HAAGA-HELIA
ammattikorkeakoulu

Guild Charter - Kotisivupalvelu videopeliyhteisöille

Marko Hannila

Krista Wikström

Opinnäytetyö

Tietojenkäsittelyn Koulutusohjelma

7.6.2014

HAAGA-HELIA
ammattikorkeakoulu

Tiivistelmä

8.5.2014

Tietojenkäsittelynkoulutusohjelma

Tekijä tai tekijät Marko Hannila Krista Wikström	Ryhmä tai aloitusvuosi 2011
Opinnäytetyön nimi Guild Charter - Kotisivupalvelu videopeliyhteisölle	Sivu- ja liitesivumäärä 30 + 1
Ohjaaja Sirpa Marttila	

Projektin tavoitteena oli luoda alfa-versio palvelusta, josta videopelien pelaajayhteisöt voivat hankkia kotisivut. Sivuston on myös tarkoitus palvella yksittäisiä pelaajia, jotka etsivät itselleen sopivaa yhteisöä. Sivusto suunnataan aluksi Guild Wars 2 -videopelin yhteisöille. Tämän takia sivustolle integroidaan pelin API, jonka kautta haetaan tietoja yhteisöjen pelikilloista.

Projektissa toteutettiin rekisteröityminen sivustolle, alasivuston hankinta kotisivuiksi sekä keskustelualue ja oikeuksienhallinta alasivustolle. Tässä opinnäytetyössä käyttöliittymä on huomioitu vain välttämättömien tiedonsyöttökenttien ja linkkien osalta. Käyttöliittymän suunnittelu ja toteutus tehdään toisessa opinnäytetyössä.

Työ toteutettiin käyttäen Zend Framework 2 -viitekehystä, Doctrine 2 ORM -kirjastoa ja noudattaen MVC-arkkitehtuuria. Työ aloitettiin lokakuussa 2013 ja se päätettiin toukokuussa 2014. Työtä tehtiin kuukauden jaksoissa ja jokaisen jakson päätteeksi pidettiin ohjauskokous. Työn etenemistä seurattiin ohjauskokouksia varten laadittujen edistymisraporttien avulla.

Projektin tuloksena valmistui sivusto, joka sisältää edellä mainitut keskeisimmät toiminnot. Projektin aikana parannettiin koodin nimeämiskäytäntöjä ja rakennetta. Sivustoa jatkokehitetään opinnäytetyön valmistuttua.

Asiasanat
PHP, keskustelualue, foorumi, produkti, ORM

Abstract

HAAGA-HELIA
ammattikorkeakoulu

8.5.2014

Degree Programme in Information Technology

Authors Marko Hannila Krista Wikström	Group or year of entry 2011
The title of thesis Guild Charter - Website hosting service for video-game communities	Number of report pages and attachment pages 30 + 1
Advisor(s) Sirpa Marttila	

The goal of the project was to create an alpha version of a service through which videogame communities can acquire a web site for themselves. The main site is also meant to serve individual players who want to find communities suiting their personal tastes. In the beginning, the site will be aimed at communities that play the Guild Wars 2 videogame. To fulfill the needs of these players better, the game's API will be integrated into the site. The API can be used to fetch information about the community's in-game guild.

In this project, the following features were developed on the site: registration, acquiring a subdomain site for the community, forum on the subdomain, and rights management for roles within the subdomain. The user interface is not the focus of this thesis, and has not been addressed, other than where required input fields and links are concerned. Designing and developing the user interface are addressed in another thesis.

The main tools and techniques used in the project were Zend Framework 2, Doctrine 2 ORM library, and MVC architecture. The work began in October 2013 and concluded in May 2014. The work was divided into one-month periods, and after each, a directive meeting was held. The advancement of the project was monitored with progress reports that were drafted for each meeting.

The result of this thesis was a web site that has the fore mentioned features. During the project, the structure of the code and the naming practices were improved. The site will be further developed after the work for the thesis has concluded.

Key words
PHP, forum, product, ORM

Sisällys

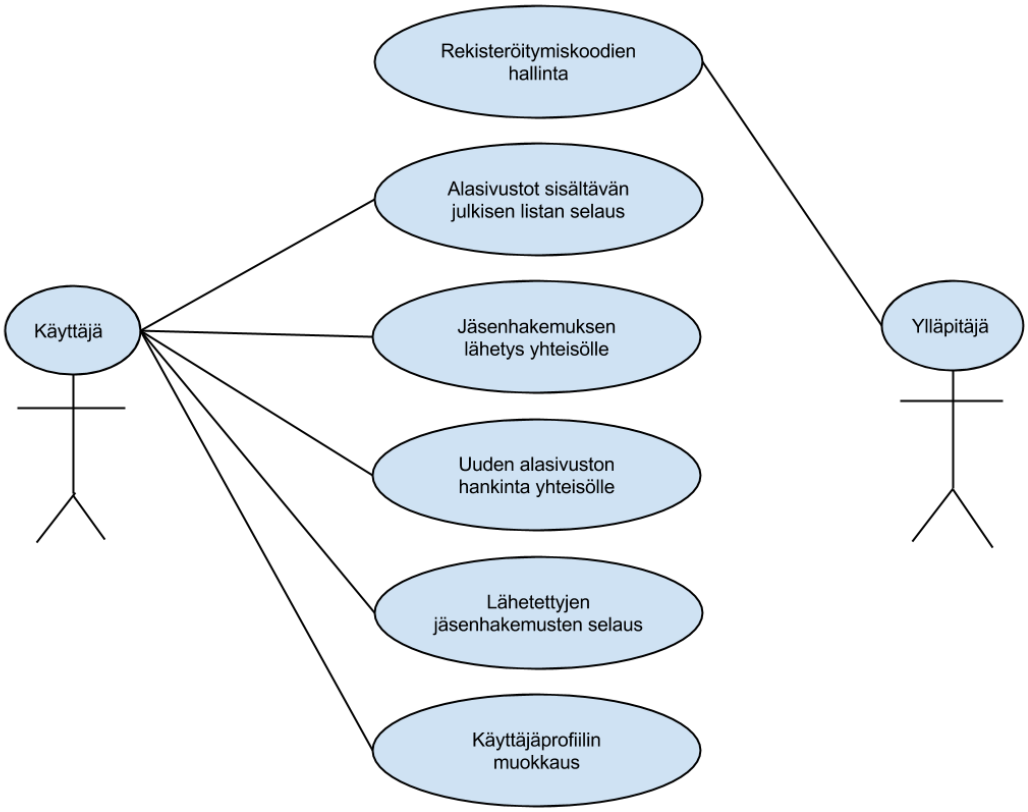
1 Johdanto 1
1.1 Tavoitteet 1
1.2 Opinnäytetyön rajaus 3
1.3 Käsitteet 3
2 Tietoperusta 5
2.1 Git 5
2.2 Vagrant 6
2.3 Composer 7
2.4 MVC 8
2.5 ORM 9
2.6 Viitekehys 11
2.7 Zend Framework 2 11
3 Guild Charter -projekti 13
3.1 Tavoite 13
3.2 Tietokannat 13
3.3 Pääsivusto 15
3.4 Guild Wars 2 API 16
3.5 Keskustelualue 17
3.6 Roolit 18
3.7 Tietoturva 20
3.8 Toteutuskuvaus 21
4 Yhteenveto 23
4.1 Tulokset 23
4.2 Parityöskentely 25
4.3 Haasteet kehitystyössä 26
4.4 Oma oppiminen, Krista 27
4.5 Oma oppiminen, Marko 28
4.6 Jatkokehitys 28
Lähteet 31
Liitteet 34
Liite 1. Relaatiokaavio (salainen) 34

1 Johdanto

Projektissa tehtiin verkkosivusto, joka toimii pelaajien kokoontumis- ja keskustelualueena. Projekti käynnistettiin, koska videopeliyhteisöille suunnatulla kotisivupalvelulla uskottiin olevan kysyntää. Ohjelmisto toteutettiin PHP:lla käyttäen Zend Framework 2 PHP-viitekehystä ja Doctrine 2 -kirjastoa. Sovellus noudattaa MVC-arkkitehtuuria. Opinnäytetyö on alfa-versio lopputuotteesta ja kehitystyötä jatketaan opinnäytetyön valmistuttua.

1.1 Tavoitteet

Projektin tavoitteena oli tuottaa alfa-versio sivustosta, joka tarjoaa sosiaalisen kokoontumispaikan ja keskustelualueen MMO-pelejä (Massive Multiplayer Online) pelaaville yhteisöille. Palvelun avulla peliyhteisöt voivat järjestää erilaisia yhteisiä tapahtumia ja kommunikoida pelien ulkopuolella. Yhteisöjen alasivustot kootaan yhdeksi listaksi, josta käyttäjät voivat etsiä itselleen sopivaa yhteisöä ja lähettää liittymispyynnön. Yhteisön jäsenmäärää ei rajata ja käyttäjä voi kuulua moneen eri yhteisöön. Pääsivusto toimii palvelun keskuksena, joka sisältää perustoiminnot palvelun käytölle (Kuvio 1).



Kuvio 1. Käyttötapauskartta pääsivustosta.

Sivuston alfa-versioon kuuluu pääominaisuutena keskustelualue, johon on mahdollista määrittää pääsyoikeudet käyttäjäryhmille aluekohtaisesti. Sivustolla on päätietokanta ja jokaisella alasivustolla on oma tietokantansa. Projektissa tehdään sovelluksen dokumentaatio, jotta palvelun jatkokehitys helpottuisi ja uusien kehittäjien liittyminen projektiin sujuisi nopeasti. Projektin lopputuotetta voidaan myös käyttää taidonnäytetyönä tekijöiden CV:issä.

Markkinoilla oleviin kilpailijoihin verrattuna palvelun alfa-versio ei tuo juurikaan uusia ominaisuuksia tarjolle. Tämä johtuu pääosin siitä, että kilpailijoiden palveluita on kehitetty huomattavasti kauemmin ja ne ovat sen takia kehittyneempiä. Kilpailijoiden palvelut tarjoavat yleensä itsenäisiä kotisivuja. Guild Charterin tarjoamat kotisivut ovat osa koko sivustoa, erillisen sivuston sijaan. Tältä kannalta katsottuna Guild Charterissa on sosiaalisen median piirteitä, mutta se on suunnattu yhteisöille yksilöiden sijaan.

1.2 Opinnäytetyön rajaus

Toteutettu projekti on jaettu kahdeksi opinnäytetyöksi, joista toinen käsittelee käyttöliittymän suunnittelua ja toteutusta ja toinen sovelluksen toiminnallisuuksien suunnittelua ja toteutusta. Tässä opinnäytetyössä on keskitytty toiminnallisuuksien toteuttamiseen, joten käyttöliittymässä on huomioitu vain välttämättömät syötekentät ja linkit. Näin ollen käyttöliittymän toteutus puuttuu myös sivustosta otetuista kuvakaappauksissa.

1.3 Käsitteet

Alakeskustelupalsta	Keskustelupalsta, joka on toisen keskustelupalstan alla.
Alasivusto	Yhteisön oma alue palvelussa. Sisältää yhteisön oman keskustelualueen.
Keskustelualue	Alasivuston alue, jossa käyttäjät voivat keskustella.
Keskusteluketju	Kokoaa käyttäjien viestit keskusteluksi.
Keskustelupalsta	Pitää sisällään keskusteluketjuja ja keskustelupalstoja.
Keskustelupalstaryhmä	Ryhmä keskustelupalstoja keskustelualueen juuressa.
Käyttäjä	Henkilö, joka on rekisteröitynyt sivustolle.
MMO-peli	Massive Multiplayer Online -peli on videopeli, jota pelataan verkossa muiden pelaajien kanssa. MMO-pelit keskittyvät vahvasti pelaajien keskinäiseen viestintään ja vuorovaikutukseen. Pelien sosiaalisesta luonteesta johtuen pelaajat muodostavat peleissä usein yhteisöjä.
Omistaja	Käyttäjä, jonka nimissä alasivusto on. Omistaja vastaa ylläpitäjä-roolien määrityksistä ja on itse oletuksena ylläpitäjä.
Pääsivusto	Sivuston alue, jossa käyttäjä voi päivittää profiiliaan ja hankkia alasivuston.

Vieraileva käyttäjä	Sivustolle kirjautunut henkilö, joka on vierailemassa alasivustolla, jonka jäsen hän ei kuitenkaan ole.
Vierailija	Henkilö, joka selaa sivustoa kirjautumatta sisälle.
Yhteisö	Käyttäjistä koostuva ryhmä, jolla on oma alasivusto.
Ylläpitäjä	Alasivuston hallinnoinnista vastaava henkilö, joka on saanut valtuudet omistajalta. Virallisen alasivuston ylläpitäjä on ylläpitäjä myös pääsivustolla.

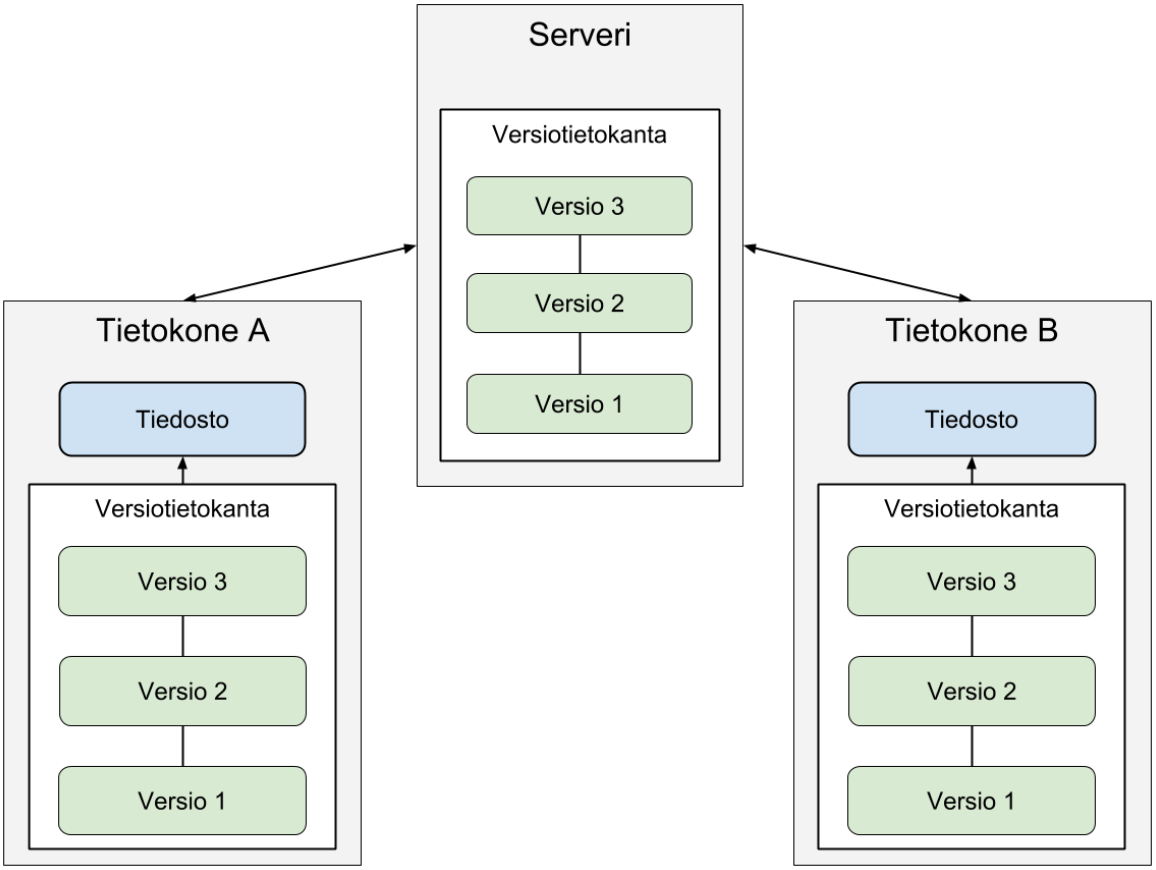
2 Tietoperusta

Sovelluksissa käytetään usein valmiita koodikirjastoja ja viitekehyksiä. Koodi rakennetaan usein jollain yleisesti tunnetulla arkkitehtuurilla ja koodiratkaisut tehdään soveltaen vakiintuneita tekniikoita ja käytäntöjä. Tämä mahdollistaa sen, että kehittäjä voi siirtyä toisiin projekteihin ja hän silti pystyy ymmärtämään, kuinka sovellus toimii.

Kirjastoja, viitekehyksia, arkkitehtuureja, tekniikoita, käytäntöjä ja ohjelmointikieliä on lukemattomat määrät, joten jokaisen projektin kohdalla joudutaan valitsemaan projektiin sopivat työvälineet. Usein työvälineiden valintaan vaikuttavat merkittävästi kehitystiimin taidot ja osaaminen, ellei kehitystiimiä koota vasta työvälineiden valinnan jälkeen. (Riehle 2000, 1-2.)

2.1 Git

Git on hajautettu versionhallintajärjestelmä, joka on kehittynein nykyisistä versionhallintatavoista. Versionhallinta tarkoittaa yksinkertaisimmillaan sitä, että kehittäjä ottaa projektin hakemistosta varmuuskopion tietyin aikavälein. Hajautetussa versionhallinassa palvelin vastaa keskitetysti versionhallinnasta, mutta kehittäjät kopioivat palvelimelta koko versionhallinnan omille tietokoneilleen. (Chacon 2009, 1-3; Kuvio 2.) Kun kehittäjät tekevät muutoksia projektiin ja haluavat saattaa muutokset muiden kehittäjien saataville, he työntävät omalla koneellaan olevan versionhallinnan palvelimelle. Jos joku muu on työntänyt palvelimelle muutoksia, joita kehittäjä ei ole hakenut, hylkää palvelin toiminnon. Tällöin uudet muutokset haetaan palvelimelta, jonka jälkeen omatkin muutokset voidaan työntää palvelimelle. (Chacon 2009, 34.) Hajautettu versionhallinta minimoi riskin menettää kehitystyön tulos, jos versionhallinnasta vastaava palvelin hajoaa (Chacon 2009, 3).



Kuvio 2. Hajautettu versionhallinta (Chacon 2009, 3.)

Hajautettua versionhallintaa edelsi keskitetty versionhallinta. Se eroaa hajautetusta versionhallinnasta siten, että palvelimelta ei kopioida koko versionhallintaa vaan ainoastaan tiedostot, jotka tarvitaan. Hajautetun ja keskitetyn versionhallinan avulla tiedostot tai koko projekti voidaan palauttaa aiempaan versioon. Lisäksi on mahdollista seurata projektiin tehtyjä muutoksia ja vertailla versioiden välisiä eroja. Molempien versionhallintatapojen avulla projektissa voidaan selvittää, kuka on tehnyt muutoksen, milloin muutos on tehty ja mitä tarkalleen on muutettu. Jos kehitettävä sovellus ei toimi viimeisen muutoksen jälkeen, koodi voidaan palauttaa muutosta edeltävään tilaan, jolloin projektin kehitys ei keskeydy muilla kehittäjillä. Edellä mainittuja ominaisuuksia voidaan hajautetussa versionhallinnassa käyttää omalla koneella. Keskitetyssä versionhallinnassa tietoja on tarkasteltava palvelimelta. (Chacon 2009, 1-3.)

2.2 Vagrant

Vagrant on työkalu, jolla pystytetään kehitysympäristöjä (HashiCorp a). Sama kehitysympäristö voidaan asentaa Vagrantin avulla jokaisen kehittäjän koneelle, vaikka koneet

toimisivat eri käyttöjärjestelmillä. Vagrantin tukemat käyttöjärjestelmät ovat Linux, OS X ja Windows. Kehitysympäristö pystytetään virtuaalikoneeseen, jonne Vagrant asentaa kaiken tarpeellisen ja synkronoi projektin tiedostot virtuaalikoneeseen. Kun Vagrantin asennustiedostoon on määritelty ympäristön asennusohjeet, asennustiedosto voidaan liittää versiohallintaan. Tämän jälkeen muut kehittäjät voivat pystyttää kehitysympäristön ajamalla asennustiedoston Vagrantilla. Jokainen kehittäjä voi näin luottaa siihen, että sovellus toimii samoin kuin tuotantopalvelimella ja muiden kehittäjien tietokoneilla. (HashiCorp c.) Muun muassa Nokia, Mozilla, Yammer, O'Reilly, Disqus ja BBC käyttävät Vagrantia kehitystyössään (HashiCorp b).

Vagrantin asetustiedoston voi kirjoittaa joko itse tai apuna voi käyttää PuPHPetia tai muuta vastaavaa palvelua. PuPHPetin avulla asetukset saa valittua helposti, jonka jälkeen asetustiedostot ladataan zip-pakettina. Paketin purkamisen jälkeen ympäristö pystytetään ajamalla asennustiedostot Vagrantilla. (Treminio 2013.)

2.3 Composer

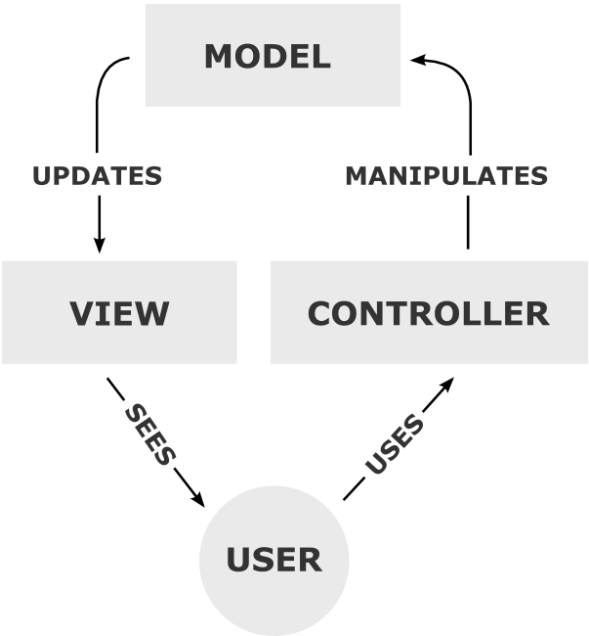
Composer on työkalu, jolla hallitaan koodikirjastojen riippuvuuksia PHP-projekteissa. Se nopeuttaa projektin vaatimien kirjastojen asennusta ja pitää huolta, että kirjastoista ladataan oikeat versiot. Composer helpottaa työskentelyä huomattavasti silloin, kun kehitettävä sovellus riippuu monista ulkoisista kirjastoista, ja ne edelleen muista kirjastoista. Riippuvuuksien hallinnan avulla projektiin on helppo sisällyttää uusia kirjastoja. Tällöin kehittäjien ei tarvitse itse selvittää, mitä vaatimuksia uusilla kirjastoilla on. Composeria voidaan käyttää Linux, OS X tai Windows -käyttöjärjestelmällä. (Composer.)

Composerille voidaan esimerkiksi määrittää, että projekti vaatii Doctrine Modulen. Doctrine Module on moduuli Zend Framework 2 -viitekehykselle. Sen avulla Doctrine saadaan helposti käyttöön sovelluksessa, joka käyttää Zend Framework 2 -viitekehystä. Composer lataa projektiin Doctrine Modulen ja tarkistaa sen riippuvuudet, joihin sisältyy Doctrine ja monia Zend Framework 2:n osia. Ladattuaan Doctrinen Composer tarkistaa edelleen sen riippuvuudet ja lataa vaaditut kirjastot. Sama toistetaan kaikille ladatuille kirjastoille, kunnes jokainen ladattu kirjasto on käyty läpi ja sen vaatimukset täytetty.

2.4 MVC

MVC eli Model-View-Controller -arkkitehtuuri jakaa ohjelmiston kolmeen osaan: malliin (model), näkymään (view) ja ohjaimeen (controller), kuten kuviossa 3 on esitetty. MVC-arkkitehtuuri on suosittu ohjelmistonkehittäjien keskuudessa, koska se helpottaa koodin ylläpitoa ja uudelleenkäytettävyyttä selkeän rakenteensa ansiosta (Google). Rakenne muodostuu arkkitehtuurin tavasta eritellä käyttäjälle näkyvä käyttöliittymä ja sovelluksen liiketoimintalogiikka toisistaan.

Erottelusta on hyötyä nykypäivän sovelluskehityksessä monin tavoin. Esimerkiksi sovelluksen käyttöliittymän pitää usein mukautua tietokoneen tai mobiililaitteen näytön resoluutioon. Ohjain voidaan tällöin määrittää palauttamaan eri näkymä resoluution mukaan, vaikka mallin koodi pysyy täysin samana. (Pastor 2010.) MVC-arkkitehtuurin mukainen jako mahdollistaa käyttöliittymän ja toiminnallisuuksien rakentamisen samanaikaisesti erillään toisistaan, jolloin töitä voidaan hajauttaa helposti eri henkilöille (Google). MVC-arkkitehtuurin laajan käytön seurauksena monet viitekehykset, kuten Zend Framework 2, rakentavat toimintansa sen pohjalta (Php Frameworks).



Kuvio 3. MVC-arkkitehtuurin kuvaus (Frey 2010).

Näkymä on rajapinta sovelluksen ja käyttäjän välillä. Suosituksena on, että näkymä ei sisällä lainkaan liiketoimintalogiikkaa. Näkymä ottaa vastaan tietoa mallilta, jolloin näkymä muuttuu mallin palauttaman tiedon mukaan. (Strong 2013; Qian, Fu, Tao, Xu, & Díaz-Herrera 2009, 201-202.)

Ohjain vastaanottaa selaimesta tulevia syötteitä ja käsittelee ne. Se välittää tehtäviä mallille ja palauttaa uuden näkymän selaimelle (Qian ym. 2009, 205.). Ohjain päivittää näkymän, kun mallin sisällä oleva tieto muuttuu. Kun käyttäjä on vuorovaikutuksessa näkymän kanssa, päivittää ohjain mallin tietoja. (Google.)

Malli on MVC arkkitehtuurin alin taso. Se sisältää liiketoimintalogiikan ytimen, mikä tarkoittaa monissa tapauksissa vuorovaikutusta tietokantoihin. Malli vastaa tiedon hakemisen lisäksi myös tiedon muokkaamisesta. Ohjain lähettää mallille pyynnön tallentaa tai hakea tietoa tietokannasta. (Strong 2013.) Kun sovellus käsittelee tietoa, säilyttää se tiedon mallin olioissa (Google). Esimerkiksi kun käyttäjä rekisteröityy sovellukseen, tallennetaan käyttäjän tiedot ensin malliin. Käyttäjän tiedot tallentuvat tietokantaan vasta, kun sovellus on tarkistanut tiedot ja suorittanut muut rekisteröitymiseen liittyvät tehtävät.

2.5 ORM

ORM eli object-relational mapping on ohjelmistokehityksessä käytetty tekniikka, jolla välitetään tietoa relaatiokannan ja ohjelmakoodin välillä. Tekniikka perustuu olioiden ja tietokannan yhdistämiseen. Tällöin kehittäjä pyytää käytettyä kirjastoa muodostamaan halutun olion ja kirjasto muodostaa SQL-kyselyt määriteltyjen yhteyksien perusteella. (Janssen.)

Tietokannat tukevat usein vain yksinkertaisten tietojen, kuten numeroiden tai kirjainsarjojen tallennusta. Olio-ohjelmoinnissa tietoja käsitellään kuitenkin olioiden välityksellä. ORM-tekniikan perusajatuksena on, että olio muunnetaan tallennettaessa muotoon, jossa sen sisältämät tiedot voidaan tallentaa tietokantaan. Kun tietoja haetaan tietokannasta, ne muunnetaan takaisin oliomuotoon. (Janssen; Doctrine Team 2013.)

Mappaus tehdään yhdistämällä luokan jokainen attribuutti nollaan tai useampaan sarakkeeseen tietokannassa. Jos olion attribuutissa olevaa tietoa ei tarvitse tallentaa tietokantaan, ei kyseistä attribuuttia mapata. Osa attribuuteista sisältää viitteen toiseen olioon, mikä vastaa relaatiokannassa olevaa viiteavainyhteyttä. Attribuutin mappaus on yksinkertaisin, kun sarakkeen tietotyyppi on sama tai vastaava kuin attribuutin. (Ambler.)

Esimerkiksi Käyttäjä-oliolla voi olla attribuutit id, käyttäjänimi, sähköposti, salasana, osoite ja kaverit. Näistä id, käyttäjänimi, sähköposti ja salasana tallennetaan tietokannan Käyttäjä-tauluun. Osoite-attribuutti sisältää viitteen Osoite-olioon, jota vastaa Osoite-taulu tietokannassa. Kaverit-attribuutti on lista viittauksista toisiin Käyttäjä-olioihin. Tieto viitteista on tallennettu tietokannassa välitauluun. Välitaulu viittaa Käyttäjä-tauluun kahdesti. Toinen viite kertoo kuka on kaveri ja toinen kenen kaverista on kyse. Käyttäjä-oliolta voidaan pyytää jokin perustiedoista: id, käyttäjänimi, sähköposti tai salasana. Palautettu tieto on numero, teksti, aika tai jokin muu yksinkertainen tietotyyppi. Kun pyydetään osoitetta, saadaan Osoite-olio, jolta voidaan puolestaan pyytää sen omia attribuutteja. Näitä voivat olla esimerkiksi katuosoite, kaupunki ja maa. Osa näistäkin attribuuteista voi olla viitteitä toisiin olioihin. Kaverit-attribuutin sisältämää listaa voidaan käydä läpi ja muodostaa siitä esimerkiksi taulukko, jossa on jokaisen kaverin käyttäjänimi, sähköposti sekä kaverin osoitetiedoista maa ja kaupunki.

Doctrine ORM on PHP-kirjasto ORM-tekniikan käyttöön. Se tarjoaa kehittäjälle mahdollisuuden keskittyä paremmin kehitystyöhön. Kehittäjä voi jättää tietokannan ylläpidon Doctrinen vastuulle ja keskittyä liiketoimintaloogiikan toteuttamiseen. Doctrine antaa kehittäjän käyttöön DQL:n (Doctrine-Query-Language), joka on OQL (Object-Query-Language). Kun kehittäjä tarvitsee yksinkertaista select-, insert-, update- tai delete-kyselyä monimutkaisemman kyselyn, voidaan se kirjoittaa DQL:lla. DQL muistuttaa SQL:a, mutta se ei ole tietoinen tietokannan tauluista tai sarakkeista, vaan sillä viitataan olioihin ja olioiden attribuutteihin. Oliot puolestaan on yhdistetty ORM:n mukaisesti tietokannan tauluihin, jolloin Doctrine muodostaa SQL-kyselyn DQL-kyselyn perusteella. (Doctrine Team 2013.)

2.6 Viitekehys

Viitekehys on kehitysalusta, joka tarjoaa uudelleenkäytettävän koodipohjan ohjelmiston kehitykselle. Viitekehys tarjoaa koodikirjastoja, työkaluja ja ohjelmointirajapintoja kehittäjän käyttöön. (Riehle 2000, 1-2.) Viitekehysten käytön etuna on ennalta kehitetyn ja testatun koodin käyttö, mikä keventää kehitys- ja testaustyötä ja auttaa uusia sovelluksia toimimaan luotettavammin. Hyvin suunniteltu viitekehys voi auttaa yleistämään hyviä ohjelmointikäytäntöjä, suunnittelumalleja ja ohjelmistokehitystyökaluja. Lisäksi sovelluksen tehokkuus voi parantua pelkällä viitekehyksen päivityksellä. (Baker 2009.)

Viitekehyksien toimintaa on mahdollista laajentaa vastaamaan paremmin kehitettävän sovelluksen tarpeita. Tällöin viitekehyksen koodia ei muokata, vaan sitä jatketaan omalla koodilla. (Baker 2009.) Sovelluksen ylläpito helpottuu, kun käytetty viitekehys noudattaa selkeää rakennetta, kuten esimerkiksi MVC-arkkitehtuuria. Koodin selkeää rakennetta kannattaa ylläpitää, jotta virheiden löytäminen ja korjaaminen on helpompaa. (Allen, Lo & Brown 2009, 4-8.)

Viitekehyksien kehittäminen on kuitenkin hyvin työlästä ja aikaa vievää, minkä vuoksi käytetään usein valmista viitekehystä sen sijaan, että kehitettäisiin oma viitekehys. Viitekehysten käytön opettelukaan ei ole välttämättä helppoa. Päivitykset saattavat myös laajentaa ja monimutkaistaa viitekehystä, mikä tekee oppimisen vieläkin vaikeammaksi. (Baker 2009.)

2.7 Zend Framework 2

Zend Framework 2 on PHP-viitekehys. Sen toiminta on suunniteltu laajennettavaksi moduuleilla, jotka voivat olla viitekehyksen kehittäjien tai ulkopuolisten tahojen tuottamia. Zend Framework 2 -yhteisö kehittää moduuleja moniin eri tarkoituksiin. Moduulin kehittäjä jakaa moduulin usein myös muiden kehittäjien käyttöön.

Esimerkiksi ZFCUser-moduuli mahdollistaa sisäänkirjautumisen, rekisteröitymisen ja käyttäjän kirjautumistilan seurannan. Se on suunniteltu helppokäyttöiseksi ja sen muokkaaminen on vaivatonta. (Coury 2014.) ZFCUser-moduulin käyttöönotto on nopeaa. Se asennetaan Composerin kautta ja lisätään sovelluksen aktiivisiin moduuleihin (Stosic

2013). BjyAuthorize on puolestaan moduuli, joka jatkaa ZFCUser-moduulin toimintaa. BjyAuthorize lisää auktorisointiominaisuuden, jolla voidaan helposti ja nopeasti rajata pääsyä ohjaimiin ja niiden metodeihin käyttäjän roolien perusteella.

3 Guild Charter -projekti

Opinnäytetyön lopputuote on Guild Charter -sivuston alfa-versio. Sivusto sisältää rekisteröitymisen, sisäänkirjautumisen, oman yhteisön luonnin ja perustan yhteisön toiminnalle ja hallinnalle. Perustoimintoihin kuuluvat rekisteröityminen, sisäänkirjautuminen, viestien ja keskusteluketjujen lisääminen keskustelualueelle, omien tietojen päivittäminen ja omien keskusteluketjuviestien muokkaaminen.

Hallintatoimintoihin kuuluvat muiden käyttäjien viestien muokkaaminen, keskustelupalstojen ja keskusteluketjujen siirtäminen keskustelualueella, käyttäjien roolien hallinta ja sitä kautta käyttöoikeuksien hallinta.

3.1 Tavoite

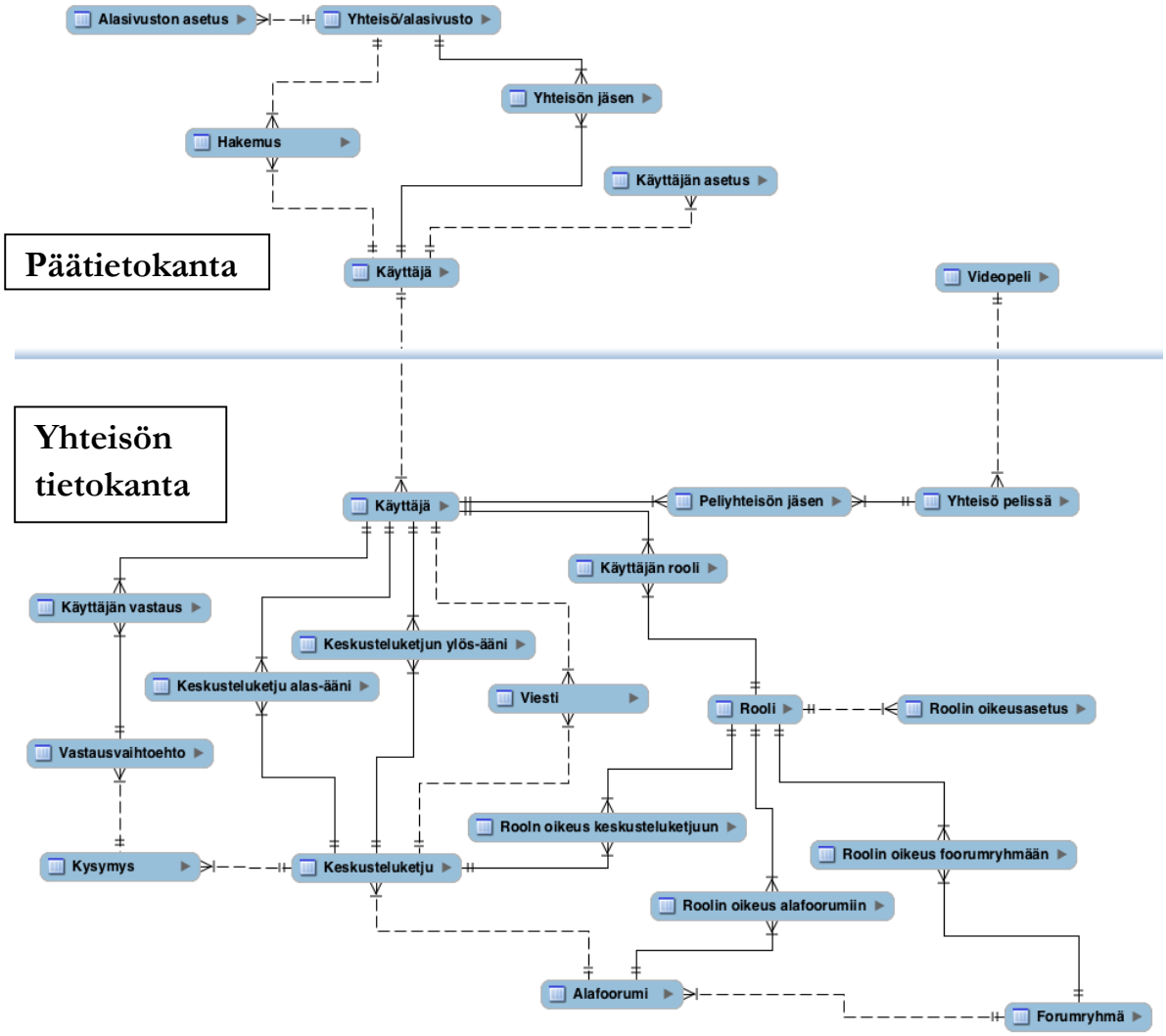
Projekti syntyi tarpeesta saada peliyhteisölle verkkosivut, jotka sisältävät muutakin kuin vain keskustelualueen. Saatavilla on monia valmiita keskustelualuepalikoita, mutta toimintojen täydentäminen niihin voi olla vaikeaa, mikäli tarvittavia ominaisuuksia puuttuu. Useat sivustot, jotka tarjoavat keskustelualueita ilmaiseksi tai pientä maksua vastaan, keskittyvät yksinomaan keskustelualueisiin. Näille sivustoille ei voi lisätä muuta sisältöä kuten esimerkiksi kalenteria tulevista tapahtumista.

Tästä johtuen päätettiin toteuttaa MMO-pelien pelaajille suunnattu kotisivupalvelu. Sivusto toimii myös yhteisenä kokouspaikkana pelaajille. Sivusto ei ainoastaan keskity vain kotisivujen tarjoamiseen, vaan auttaa pelaajia löytämään itselleen sopivia peliyhteisöjä sekä pitämään yhteyttä pelikavereihin pelien ulkopuolella. Projektin tavoitteena oli tehdä perusta kattavalle palvelulle, jota pystyvät hyödyntämään niin pienet kuin suuretkin videopeliyhteisöt. Oman yhteisön ylläpito on maksutonta palvelun ollessa alfa-vaiheessa.

3.2 Tietokannat

Sovellus hyödyntää toiminnassaan useita tietokantoja. Päätietokantaa ja jokaisen yhteisön omaa tietokantaa, jotka on esitetty tiivistetysti kuviossa 4. Tarkempi relaatiokaavio

tietokannasta on liitteenä 1. Kun käyttäjä lähettää alasivuston hankintalomakkeen palvelimelle ja sovellus on tarkastanut lomakkeen tiedot, luodaan yhteisön tietokanta automaatisesti ajamalla sql-tiedosto. Sql-tiedostossa on sekä yhteisön tietokannan taulujen luontilauseet, että tiedonsyöttölauseet oletustiedoille. Oletustietoja ovat oletusroolit ja alasivuston oletusasetukset.



Kuvio 4. Yksinkertaistettu tietokantojen relaatiokaavio

Päätietokannan avulla hallinnoidaan yleisiä tietoja käyttäjistä ja ylläpidetään rekisteriä yhteisöistä. Yhteisön omaan tietokantaan tallennetaan yhteisön alasivuston sisältö. Relaatiotietokannat eivät voi ylläpitää yhteyksiä toisiin tietokantoihin, joten yhteyksien ylläpito tietokantojen välillä on Doctrinen vastuulla. Tiedot tallentuvat tietokantoihin käyttäjien syötteiden ja Guild Wars 2 -pelin API:n kautta saatavien tietojen perusteella.

Guild Wars 2 API tarjoaa tietoja pelissä olevista esineistä, killoista, kartoista ja taisteluiden kulusta pelimaailmojen välillä. Sovelluksen alfa-versiossa tallennetaan kuitenkin vain tieto yhteisön killasta tai killoista.

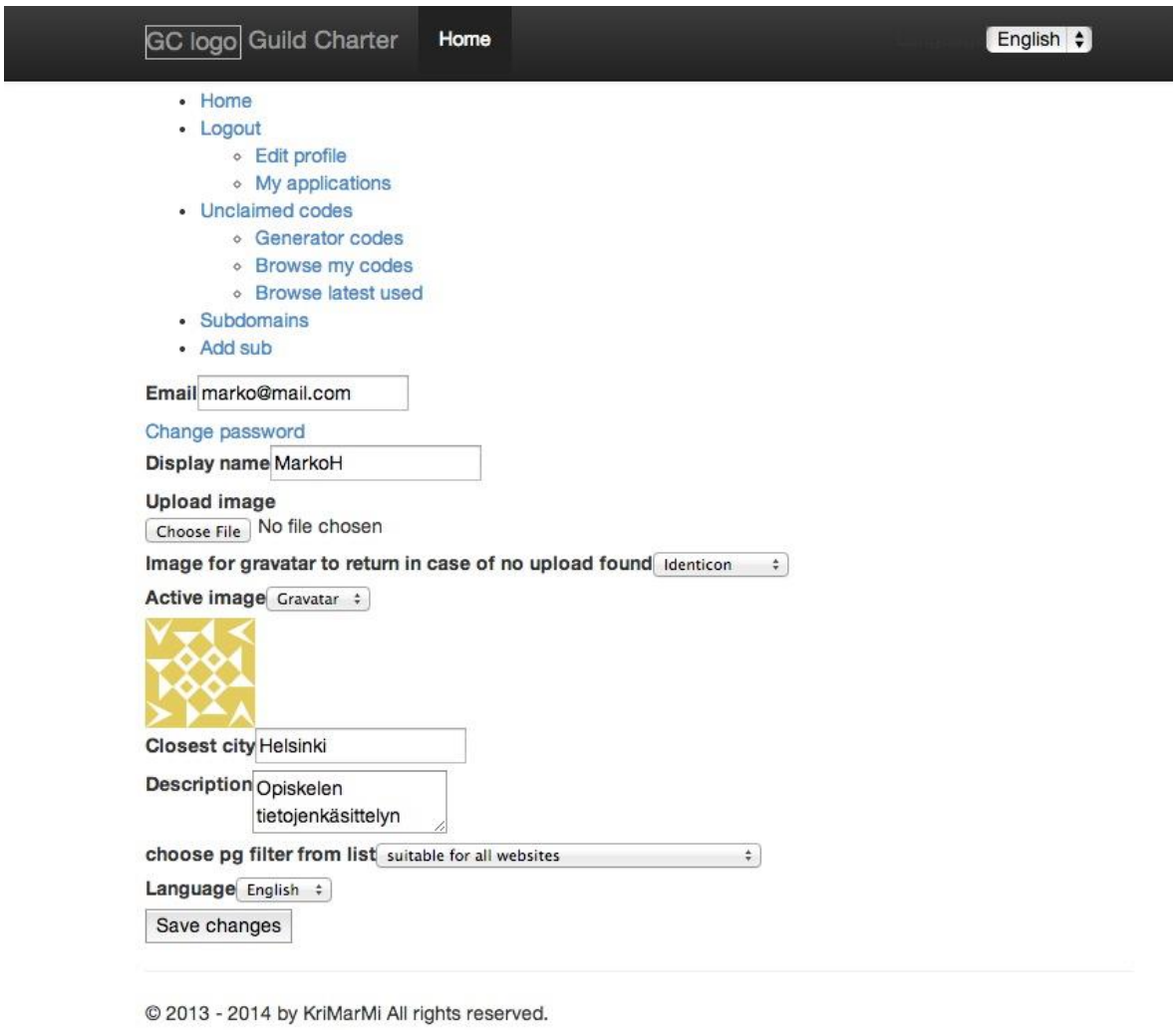
3.3 Pääsivusto

Jotta käyttäjä voi liittyä yhteisön jäseneksi, on palvelun mahdollistettava yhteisön löytäminen. Tämä toteutettiin muodostamalla julkinen lista yhteisöistä, jotka eivät ole erikseen asettaneet itseään piilotetuiksi. Listassa näkyvät tiivistetysti tarpeellisimmat tiedot kuten yhteisön nimi, lyhyt kuvaus yhteisöstä ja pg-arvo (parental guidance) alasivuston sisällölle. Pg-arvo ilmaisee, onko alasivustolla esimerkiksi kiroilua tai väkivaltaista sisältöä, jonka perusteella käyttäjä voi suodattaa yhteisöjä listalta ja arvioida yhteisön sopivuutta itselleen. Kun käyttäjä löytää mielenkiintoisen yhteisön, hän voi siirtyä täyttämään hakemuslomaketta Lähetä hakemus -linkin kautta.

Saapuneet hakemukset näytetään alasivustolla käyttäjälle, jolle on annettu oikeudet käsitellä yhteisölle saapuneita hakemuksia. Hän voi selata hakemuksia ja hyväksyä tai hylätä niitä. Ylläpitäjällä on aina oikeus käsitellä hakemuksia, mutta hän voi myös luoda roolin, jolle hän antaa oikeudet hakemusten käsittelyyn. Kun hakemus hyväksytään, tallennetaan uuden jäsenen tiedot alasivuston tietokantaan.

Käyttäjä voi myös siirtyä selaamaan yhteisön alasivustoa valitsemalla yhteisön nimen listasta. Alasivuston etusivulla on täysi kuvaus yhteisöstä. Vieraileva käyttäjä voi selata ainoastaan alasivuston julkisia osia. Oletuksena vain alasivuston etusivu on julkinen, mutta ylläpitäjät voivat avata alasivuston muitakin osia julkisiksi.

Rekisteröityneellä käyttäjällä on käytössään oma profiilisivunsa, joka on esitetty kuvassa 1. Sivulla käyttäjä voi päivittää rekisteröitymisen yhteydessä antamiaan tietoja. Käyttäjä voi täydentää profiiliaan kirjoittamalla kuvauksen itsestään tai lisäämällä profiilikuvan. Käyttäjä voi valita aktiivisen profiilikuvan kolmesta vaihtoehdosta. Kuvana voidaan käyttää palvelun oletusprofiilikuvaa, kuva voidaan hakea Gravatar-kuvapalvelusta tai käyttäjä voi valita sivustolle aiemmin lataamansa kuvan. Alasivuston jäsenellä on käytössään alasivustokohtainen profiilisivu. Alasivuston profiilisivulla käyttäjä voi valita näyttönimen, kuvan ja kuvauksen, jotka näkyvät vain kyseisellä alasivustolla.



Kuva 1. Sivu, jolla päivitetään oman profiilin tietoja.

Rekisteröitymiseen lisättiin jälkikäteen kenttä rekisteröitymiskoodille. Koodivaatimus lisätiin, sillä pääsyä sovelluksen alfa-versioon haluttiin rajoittaa. Ainoastaan virallisen alasivuston ylläpitäjillä on oikeus luoda uusia koodeja. Rekisteröitymiskoodin voi saada vain suoraan kehittäjiltä, jolloin pystytään hallitsemaan tarkasti, kuka pääsee testikäyttämään sivustoa. Rekisteröitymiskoodi poistetaan, kun sivusto on julkaisukelpoinen ja käyttäjien määrää ei haluta enää rajoittaa.

3.4 Guild Wars 2 API

Ensimmäiseksi sovellukseen integroitavaksi ohjelmointirajapinnaksi (API) valittiin Guild Wars 2 -pelin API, sillä se oli ainoa kehittäjille jo entuudestaan tuttu ohjelmointi-

rajapinta. Sivuston alfa-versioon ei integroida muita ohjelmointirajapintoja. Sovellus hakee ohjelmointirajapinnan avulla tietoja yhteisön Guild Wars 2- killasta. Ylläpitäjien ei tarvitse päivittää ohjelmointirajapinnan kautta saatavia tietoja, vaan sovellus hakee ne automattisesti palveluun syötetyn killan nimen perusteella.

Tällä hetkellä ohjelmointirajapinnan kautta killoista saatavat tiedot ovat nimi, lyhenne ja logo. Killan tietoja voi hakea ohjelmointirajapinnasta myös killan id:n eli tunnisteen perusteella. Koska peruskäyttäjän on vaikea selvittää killan tunnistettta, päätettiin tietojen haku tehdä killan nimen mukaan.

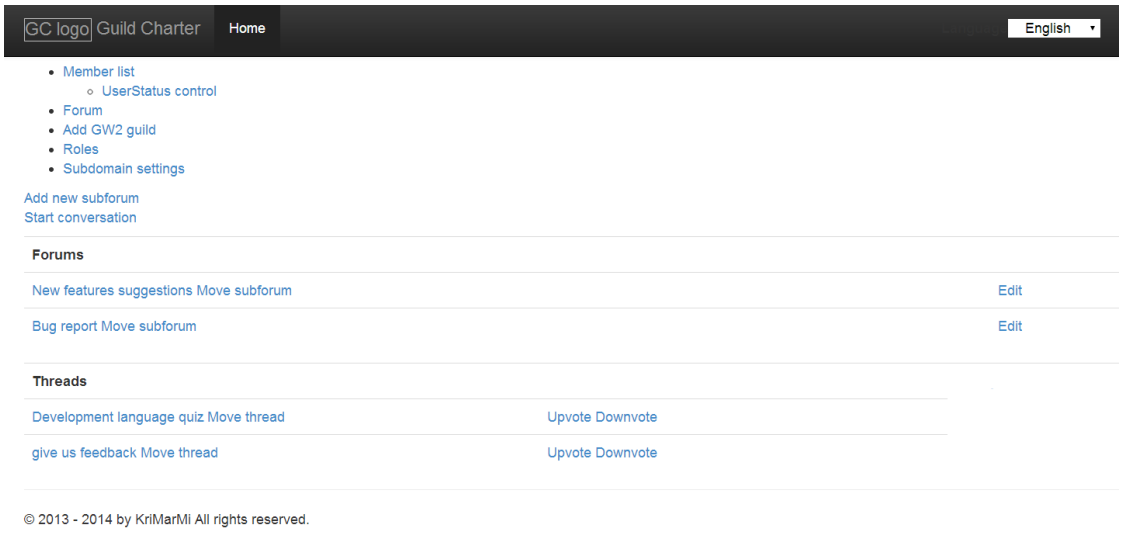
3.5 Keskustelualue

Keskustelualue päätettiin luoda itse alusta alkaen, jotta sen jatkokehitys olisi vähemmän sidoksissa ulkopuoliseen koodiin. Jos sovelluksessa käytettäisiin valmista keskustelualuepalikkaa, sen toimintoja jouduttaisiin muokkaamaan ja uusia toimintoja lisäämään. Tällöin palikan päivitys saattaisi helposti rikkoa sovellusta varten muokatut ja lisätyt toiminnot. Jos palikan ei todettaisi soveltuvan sivustolle, sen vaihtaminen tai poistaminen voisi viedä liikaa aikaa ja tehdä koodista sekavaa. Tämä koskee kaikkia muitakin ulkoista koodia olevia palikoita. Projektissa koettiin järkeväksi, että kokoonpanoon ei lisätä uutta palikkaa keskustelualuetta varten.

Kun keskustelualueen perustoiminnot olivat valmiit, toteutettiin ensimmäiseksi kyselytyyppinen viestiketju. Se on viestiketju, joka sisältää kysymyksen ja vastausvaihtoehdot. Käyttäjät eivät tällöin vain lisää viestejä viestiketjuun, vaan he myös vastaavat kysymykseen. Kyselytyyppinen viestiketju päätettiin toteuttaa, koska siitä arveltiin olevan hyötyä jatkokehityksessä. Toimintoa voidaan hyödyntää esimerkiksi kysymällä käyttäjiltä, mitkä suunnitellut ominaisuudet ovat toivotuimpia. Toiminto on yhteisöjen käytettävissä niiden omilla alasivustoilla.

Toisena lisäominaisuutena toteutettiin keskusteluketjujen järjestäminen käyttäjien äänten perusteella. Kun ominaisuus on asetettu keskustelupalstalla käyttöön, käyttäjät voivat äänestää keskusteluketjuja ylös tai alas (Kuva 2). Sivuston alfa-versiossa keskusteluketjut järjestetään suoraan annettujen äänten perusteella. Ensin sovellus laskee jokaisen

keskusteluketjun ylös-äänet ja alas-äänet, jonka jälkeen ylös-äänistä vähennetään alas-äänet. Tämän jälkeen keskusteluketjut järjestetään erotusten tulosten mukaan.



Kuva 2. Alakeskustelualue, jossa keskusteluketjut on järjestetty käyttäjien antamien äänten perusteella.

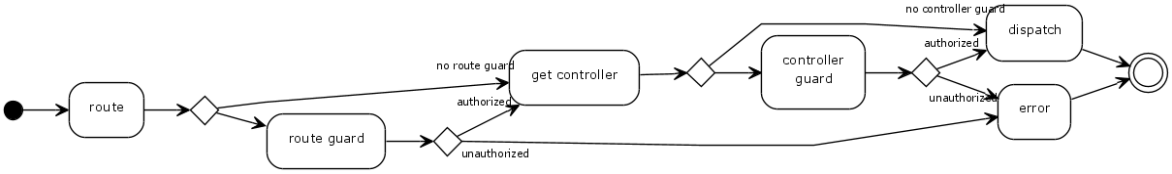
Yhteisö voi rakentaa keskustelualueensa vastaamaan yhteisön tarpeita. Esimerkiksi suuri yhteisö voi pilkkoa keskustelupalstoja alakeskustelupalstoiksi, jolloin koko keskustelualueen hallinta on helpompaa. Koska suuri yhteisö tuottaa paljon sisältöä, helpottaa keskustelualueen tarkka jaottelu sisällön selaamista. Pieni yhteisö voi puolestaan tehdä jaottelun pelkillä keskusteluryhmillä ja keskustelupalstoilla. Koska keskusteluketjut ovat aina keskustelupalstojen alla, käyttäjät eivät voi luoda keskusteluketjuja keskustelualueen juureen. Ratkaisu pakottaa yhteisön jakamaan keskustelualueensa keskustelupalstoiksi, jolloin sisällön selaaminen on helpompaa.

3.6 Roolit

Roolien oikeuksien hallinta toteutettiin BjyAuthorize-moduulilla, joka toimii palomuurin tavoin estäen luvattoman pääsyn ohjaimiin tai reitteihin (Bjyoungblood). Reitit ovat Zend Framework 2:n tapa välittää palvelimelle tulevat pyynnöt ohjaimiin. Viitekehys pyrkii löytämään reitin, joka soveltuu vastaanottamaan selaimelta tulleen pyynnön. Reitille siis määritellään, mitkä osoitteet se vastaanottaa. Reitille asetetaan myös ohjain,

joka käsittelee reitille välitetyt pyynnöt. Kuviossa 5 on esitetty, kuinka pyyntö käsitellään, kun auktorisointi ei ole käytössä. Kuviossa 6 on esitetty pyynnön käsittely, kun BjyAuthorize vastaa auktorisoinnista. BjyAuthorize tarkistaa, onko käyttäjällä rooliensa kautta oikeus pyydettyyn sivuun. Reitit välittävät pyynnöt aina johonkin ohjaimeen, jolloin ohjaimeen voi tulla pyyntöjä useista eri reiteistä. Ohjaimen toiminnallisuus on kuitenkin aina sama ja juuri toiminnallisuus halutaan suojata käyttäjiltä, joilla ei ole oikeutta toiminnon käyttöön. Toiminnallisuus suojataan paremmin, kun suojaus asetetaan ohjaimelle reittien sijaan. Tällöin uudetkin reitit ohjaimeen tulevat suojatuiksi jo ennen niiden luontia.

Kuvio 5. Normaali Zend\Mvc mallin kulku (Youngblood, B. 2013).



Kuvio 6. Kuvio 6. Zend\Mvc mallin kulku BjyAuthorizen kanssa (Youngblood, B. 2013).

Alasivuston ylläpitäjä voi roolien avulla hallita muiden jäsenten pääsyä tietyille alasivuston alueille. Roolien avulla voidaan myös määrittää tarkemmin oikeuksia alasivuston sisältöön, esimerkiksi keskustelalueen tiettyihin keskustelupalstoihin tai keskusteluketjuihin. Ylläpitäjä voi siis lisätä ja muokata rooleja siten, että ne tukevat yhteisön hierarkiaa ja toimivat joustavasti yhteisön tarpeiden mukaan.

Roolit voivat periä oikeuksia toisilta rooleilta, mutta kuitenkin vain yhdeltä roolilta kerrallaan. Käyttäjillä tosin voi olla useita rooleja. Roolin on mahdollista periä oikeuksia useammalta roolilta, jos perittävä rooli perii oikeuksia kolmannelta roolilta. Esimerkiksi rooli A perii oikeuksia rooli B:ltä. Rooli B perii oikeuksia rooli C:ltä. Rooli A perii rooli C:n oikeudet rooli B:n kautta. Pitkä periytymisketju tekee kuitenkin roolin oikeuksien seurannan vaikeaksi, joten periytymistä on hyvä käyttää harkiten. Roolit, jotka luodaan

tietokannan muodostamisen yhteydessä, eivät peri oikeuksia muilta rooleilta. Oikeuksien perintä jätetiin pois, jotta roolien hallinta olisi mahdollisimman selkeää ylläpitäjille. Tietokannassa valmiina olevat roolit ovat vierailija, vieraileva käyttäjä, käyttäjä, ylläpitäjä ja omistaja. Kaikilla yhteisön jäsenillä on Käyttäjä-rooli, jonka kautta käyttäjä saa perusoikeudet alasivuston selaamiseen. Vierailija-rooli annetaan käyttäjälle, joka ei ole kirjautunut sivustolle. Sitä ei koskaan käytetä rekisteröityneen käyttäjän roolina. Vieraileva käyttäjä -rooli annetaan kirjautuneelle käyttäjälle, joka selaa alasivustoa, jonka jäsen hän ei ole. Näin yhteisö voi rajoittaa pääsyn alasivustolleen erikseen vierailijoilta ja vierailevilta käyttäjiltä. Muut roolit, kuten ylläpitäjä, omistaja ja ylläpitäjien luomat roolit, täydentävät käyttäjän oikeuksia.

3.7 Tietoturva

Ylläpitäjällä on täydet oikeudet alasivustoon, mutta hän ei voi poistaa ylläpito-oikeutta itseltään. Hän ei voi myöskään antaa Ylläpitäjä-roolia muille käyttäjille, sillä ainoastaan omistajalla on oikeus jakaa ylläpito-oikeuksia. Ylläpitäjä voi alasivuston asetuksia säätämällä valita, mitkä alasivuston osat ovat käytössä. Sivuston alfa-versiossa ei ole muita osia kuin keskustelualue, joka on oletuksena käytössä.

Sivusto keskittyy pääasiassa yhteisön jäsenten tuottamaan sisältöön. Tämä tarkoittaa sitä, että sivustolle syötetään jatkuvasti uutta tietoa. Osa tiedoista voi olla suunniteltu vahingoittamaan sivuston toimintaa tai murtamaan sen tietosuoja. Tämän vuoksi kaikki käyttäjän lähettämä tieto validoidaan aina palvelimella ennen sen käsittelyä. Kun sovellus vastaanottaa käyttäjän täyttämän lomakkeen, se tarkistaa, että kaikki vaaditut kentät on täytetty ja että kenttien sisältö on oletettua. Esimerkiksi pudostusvalikon arvosta tarkistetaan, että saatu arvo löytyy annetuista vaihtoehdoista. Päivämäärästästä tarkistetaan puolestaan tietyissä tilanteissa, että annettu päivämäärä on tulevaisuudessa. Näin voidaan välttää mahdolliset injektiohyökkäykset ja pitää tietokanta eheänä. Zend Framework 2 tarjoaa sovelluksen käyttöön monia valmiita validointi- ja suodatusehtoja, joita jouduttiin kuitenkin täydentämään sovelluksen omiin tarpeisiin kehitetyillä validaattoreilla ja filttereillä. Lisäksi Doctrine 2 antaa oman suojansa tietokantaan kohdistuvia injektiohyökkäyksiä vastaan.

Sovelluksessa on käytössä ohjaimiin perustuva auktorisointi. Oletuksena auktorisointi evää kaikki pyynnöt, joten jokaiselle ohjaimelle tai sen metodille on määriteltävä roolit, joiden on sallittu käyttää ohjainta tai sen metodia. Aina kun käyttäjä pyytää palvelimelta uutta sivua, pyyntö kulkee ohjaimen kautta. Näin jokainen ohjain on täysin suojattu, kunnes sille on määritelty roolit, joilla on oikeus käyttää sitä.

3.8 Toteutuskuvaus

Projekti eteni hyvin alusta alkaen, sillä työskentely aloitettiin toiminnallisuuksien suunnittelulla ja relaatiokannan luonnilla. Sovelluksen kooditason rakenne suunniteltiin ennen ohjelmointityön aloitusta. Kun koodaus aloitettiin, luotiin ensimmäisenä kaikki olioluokat ja ne mapattiin vastaamaan luotua kantaa. Tehty suunnittelu- ja pohjustustyö tietokannan ja Zend Framework 2:n kanssa mahdollistivat kehitystyön sulavuuden ja tasaisen etenemisen. Jos alussa ei olisi tehty pohjustusta, olisi liiketoimintalogiikan kehitys edennyt hitaammin. Etukäteen luotuina oliot olivat molempien kehittäjien käytettävissä alusta alkaen, jolloin myös pienennettiin riskiä tehdä sama työ kahdesti. Näin kehittäjät pystyivät keskittymään liiketoimintalogiikan toteutukseen, sillä heidän ei tarvinnut miettiä tietokantayhteyksiä toimintalogiikan ohella.

Sisäänkirjautuminen ja rekisteröityminen sivustolle toteutettiin ZFCUser-moduulilla. Koska sovellus ja tietokanta ovat monimutkaisia, moduulin toimintaa oli täydennettävä, jotta se täytti sovelluksen tarpeet. Moduulin toimintaa muokattiin lisäämällä käyttäjän tietojen tallentuminen istunnon muistiin sisäänkirjautumisen yhteydessä. Lisäksi rekisteröitymislomakkeeseen lisättiin rekisteröitymiskoodi, jota ilman rekisteröitymistä ei hyväksytä. Rekisteröitymiskoodin saa alfa-vaiheessa ainoastaan suoraan kehittäjiltä, jotka voivat sovelluksen kautta luoda uusia koodeja tarpeen mukaan. Rekisteröitymisen yhteydessä tietokantaan tallentuvat käyttäjän antamat tiedot, joita ovat sähköposti, käyttäjätunnus ja salasana. Salasana suolataan ja kryptataan ennen kantaan tallentumista. Rekisteröitymistä täydennettiin lisäämällä rekisteröitymispäivä tallentuviin tietoihin. Oikeuksien tarkastelussa päätettiin käyttää BjyAuthorize-moduulia, koska auktorisointiprosessin rakentaminen itse olisi ollut työlästä. Lisäksi itse toteutettu auktorisointi sisältää luultavammin virheitä kuin moduuli, joka on jo todettu toimivaksi. Moduulin avulla rajoitetaan pääsyä ohjaimiin ja ohjainten metodeihin käyttäjän roolien mukaan. Tällöin

esimerkiksi voidaan rajata pääsy osaan palvelua vain kirjautuneille käyttäjille tai näyttää alasivuston hallintapaneeli vain ylläpitäjille.

Keskustelualuetta lähdettiin rakentamaan jo projektin alkuvaiheessa, koska sen arvioitiin vievän paljon aikaa. Keskustelualueen juuressa on keskustelupalstaryhmiä, jotka kokoavat sisälleen samaan aihe-alueeseen kuuluvat keskustelupalstat. Keskustelupalstat voivat sisältää keskusteluketjujen lisäksi alakeskustelupalstoja. Keskusteluryhmiä voi olla yksi tai useampia riippuen keskustelualueen koosta ja luonteesta.

4 Yhteenveto

Kokonaisuudessaan projekti eteni odotetusti ja tavoitteet saavutettiin. Käyttöliittymän toteutuksen viivästyminen oli odottamaton käänne, mutta se ei vaikuttanut opinnäytetyön valmistumiseen. Sovellus on annettu testikäyttöön muutamalle henkilölle ja testikäyttäjien määrää on tarkoitus kasvattaa. Kehitystyön ongelmatilanteet selvitettiin usein nopeasti ja ratkaisut olivat kestäviä. Sovelluksen jatkokehitystä varten suunniteltiin kokonaan uusia ominaisuuksia ja parannuksia olemassa oleviin toimintoihin.

4.1 Tulokset

Työssä laadittiin projektisuunnitelma, relaatiokaavio tietokannoista, editysmisraportit, suunitteludokumentti, määritysdokumentti ja vaatimusmääritysdokumentti. Projektin aikana koodin rakennetta kehitettiin ja uutta rakennetta noudatettiin siitä eteenpäin. Jo valmiiden osien uudelleenkirjoitus päätettiin kuitenkin jättää toteutettavaksi opinnäytetyön valmistumisen jälkeen.

Sivustolle kehitettiin suuri määrä toimintoja, joista tärkeimmät on käsitelty luvussa 2. Sen vuoksi toiminnot on kuvattu tässä luvussa hyvin lyhyesti.

Rekisteröityminen sivustolle	Käyttäjä määrittää itselleen käyttäjätunnuksen, salasanan ja antaa sähköpostiosoitteen. Käyttäjän on annettava rekisteröitymiskoodi, jotta hän voi rekisteröityä palvelun alfa-versioon.
Sisäänkirjautuminen sivustolle	Käyttäjä voi kirjautua sivustolle sähköpostiosoitteella ja salasanalla.
Profiilin editointi	Käyttäjä pystyy päivittämään omia tietojaan profiilisivulla. Muokattavia tietoja ovat esimerkiksi sähköpostiosoite, salasana, pg-arvo, kuva ja lähin kaupunki.
Julkinen lista alasivustoista	Lista alasivustoista, joita ei ole määritelty näkymättömiksi. Kirjautuneet käyttäjät voivat lähettää pyynnön päästä yhteisön jäseneksi.

Hakemuksen lähetys yhteisölle	Käyttäjä valitsee julkiselta listalta haluamansa alasivuston ja lähettää sinne liittymispyynnön. Hakemuksessa on lähettäjän näyttönimi ja mahdollinen saateviesti.
Alasivuston hankinta yhteisön käyttöön	Käyttäjä voi halutessaan hankkia alasivuston omalle peliyhteisölleen. Sovelluksen alfa-versiossa jokainen käyttäjä voi luoda vain yhden alasivuston.
Alasivuston tietojen päivitys	Ylläpitäjä voi päivittää alasivuston kuvausta.
Alasivuston asetusten muokkaus	Ylläpitäjä voi säätää roolien oikeuksia, päättää näkyykö alasivusto pääsivuston jukisessa alasivustolistassa ja valita alasivuston aktiiviset osat. Alfa-versiossa ainoa alasivuston osa on keskustelualue.
Roolien lisäys, muokkaus ja poisto	Ylläpitäjä voi luoda uusia rooleja ja muokata niiden oikeuksia. Oletusrooleja ei voi poistaa ja niistä vain muutaman oikeuksia voidaan muokata. Tällä turvataan alasivuston toimivuus.
Hakemusten hyväksyntä ja hylkäys	Henkilö, jolla on oikeus hakemusten käsittelyyn, voi hyväksyä tai hylätä alasivustolle saapuneita hakemuksia.
Ylläpitäjä-roolin hallinta	Omistaja voi antaa ylläpitäjä-roolin muille käyttäjille tai itselleen. Hän voi myös poistaa ylläpitäjä-roolin käyttäjältä.
Roolin omaavien käyttäjien hallinta	Ylläpitäjä voi määrittää ja poistaa rooleja käyttäjiltä muokkaamalla roolin käyttäjälistaa.
Keskustelupalstan lisäys, muokkaus ja siirto	Ylläpitäjä voi luoda keskustelupalstoja keskustelualueelle. Keskustelupalstan nimeä voidaan muokata jälkikäteen. Keskustelupalsta voidaan siirtää toisen keskustelupalstan tai keskustelupalstaryhmän sisälle.

Keskusteluketjujen äänestys keskustelupalstalla	Ylläpitäjä voi asettaa keskusteluketjujen äänestystoiminnon keskustelupalstalle. Tällöin keskusteluketjujen äänestys on käytössä ja keskusteluketjut järjestyvät käyttäjien antamien äänten perusteella.
Keskusteluketjun lisäys, muokkaus ja siirto	Käyttäjä voi lisätä alasivuston keskustelualueelle keskusteluketjuja. Ylläpitäjä voi muokata keskusteluketjun nimeä tai siirtää keskusteluketjun toisen keskustelupalstan sisälle.
Keskusteluketjujen selaaminen	Viestit on järjestetty keskusteluketjussa aikajärjestyksessä vanhimmasta uusimpaan. Viestin yhteydessä näkyy, milloin se on lähetetty ja kuka sen on lähettänyt.
Viestien lisäys keskusteluketjuun	Viestien lisäys olemassa olevaan keskusteluketjuun.
Loki keskustelualueen viestien muokkauksista	Käyttäjä voi tarkastella keskustelualueen viesteihin tehtyjä muutoksia ja alkuperäistä viestiä lokin avulla.
Käyttäjän lukitseminen väliaikaisesti ulos keskustelualueelta	Ylläpitäjä tai käyttäjä, jolla on muokkausoikeus keskustelualueella, voi estää väliaikaisesti häiriköivän käyttäjän pääsyn keskustelualueelle.
Yhteisön Guild Wars 2 killan tietojen haku	Ylläpitäjä voi tallentaa killan nimen, logon ja lyhenteen sivustolle. Kun sivustolle annetaan killan nimi, sovellus hakee logon ja lyhenteen Guild Wars 2 API:n kautta.
Lokalisointi	Sivuston alfa-versio tukee suomea ja amerikanenglantia. Päivämäärät ja kellonajat muotoillaan valitun kielen mukaan.

4.2 Parityöskentely

Opinnäytetyö tehtiin parityönä. Varsinkin projektin alkuvaiheessa töitä tehtiin kokoontumalla samaan paikkaan, jotta keskustelu kehitettävistä toiminnoista olisi helpompaa. Projektin edetessä tehtiin enemmän etätyötä ja yhteydenpito toteutettiin verkkopuhe-

luiden avulla. Työt jaettiin osa-alueittain, jolloin vältyttiin päällekkäiseltä työltä. Kummallakin tekijällä oli omat vastuualueensa. Koko projektin ajan tehtiin tiivistä yhteistyötä ja keskusteltiin eri ratkaisutavoista niin ohjelman käytettävyyden kuin myös koodin toteutustavan osalta.

Vastuuhenkilö toteutti aina suurimman osan vastuualueensa ominaisuuksista, vaikka välillä toimintoja kehitettiin omien vastuualueiden ulkopuolella. Kaikki osallistuivat alkutöihin, mutta työympäristön pystytys ja sovelluksen alustus olivat Markon vastuulla. Alkutöihin kuuluivat virtuaalikoneen pystytys Vagrantin avulla, relaatiotietokannan ja sovelluksen rakenteen suunnittelu, olioiden luonti ja niiden yhdistäminen tietokantaan sekä rekisteröityminen ja sisäänkirjautuminen. Markon muina vastuualueina olivat alasivuston hankinta, lokalisointi, Guild Wars 2 API, alfa-koodien hallinta ja käyttö, auktorisointi, hakemusten lähetys ja saapuneiden hakemusten käsittely. Kristan päävastuualueina olivat keskustelualue, roolien hallinta ja roolien oikeuksien hallinta. Nämä olivat laajoja kokonaisuuksia sisältäen monia toimintoja, joten vastuualueiden tasapainottamiseksi Kristalle ei asetettu enempää vastuualueita. Omien vastuualueidensa lisäksi molemmat kehittivät osin muita toimintoja kuten profiilin tietojen päivitystä ja alasivuston asetusten hallintaa.

4.3 Haasteet kehitystyössä

Projektissa tuli vastaan muutamia haastavia tilanteita. Eräs niistä aiheutui Zend Framework 2 -viitekehyksessä olevasta virheestä. Virhe ilmeni, kun toteutettiin kysymyskeskusteluketjun luontia keskustelualueelle. Kysymysten ja vastausvaihtoehtojen määrittelyssä käytetyt tiedonsyöttökentät eivät muodostuneet näkymään. Virhe korjaantui viitekehykseen tulleessa päivityksessä. Tämän jälkeen ilmeni toinen ongelma. Kaikki lomakkeen tiedot eivät tallentuneet oikein olioihin. Osa tiedoista puuttui, mutta tiedon siirtoa lomakkeesta olioihin ei haluttu kirjoittaa itse. Vaikka tiedon siirto olisi ollut helppo kirjoittaa, ei se olisi ollut yhtä joustava muutoksille kuin viitekehyksen tekemä automaattinen tiedonsiirto. Ratkaisuksi valittiin toiminnon yksinkertaistaminen, kunnes virheet viitekehyksessä on korjattu. Keskusteluketjuun voi toistaiseksi valita vain yhden kysymyksen, jolle määritellään vastausvaihtoehdot. Keskusteluketjuihin lisätään myöhemmin mahdollisuus esittää useampia kysymyksiä. Tämä aiheuttaa sen, että koodia on

uudelleenkirjoitettava toimintoa täydennettäessä. Tämä on kuitenkin pieni vaiva suhteutettuna siihen, että ominaisuus saatiin toimimaan.

Käyttöliittymän kehityksen siirtyminen myöhemmäksi toi omat haasteensa. Koska sivuston rakenteen suunnittelu oli osa käyttöliittymäopinnäytetyötä, osa palvelinkoodista voidaan joutua kirjoittamaan uudelleen, jos rakenteeseen tehdään laajoja muutoksia. Tästä syystä rakennetta pyrittiin suunnittelemaan samalla, kun kehitettiin toimintoja.

Vieraileva käyttäjä -roolin asetus käyttäjälle, joka on vierailemassa alasivustolla, osoittautui haastavammaksi kuin arvioitiin. Kirjautuneelle käyttäjälle määritetään roolit tietokannassa olevien tietojen mukaan. Tiedot rooleista tallennetaan alasivuston tietokantaan, jolloin vierailevalla käyttäjällä ei ole rooleja. Jokaiselle vierailevalle käyttäjälle ei ole järkevää tallentaa tietoa rooleista. Näin vältytään siltä, että tietokantaan muodostuu paljon turhaa tietoa. Ongelman ratkaisemiseksi päätettiin muokata BjyAuthorize-moduulin toimintaa. Metodi, joka välittää käyttäjän roolit, kirjoitettiin uudelleen pienin muokkauksin, jotta roolittomalle kirjautuneelle käyttäjälle palautuu Vieraileva käyttäjä -rooli.

4.4 Oma oppiminen, Krista

Tietoni PHP:sta ja Zend Framework 2 -viitekehyksestä olivat projektin alussa vähäiset, minkä vuoksi Markolta kului hyvin paljon aikaa opettamiseeni. Projektin edetessä itsevarmuuteni kasvoi ja kykyni työskennellä itsenäisesti kehittyi. Päätökset koodiratkaisuista olivat kuitenkin usein Markon vastuulla.

Kehitystyön edetessä aloin huomata projektin alussa tehdyn määrittelytyön tärkeyden työmäärän ja ajankäytön hallitsemisessa. Sen ansiosta projekti eteni hallitusti ja olimme pystyneet ottamaan toimintoja kehitettäväksi sopivan määrän käytettävissä olevaan työaikaan nähden.

Ennen opinnäytetyön aloittamista en ollut miettinyt kunnolla, kuinka paljon työtunteja tulisi kulumaan yksinkertaisten toimintojen kuten listauksien ja poistojen toteuteuttamiseen. Vastuullani oli keskustelualueen, alasivuston asetusten ja roolien hallinnan kehittäminen. Roolien hallinta piti sisällään roolien luonnin, muokkauksen, poiston ja roolien oikeuksien muokkauksen. Keskustelualueen toteutus eteni hyvin, mutta alasivuston

asetukset ja roolien hallinta osoittautuivat haasteellisiksi. BjyAuthorize kuitenkin helpotti roolien oikeuksien hallintaa huomattavasti.

4.5 Oma oppiminen, Marko

Opinnäytetyön aikana kehitin eniten osaamistani Zend Framework 2 -viitekehyksestä. Olin aiemmin käyttänyt viitekehyksen lokalisointityökaluista vain kielikäännöstyökalua, joten päivämäärien ja sanojen monikkomuotojen muotoilu valitun kielen mukaan olivat uusia asioita. Opin paljon reitityksistä ja viitekehyksen toimintojen täydentämisestä kuten omien ViewHelpereiden ja Validaattoreiden kirjoituksesta. Toimintojen täydentäminen osoittautui hyvin tarpeelliseksi projektissa. ViewHelperit ovat aputyökaluja, joilla muotoillaan tietoa näkymässä ennen näkymän lähetetysta selaimelle. Validaattorit ovat lomakkeen kenttiin sidottavia tarkistusehtoja, joilla varmistetaan, että tieto on oikeassa muodossa. Myös Git-versiohallinnan käyttö on tullut minulle paljon tutummaksi. Sen käyttö projektissa suunnitellaan uudelleen opinnäytetyön jälkeen.

Vagrant oli minulle uusi työkalu. Sen opiskeluun ei ollut projektin alussa käytettävissä montaa tuntia, joten tulen tutustumaan sen käyttöön myöhemmin lisää. Vagrantin käyttö oli haastavaa, mutta se helpotti työtä huomattavasti, kun ympäristö oli onnistuneesti pystyssä usealla eri koneella. Erityisesti hyöty Vagrantista näkyi, kun tein jälkikäteen hienosäätöä kehitysympäristöön. Hyödyimme Vagrantista uudelleen, kun tuotantopalvelimen käyttöjärjestelmäksi valittiin CentOS, jonka jälkeen kehitysympäristön käyttöjärjestelmä oli vaihdettava samaksi. Käytimme Puphpet.com -palvelua Vagrant-asetustiedoston luomiseen. Tämä helpotti huomattavasti kehitysympäristön asentamista.

4.6 Jatkokehitys

Sovelluksen jatkokehittämistä varten olisi tärkeää refaktoroida koko projekti. Työn aikana on huomattu parempia tapoja järjestää koodi ja nimetä tiedostoja. Refaktorointia ei tehty opinnäytetyön aikana, sillä aika ei olisi riittänyt siihen. Refaktorointi olisi kuitenkin tarpeen tehdä mahdollisimman pian, jotta jatkokehitys nopeutuu.

Yksityisviestit olisivat käyttäjille hyödyllinen ominaisuus tilanteissa, joissa keskustelua ei haluta käydä julkisella kanavalla kuten keskustelualueella. Jos yksityisviestit toimisivat pääsivuston kautta, käyttäjä voisi olla yhteydessä muihin yhteisöihin kuuluviin käyttäjiin. Tällöin esimerkiksi yhteisön edustaja voisi olla yhteydessä jäsenhakemuksen lähettäjään ennen kuin hakemus hyväksytään.

Keskustelualueen toimintaa olisi myös hyvä kehittää. Esimerkiksi viestiketjujen järjestys äänestyksen perusteella vaatisi monimutkaisemman laskenta-algoritmin, joka ottaisi huomioon keskusteluketjun iän. Nykyisellään eniten ääniä saanut keskusteluketju pysyy aina ylimpänä, jolloin vanhat ja suositut keskusteluketjut pysyvät keskustelupalstalla aina ensimmäisinä ja uudet keskusteluketjut jäävät helposti käyttäjiltä huomaamatta. Lisäksi viestien muutoslokin selausoikeus voisi olla ylläpitäjän asetettavissa. Näin ylläpitäjä voisi estää peruskäyttäjää näkemästä muutoksia, vaikka peruskäyttäjällä olisikin oletuksena oikeus muutoslokin lukuun.

Kalenteri auttaisi yhteisön jäseniä seuraamaan tulevia tapahtumia, joita yhteisö järjestää. Kalenterin merkinnät voisi liittää tiettyihin peleihin, jolloin käyttäjä voisi suodattaa kalenterista ne merkinnät, joista hän on kiinnostunut. Näin helpotettaisiin yhteisön johtamista, joka on usein haasteellista. Yhteisön jäsenten äänten kantautuminen johtohenkilöille saattaa usein rajoittua vain tiettyyn sisäpiiriin, jolloin uudet ja vähemmän aktiiviset jäsenet eivät saa mielipiteitään kuuluviin. Keskustelualueella voi muodostaa kyselykeskusteluketjuja, mutta ne saattavat hukkua muiden keskusteluketjujen joukkoon. Erillinen kyselytoiminto voisi palvella hyvin tilanteissa, joissa halutaan palautetta koko yhteisöltä. Aktiiviset kyselyt olisi helppo sisällyttää esimerkiksi yhteisön etusivulle, jolloin ne tavoittavat mahdollisimman monta jäsentä. Lisäksi kyselytoimintoa voitaisiin käyttää kyselykeskusteluketjun sijasta käyttäjäpalautteen keräämiseen.

Guild Wars 2 API on jo osittain integroitu palveluun, mutta koska API:a kehitetään vielä, sen kehitystä on seurattava pelin virallisella keskustelualueella ja virallisessa wikissä. Sivuston rajapinta API:n käyttöön pitäisi päivittää, kun API:sta julkaistaan uusi versio. API-tuki olisi hyvä laajentaa kattamaan muita pelejä. Esimerkiksi EVE Online

on hyvin vahvasti yhteisöjen varaan rakennettu MMO-peli, jolla on varsin pitkälle kehitetty API. API:n integrointi auttaisi varmasti houkuttelemaan uusia käyttäjiä ja yhteisöjä palveluun.

Hyvin tärkeä jatkokehityshanke on yhteisön vuokrauksen laskutus. Jos sivusto menestyy ja käyttäjämäärät kasvavat, palvelun ylläpito vaatii enemmän työtä. Tällöin tarvitaan tuloja, jotta voidaan palkata työntekijöitä kehittämään palvelua ja kommunikoimaan asiakkaiden kanssa. Toinen vaihtoehto laskutukselle ovat mainokset, mutta silloin sivuston visuaalinen ilme kärsisi, sillä suurin osa mainoksista ei yleensä istu sivustojen ulkoasuun kovin hyvin. Koska yhteisön ylläpito palvelussa tulee toimimaan vuokrausperiaatteella, täytyy asiakkaita pystyä laskuttamaan automaattisesti tietyin aikavälein, luultavimmin kuukausittain. Tämä vaatisi asiakkaiden luottokorttitietojen tallennusta, mikä pitäisi hoitaa erityisen huolellisesti. Tämän ominaisuuden toteutus olisi nykyiselle kehitystiimille liian suuri työ. Ominaisuus vaatisi todella tietoturvallista järjestelmää, jotta luottokorttitiedot eivät joutuisi vääriin käsiin. Luottokorttitietojen säilytys lisäisi riskiä joutua tietomurron kohteeksi, sillä luottokorttitiedot ovat houkuttava kohde rikollisille. Näin ollen olisi suositeltavaa tutkia muita vaihtoehtoja laskutuksen hoitamiseksi. On olemassa palveluita, jotka hoitavat laskutuksen palveluntarjoajan puolesta ja pitävät osan maksusta korvauksena laskutuksen hoidosta. Tälläinen palvelu siirtäisi luottokorttitietojen säilytyksestä koituvan vastuun pois Guild Charter -palvelulta.

Lähteet

Allen, R., Lo, N. & Brown, S. 2009. Zend Framework in Action. Manning Publications. Greenwich.

Ambler, S. Mapping Objects to Relational Databases: O/R Mapping In Detail. Ambysoft. Luettavissa: http://www.agiledata.org/essays/mappingObjects.html. Luettu: 17.3.2014.

Apple. 2013. Model-View-Controller. Luettavissa: https://developer.apple.com/library/mac/documentation/general/conceptual/devpedia-cocoacore/MVC.html. Luettu: 8.3.2014.

Baker, M. 2009. What is a Software Framework? And why should you like 'em? Luettavissa: http://info.cimetrix.com/blog/bid/22339/What-is-a-Software-Framework-And-why-should-you-like-em. Luettu: 8.3.2014.

Chacon, S. 2009. Pro Git. Luettavissa: https://github.s3.amazonaws.com/media/progit.en.pdf. Luettu: 13.4.2014.

Composer. Introduction. Luettavissa: https://getcomposer.org/doc/00-intro.md. Luettu: 29.3.2014.

Coury, E. 2014. ZfcUser. Luettavissa: https://github.com/ZF-Commons/ZfcUser. Luettu: 02.04.2014.

Doctrine Team. 2013. Getting Started With Doctrine. Luettavissa: http://docs.doctrine-project.org/projects/doctrine-orm/en/latest/tutorials/getting-started.html. Luettu: 2.4.2014.

Frey, R. 2010. MVC-Process.svg. Luettavissa: https://commons.wikimedia.org/wiki/File:MVC-Process.svg. Luettu: 30.3.2014.

Google. MVC Architecture. Luettavissa: http://developer.chrome.com/apps/app_frameworks. Luettu: 8.3.2014.

HashiCorp. a. About. Luettavissa: https://www.vagrantup.com/about.html. Luettu 29.3.2014.

HashiCorp. b. Trusted by. Luettavissa: https://www.vagrantup.com/. Luettu: 29.3.2014.

HashiCorp. c. Why vagrant? Luettavissa: https://docs.vagrantup.com/v2/why-vagrant/index.html. Luettu: 29.3.2014.

Janssen, C. Object-Relational Mapping (ORM). Luettavissa: http://www.techopedia.com/definition/24200/object-relational-mapping--orm. Luettu: 26.4.2014

Pastor, P. 2010. MVC for Noobs. Luettavissa: http://code.tutsplus.com/tutorials/mvc-for-noobs--net-10488. Luettu: 8.3.2014.

Php Frameworks. Luettavissa: http://www.phpframeworks.com/. Luettu: 16.3.2014.

Qian, K., Fu, X., Tao, L., Xu, C. & Díaz-Herrera J. 2009. Software Architecture and Design Illuminated. Jones and Bartlett Illuminated series. Jones & Bartlett Publishers. Sudbury, MA.

Riehle, D. 2000. Framework Desing: A Role Modeling Approach Ph D. Thesis, No. 13509. Zurich Sveitsi. http://www.riehle.org/computer-science/research/dissertation/diss-a4.pdf. Luettu 02.04.2014.

Stosic, M. 2013. ZfcUser, BjyAuthorize and Doctrine working together. Luettavissa: http://samminds.com/2013/03/zfcuser-bjyauthorize-and-doctrine-working-together/. Luettu 02.04.2014.

Strong, C. 2013. A Really Simple Explanation Of MVC. Luettavissa: http://www.strongandagile.co.uk/index.php/a-really-simple-explanation-of-mvc/. Luettu: 8.3.2014.

Treminio, J. 2013. Introduction to Vagrant/Puppet and introducing PuPHPet - A simple to use Vagrant/Puppet GUI Configurator! Luettavissa: https://jtreminio.com/2013/05/introduction_to_vagrant_puppet_and_introducing_puphpet_a_simple_to_use_vagrant_puppet_gui_configurator/. Luettu: 20.04.2014.

Youngblood, B. 2013. BjyAuthorize - Acl security for ZF2. Luettavissa: https://github.com/bjyoungblood/BjyAuthorize. Luettu 19.03.2014.

Liitteet

Liite 1. Relaatiokaavio (salainen)