee PostgreSQL:n tapaan täysin transaktioita ja ACIDominaisuuksia, jotka pyrkivät varmistamaan tiedon eheyttä [18].

Skaalautuvuuden näkökulmasta tietokannan hallintajärjestelmien välillä ei ole suurta eroa ja molemmat tarjoavat siihen samat tekniikat. PostgreSQL mahdollistaa kuitenkin omien tietotyyppien ja operaatioiden luomisen, joka on selkeä etu MariaDB:hen verrattuna. Lisäksi PostgreSQL tarjoaa valmiina graafisen käyttöliittymän tietokannan käsittelyyn, toisin kuin MariaDB. MariaDB on kuitenkin aiemmissa testeissä osoittautunut nopeammaksi kuin MySQL, joten voidaan olettaa sen olevan myös jonkin verran suorituskykyisempi kuin PostgreSQL [27]. PostgreSQL on luotettava ja hyvä suojaamaan tietoa, kun taas MariaDB on joustava ja sen tarjoamat useat tiedontallennusmoottorit mahdollistavat suorituskuorman jakamisen monella tavalla. Molemmat tietokannan hallintajärjestelmät soveltuisivat hyvin toteuttavaan järjestelmään, mutta tässä toteutuksessa suorituskyvyn todettiin olevan toteutuksen kannalta tärkeää, joten näistä vaihtoehdoista MariaDB todettiin olevan parempi tietokannan hallintajärjestelmä toteutukseen.

4.2.4 Tietokannan hallintajärjestelmän valinta

Edellisissä luvuissa todettiin relaatiotietokantojen soveltuvan paremmin toteutukseen kuin NoSQL-tietokannat. Samoin tarkasteltavista relaatiotietokannoista MariaDB:n todettiin olevan parempi vaihtoehto kuin MySQL ja PostgreSQL, joten toteutukseen tietokannan hallintajärjestelmäksi valittiin MariaDB. Valintaan vaikuttaneet tekijät ja MariaDB:n vahvuudet verrattuna muihin tietokannan hallintajärjestelmiin on esitetty taulukossa 4.1.

Taulukko 4.1. MariaDB:n vahvuudet ja heikkoudet

Tietokannan hallintajärjestelmä	MariaDB:n vahvuudet	MariaDB:n heikkoudet
NoSQL-järjestelmät	<ul style="list-style-type: none"> ❖ Tiedon yhdistäminen on tehokkaampaa ja helpompaa ❖ Tiedon eheys on paremmin varmistettu ❖ Tunnetut standardit helpottavat ylläpitoa ja tiedon siirtoa 	<ul style="list-style-type: none"> ❖ Huonompi skaalautuvuus. ❖ Tarvitsee määritellä skeema ennen kuin tietoa voidaan tallentaa tietokantaan.
MySQL	<ul style="list-style-type: none"> ❖ Sisältää saman toiminnallisuuden, mutta ei huonoja puolia. ❖ Sisältää MySQL:n kaupallisen version ominaisuuksia. ❖ Suorituskykyisempi ❖ Sisältää uutta toiminnallisuutta, kuten dynaamiset sarakkeet ja tarkempia aikaleimoja. 	<ul style="list-style-type: none"> ❖ MySQL:n uuden version ilmestyessä kestää useista viikoista kuukausiin, ennen kuin kaikki samat toiminnallisuudet ovat saatu päivitettyä myös MariaDB:hen.
PostgreSQL	<ul style="list-style-type: none"> ❖ Aiemmissä mittauksissa MySQL - pohjaiset tietokannan hallintajärjestelmät ovat olleet suorituskykyisempiä kuin PostgreSQL. ❖ Tiedontallennusmoottorit mahdollistavat sekä hyvän suorituskyvyn että ACID - ominaisuudet. Niiden avulla tietokanta on myös helpommin mukautettavissa tiettyyn käyttötarkoitukseen. 	<ul style="list-style-type: none"> ❖ Ei ole mahdollista luoda omia tietotyyppejä. ❖ Ei ole valmista tukea graafiselle käyttöliittymälle.

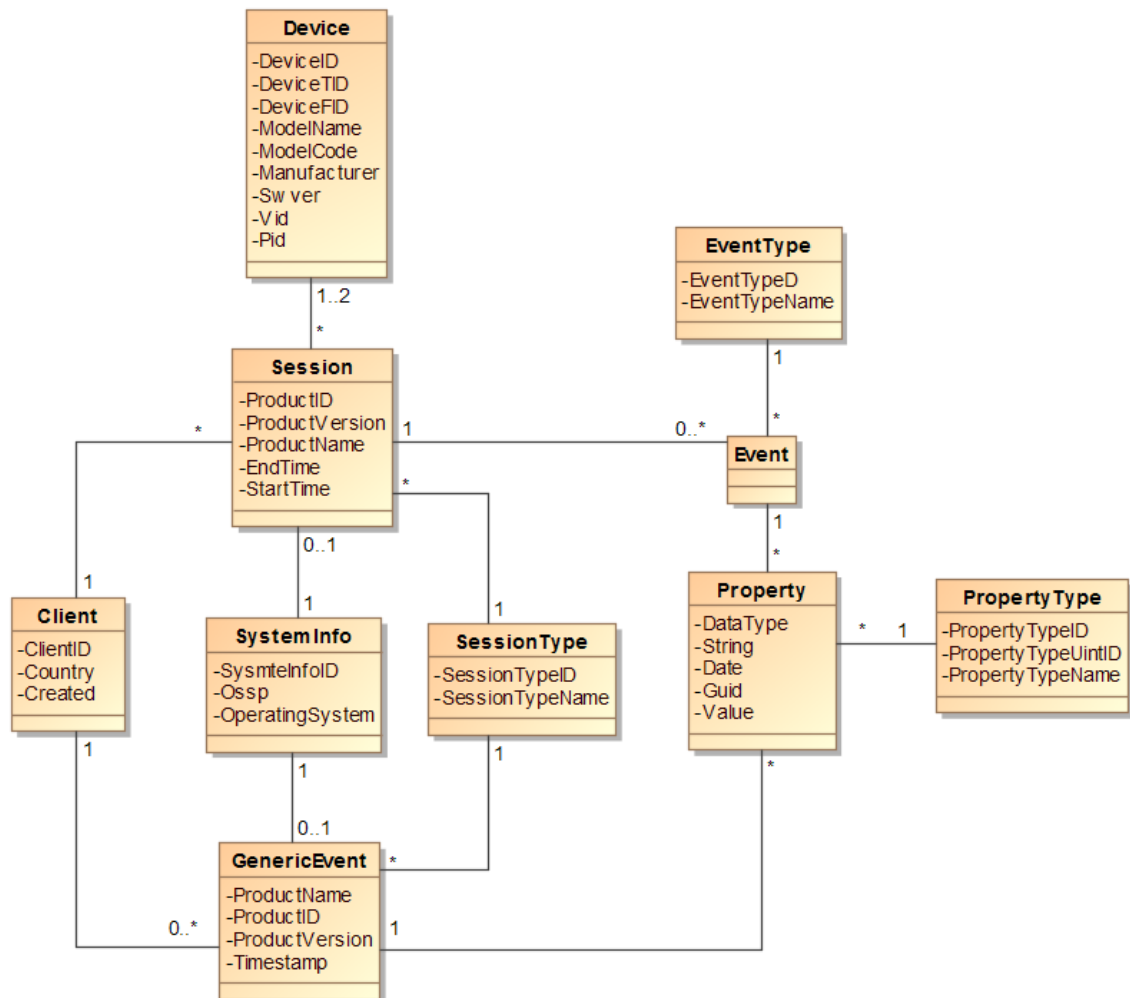
4.3 Sovelluksen tietomalli

Valittuun MariaDB-tietokannan hallintajärjestelmään suunniteltiin yleiskäyttöinen tietokanta, jonka mallia voidaan hyödyntää myös yrityksen tulevissa sovelluksissa ja niistä kerättävien tietojen tallentamisessa. Lisäksi tietokanta suunniteltiin mahdollisimman suorituskykyiseksi, jotta se suoriutuisi tulevaisuudessa hyvin myös suurista tietomääristä.

Itse käsitteellinen mallintaminen toteutettiin iteratiivisena prosessina, joka kehittyi ja muuttui projektin aikana useaan otteeseen. Käsittekaavion ja tietokannan suunnittelussa käytettiin niin sanottua Top-Down -lähestymistapaa, jossa pyrittiin tunnistamaan ensin isoja kokonaisuuksia, joihin myöhemmin pyrittiin lisäämään yksityiskohtaisempaa tietoa. Suunnittelussa olisi voitu hyödyntää myös Bottom-Down -menetelmää eli pienistä yksityiskohdista olisi siirrytty isompiin kokonaisuuksiin. Tämän menetelmän

riskinä oli kuitenkin tarpeeton tiedon toistuminen, jonka vuoksi sitä ei tässä projektissa käytetty.

Käsitekaaviolla tarkoitetaan kohdealueen vääristymätöntä ja täydellistä esitystä, jossa toteutusnäkökohdat jätetään huomioimatta [28]. Toteutetun järjestelmän tietokannan käsitekaaviota on kuvattu UML-luokkakaaviona kuvassa 4.1.



Kuva 4.1. Tietokannan käsitekaavio

Toteutettava järjestelmä sisältää asiakkaita, joilta tallennetaan sekä yleistä tietoa että sovellukseen käyttöön liittyvää tietoa. Tiedot pyrkivät kertomaan kuinka usein, mistä ja miten sovellusta käytetään. Järjestelmään tallennetaan asiakkaita yksilöivä tieto sekä sovelluksen käyttöympäristöön että käytettyyn laitteeseen liittyvää tietoa. Lisäksi järjestelmä sisältää jokaiseen sovelluksen käyttökertaan liittyvät tiedot sekä mitä ominaisuuksia asiakas on järjestelmässä käyttänyt.

4.3.1 Asiakas

Käsitekaaviossa oleellinen käsite on Client, joka kuvaa järjestelmässä PiceaHub-sovelluksen asiakasta. Se sisältää asiakkaan yksilöivän tunnuksen, maan ja luontiajan.

Client -käsitteen avulla voidaan tietokannasta tutkia, kuinka paljon sovelluksella on käyttäjiä ja mistä päin maailmaa he ovat kotoisin. Lisäksi käsitteen avulla pystytään seuraamaan, kuinka käyttäjämäärät kehittyvät tietyllä ajan jaksolla. Client-käsitteen kautta päästään käsiksi muihin asiakkaan ja sovelluksen käyttötietoihin.

4.3.2 Istunto ja yleinen tapahtuma

Client-käsitteeseen liittyy kaksi käsitettä: istunto (Session) ja yleinen tapahtuma (GenericEvent). Näiden kahden käsitteen avulla saadaan PiceaHub-sovelluksen käyttäjiltä tiettyyn käyttökertaan liittyvä yleinen tieto, kuten sovelluksen käyttöaika, käytettävän sovelluksen tyyppi ja sovelluksen versio. Sovelluksen tyyppi on erotettu käsittekaaviossa omaksi attribuutikseen, koska samaa tietokannan rakennetta haluttiin hyödyntää myös tulevilla yrityksen sovelluksissa.

Yleinen tapahtuma haluttiin erottaa istunnosta ja tapahtumista (Event), sillä ne sisältävät tietoa, jotka eivät varsinaisesti liity sovelluksen oikeisiin tapahtumiin tai istuntoihin. Yleinen tapahtuma sisältää tietoa esimerkiksi käyttäjän lähettämästä palautteesta, joka ei ole sovelluksen oikea ominaisuus.

4.3.3 Käyttöjärjestelmän tiedot

Yleiseen tapahtumaan ja istuntoon liittyy asiakkaan käyttämän tietokoneen käyttöjärjestelmän tiedot (SystemInfo). Käyttöjärjestelmän tiedot sisältävät käyttöjärjestelmän yksilöivän tunnuksen sekä nimen. Näiden tietojen avulla voidaan tunnistaa mahdolliset käyttöjärjestelmästä johtuvat ongelmat tai puutteet.

4.3.4 Laitetiedot

Istuntoon liittyy myös asiakkaan laitetiedot (Device). Laitetiedoista saadaan selville laitteen tyyppi, joka voi olla esimerkiksi tietokone, muistitikku tai mobiililaitte. Jokaiseen laitteeseen liittyy laitteen yksilöivä tieto, mallikoodi, mallin nimi ja laitteen valmistaja. Laitetietojen avulla voidaan tunnistaa mahdollisista laitteista johtuvat ongelmatilanteet tai puutteet.

4.3.5 Tapahtuma

Istuntoon eli sovelluksen käyttökertaan liittyy tapahtumia (Event), jotka kuvaavat mitä tapahtumia ja ominaisuuksia sovelluksessa on tietyn käyttökerran aikana käytetty. Varsinainen tapahtuma ei kuvaa tapahtuman tyyppiä, mutta sen avulla päästään käsiksi sen kuvaustauluun (EventType), joka sisältää kaikkien tapahtumien nimet ja tunnisteet. Tapahtumien varsinaiset tiedot sijaitsevat tapahtuman ominaisuuksissa (Property).

4.3.6 Ominaisuus

Tapahtumiin ja yleiseen tapahtumaan liittyvät ominaisuudet (Property) mahdollistavat tietokannan yleiskäyttöisen rakenteen ja sen, että tapahtumat (Event) voivat sisältää minkä tyyppistä tietoa tahansa. Ominaisuus sisältää tietotyypin (DataType), joka kertoo minkä tyyppistä tietoa tiettyyn tapahtumaan liittyy. Tietotyyppi voi olla tyypiltään merkkijono, päivämäärä, kokonaisluku tai ominaisuuden yksilöllinen tunnuskoodi. Tietotyyppi kertoo myös, missä ominaisuuden attribuuteista varsinainen tieto sijaitsee.

4.3.7 Metatieto

Istuntoon, tapahtumaan, yleiseen tapahtumaan ja ominaisuuksiin liittyy metatietoa eli tietoa tiedosta. Tässä toteutuksessa metatieto on toteutettu kuvaavina käsitteinä, jotka sisältävät käsitteisiin liittyviä yksilöllisiä merkkijonotunnuksia sekä hakuja nopeuttavia kokonaislukuja. Lisäksi kuvaaviin käsitteisiin on sisällytetty yksilöllisiin tunnuksiin liittyviä nimiä, joiden avulla voidaan tehdä erilaisia koosteita WWW-sivustossa.

4.4 Tietokannan fyysinen suunnittelu

Ennen tietokannan fyysistä suunnittelua suoritettiin tietokannan looginen suunnittelu, jossa luotu käsittekaavio muunnettiin relaatiokaavioksi. Relaatiokaavion tarkoituksena oli varmistaa, että toteutettava tietokanta on riippumaton tietokannan hallintajärjestelmästä. Relaatiokaavion avulla nähtiin myös mahdollisia toteutusvaihtoehtoja, joita voitiin hyödyntää varsinaisessa tietokannan fyysisessä suunnittelussa.

Fyysisen suunnittelun päätavoitteiksi asetettiin tietokannan yleiskäyttöisyys ja suorituskyky, jossa riippuvuuksia ja toisteista tietoa on mahdollisimman vähän. Toisteisen tiedon ongelmana on yleensä, että se vie turhaa levytilaa palvelimelta ja vaikeuttaa tietokannan ylläpitoa. Lisäksi tietokantaa päivitettäessä muutokset joudutaan tekemään useaan paikkaan. Tiedon toisteisuuden vähentämiseksi tietokannan käsittekaavioille ja relaatiokaavioille suoritettiin normalisointia ennen kuin tietokantaa alettiin varsinaisesti toteuttaa. Normalisointi toteutettiin kolmanteen normaalimuotoon, jota pidetään suurimpana tarpeellisena tasona sovelluksille [29].

Ennen kuin tietokantaa ja tietokannan tauluja voitiin luoda, valittiin tietokannan tauluille toteutukseen sopivat tiedontallennusmoottorit. MariaDB tarjoaa tauluille useita tiedontallennusmoottoreita, kuten XtraDB, MyiSAM, Aria, Archive ja ScaleDB. Näistä vaihtoehdoista suorituskyvyn kannalta parhaita ovat Aria ja MyiSAM, mutta luotettavuus niissä on huonompi kuin XtraDB:ssä. Tämän vuoksi toteutukseen valittiin XtraDB, sillä se tarjoaa hyvän suorituskyvyn lisäksi myös tuen transaktioille ja sen ACID -ominaisuudet pyrkivät turvaamaan tiedon eheyttä.

Tiedontallennusmoottorin valinnan jälkeen relaatiokaaviosta muunnettiin relaatiot tauluiksi ja relaatioiden attribuutit taulujen sarakkeiksi. Tässä muunnosprosessissa tietokannan fyysinen toteutus ei juuri eronnut relaatiokaaviosta. Jokaiselle taululle luotiin ensiksi pääavain, joiden avulla taulun rivit voitiin erottaa toisistaan. Tietyille tauluille pääavaimeksi mietittiin PiceaHub-sovelluksen käyttötiedosta valmiina saatavaa GUID-tietotyyppiä, jolla voitiin yksilöidä taulujen rivejä. GUID eli Globally Unique Identifier on yksilöllinen 128-bittinen numerosarja, jota käytetään ohjelmistoissa tunnisteina. Tämän tietotyypin ongelmana oli kuitenkin sen tehottomuus, sillä se oli pitkä ja sisälsi numeroiden lisäksi myös merkkejä. Jotta pääavaimiin kohdistuvat haut ja liitokset olisivat tietokannassa nopeampia, päätettiin jokaiselle taululle antaa pääavaimeksi kokonaisluku. Pääavainten pituudet asetettiin sen mukaan, kuinka paljon taululle arvioitiin tulevan rivejä tulevaisuudessa. Pääavaimia luotaessa annettiin niille myös `auto_increment` -määritelmä, joka korottaa automaattisesti kyseisen sarakkeen arvoa yhdellä, kun tietokantaan luodaan uutta riviä. Tämän määritelmän avulla jokainen taulun rivin pääavain sisältää yksilöllisen kokonaislukuarvon. Pääavainten luonnin lisäksi tietokantaan määritettiin taulujen vierasavaimet, joiden avulla taulujen välille luotiin yhteyksiä. Muut taulujen sarakkeet toteutettiin tyypillisten tietotyyppien, kuten kokonaislukujen, merkkijonojen ja aikaleimojen avulla.

Tietokannan toteutuksen yhteydessä luotiin myös PHP-ohjelmointikielellä toteutettuja skriptejä, joissa tietokannan toimivuutta testattiin erilaisilla kyselyillä ja syötteillä. Testaamisen tarkoitus oli selvittää, että tietokannasta saadaan helposti haettua kaikki WWW-sivustoilla tarvittava tieto. Lisäksi testauksen yhteydessä pyrittiin suunnittelemaan kyselyitä, joita WWW-sivustot tulevat käyttämään käyttötiedon esittämiseen sovelluskehittäjille.

Kyselyiden vasteaikojen pienentämiseksi ja suorituskyvyn parantamiseksi merkittävillä kyselyillä suoritettiin hakemistoja. Hakemistojen tarkoitus on muodostaa tietokannan tauluista erillinen tietojoukko, joiden tarkoitus on tehostaa monikoiden hakemista tauluista. Pääavaimille ja vierasavaimille tietokannan hallintajärjestelmä osaa automaattisesti luoda hakemistoja, joten näille niitä ei tarvinnut luoda. Tässä toteutuksessa luotiin raskaimmille kyselyille hakemistot, joihin valittiin kyselyiden liitoksissa ja hakuehdoissa käytetyt sarakkeet. Lisäksi hakemistoissa valittiin käytettävimmät ja valikoivat ominaisuudet ensimmäiseksi, jotta niistä saataisiin mahdollisimman tehokkaita. Hakemistojen käyttöä tarkkaillaan kuitenkin edelleen ja niitä voidaan tarpeen mukaan lisätä tai poistaa.

4.4.1 Tietoturvallisuus

Tietokantaan kohdistuvat uhat voivat vaarantaa tietokannassa olevan tiedon eheyden, saatavuuden ja luettavuuden. Tietokannan eheys voi vähentyä, jos tietokantaan kohdistetaan luvattomia muutoksia joko tahallisilla tai ei-tahallisilla toimenpiteillä. Jos menetettyä tietoa ei korjata, se voi johtaa järjestelmän epätarkkaan, väärään tai

virheelliseen toimintaan [11]. Samoin tietokannan saavutettavuus voi uhkien myötä vähentyä, jolloin ohjelmat tai käyttäjät eivät pääse käsiksi heille oikeutettuihin tietoihin. Luottamuksellisuudella puolestaan tarkoitetaan tapaa, jossa tieto turvataan luvattomalta paljastumiselta tai julkaisemiselta. Luottamuksellisuuden varmistaminen on erityisen tärkeä järjestelmissä, joihin on tallennettuna arkaluontoista tietoa, kuten tässä toteutuksessa olevaa asiakkaiden tietoa.

Tietokannan eheyden, saatavuuden ja luottamuksellisuuden turvaaminen toteutettiin pääsynhallinnan avulla. Pääsynhallinnassa sovelluksille asetettiin käyttöoikeudet niiden toimintaperiaatteiden mukaan. Käyttöoikeudet määrittävät, mitä käyttäjät saavat ja voivat tehdä. Toteuttamalla sovelluskohtaisia käyttöoikeuksia voitiin varmistaa, että tietystä sovelluksesta ei voida suorittaa laittomia operaatioita.

Tässä toteutuksessa XML-jäsentäjälle ja WWW-sivustoille luotiin omat käyttöoikeudet. XML-jäsentäjälle annettiin sekä luku- että kirjoitusoikeudet, kun taas WWW-sivustossa tietokantaa voidaan käyttää ainoastaan lukuoikeuksien avulla. Näiden oikeuksien avulla saatiin parannettua erityisesti tiedon luottamuksellisuutta, sillä tieto on ainoastaan siihen oikeutettujen käytössä.

Pääsynhallinnan lisäksi tietokannan tiedon eheyttä, luottamuksellisuutta ja saatavuutta pyrittiin parantamaan ratkaisulla, jossa tietokanta sijoitettiin sisäverkossa sijaitsevalle palvelimelle. Tämän ratkaisu estää yhteydet ulkoverkosta, joka vähentää huomattavasti mahdollisia tietoturva-uhkia, kuten palvelunestohyökkäyksiä.

5 PHP-SOVELLUSTEN TOTEUTUS

Tässä diplomityössä toteutettiin kolme PHP-sovellusta, joiden tehtävinä on huolehtia PiceaHub-sovelluksesta saapuvan XML-tiedon vastaanottamisesta, tietojen jäsentämisestä tietokantaan ja tietojen näyttämisestä WWW-sivustossa. Tässä luvussa kerrotaan näiden sovellusten toteutuksista.

5.1 Toteutusvaihtoehdot

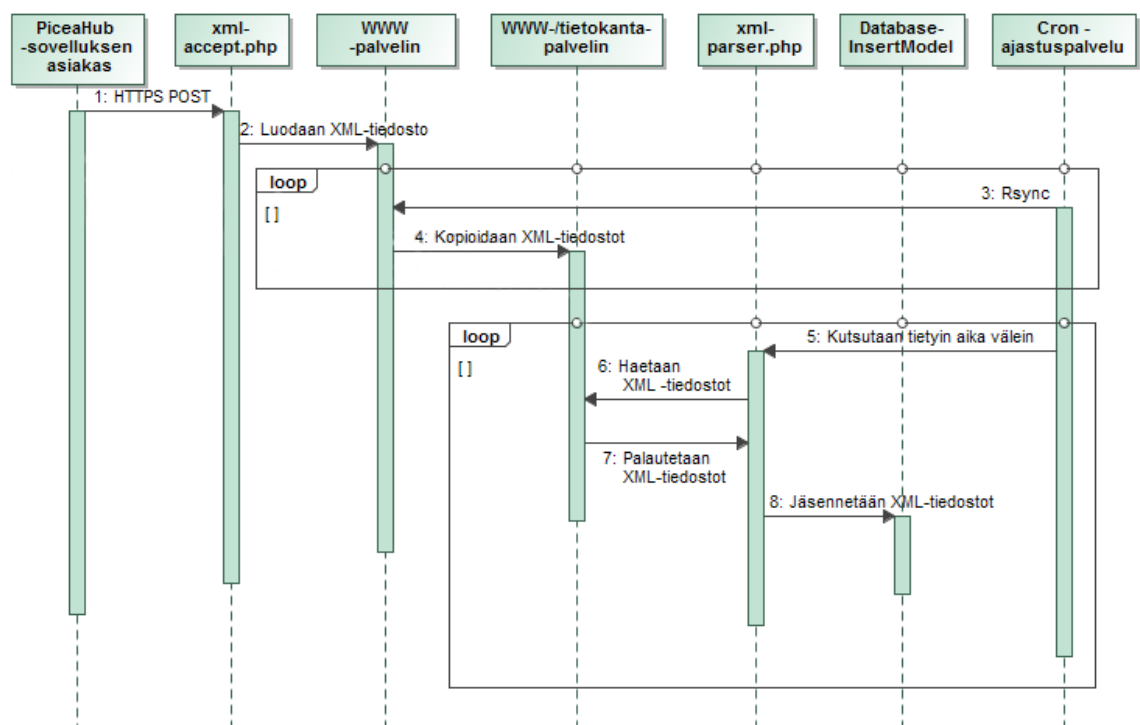
PiceaHub-sovelluksesta saatavan käyttötiedon lähetyks toteutettiin HTTPS-protokollan POST-metodin avulla, jossa tiedot välitetään pyynnön mukana XML-muodossa palvelimelle. Vastaanottamisen jälkeen tieto tallennetaan palvelimelle XML-tiedostoksi, jotka ajetaan luoduista tiedostoista tietokantaan. Tiedostojen tallentamisen tarkoitus on varmistaa järjestelmän toiminnallisuus ja mahdollistaa tietokannan rakenteen muuttaminen ilman, että tietoa katoaisi.

Tässä käyttötiedon vaihtoehtoisena tallennustapana olisi voinut käyttää XML-muotoa tehokkaampaa JSON-muotoa. JSON-tieto tarjoaa XML-tietoon verrattuna yksinkertaisemman rakenteen, jonka ansiosta sen tallentaminen tarvitsee vähemmän levytilaa ja sitä on myös PHP-koodissa tehokkaampi jäsentää. Vaikka JSON-muoto vie vähemmän levytilaa ja on tehokkaampi ratkaisu kuin XML, valittiin toteutukseen XML-formaatti, koska pelkästään tiedostoja tarkasteltaessa XML-tiedostot ovat JSON-tiedostoja luettavampia, sillä niissä tieto on selkeämmin kuvattu ja esitetty. Valintaan vaikutti myös se, että XML-tiedon käsittelyyn oli tarjolla enemmän kirjastoja ja sovelluskehittäjillä oli enemmän käytännön kokemusta XML-tiedon käsittelystä.

XML-tiedostojen jäsentäjä toteutettiin eri palvelimilla kuin XML-tiedon vastaanottaja, joka puolestaan lisäsi toteuttavan järjestelmän monimutkaisuutta. Toteutuksessa tiedostot joudutaan synkronoimaan palvelimelta toiselle ajastuspalvelujen ja apuohjelmien avulla ennen kuin käyttötietoa voidaan ajaa tietokantaan. Vaihtoehtoisena toteutustapana tässä olisi voinut käyttää vain yhtä palvelinta toteutuksen tekemiseen, jolloin erillistä synkronointia ja ajastuspalvelua ei olisi tarvinnut käyttää. Yhden palvelimen ongelmana on kuitenkin, että siihen kohdistuu suuri suorituskuorma ja XML-tiedostoihin saatetaan päästä käsiksi ulkoverkosta. Siirtämällä XML-tiedostot ja varsinainen tietokanta sisäverkon palvelimelle saatiin yksittäisen palvelimen suorituskuormaa pienennettyä ja voitiin varmistua paremmasta tietoturvallisuudesta, sillä XML-tiedostoihin ei pääse käsiksi kuin yrityksen sisäverkosta.

Toteuttamalla tiedostojen synkronointi sisäverkossa olevalle palvelimelle vähennettiin yhteen palvelimeen kohdistuvaa kuormitusta ja parannettiin tiedostojen tietoturvallisuutta, koska ulkoverkosta niihin ei päästä käsiksi. Lisäksi sisäverkossa lisätallennustilan hankinta on halvempaa kuin ulkoverkossa kiinni olevalla palvelimella.

Järjestelmän toteutusratkaisua ja siinä tapahtuvaa tiedon kulkua on kuvattu tapahtumasekvenssikaavion avulla kuvassa 5.1. PiceaHub-sovelluksen asiakkaan käyttötieto lähetetään HTTPS POST -pyynnön mukana XML-muodossa yrityksen WWW-palvelimella sijaitsevalle xml-accept.php -tiedostolle, joka luo pyynnön sisällöstä XML-tiedoston tietokantapalvelimelle. Cron-ajastuspalvelusta kutsutaan tietyin aikavälein rsync-ohjelmaa, joka kopioi XML-tiedostot palvelimelta toiselle. Jotta tiedot saadaan WWW-/tietokantapalvelimella ajettua tietokantaan, cron-ajastuspalvelu kutsuu xml-parser.php -tiedostoa hakemaan palvelimelle kopioidut tiedostot käsiteltäväksi. Tämän jälkeen käsiteltävät tiedostot jäsennetään PHP:lla toteutetun DatabaseInsertModel -luokan avulla tietokantaan.



Kuva 5.1. Tiedon kulku PiceaHub -sovelluksesta tietokantaan

5.2 XML-tiedon vastaanottaminen

Käyttötietoa lähetetään PiceaHub-sovelluksen käyttäjiltä palvelimelle kyseisen sovelluksen sulkemisen yhteydessä ja laitetta irrotettaessa tietokoneesta sovelluksen ollessa päällä. Varsinainen käyttötieto lähetetään palvelimelle HTTPS-protokollan POST-metodin avulla, jossa tiedot välitetään XML-muodossa SSL-salatuin yhteyden yli.

POST toimii pyyntö-vastaus -periaatteella ja sitä käytetään tiedon lähettämiseen määritellyille resurssille, jossa lähetettävän tiedon määrää ei ole rajoitettu [30].

Jotta POST-pyyntöjä ja niiden sisältämää XML-tietoa voitiin vastaanottaa, toteutettiin palvelimelle sovellus PHP-ohjelmointikielellä. Pyyntöjen vastaanottamisen lisäksi sovelluksen tehtävänä on luoda pyyntöjen sisällöstä XML-tiedostoja ja tallentamaan ne määrättyyn hakemistoon myöhempää käsittelyä varten. Ennen tiedostojen tallentamista sovelluksessa lisätään XML-tietoon pyynnön lähettäneen asiakkaan IP-osoite ja maa. Asiakkaan IP-osoite haetaan sovelluksessa PHP:n globaalia `$_SERVER['REMOTE_ADDR']` -taulukkomuuttujaa käyttämällä. IP-osoitetta vastaava maatieto haetaan puolestaan käyttämällä sovelluksessa ulkoisia palveluita, jotka palauttavat HTTP:n GET -pyynnöllä IP-osoitetta vastaavan maatiedon.

PiceaHub-sovelluksen käyttäjältä voi tulla useita pyyntöjä samaan aikaan, joten ennen XML-tiedoston luontia lisätään niiden nimiin niitä yksilöiviä tietoja. Yksilöivän tiedon tarkoitus on estää tiedostojen päällekirjoituksia ja helpottaa XML-tiedostojen manuaalista tarkastelua myöhemmin. Tätä tarkoitusta varten tiedoston nimeen haetaan sovelluksessa XML-tiedosta asiakkaan yksilöivä tunnus ja asiakkaan luontiaika. Lisäksi tiedostojen nimeen lisätään PHP:n satunnaisluku generaattorilla kuusinumeroinen luku, jotta estetään päällekirjoitukset varmuudella.

WWW-palvelimelle luodut XML-tiedostot synkronoidaan yrityksen sisäisessä verkossa olevalle tietokanta palvelimelle, jotta voidaan varmistua arkaluotoisen käyttötiedon tietoturvallisuudesta. XML-tiedostoihin pääsee käsiksi siis ainoastaan yrityksen sisäisestä verkosta ja sen ulkopuolisilta käyttäjiltä tietoihin pääsy on evätty. Tiedostojen varsinaisessa synkronoinnissa käytettiin rsync-ohjelmaa, joka mahdollistaa tiedostojen siirron SSH:ta käyttäen palvelimelta toiselle. Varsinainen rsync-ohjelman kutsuminen toteutettiin käyttämällä Unix-käyttöjärjestelmän cron-ajastuspalvelua, joka kutsuu rsync-ohjelmaa tietyin aikavälein.

5.3 XML-tiedostojen jäsentäminen ja tallennus

Jotta XML-tiedostot saatiin jäsenettyä ja tallennettua tietokantaan, toteutettiin palvelimelle sitä varten sovellus PHP-ohjelmointikieltä käyttämällä. Sovelluksen tehtävä on hakea luodut XML-tiedostot kansioista, jäsentää ne ja tallentaa niistä saadut tiedot tietokantoihin. Tietokantoihin lisäämisen jälkeen XML-tiedostot siirretään varmuuskopioksi toiseen kansioon. Varmuuskopiot XML-tiedostoista säilötään niin kauan, kunnes ollaan varmoja järjestelmän toimivuudesta ja siitä, että muutoksia tietokannan rakenteeseen ei enää tehdä.

PHP-sovelluksen toteutuksessa XML-tiedostot haetaan käsiteltäväksi yksi kerrallaan käyttäen PHP:n silmukoita ja funktioita. Ennen kuin tiedostoa voitiin käsitellä,

muutettiin XML-tiedosto käsiteltävään olio -muotoon PHP:n SimpleXML -moduulia käyttämällä. SimpleXML -moduulilla mahdollistettiin, että XML-tietoa voitiin käsitellä sekä elementti että attribuutti kerrallaan.

Ennen kuin XML-tietoa voidaan jäsentää ja tallentaa tietokantaan, haetaan sovelluksessa käsiteltävästä XML-oliosta tuotteen tyyppi, joka määrittelee käytettävän tietokannan. Tuotteen tyyppin hakemisen jälkeen sovelluksessa luodaan yhteys MariaDB tietokannan ja PHP-sovelluksen välille. Yhteyden luomista varten sovellukseen toteutettiin tietokantaluokka, jonka tehtävinä on huolehtia yhteyden luomisesta ja tietokantaan kohdistuvien kyselyiden suorittamisesta. Lisäksi luokka tarjoaa apufunktioita kyselyiden suorittamiseen sekä ilmoittaa virheilmoituksina kyselyiden onnistumisesta ja epäonnistumisista.

Varsinainen tietokantayhteyden luominen suoritettiin tietokanta-apuluokassa käyttämällä Mysqli-laajennusta, joka soveltuu myös MariaDB:n tietokantoihin. Mysqli (MySQL improved) tarjoaa tietokannan käsittelyä varten nimensä mukaisesti parannetun rajapinnan MySQL-laajennukseen verrattuna. Toteutuksen kannalta merkittävimpiä rajapintojen eroja ovat lisätty tuki transaktioille ja prepare-lausekkeiden luomiselle. Prepare-lausekkeiden avulla tietokannan tauluihin sijoitettavat arvot voitiin muuttaa automaattisesti oikeaan tietotyyppiin. Erityisesti tästä oli hyötyä tapauksissa, joissa taulujen sarakkeisiin sijoitettiin sekä NULL- että merkkijonoarvoja.

Jäsentäjä -ohjelman varsinainen kutsuminen toteutettiin käyttämällä Unix-pohjaisen käyttöjärjestelmän Cron-ajastuspalvelua, joka kutsuu jäsentäjää ajamaan XML-tiedostot tietokantaan tietyin aikavälein.

5.4 WWW-sivuston toteutus

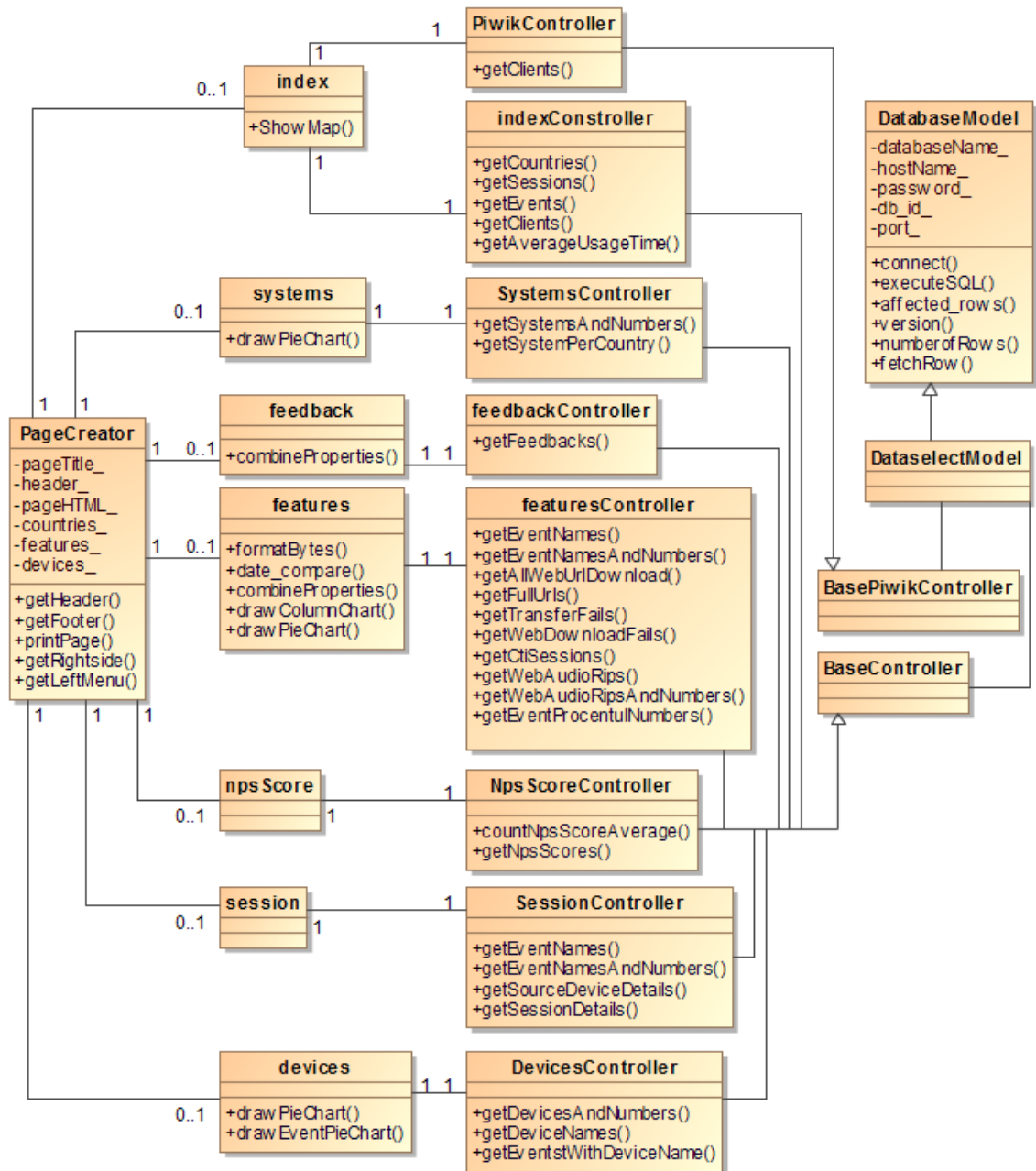
Tässä luvussa kuvataan PiceaHub-sovelluksen käyttötietoa sisältävien WWW-sivustojen toteutusta. Ensiksi luvussa kuvataan sivustojen yleistä arkkitehtuuria ja käyttöliittymää, jonka jälkeen kerrotaan tarkemmista toteutuksen yksityiskohdista ja tietoturvasta.

5.4.1 Arkkitehtuuri

WWW-sivustojen toteutuksen tavoitteena oli luoda yleiskäyttöinen ja helposti ylläpidettävä sivusto, jonka pohjaa voitaisiin hyödyntää myös tulevaisuudessa yrityksen tuotteissa ja niiden eri versioissa. Jotta sivustoista saataisiin yleiskäyttöinen ja ylläpidettävä, päätettiin sivustot toteuttaa MVC-arkkitehtuurin mukaisesti.

MVC-arkkitehtuurista on olemassa useita versioita ja myös tässä toteutuksessa on käytetty tämän mallin muunnelmää, jossa ohjain käyttää mallia tietokannan käsittelemiseen ja muuntaa tietokannasta saadut tiedot näkymälle sopivaan muotoon.

Näkymän tehtävä on näyttää ohjaimelta saatava tieto WWW-sivuston käyttäjille kirjastojen ja taulukoiden avulla. WWW-sivuston arkkitehtuuria on kuvattu kuvassa 5.2, jossa on käytetty UML-luokkakaavionaatiota.

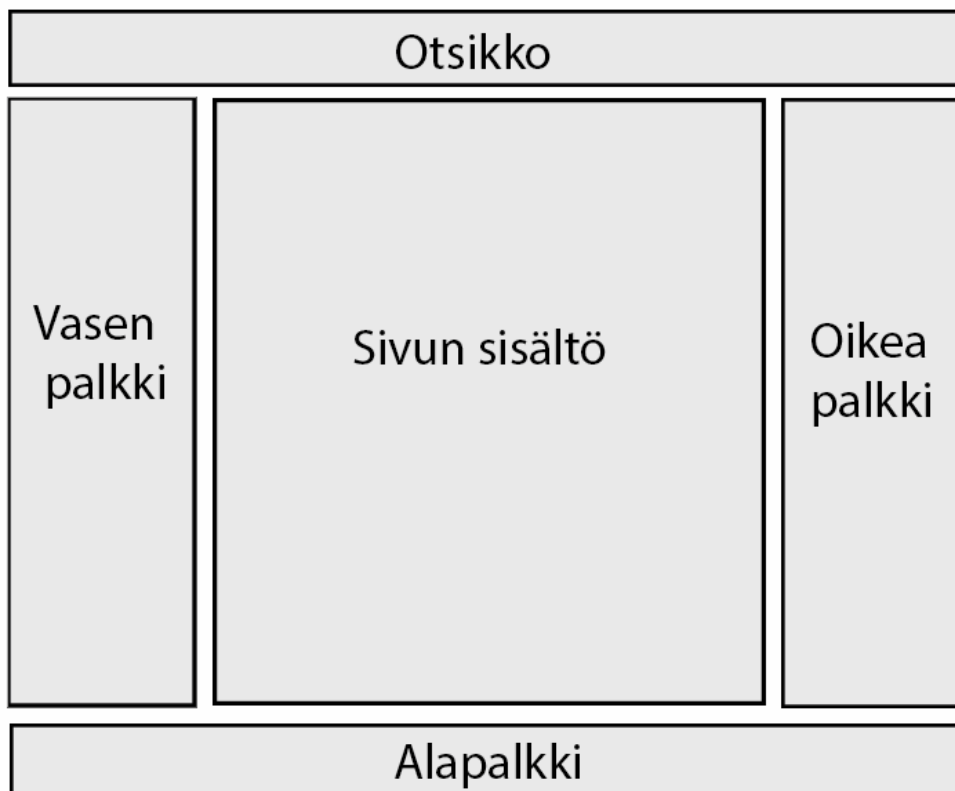


Kuva 5.2. WWW-sivuston arkkitehtuuria mallintava luokkakaavio

5.4.2 Käyttöliittymä

Käyttöliittymän toteutuksessa käytettiin visuaalisen suunnittelun periaatteita, joiden mukaan sovelluksen ulkoasun tulee olla yksinkertainen, selkeä, johdonmukainen ja miellyttävä [31]. Jotta näitä ominaisuuksia saatiin tuettua, erotettiin käyttöliittymä kuvan 5.3 mukaisesti viiteen osaan: otsikkopalkkiin, vasempaan palkkiin, sisältöalueeseen, oikeaan palkkiin ja alapalkkiin.

Otsikkopalkki toteutettiin näyttämään tiedot sovelluksesta, josta käyttötietoa kerätään sekä linkit muihin yritykseen liittyville WWW-sivuille. Toteutettua vasenta palkkia voidaan kutsua myös päänavigointipalkiksi, joka sisältää linkit kaikille sivuston alueille. Oikeanpuoleinen palkki sisältää tiettyyn yksittäiseen ja tarkasteltavaan sivuun liittyvää lisätietoa ja mahdollisia linkkejä tarkasteltavan sivun lisätietojen tarkasteluun. Sivun sisältöalue toteutettiin nimensä mukaisesti näyttämään sivulla sivun sisältö, kuten taulukot ja kuvaajat sekä valitun sivun nimi. Alapalkki toteutettiin WWW-sivustoille tyypilliseen tapaan esittämään tekijänoikeuksiin liittyvät tiedot.



Kuva 5.3. WWW-sivujen käyttöliittymä

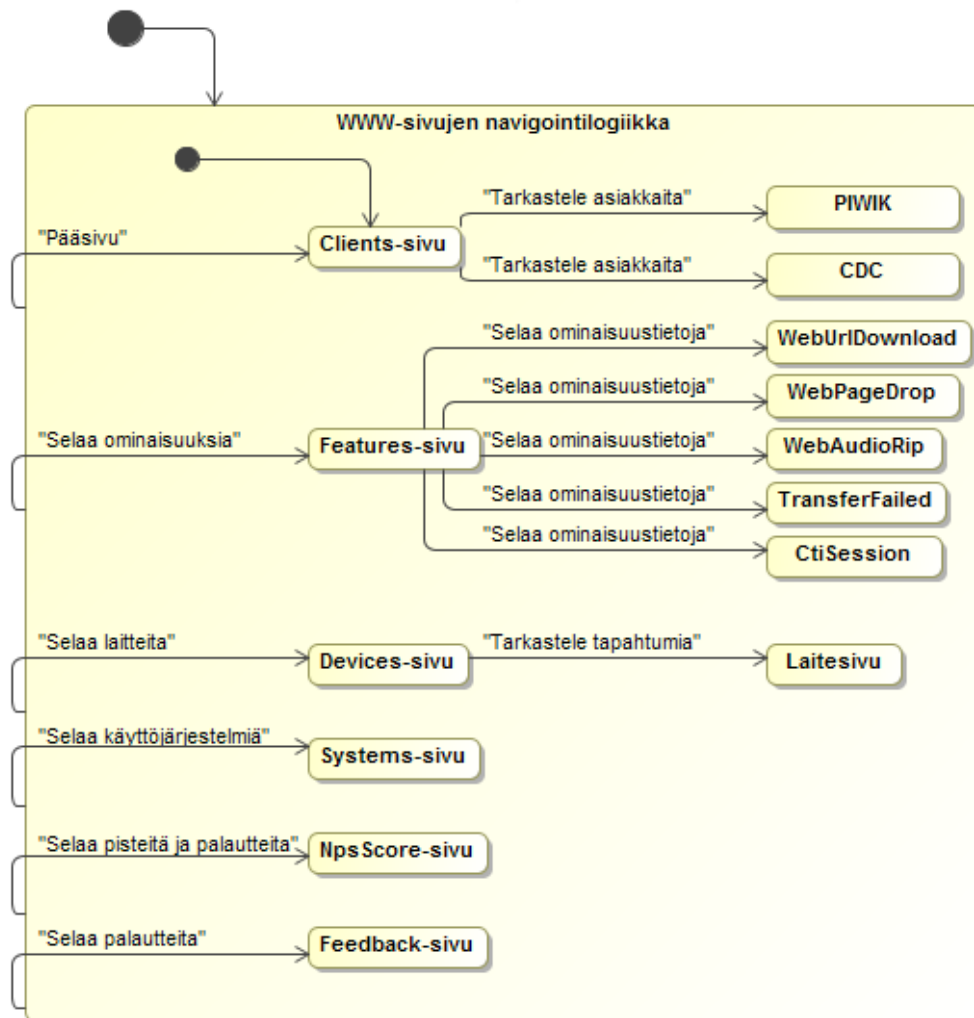
Käyttöliittymän toteutuksessa käytettiin sekä HTML-kuvailukieltä että CSS-tyylikieltä, jotta sisältö ja ulkoasu saatiin erotettua toisistaan. Erotuksen tarkoituksena on helpottaa WWW-sivuston ylläpitoa. WWW-sivuston toteutuksessa CSS-tyylikieltä käytettiin ulkoisen tyylitiedoston avulla. Tämä sen vuoksi, että dokumenttikohtaiset tyylitiedostot vaikeuttavat ulkoasun ylläpitoa ja elementtikohtaiset tyylimääritykset olisivat vieneet pohjan sisällön ja ulkoasun erottamiselta. Tyylitiedoston käyttö mahdollistaa myös sen, että sivustojen ulkoasua voidaan myöhemmin muokata yhdestä paikasta käsin ja suurten muutosten tekeminen yhdestä paikasta käsin on helpompaa.

Käyttöliittymän toteutuksessa käytettiin myös JavaScriptiä ja Google Chart -työkalua käyttötietoa sisältävien kuvaajien piirtämiseen. Google Chart on työkalu, jonka avulla

voidaan luoda erilaisia kuvaajia ja sisällyttää ne WWW-sivustoille. Diagrammien esittämiseen WWW-sivustoilla käytettiin Google Chart -työkalun tarjoamaa kirjastoja, kuten Google Ajax ja Google Visualization. Näistä kirjastoista Google Ajax -kirjasto tulee ajaa ensin, koska sen avulla mahdollistetaan Google Visualization -kirjaston ajaminen. Google Visualization -kirjasto puolestaan sisältää tärkeimmät metodit ja luokat, joita tarvittiin WWW-sivustolla näytettävien kuvaajien luomisessa ja käsittelemisessä.

5.4.3 Sovelluslogiikka

Kuvan 5.2 mukaisesti WWW-sivujen arkkitehtuuri jaettiin kuuteen sivuun, jotka toteutettiin MVC -mallin (Model-View-Controller) mukaisesti mallien, näkymien ja ohjainten avulla. Toteutuksessa käytettiin myös globaalia navigointijärjestelmää, johon on sisällytetty navigointi kaikille sivuille ja toimintoihin käyttäjän sijainnista riippumatta. WWW-sivujen tarkempi navigointilogiikka on kuvattu kuvassa 5.4.



Kuva 5.4. WWW-sivujen navigointilogiikka

Jokaiseen toteutettuun sivuun liittyy vähintään yksi ohjain. Lisäksi jokainen ohjain käyttää vain yhtä mallia, joka sisältää kaiken tietokannan kyselyihin liittyvän toiminnallisuuden. Ohjain puolestaan välittää mallista saadut tiedon näkymälle, joka näyttää yhden sivun ja tietokannasta saadut tiedot WWW-sivuston käyttäjälle PageCreator-luokan avulla.

PageCreator-luokka on suunniteltu huolehtimaan kaikkien näkymien sivujen luomisesta. Sivujen luomiseen kuuluvat sivun eri osa-alueiden, kuten alapalkin, yläpalkin, vasemman palkin, oikean palkin ja varsinaisen sisällön tietojen yhdistäminen yhdeksi HTML-sivuksi kuvan 5.3 mukaisesti. Joissain näkymissä luokalle välitetään myös tietoja, kuten käyttäjien maat, jolloin niitä voidaan luokan jäsenmuuttajien avulla näyttää sivun oikeassa tai vasemmassa palkissa.

WWW-sivustoissa tietokantayhteyden luominen sovelluksen ja tietokannan välille toteutettiin käyttämällä apuna XML-tiedon jäsentäjässä luotua tietokanta-apuluokkaa. Tästä apuluokasta luotiin WWW-sivustoja varten uusi versio DataselectModel -luokka, jonka tehtävänä on hakea tietokannasta WWW-sivustoilla tarvittavia tietoja ja käyttää luvussa 5.3 kuvattua Mysqli-laajennusta kyselyiden suorittamiseen.

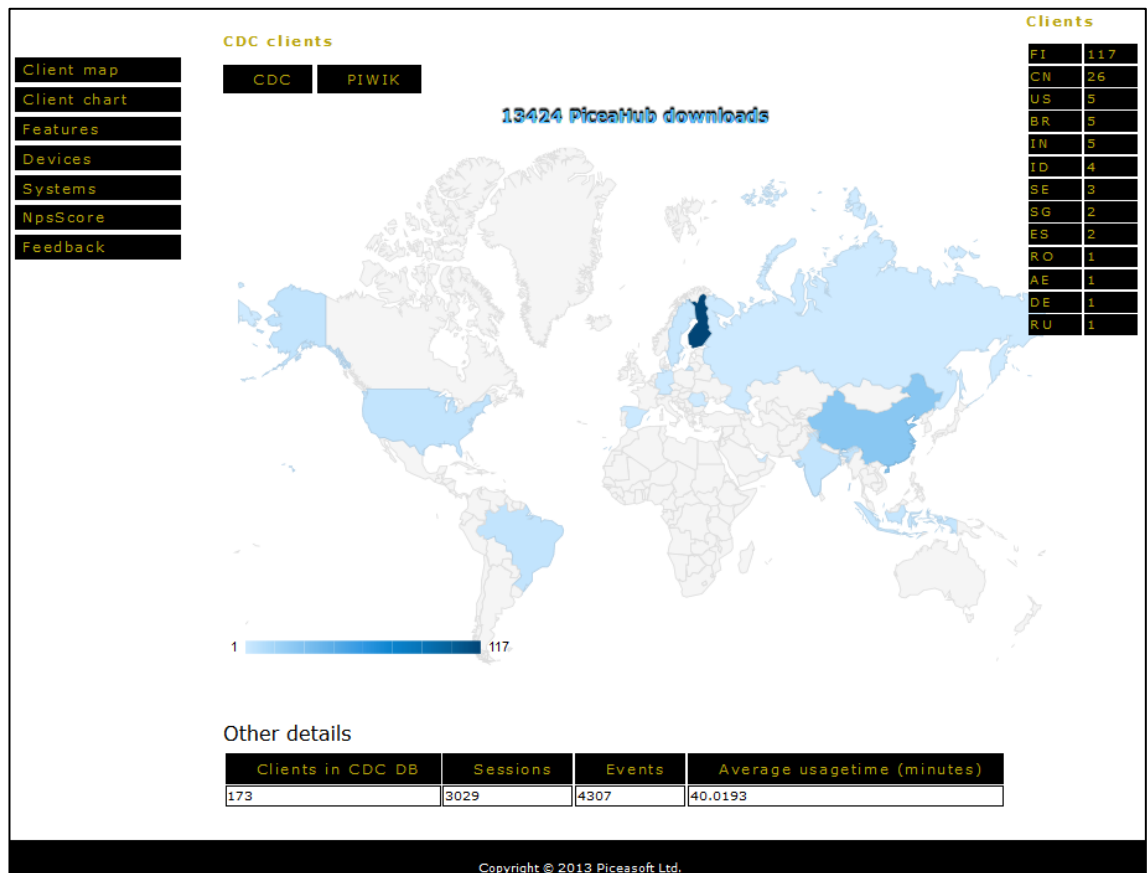
Clients-sivu

Clients-sivulla näytetään PiceaHub-sovelluksen kaksi asiakasryhmää. Toinen asiakasryhmistä on PiceaHub-sovelluksen ladanneet asiakkaat ja toinen on asiakkaat, joilta varsinaista käyttötietoa saadaan kerättyä. Nämä kaksi asiakasryhmää on erotettuja toisistaan, koska kaikilta asiakkailta ei saada kerättyä käyttötietoa ja käyttötiedon lähettäminen on sovelluksessa vapaaehtoista. Asiakkaiden määrän lisäksi sivulla näytetään asiakkaiden maatiedot sekä taulukoissa että kahden Google Chartin Geochart -kuvaajan avulla. Geochart-kuvaaja sisältää kaikki maapallon maat, jossa maiden värien tummuusaste kuvaa käyttäjien määrää suhteessa muihin maihin. Siirtämällä hiiren kursoria maan päälle, kartta näyttää puhekuplassa asiakkaiden määrän numeroina. Käyttäjämäärien lisäksi Clients-sivulla näytetään taulukossa sovellusten käyttökerrat, tapahtumien eli ominaisuuksien määrän ja sovelluksen keskimääräinen käyttöaika.

Clients-sivuston toteutus eroaa muista sivustoista siinä, että se käyttää kahta ohjainta ja kahta tietokantaa tiedon käsittelyyn. Toinen ohjaimista käyttää varsinaista käyttötietoa sisältävää tietokantaa ja toinen PIWIK-tietokantaa. PIWIK on yrityksen käytössä oleva ohjelmisto, jonka avulla saadaan selville muun muassa PiceaHub-sovelluksen latausten määrä. PIWIK tietokanta sijaitsee samassa MariaDB-tietokannan hallintajärjestelmässä, jossa myös käyttötietoa sisältävät tietokannat sijaitsevat.

Clients-sivuston toteutuksessa molemmat ohjaimet käyttävät mallia tiedon hakemiseen tietokannasta ja välittävät saadut tiedot näkymälle. Ohjainten avulla sovelluksessa kutsutaan mallia, joka puolestaan hakee tietokannoista asiakkaiden maat, asiakkaiden

lukumäärät, istunnot, tapahtumat ja sovelluksen keskimääräisen käyttöajan. Näkymän tehtävänä on näyttää ohjaimelta saatavat tiedot WWW-sivuston käyttäjille taulukkoja ja Google Chartin karttakuvaajia käyttämällä. Kuvassa 5.5 on esitettyä Clients-sivu.

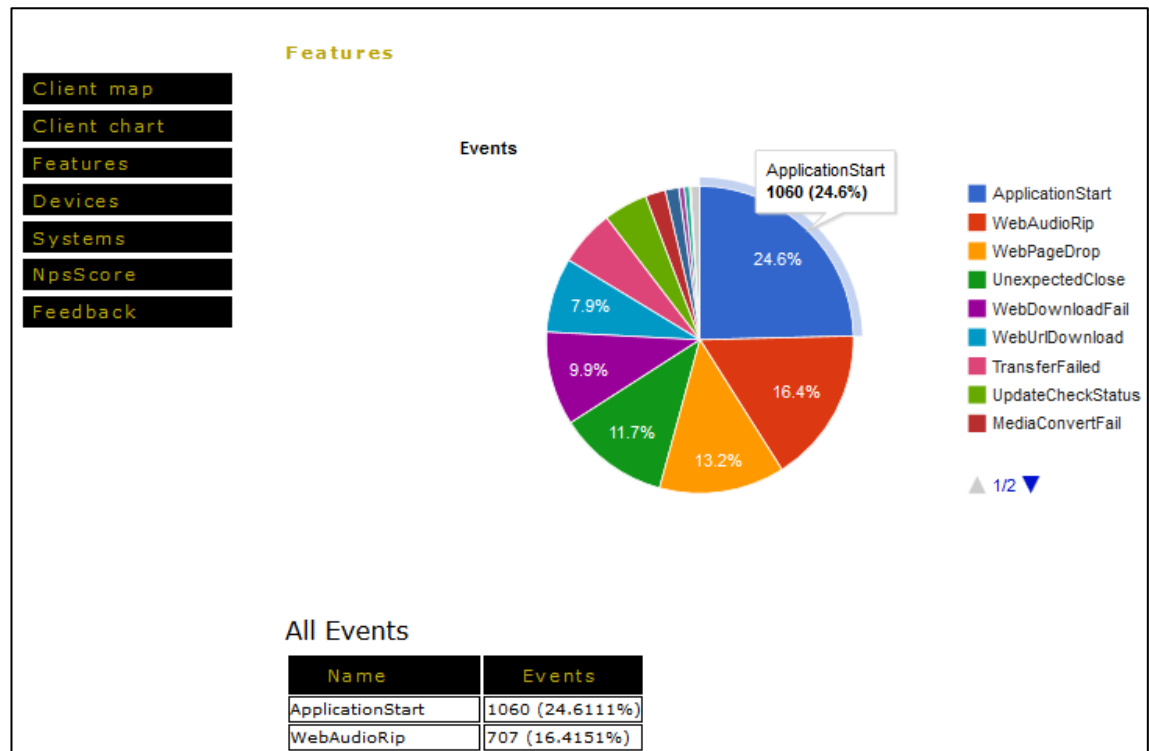


Kuva 5.5. PiceaHubin demoversion WWW-sivujen clients-sivu

Features-sivu

Features-sivu sisältää PiceaHub-sovelluksen asiakkailta saatujen tapahtumien eli sovelluksen ominaisuuksien tietoja. Ominaisuuksista on nähtävissä ominaisuuden nimi ja sekä ominaisuuden että niiden prosentuaaliset osuudet kokonaismäärästä. Ominaisuudet näytetään WWW-sivuston käyttäjille näkymän luomassa taulukossa ja Google Chartin ympyrädiagrammin avulla. Tämä sivun tarkoitus on näyttää kehittäjille mitä PiceaHub-sovelluksen käyttäjät käyttävät ja miten niitä käytetään. Lisäksi sivun avulla nähdään, tapahtuuko sovelluksen käytössä poikkeavuuksia tai virhetilanteita, jotta sovelluskehittäjät voivat ne helpommin korjata.

Sivulla oleva oikea palkki sisältää jokaisen ominaisuuden linkin, joiden avulla päästään tutkimaan tarkemmin ominaisuuksiin liittyviä tietoja. Varsinainen linkki on toteutettu HTTP:n GET-metodia käyttämällä, jossa ominaisuuden tyyppi välitetään. Näkymä sisältää toiminnallisuuden, jonka hakee GET-pyynnöstä ominaisuuden tyyppin ja näyttää sitä vastaavat näkymät WWW-sivuston käyttäjälle. Kuvassa 5.6 on esitettyä Features-sivu.



Kuva 5.6. PiceaHubin demoversion WWW-sivujen Features-sivu

Alla olevissa kappaleissa esitetään Features-sivulla näkyviä ominaisuuksia, joiden avulla sivulla näytetään miten ja mitä PiceaHub-sovelluksen ominaisuuksia käyttäjät ovat käyttäneet.

WebUrlDownload-ominaisuus

Tämä ominaisuuden avulla Features-sivulla saadaan tieto PiceaHub-sovelluksen käyttäjien lataamista videoista. Sivustolla näytetään, mistä eri isäntäosoitteista PiceaHub-sovellusta voidaan käyttää ja miten käyttäjät lataavat videoita, jotta PiceaHubin tuotekehitys saadaan keskitettyä oikeille alueille.

Tietojen näyttäminen WWW-sivustolla toteutettiin käyttämällä ohjainta ja mallia tiedon välittämiseen näkymälle, joka puolestaan näyttää saadut tiedot käyttäjille taulukoissa. Taulukossa näytettäviin tietoihin kuuluvat isäntäosoitteet sekä ladattujen tiedostojen tyyppi, nimi ja operaatiotunnus. Näkymän tehtäviin kuuluu myös suosituimpien isäntäosoitteiden näyttäminen Google Chartin pylväskuvaajan avulla.

WebPageDrop-ominaisuus

WebPageDrop -ominaisuuden avulla saadaan selville suosituimmat URL-osoitteet, jossa PiceaHub-sovellusta on käytetty. Suosituimmat URL-osoitteet ja niiden katselukerrat näytetään käyttäjälle näkymässä taulukon ja Google Chartin kolumni -kuvaajan avulla. Taulukossa oleva URL-osoite on suora linkki videoihin.

WebAudioRip-ominaisuus

WebAudioRip on PiceaHubin ominaisuus, joka syntyy käyttäjän muuntaessa esimerkiksi Youtube-palvelussa olevaa videota mp3-formaattiin. Sivustossa tämän ominaisuuden avulla saadaan selville suosituimmat osoitteet, jossa sovellusta ja kyseistä ominaisuutta on käytetty. Osoitteet esitetään sekä taulukossa että kolumnikuvaajan avulla.

TransferFailed-ominaisuus

Tämän ominaisuuden avulla saadaan selville mahdollisia syitä ja tietoja, jotka ovat johtaneet PiceaHub-sovelluksessa tapahtuvan tiedonsiirrossa epäonnistumiseen. Sivulla näytettävä tieto auttaa PiceaHub-sovelluksen kehittäjiä paikallistamaan virheitä ja parantamaan sovellusta. Tiedoston siirron epäonnistumiseen voi vaikuttaa useita erilaisia tekijöitä ja sen vuoksi tässä ominaisuudessa pyritään näyttämään mahdollisimman paljon erilaisia tietoja, jotka voivat johtaa tiedoston siirron epäonnistumiseen.

Tiedoston siirrossa tapahtuneet epäonnistumiset näytetään näkymän avulla taulukossa, joka sisältää muun muassa siirrossa epäonnistuneiden tiedostojen nimet, virhekoodit, päivämäärät sekä linkit tarkempiin istuntotietoihin. Tarkemmat istuntotiedot puolestaan sisältävät asiakkaan yksilöivän tunnuksen, käyttöjärjestelmän, käytetyn tuotteen tunnuksen, istunnon aloitusajan, istunnon lopetusajan, tuotteen version sekä tarkemmat laitetiedot, kuten mallin koodin ja malli nimen.

CtiSession-ominaisuus

CtiSession-ominaisuuden avulla saadaan selville Internet-ominaisuuden käyttöön liittyvää tietoa. Tiedon avulla saadaan selville muun muassa kuinka usein ja kauan Internet-ominaisuutta keskimäärin käytetään. Internet-ominaisuuden käyttökertojen ja yhteyden keston lisäksi sivulla näytetään lähetetyn tiedonmäärän, vastaanotetun tiedonmäärän sekä operaattorin tunnus. Tämän ominaisuuden tarkoitus on selvittää, että kyseinen ominaisuus toimii toivotulla tavalla.

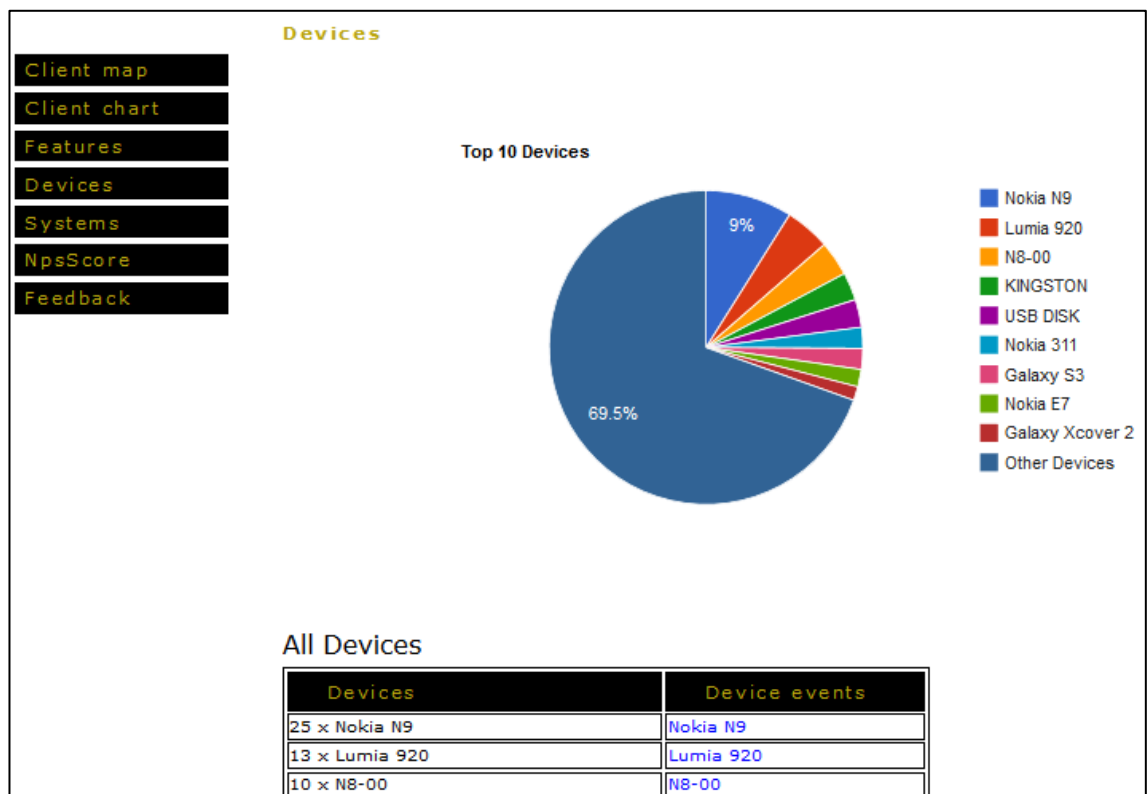
Devices-sivu

Devices-sivu sisältää kaikki laitteet, jolla PiceaHub-sovellusta on käytetty. Sivulla näytetään laitteiden määrät taulukossa ja jokaista laitetta voidaan tarkastella taulukon sisältämien linkkien avulla. Yksittäistä laitetyyppiä voidaan sivulla tarkastella ja tarkasteltavasti laitetypistä saadaan selville, mitä ominaisuuksia valitulla laitetypillä on käytetty.

Tämän sivun toteutuksessa käytettiin omaa ohjainta, joka kutsuu mallia tietokannassa olevien tietojen hakemiseen ja välittää tiedot näkymille. Näkymä esittää tietokannasta saadut laitteet ja niiden määrät sekä taulukon että Google Chartin ympyrädiagrammin avulla. Ympyrädiagrammi sisältää kymmenen suosituinta laitetta ja niiden

prosentuaaliset osuudet laitteiden kokonaismäärästä. Suosituimpien laitteiden lisäksi kuvaaja sisältää muiden laitteiden yhteenlasketun määrän.

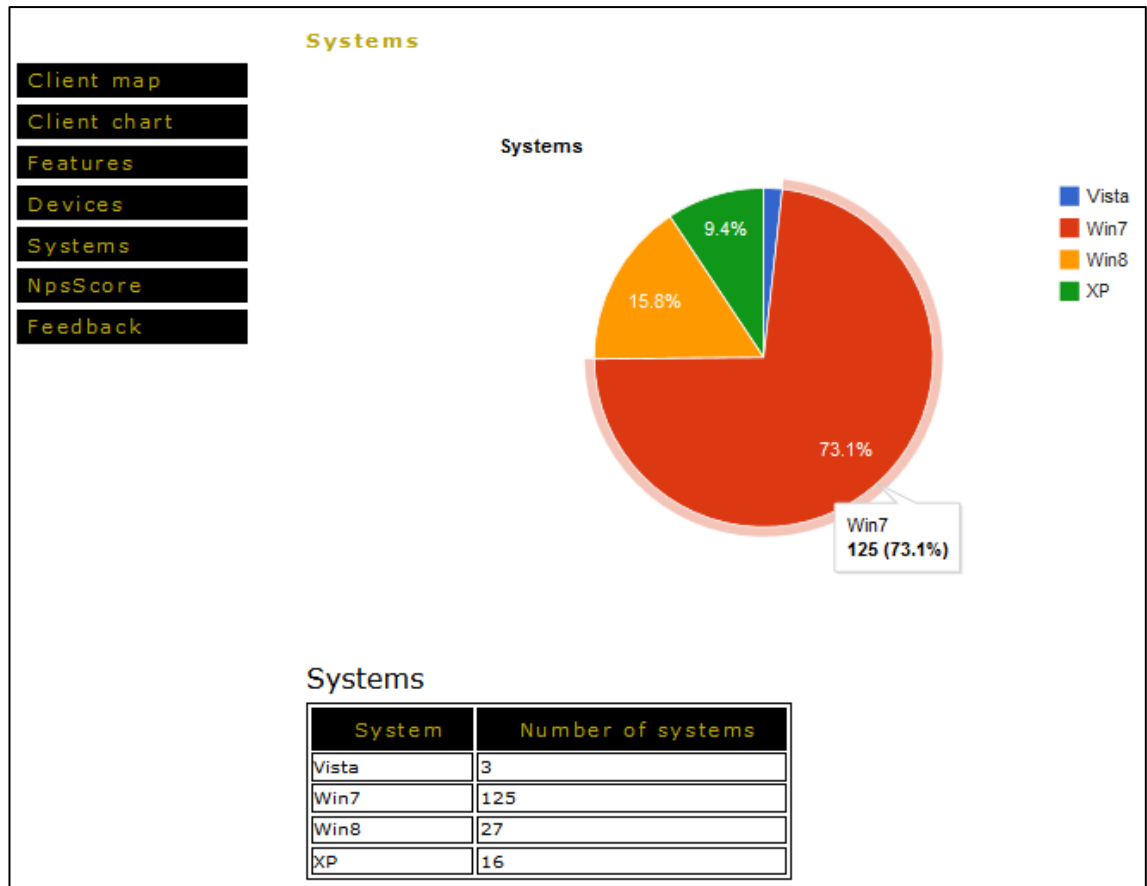
Taulukossa olevia laitteita voidaan sivustolla tarkastella tarkemmin niiden sisältämien linkkien avulla. Linkit toimivat HTTP GET-metodin avulla, jossa linkkiä painettaessa lähetetään laitetiedon sisältävä GET -pyyntö, joka vastaanotetaan ja tunnistetaan näkymässä. Näkymä näyttää tietyltä laitteen tapahtumat ja niiden lukumäärät Features-sivun mukaisesti sekä taulukossa että ympyrädiagrammissa. Kuvassa 5.7 on esitettynä Devices-sivu.



Kuva 5.7. PiceaHubin demoversion WWW-sivujen Devices-sivu

Systems-sivu

Systems-sivulla näytetään asiakkaiden käyttämät käyttöjärjestelmät ja niiden lukumäärät sekä taulukossa että ympyrädiagrammin avulla. Käyttöjärjestelmätietojen tarkoitus on tunnistaa mahdolliset PiceaHub-sovelluksen käyttöjärjestelmistä johtuvat virhetilanteet ja näyttää millaisia käyttöjärjestelmiä ylipäätään käytetään. Kuvassa 5.8 on esitettynä Systems-sivu.



Kuva 5.8. PiceaHubin demoversion WWW-sivujen Systems-sivu

NpsScore-sivu

NpsScore on sivu, jonka avulla nähdään käyttäjiltä saatu palaute sekä sanallisessa että pisteiden muodossa. Palautekysely ilmestyy PiceaHub-sovelluksen käyttäjälle, kun he ovat käyttäneet sovellusta tietyn aikaa. Palautekyselyssä sovellukselle annetaan tähtiä yhdestä viiteen sekä vapaaehtoinen sanallinen palaute. Yksittäisten palautteiden lisäksi sivustolla näytetään käyttäjien antamien pisteiden keskiarvo.

Käyttäjiltä saatu palaute näytetään aikajärjestyksessä sivulla olevassa taulukossa, jossa uusin palaute on ensin. Yksittäisen käyttäjän palautteessa näytetään lähetetyn palautteen numero, palautteen viesti, päivämäärä, annetut pisteet sekä maa, josta käyttäjä on kotoisin. Kuvassa 5.9 on esitettyä NpsScore-sivu.

NpsScore

Average points = 4.6383 (NpsScore = 81.914893617021)

#	Message	Time	Points 1-5	Country
47	Great	2013-04-15 21:24:01	5	FI
46	ITS WORKING NORMALY AND I LIKE IT	2013-04-13 01:53:09	5	RU

Kuva 5.9. PiceaHubin demoversion WWW-sivujen NpsScore-sivu

Feedback-sivu

Feedback-sivulla näytetään muu käyttäjiltä saatu palaute, joka ei liity NpsScoreen. Muulla palautteella tarkoitetaan palautetta, jonka käyttäjä voi milloin tahansa lähettää ohjelmaa käyttämällä. Palautteen avulla sovelluskehittäjät saavat käyttäjiltä vapaamuotoista palautetta, kuten kehitysideoita ja tarkempia kuvauksia sovelluksen mahdollisista ongelmista. Kuvassa 5.10 on esitettyä feedback-sivu.

Feedback				
Timestamp	Subject	Feedback message	Email	Country
2013-04-19 14:29:15	Testing new demo release	Version 0.0.565.2578		FI
2013-04-16 19:03:45	Testi kotoa	Tää on versio 0.0.548.2386		FI
2013-04-09 20:36:57	Uusin paketti (548) asennettu	Feedbakkia		FI
2013-04-09 15:53:29	Testing new beta candidate	Checking if this message pops up to the demo feedback pages!		FI

Kuva 5.10. PiceaHub demoversion WWW-sivujen feedback-sivu

5.4.4 Tietoturvallisuus

SQL-injektio on tietokannoille tyypillinen tietoturvauhka, jolla tarkoitetaan sovelluksessa olevien SQL-kyselyjen manipulointia hyökkäyksen suorittajan eduksi. Kyselyjen manipulointi suoritetaan sovelluksessa tyypillisesti lomakkeen tai sivun osoitteen avulla. SQL-injektio voi vaurioittaa tietokantaa monella tavalla, kuten käsittelemällä tietokantaa luvattomasti tai hakemalla arkaluontoista tietoa. Joissain tapauksissa SQL-injektioilla voidaan suorittaa palvelunestohyökkäys, jossa käyttäjiltä estetään pääsy verkossa olevaan sovellukseen kuormittamalla puskuria tai varaamalla resursseja.

Tässä toteutuksessa tietokannan SQL-injektoiden torjumiseksi suoritettiin käyttäjien syötteiden tarkistuksia, jossa syötteet tehtiin vaarattomaksi. Erityisesti GET-pyyntöjen sisältämät arvot muokattiin siten, että ne eivät sisällä suoritettavaa SQL-kieltä.

6 ARVIOINTI

Tässä luvussa arvioidaan toteutettua tietokantaan, PHP-sovelluksia ja WWW-sivustoja. Arvioinnin tarkoitus on pohtia mahdollisia toteutuksen hyviä ja huonoja puolia sekä tulevaisuuden tuomia mahdollisia haasteita.

6.1 Tietokannan arviointi

Tietokannan suunnittelun ja toteutuksen tavoitteena oli yleiskäyttöinen ja suorituskykyinen tietokanta. Yleiskäyttöisellä tietokannalla tarkoitetaan tässä toteutuksessa tietokantaa, jonka rakennetta voidaan hyödyntää myös yrityksen muissa ja tulevaisuudessa tuotteissa. Suorituskyvyllä puolestaan tarkoitetaan, että tietokanta toimii tehokkaasti myös suurilla tietomäärillä ja isolla raskauskuormalla. Yleisesti katsottuna tietokannan suorituskykyä tai yleiskäyttöisyyttä lisättäessä toinen ominaisuuksista heikkenee.

6.1.1 Yleiskäyttöisyys

Tietokannan yleiskäyttöisyys pyrittiin tässä toteutuksessa varmistamaan toteuttamalla tietokannan rakenne sellaiseksi, että se sisältää sovelluksille mahdollisimman paljon yhteisiä tietokokonaisuuksia ja tauluja. Lisäksi rakenne pyrittiin tekemään sellaiseksi, että sovellusten kehittyessä ja uusien tapahtumatyyppien syntyessä, voidaan ne tallentaa tietokantaan.

Uusien tapahtumatyyppien tallentaminen tietokantaan mahdollistettiin tapahtumaan liittyvän ominaisuus-taulun avulla. Tietokannassa tapahtuma rivi koostuu n-määrästä ominaisuus-rivejä. Ominaisuus-taulu toimii periaatteella, jossa yhdessä sen kentistä on tieto, mistä sen taulun kentistä (merkkijono, aikaleima tai kokonaisluku) varsinainen tieto löytyy. Tämä toteutusratkaisu mahdollistaa sen, että myös tulevaisuudessa kaikki tapahtumiin liittyvä tieto voidaan muodostaa ominaisuus -taulussa olevien tietotyyppien avulla. Toisaalta ominaisuus-taulu asettaa haasteen tietokannan suorituskyvylle, sillä haettaessa esimerkiksi tietyn tapahtuma-rivin tiedot, joudutaan ne yhdistämään PHP-ohjelmassa useasta ominaisuus-rivistä. Tämän tapainen ominaisuuksien yhdistely ei ole paras tapa tietokannan suorituskyvyn kannalta, mutta tässä toteutuksessa yleiskäyttöisyys katsottiin suorituskykyä tärkeämmäksi.

6.1.2 Suorituskyky

Tietokannan suorituskyvyn maksimoimiseksi pyrittiin tietokannan mallista tekemään yksinkertainen ja johdonmukainen. Ylläpidettävyyden helpottamiseksi tietokannalle suoritettiin myös normalisointi, joka puolestaan asettaa haasteen tietokannan suorituskyvylle, sillä tietoa joudutaan hakemaan usean taulun tietoja yhdistelemällä.

Suorituskyvyn parantamiseksi tietokannassa käytettiin kuvaustauluja, jotka sisältävät metatietoa eli tietoa tiedosta. Kuvaustaulut toimivat periaatteella, että niihin tallennetaan yksilöiviä tunnuksia, joita käytetään tietokannassa haettaessa tiettyä istunto-, tapahtumatai ominaisuustyyppiä. Yksilöivät tunnukset saadaan suoraan PiceaHub-sovelluksesta ja ne esitetään tauluissa merkkijonoina. Tietokantaan kohdistuvissa kyselyissä merkkijonovertailut ovat kokonaislukuvertailuja hitaampia, joten kuvaustaulussa merkkijonotunnukset ovat yhdistetty tiettyyn kokonaislukuarvoon. Kuvaustaulun ideana on nopeuttaa istuntoihin, tapahtumiin ja ominaisuuksiin kohdistuvia hakuja, sillä niiden tyypit voidaan hakea kuvaustaulun kokonaislukuarvon avulla. Lisäksi kuvaustaulut mahdollistavat koosteiden tekemisen WWW-sivustolla, sillä kuvaustaulut sisältävät tunnuksia vastaavat selitykset.

Suorituskyvyn parantamiseksi eräänä toteutusvaihtoehtona tietokannan rakenteeseen mietittiin MariaDB:n ominaisuutta, jossa tietokannan tauluille voidaan asettaa dynaamisia sarakkeita. Dynaamisella sarakkeella tarkoitetaan siis tietokannassa taulun saraketta, jossa sarakkeen määrät ja tietotyypit voivat vaihdella eri riveillä. Dynaamisia sarakkeita käytetään tyypillisesti tilanteissa, jossa tiedolla on suuri määrä erilaisia attribuutteja, joita ei välttämättä tiedetä etukäteen. Tietokannan toteutuksessa dynaamisia sarakkeita olisi voitu käyttää tapahtuma-tauluissa, jolloin ominaisuus-tauluja ei olisi tarvinnut käyttää. Tämä toteutusratkaisu olisi auttanut suorituskyvyn parantamisessa, koska tapahtumien tietoja ei tarvitsisi kerätä useasta ominaisuudesta. Dynaamisia sarakkeita ei toteutuksessa kuitenkaan käytetty, sillä MariaDB:n dokumentaatio suosittelee dynaamisia sarakkeita käytettäväksi ainoastaan silloin, kun normaaleita sarakkeita ei ole mahdollista käyttää. Tulevaisuudessa dynaamisia sarakkeita ja niiden vaikutusta suorituskykyyn voidaan tutkia paremmin, jos nykyinen toteutusratkaisu koetaan suorituskyvyn kannalta liian hitaaksi. Tällä hetkellä MariaDB:n vakaa versio ei tue hakemistojen luontia dynaamisille sarakkeille, mutta tulevissa versioissa myös tämä toiminnallisuus on mahdollista.

6.1.3 Skaalautuvuus

Tietokannan hallintajärjestelmän eräänä vaatimuksena on hyvä skaalautuvuus, jolla mahdollistetaan suorituskyvyn säilyminen tiedon määrän kasvaessa tietokannassa. Tyypillisesti tietokannan skaalautuvuutta voidaan parantaa suorittamalla sekä horisontaalisesta että vertikaalisesta skaalautumista. Vertikaalisella skaalautuvuudella tarkoitetaan tietokannan suorituskyvyn kasvua päivittämällä olemassa olevaa palvelinta

esimerkiksi ostamalla siihen lisää muistia tai prosessoriytimiä. Horisontaalisella skaalautuvuudella puolestaan tarkoitetaan suorituskyvyn parantamista jakamalla suorituskuormaa usealle tietokoneelle. Merkittävimmät erot näiden kahden skaalautumistekniikan välillä ovat, että vertikaalista skaalautumisella voidaan saavuttaa 10–30-kertainen suorituskyvyn kasvu, kun taas horisontaalisesti skaalautumalla tarjoaa potentiaalisesti rajattoman suorituskyvyn kasvun [32].

Nykyinen yhden tietokantapalvelimen toteutusratkaisu tulee riittämään todennäköisesti pitkään, mutta tulevaisuuden kannalta on hyvä varautua tilanteeseen, jossa suorituskyky ja tallennuskapasiteetti eivät enää riitä tiedon tallentamiseen. Vertikaalisesti skaalautumista voidaan suorittaa vain tiettyyn pisteeseen asti, joten tulevaisuuden kannalta on hyvä varautua horisontaalisesti skaalautumiseen.

Tässä toteutuksessa horisontaalista skaalautumista voidaan suorittaa kahdella tavalla: replikoimalla ja osittamalla. Näistä menetelmistä replikointi on yksinkertaisin tapa, siinä sama tieto jaetaan usealle palvelimelle. Se soveltuu erityisesti lukupainotteisiin sovelluksiin. Tulevaisuudessa tietokantaan kohdistuvaa suorituskuormaa on vaikea arvioida, mutta se tulee todennäköisesti olemaan kirjoituspainotteista, koska WWW-sovelluksen käyttäjämäärät ovat vähäisiä. Tämän vuoksi tulevaisuudessa merkittävä tekniikka skaalautuvuuden parantamiseksi on osittaminen. Tässä toteutuksessa osittaminen on mahdollistettu toteuttamalla tietokannan rakenne mahdollisimman yksinkertaiseksi, jossa on tauluja ja niiden välisiä riippuvuuksia on vähän. Yksinkertainen tietokannan rakenne helpottaa huomattavasti tiedon jakamista eri palvelimilla tai kovalevyillä sijaitseviin tietokantoihin, sillä tietokokonaisuudet ovat helpommin erotettavissa.

Noodilla tarkoitetaan MariaDB:n arkkitehtuurissa funktionaalista yksikköä, kuten esimerkiksi yhtä tietokantapalvelinta. Osittamisessa suorituskuormaa jaetaan ja tallennetaan usealla noodille jakamalla tietoa pieniin osiin. Ositettavan tiedon jakamisesta tulevaisuudessa vastaa edellä mainittu XML-jäsentäjä, jonka on tiedettävä periaate tiedon jakamiseen ja osattava ottaa yhteys oikeaan tietokantaan. Tyypillisesti asiakaskeisissä järjestelmissä tietoa jaetaan noodien välillä maa- tai mannerkohtaisesti. Tätä toteutusratkaisua tullaan tarpeen vaatiessa todennäköisesti myös tässä järjestelmässä hyödyntämään.

6.2 PHP-sovellusten arviointi

XML-tiedon vastaanottajan, XML-jäsentäjän ja WWW-sivustojen toteutuksessa käytettiin PHP-ohjelmointikielellä, koska sen mahdollisti kaikkien sovellusten toteutuksen samalla ohjelmointikielellä. Valintaan vaikutti myös se, että ohjelmat toteutettiin palvelimelle ja PHP on palvelimen puolella suoritettava skriptikieli.

6.2.1 XML-tiedon vastaanottaja

XML-tiedon vastaanottajan toteutus on koodimäärältään lyhyt ja selkeä, se on myös sen parhaimpia puolia. HTTPS POST -pyyntöjen vastaanottamisen ja XML-tiedostojen luonnin lisäksi sovelluksen tehtävänä on hakea vastaanotetusta pyynnöstä IP-osoite, jolle sovellus hakee maatiedon ulkoista palvelua käyttämällä ja tallentaa saadun tiedon XML-tietoon. Tämän toteutusratkaisun huonona puolena on, että maatiedon oikeellisuudesta on vaikea saada täyttä varmuutta.

PiceaHub-sovelluksen käyttäjän maatiedon hakemiseen mietittiin toteutusvaihtoehtoa, jossa maatieta saataisiin sovelluksen käyttäjän käyttöjärjestelmästä. Käyttöjärjestelmästä saatava aikavyöhykkeen maatieta osoittautui kuitenkin vaikeasti saatavaksi ja käyttöjärjestelmän kieli liian epätarkaksi toteutukseen, joten maatiedon hakeminen ulkoisista palveluista todettiin parhaimmaksi ja tarkimmaksi toteutusvaihtoehdoksi. Maatiedon tarkkuutta voitaisiin jatkossa mahdollisesti parantaa käyttämällä useampia palveluita maatiedon hakemiseen IP-osoitteen perusteella, joista tämän jälkeen valittaisiin eniten hakuosumia saatu maa.

XML-tiedon vastaanottajan tietoturvallisuutta voitaisiin jatkossa parantaa tarkistussummien avulla, jossa PiceaHub-sovelluksesta lähetettävästä XML-tiedosta luotaisiin tarkistussumma, jonka oikeellisuus puolestaan tarkistettaisiin XML-tiedon vastaanottajassa. Tämän menetelmän avulla saataisiin varmistettua samalla myös lähetettävän tiedon eheys.

6.2.2 XML-tiedostojen jäsentäjä

XML-tiedostojen jäsentäjässä haetaan tietyin aikavälein XML-tiedostoja käsiteltäväksi ja jäsennetään niiden sisältämät tiedot tietokantaan. XML-tiedostoja jäsennettäessä pyritään aina ensimmäiseksi tunnistamaan käytettävä tietokanta, joka vaihtelee XML-tiedosta saatavan tuotteen tyyppin ja version mukaan. Tämä jäsentäjässä suoritettava tietokannan tunnistaminen on myös jäsentäjän heikkous, sillä XML-tiedossa ei ole tietoa käytettävästä tietokannasta, vaan ainoastaan tuotteen versio. Tuotteen versio puolestaan määrittää, mitä tietokantaa käytetään, jolloin vastaava tieto tulee olla myös jäsentäjässä, jotta tieto voidaan sijoittaa oikeaan tietokantaan. Tämä puolestaan aiheuttaa osittain sen, että sovellus on virheille herkkä. Esimerkiksi XML-tiedostoa jäsennettäessä, jonka versiota ei ole päivitetty jäsentäjään, jää sijaitsemaansa hakemistoon, kunnes tietokannan tiedot ovat saatu päivitettyä jäsentäjään. Tämä puolestaan aiheuttaa ongelman, että samaa tiedostoa saatetaan käsitellä useita kertoja jäsentäjässä turhaan.

Sovelluksen vikasietoisuutta ja ylläpidettävyyttä voitaisiin jatkossa parantaa muuttamalla sovelluksessa tietokantojen luominen ja käytettävän tietokannan valinta automaattiseksi. Tämä edellyttää sitä, että XML-tieto sisältää valmiina käytettävän

tietokannan tiedot. Tämä ratkaisu mahdollistaisi sen, että käytettäviä tietokantoja ei tarvitsisi erikseen muuttaa jäsentäjään.

XML-tiedostojen jäsentäjän tietoturvallisuutta voitaisiin myös jatkossa parantaa tarkemmilla syötteiden tarkistuksilla. Tällä hetkellä sovelluksessa luotetaan, että jäsenennettävä tieto on laillista. Tähän ongelmaan on osittain varauduttu luvussa 5.3 kuvatuilla `prepare` -lausekkeilla. Jatkossa voitaisiin toteuttaa enemmän syötteiden tarkistuksia, jotta XML-tiedostojen kautta suoritettavat SQL-injektiot eivät olisi missään muodossa mahdollisia.

6.2.3 WWW-sivustot

WWW-sivustojen tarkoitus on näyttää tietokannassa oleva käyttötieto kehittäjien kannalta hyödyllisessä muodossa. Tämä mahdollistettiin MVC-arkkitehtuurin avulla, jossa käyttöliittymälogiikka erotettiin sovelluslogiikasta. MVC-arkkitehtuurin mahdollistaa erityisesti uusien näkymien nopean luomisen. Erityisesti näkymiä, joissa käytetään samaa tietoa kuin muissa näkymissä, on nopea toteuttaa. Lisäksi MVC-arkkitehtuuri yhdessä CSS-tyylitiedoston kanssa tekevät sivuista helposti ylläpidettävän, jossa ulkoasua voidaan muokata yhdestä tiedostosta käsin.

Tulevaisuudessa WWW-sivustojen yleiskäyttöisyyttä ja ylläpidettävyyttä voidaan edelleen parantaa toteuttamalla kaikille yrityksen tuotteiden WWW-sivuille yhteisiä luokkia ja funktioita. Tällä hetkellä WWW-sivujen tekeminen uusille tuotteille toteutetaan kopioimalla vanha WWW-sivustojen toteutus, joihin tämän jälkeen päivitetään uuden käytettävän tietokannan tiedot. Tämä aiheuttaa sen, että WWW-sivuille tehtävät muutokset joudutaan päivittämään melkein kaikkiin toteutettuihin sivuihin.

Hyvän tietoturvallisuuden takaamiseksi WWW-sivustot ovat tällä hetkellä sijoitettu yrityksen sisäverkkoon. Tulevaisuudessa yrityksen eri tuotteille joudutaan todennäköisesti sallimaan pääsy myös ulkoverkosta, jolloin tietoturvallisuutta joudutaan myös kehittämään paremmaksi. Tähän ongelmaan liittyen WWW-sivuilla tullaan tulevaisuudessa toteuttamaan autentikointi, jossa sivujen käyttäjiä pyritään tunnistamaan.

Tietokannan tietomäärän kasvaessa suureksi ja uusien WWW-sivujen ominaisuuksien myötä voi WWW-sivujen suorituskyky heikentyä. Tähän liittyen jatkossa suorituskykyä voidaan parantaa PHP:n laajennusten, kuten PHP-kiihdyttimen avulla, jotka ovat erityisesti suunniteltu PHP-kielillä kirjoitettujen ohjelmien suorituskyvyn parantamiseen [33].

7 YHTEENVETO

Diplomityön tavoitteena oli toteuttaa järjestelmä, joka mahdollistaa käyttötiedon keräämisen, tallentamisen ja analysoidun tiedon esittämisen loppukehittäjille. Käyttötiedon keräämisen tarkoitus oli auttaa sovelluskehittäjiä parantamaan olemassa olevaa sovellusta, korjaamaan sen ongelmia sekä suuntamaan tuotekehitys oikealle alueelle.

Valmiiseen järjestelmään ja tavoitteisiin päästiin useiden vaiheiden kautta. Työssä toteutettiin sovellukset, jotka vastaanottavat PiceaHub-sovelluksesta lähetettävää tietoa ja tallentavat vastaanotetun tiedon tietokantaan sekä näyttävät käyttötiedon analysoidussa muodossa WWW-sivuilla sovelluskehittäjille. Tietokannasta saatiin toteutettua yksinkertainen ja yleiskäyttöinen ratkaisu, jota voidaan hyödyntää myös yrityksen tulevien tuotteiden käyttötiedon keräämiseen. WWW-sivustojen MVC-arkkitehtuuri mahdollistaa niiden yleiskäyttöisyyden, jonka ansiosta sivustoja voidaan helposti monistaa uusille tuoteversioille ja tuotteille.

Toteutetun järjestelmän kehitys kuitenkin jatkuu edelleen ja sitä tullaan tulevaisuudessa kehittämään paljon. Erityisesti WWW-sivuihin tullaan lisäämään uutta toiminnallisuutta ja niiden tietoturvallisuutta parannetaan autentikoinnin avulla. Tulevaisuudessa jää myös nähtäväksi, kuinka paljon ja millä nopeudella tietoa tallennetaan järjestelmään ja miten sekä tietokanta että WWW-sivut suoriutuvat kyselyistä tietomäärän kasvaessa suureksi.

LÄHTEET

- [1] R, Atterer., Wnuk, M., A, Schmidt. Knowing the users every move: user activity tracking for website usability evaluation and implicit interaction, In Proceedings of the International Conference on World WideWeb. Association for Computing Machinery, 2006. p. 203.
- [2] Tiziana., M. Karusseit, M. Community Usage of the Online Conference Service: an Experience Report from three CS Conferences. Proc. I3E 2002. pp. 497-511.
- [3] Marciuska, S., Gencel, C., Abrahamsson, P. Exploring How Feature Usage Relates to Customer Perceived Value: A Case Study in a Startup Company. Free University of Bolzano-Bozen. 12 p.
- [4] Marciuska, S., Gencel, C., Abrahamsson, P. Feature Usage Diagram for Feature Reduction. Free University of Bolzano-Bozen. 15 p.
- [5] Laine, J. Django ja CodeIgniter. Kandidaatintyö. Tampere 2012. Tampereen teknillinen yliopisto. 32 s.
- [6] Koch, P. General introduction. [WWW]. [Viitattu 02.05.2013]. Saatavissa: <http://www.quirksmode.org/js/intro.html>
- [7] Korpela, J. Web-julkaisemisen opas - Javascript. [WWW]. [Viitattu 02.05.2013]. Saatavissa: <http://www.cs.tut.fi/~jkorpela/webjulk/3.2.html>
- [8] Ollikainen, V. Tiedonhallinnan perusteet - Relaatiomalli. [WWW]. [Viitattu 02.05.2013]. Saatavissa: <http://users.metropolia.fi/~olliv/Sovellusohjelmat/Materiaalit/Relaatiomalli.php>
- [9] Peltonen, J. OHJ-3300 Johdatus tietokantoihin - Relaatiotietokannat. Tampere 2009, Tampereen teknillinen yliopisto.
- [10] W3Schools. Introduction to SQL. [WWW]. [Viitattu 14.01.2013]. Saatavissa: http://www.w3schools.com/sql/sql_intro.asp
- [11] Teorey, Toby, J., Database Modeling & Design, The Fundamental Principles, Sixth Edition. United States Of America 2011, Addison-Wesley. 1200 p.
- [12] Tampereen teknillinen yliopisto. OHJ-5101 Web-ohjelmointi - Web-ohjelmointikehykset ja sivupohjamootorit. [WWW]. [Viitattu 02.05.2013]. Saatavissa: <http://www.cs.tut.fi/~seitti/media/pdfs/luento-3.pdf>

- [13] Koskinen, J. Ohjelmistoarkkitehtuurit 2012-2013 - Arkkitehtuurityylit1. [WWW]. [Viitattu 27.03.2013]. Saatavissa: <http://www.cs.tut.fi/~ohar/luennot/luennot2012/Ohar7%20Arkkitehtuurityylit1.pdf>
- [14] Koskinen, J. Protokollan toteutuksia, valmiita järjestelmiä. [WWW]. [Viitattu 27.03.2013]. Saatavissa: <http://www.cs.tut.fi/~8306000/pt.html>
- [15] If.fi. SSL-suojaus. [WWW]. [Viitattu 27.03.2013]. Saatavissa: <http://www.if.fi/web/fi/henkiloasiakkaat/ifkansio/pages/ssl-suojaus.aspx>
- [16] Forta, B. MariaDB Crash Course. United States of America 2011, Addison Wesley. 290 p.
- [17] Schwartz, B., Zaitsev, P., Tkachenko, V. High Performance MySQL, Third Edition. United States of America 2012, O'Reilly Media. 828 p.
- [18] Monty Program Ab. AskMonty knowledgebase. [WWW]. [Viitattu 14.01.2013]. Saatavissa: <https://kb.askmonty.org/en/>
- [19] Widenius, M. MariaDB or NoSQL. [WWW]. [Viitattu 02.05.2013]. Saatavissa <http://www.cs.tut.fi/tapahtumat/olio2012/Monty.pdf>
- [20] Riggs, S., Krosing, H. PostgreSQL 9 Administration Cookbook. Packt Publishing
- [21] Fowler, M., Sadalage, P. NoSQL Distilled - A Brief Guide to the Emerging World of Polyglot Persistence. United States of America 2012, Addison-Wesley. 192 p.
- [22] Cattell, R. Scalable SQL and NoSQL Data Stores. United States of America 2010, ACM Digital library. Volume 39 Issue 4, p. 12-27.
- [23] Bartholomew, D. SQL vs. NoSQL. [WWW]. [Viitattu 23.03.2013]. Saatavissa: <http://www.linuxjournal.com/article/10770>
- [24] Bartholomew, D. MariaDB vs. MySQL. [WWW]. [Viitattu 07.04.2013]. Saatavissa: <http://www.skysql.com/sites/default/files/whitepapers/mariadb-vs-mysql.pdf>
- [25] Tzanidakis, M. Replacing MySQL with MariaDB. [WWW]. [Viitattu 07.04.2013]. Saatavissa: <http://www.openlogic.com/wazi/bid/272031/Replacing-MySQL-with-MariaDB>

- [26] Networkworld. Six free databases with commercial-quality features. [WWW]. [Viitattu 14.01.2013]. Saatavissa: <http://www.networkworld.com/reviews/2012/120312-databases-test-263922.html?page=3>
- [27] Schroder, C. PostgreSQL vs. MySQL. [WWW]. [Viitattu 14.01.2013]. Saatavissa: <http://www.openlogic.com/wazi/bid/188125/PostgreSQL-vs-MySQL-Which-Is-the-Best-Open-Source-Database>
- [28] Peltonen, J. OHJ-3321 Tietokantojen suunnittelu - luento 1. Tampere 2013, Tampereen teknillinen yliopisto.
- [29] Microsoft. Tietokannan normalisoinnin perusteiden kuvaus. [WWW]. [Viitattu 02.05.2013]. Saatavissa: <http://support.microsoft.com/kb/283878/fi>
- [30] W3schools. HTTP Methods: Get vs. POST. [WWW]. [Viitattu 02.05.2013]. Saatavissa: http://www.w3schools.com/tags/ref_httpmethods.asp#gsc.tab=0w3post
- [31] Jumisko-Pyykkö, S., Järvelä, J. IHTE-2100 Introduction to User Interface Design 2011 - Lecture 3. Tampere 2013, Tampere University of Technology.
- [32] Paavolainen, S. Skaalautuvuuden ABC, osa 8. [WWW]. [Viitattu 02.05.2013]. Saatavissa: <http://blog.codento.com/2011/03/skaalautuvuuden-abc-osa-8-yksi-tietokanta-kaksi-tietokantaa-monta-tietokantaa/>
- [33] Jelastic. PHP accelerators. [WWW]. [Viitattu 02.05.2013]. Saatavissa: <http://jelastic.com/fi/docs/php-accelerators>