Tuukka Teppola

PROTOTYYPPIJÄRJESTELMÄ MOBIILIPELIEN RISTIINMARKKINOINTIIN

PROTOTYYPPIJÄRJESTELMÄ MOBIILIPELIEN RISTIINMARKKINOINTIIN

Tuukka Teppola
Opinnäytetyö
Kevät 2015
Tietotekniikan koulutusohjelma
Oulun ammattikorkeakoulu

TIIVISTELMÄ

Oulun ammattikorkeakoulu
Tietotekniikan koulutusohjelma, ohjelmistokehityksen suuntautumisvaihtoehto

---

Tekijä: Tuukka Teppola
Opinnäytetyön nimi: Prototyyppijärjestelmä mobiilipelien ristiinmarkkinointiin
Työn ohjaaja: Juha Alakärppä
Työn valmistumislukukausi ja -vuosi: Kevät 2015
Sivumäärä: 49 + 2 liitettä

Työn tavoitteena oli suunnitella ja toteuttaa oululaiselle startup-peliyritykselle Frozen Visionille prototyyppijärjestelmä, jonka avulla mobiilipelit voisivat ristiinmarkkinoida toisiaan niihin ladattavilla mainoksilla. Mainokset ovat pelissä lista kuvalinkkejä sovelluskauppaan, josta mainostettava peli voidaan ladata.

Järjestelmä koostuu kahdesta PHP:llä ja Yii-sovelluskehyksellä toteutetusta verkkosovelluksesta sekä Unity 3D:llä toteutetusta päätelaitekomponentista. Verkkosovelluksista toinen on graafisella käyttöliittymällä varustettu sovellus mainosten hallinnointiin sekä niihin liittyvän datan seurantaan ja toinen niin sanottu REST API (verkkorajapinta), jonka kautta päätelaitekomponentti on yhteydessä mainostietokantaan. Mainostietokanta toteutettiin MySQL-relaatiotietokantana.

Prototyypille asetetut vaatimukset toteutettiin aikataulun mukaisesti. Alkuperäisestä suunnitelmasta poiketen jopa järjestelmän käyttöönottoa pilvipalvelussa ehdittiin testaamaan Microsoft Azuressa. Toteutetun järjestelmän avulla Frozen Visionilla on nyt mahdollisuus edelleen markkinoida ristiinmarkkinointiverkostoaan muille peliyrityksille. Lisäksi tämän työn tuloksista on hyötyä järjestelmän jatkokehityksen kannalta.

Asiasanat: Frozen Alliance, ristiinmarkkinointi, mobiilipelit, prototyyppi, järjestelmäriippumattomuus, pilvipalvelut

ABSTRACT

Oulu University of Applied Sciences
Bachelor of Engineering, Information Technology

Author: Tuukka Teppola
Title of thesis: Prototype system for cross-promotion of mobile games
Supervisor: Juha Alakärppä
Term and year when the thesis was submitted: Spring 2015
Pages: 49 + 2 appendices

The goal of this thesis was to design and implement a prototype system for Oulu based gaming start-up Frozen Vision. The system is for cross-marketing mobile games through an advert list inside every game that leads to a market place for downloading the advertised game.

The system consists of two web applications made with PHP and Yii framework and a client side component made with Unity 3D. The first one of the web applications is a management tool for the adverts and the second one is a REST API for the client side component. All the system data is stored in a MySQL relational database.

All the system requirements were met on time. Contrary to the original plan we were also able to test cloud deployment of the system to Microsoft Azure. With the help of this prototype Frozen Vision can now market their cross-marketing platform to possible partners. In addition, the results of this thesis will be useful for further development of the system.

Keywords: Frozen Alliance, cross-marketing, cross-platform, mobile games, prototype, cloud services

ALKULAUSE

Haluan kiittää Asmo Salorantaa mielenkiintoisesta ja monipuolisesta opinnäytetyön aiheesta. Kiitos myös ohjaajalleni Juha Alakärpälle, jolta Asmon lisäksi sain todella rohkaisevaa ja rakentavaa palautetta koko projektin ajan. Erityisen mieluisasta työilmapiiristä puolestaan kiitos kuuluu koko Frozen Visionin henkilöstölle.

2.4.2015

Tuukka Teppola

SISÄLLYS

SANASTO

AJAX	Tapa toteuttaa vuorovaikutteisuutta verkkosovelluksiin ilman, että verkkosivua täytyy ladata uudelleen.
API	Ohjelmistorajapinta, jonka avulla eri sovellukset voivat olla yhteydessä toisiinsa.
CRUD	Lyhenne sanoista create, read, update ja delete, jolla kuvataan tiedonhallintaan liittyviä operaatioita.
DAO	Tiedonhallintanobjekti (Data Access Object), jolla tietokantaa voidaan hallita ohjelmakoodista.
HTTP	Siirtoprotokolla, jota verkossa olevat palvelimet ja päätelaitteet voivat käyttää tiedonsiirtoon.
IaaS	Lyhenne sanoista Infrastructure as a Service, jolla kuvataan palvelua, jossa palvelimien hallinta on ulkoistettu pilvipalveluun.
IIS	Microsoftin kehittämä verkkopalvelinohjelmisto (Internet Information Services).
JSON	Yksinkertainen tiedonsiirtoon tarkoitettu merkintäkieli, jota myös ihminen pystyy helposti lukemaan ja kirjoittamaan.
MVC	Ohjelmistoarkkitehtuurityyli, jossa ohjelmisto jaetaan tiedonhallintaan, näkymiin ja sovelluksen käsittelyyn liittyviin osiin kehityksen hajauttamista varten.
PaaS	Lyhenne sanoista Platform as a Service, jolla kuvataan ohjelmiston julkaisualustaa tarjoavaa pilvipalvelua.
REST	Ohjelmistoarkkitehtuurityyli verkkorajapintojen toteuttamiseen, joka noudattaa HTTP-protokollaa.

SQL	Kyselykieli relaatiotietokantojen hallinnointiin (Structured Query Language).
XAMPP	Verkkokehitysympäristö, joka sisältää useita verkkopalvelinohjelmistoja.
XML	Merkintäkieli, jota käytetään mm. tiedonsiirtoon ja dokumenttien tallennusformaattina.
Yii	MVC-arkkitehtuuria noudattava sovelluskehys verkkosovellusten kehittämiseen PHP-ohjelmointikielellä.

1 JOHDANTO

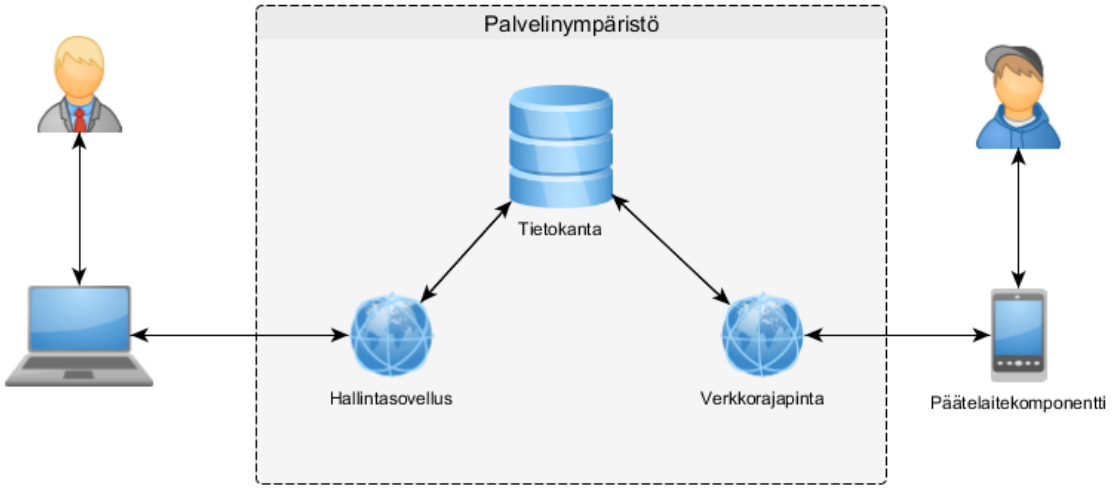
Menestykseen mobiilipelimarkkinoilla vaaditaan riittävää näkyvyyttä pelille, jotta asiakkaat löytävät juuri tuon pelin lukemattomien muiden vaihtoehtojen joukosta. Yksi tapa saada aikaan näkyvyyttä on ristiinmarkkinoida peliä toisten tuotteiden kautta. Esimerkiksi kempeleläisen Fingersoftin Hill Climb Racingin menestykseen johtaneet lataukset perustuivat kamerasovelluksilla toteutettuun markkinointiin (1). Monet peliyritykset toimivat myös julkaisijoina, jolloin niiden julkaisemat pelit saavat näkyvyyttä muiden julkaisujen kautta.

Frozen Vision on oululainen startup-peliyritys, joka sai alkunsa Oulun ammattikorkeakoulun Oulu Game LAB -ohjelmassa. Frozen Visionin tavoitteena on saada näkyvyyttä ja pelaajia mobiilipeleilleen ilman ulkopuolista julkaisijaa. Tätä varten he ovat rakentamassa Frozen Alliance -ristiinmarkkinointiverkostoa. Frozen Alliancen tarkoituksena on toimia palveluna, jonka kautta suomalaiset peliyritykset voivat ristiinmarkkinoida mobiilipelejään vaivattomasti ja ilman kustannuksia, joita vastaavanlaiset muut palvelut vaativat. Mukaan pyrkiville peleille tullaan kuitenkin asettamaan valintakriteerejä, jotta pelaaja voi luottaa verkostossa mainostettavien pelien laadukkuuteen.

Tämän työn tarkoituksena oli suunnitella ja toteuttaa tälle verkostolle prototyyppijärjestelmä perustoiminnallisuuksineen, jonka avulla Frozen Alliancen markkinointi voitaisiin aloittaa. Prototyypin ympärille olisi myöhemmin myös helpompi rakentaa varsinainen järjestelmä, kun tämän työn tuloksia voitaisiin käyttää sen jatkokehitykseen.

2 VAATIMUSMÄÄRITTELY

Frozen Alliance -järjestelmä koostuu neljästä solmukohdasta, joista tietokanta, hallintasovellus ja verkkorajapinta sijoittuvat palvelinympäristöön. Päätelaite-komponentti on sisällytetty loppukäyttäjän päätelaitesovellukseen, joka on tässä tapauksessa mobiilipeli. (Kuva 1.)



KUVA 1. Arkkitehtuurikaavio

2.1 Tietokanta

Tietokantakomponentti toteutetaan relaatiotietokantana, jota käytetään järjes-telmään liittyvän datan hallinnointiin keskitetysti. Tietokantaan pystytään tallen-tamaan tiedot verkostoon kuuluvista pelinkehittäjistä, heidän verkostoon hyväk-sytyistä peleistään sekä niihin liittyvistä mainoksista. Lisäksi mainoksien esitys- ja klikkauskertoihin liittyvä data tallennetaan.

Tietokanta rakennetaan siten, että sitä voidaan tarvittaessa laajentaa mahdolli-simman vaivattomasti. Tietokannan taulut normalisoidaan siten, että niistä jo-kainen kuvastaa vain yhtä entiteettiä eikä sama tieto kertaannu turhaan useas-sa eri taulussa.

2.1.1 Pelinkehittäjä

Pelinkehittäjään liittyvät seuraavat tiedot ovat tallennettavissa tietokantaan:

- peliyrityksen nimi
- yhteyshenkilön nimi
- yhteyshenkilön sähköpostiosoite
- yhteyshenkilön puhelinnumero
- kotisivu
- lähiosoite
- postinumero.

Postitoimipaikkaa ei tallenneta tietokantaan, vaan käytetään ulkopuolista postinumeropalvelua, jonka kautta se tieto saadaan selville, eikä käyttäjänkään tarvitse sitä siten syöttää järjestelmään.

2.1.2 Peli

Peli yhdistetään pelinkehittäjään ja tietokantaan tallennetaan tiedot sen nimestä, julkaisualustoista sekä tehostuspäivämäärästä. Tehostuspäivämäärää käytetään järjestelmässä myöhemmin tehostamaan mainosten valikoitumista esitettäväksi päätelaitteella.

2.1.3 Mainos

Mainoksesta tietokantaan on tallennettavissa otsake, mainosteksti, esitys- ja klikkauskertojen määrä, aktiivisuuden tila sekä sen sovelluskauppaosoitteet. Mainokseen liittyy myös kuva, joten siitä tallennetaan tiedot sen sijainnista, tiedostotyypistä ja koosta.

2.1.4 Mainoksen esitykset

Tietokantaan tallennetaan tiedot esitetystä mainoksesta, esityksen ajankohdasta sekä päätelaitteen uniikista tunnisteesta ja maantieteellisestä sijainnista maakohtaisesti. Lisäksi tallennetaan tiedot pelistä ja sen laitealustasta, jolta mainos on esitetty. Esityksen ajankohtaa, päätelaitteen uniikkia tunnistetta ja esitettyä mainosta käytetään erottelemaan esitykset toisistaan.

2.1.5 Mainoksen klikkaukset

Tietokantaan tallennetaan tiedot klikatusta mainoksesta, klikkauksen ajankohdasta sekä päätelaitteen uniikista tunnisteesta ja maantieteellisestä sijainnista maakohtaisesti. Lisäksi tallennetaan tiedot pelistä ja sen laitealustasta, jolta mainosta on klikattu. Klikkauksen ajankohtaa ja päätelaitteen uniikkia tunnistetta käytetään erottamaan klikkaukset toisistaan.

2.2 Hallintasovellus

Hallintasovellus on salasanalla ja käyttäjätunnuksella suojattu verkkosovellus. Käyttäjän hakeutuessa sivustolle hänet ohjataan ensin kirjautumisnäkymään, jonka kautta hänen on mahdollista tunnistautua järjestelmään. Ennen tunnistautumista kaikki sovelluksen muu toiminnallisuus on piilotettuna ja kaikki sovellukseen kohdistuneet kutsut ohjataan kirjautumisnäkymään. Sovelluksen prototyyppivaiheessa on vain yksi pääkäyttäjätaso ja tunnukset ovat kovakoodattuna sovelluksen lähdekoodissa, kunnes varsinainen tietoturvallinen tunnistautuminen toteutetaan.

Käyttäjän tunnistauduttua järjestelmään sovellus ohjaa hänet etusivulle, jolle tässä työssä toteutetaan vain linkki Frozen Visionin kotisivuille. Sovelluksessa on navigaatiopalkki, jonka kautta käyttäjällä on nyt mahdollista siirtyä hallinnoimaan järjestelmän pelinkehittäjiä, pelejä ja mainoksia. Navigaatiopalkin kautta on käyttäjän mahdollista myös kirjautua ulos sovelluksesta.

2.2.1 Pelinkehittäjät

Sovelluksen pelinkehittäjät-osion kautta käyttäjällä on mahdollista luoda uusia pelinkehittäjiä järjestelmään, tarkastella olemassa olevien pelinkehittäjien tietoja, muokata olemassa olevia pelinkehittäjiä ja poistaa niitä järjestelmästä. Uutta pelinkehittäjää järjestelmään luotaessa on pelinkehittäjästä pystyttävä syöttämään seuraavat tiedot: yrityksen nimi, yhteyshenkilön nimi, sähköpostiosoite, puhelinnumero, kotisivu, lähiosoite ja postinumero. Jokaiselle syötettävälle tiedolle on oma tekstikenttänsä luontinäkymässä ja lisäksi on kenttä paikkakunnalle, joka ei ole muokattavissa. Kun postinumero on syötetty, paikkakuntakenttä

täytetään postinumeron perusteella automaattisesti. Vain yrityksen nimi on pakollinen tieto.

Pelinkehittäjän tietoja tarkasteltaessa sille aiemmin syötetyt tiedot esitetään. Tarkastelunäkymästä on mahdollista siirtyä muokkaamaan näitä tietoja. Tietojen muokkaaminen tapahtuu kuten pelikehittäjää luotaessa. Tarkasteltava pelinkehittäjä on mahdollista poistaa, mikäli siihen ei järjestelmässä ole yhdistettynä yhtään peliä. Kaikki järjestelmän pelinkehittäjät on mahdollista listata taulukkoon, josta niitä voidaan järjestellä ja hakea jollakin pelinkehittäjään liittyvällä ominaisuudella.

2.2.2 Pelit

Sovelluksen peliosion kautta käyttäjällä on mahdollista luoda uusia pelejä järjestelmään, tarkastella olemassa olevien pelien tietoja, muokata olemassa olevia pelejä ja poistaa niitä järjestelmästä. Uutta peliä järjestelmään luotaessa on pelistä pystyttävä syöttämään seuraavat tiedot: pelin nimi, pelinkehittäjä, tehostuspäivämäärä ja sovelluskauppalinkit järjestelmään asetetuille laitealustoille. Pelin nimi ja sovelluskauppalinkit ovat tekstikenttiä, mutta pelinkehittäjä valitaan järjestelmän pelinkehittäjistä alasvetovalikon avulla. Tehostuspäivämäärän asettamiseen käytetään päivämäärävalitsinta. Pelin nimi ja sen kehittäjä ovat pakollisia tietoja.

Pelin tietoja tarkasteltaessa sille syötetyt tiedot esitetään. Tarkastelunäkymästä on mahdollista siirtyä muokkaamaan näitä tietoja. Tietojen muokkaaminen tapahtuu kuten peliä luotaessa. Tarkasteltava peli on mahdollista poistaa, mikäli siihen ei järjestelmässä ole yhdistettynä yhtään mainosta. Kaikki järjestelmän pelit on mahdollista listata taulukkoon, josta niitä voidaan järjestellä ja hakea jollakin peliin liittyvällä ominaisuudella.

2.2.3 Mainokset

Sovelluksen mainososion kautta käyttäjällä on mahdollista luoda uusia mainoksia järjestelmään, tarkastella olemassa olevien mainoksien tietoja, muokata olemassa olevia mainoksia ja poistaa niitä järjestelmästä. Uutta mainosta järjestelmään luotaessa on mainoksesta pystyttävä syöttämään seuraavat tiedot:

mainostettava peli, otsake, mainosteksti, kieli, aktiivisuuden tila ja mainoskuva. Mainoksen otsake ja mainosteksti ovat tekstikenttiä, mutta peli valitaan järjestelmän peleistä alasvetovalikon avulla. Lisäksi mainoksen kieli ja aktiivisuuden tila asetetaan alasvetovalikoilla. Kuvan asettamista varten on luontinäkymässä tiedostovalitsin. Mainostettava peli, kieli, aktiivisuuden tila ja mainoskuva ovat pakollisia tietoja. Vain yksi peliin liittyvä mainos voi kerrallaan olla aktiivisena, joten jos mainos asetetaan aktiiviseksi, kaikki muut saman pelin mainokset asetetaan automaattisesti epäaktiivisiksi.

Mainoksen tietoja tarkasteltaessa sille syötetyt tiedot esitetään asetetun kuvan sekä esitys- ja klikkausstatistiikan kanssa. Jokainen mainokseen liittyvä esitys- ja klikkauskerta esitetään taulukossa, josta niitä voidaan järjestellä ja hakea niihin liittyvien ajankohdan, pelin, laitealustan ja kielen mukaan.

Tarkastelunäkymästä on mahdollista siirtyä muokkaamaan mainoksen tietoja. Tietojen muokkaaminen tapahtuu kuten mainosta luotaessa, mutta uusi kuva ei ole pakollinen. Tarkasteltava mainos on mahdollista poistaa, mikäli siihen ei järjestelmässä ole yhdistettynä yhtään esitys- tai klikkauskertaa. Kaikki järjestelmän mainokset on mahdollista listata taulukkoon, josta niitä voidaan järjestellä ja hakea jollakin mainokseen liittyvällä ominaisuudella.

2.3 Verkkorajapinta

Verkkorajapinta toteutetaan verkkosovelluksena ilman varsinaista graafista käyttöliittymää. Se toimii REST-arkkitehtuurityylin mukaisesti ohjelmistorajapintana tietokannan ja päätelaitekomponentin välissä. HTTP-operaatioiden avulla verkkorajapinnan kautta on mahdollista hakea järjestelmästä mainoksia esitettäväksi sekä ilmoittaa esitetyistä ja klikatuista mainoksista. Vastaukset operaatioihin verkkorajapinta antaa JSON-tiedostomuodossa.

Esitettävät mainokset noudetaan HTTP-protokollan GET-metodilla, jonka parametreina annetaan päätelaitteen laitealusta, pyydettävien mainosten lukumäärä ja pyytävän pelin tunniste. Päätelaitteen laitealustan avulla valitaan sille sopivien pelien mainoksista pyydetyn lukumäärän mukainen määrä mainoksia, jotka eivät kuitenkaan kuulu pyytävälle pelille. Ensisijaisesti valitaan sellaisten pelien

mainoksia, joille on asetettu tehostuspäivämäärä, joka on kutsuhetkellä voimassa. Tämän jälkeen valitaan mainoksia, joilla on vähiten esityskertoja. Jos ei järjestelmästä löydy pyydettyä määrää mainoksia, valitaan vain löydetyt mainokset.

Ilmoitukset mainosten esittämisistä ja klikkauksista toteutetaan POST-metodilla. Mainoksen esittämistä ilmoitettaessa annetaan parametreina esitettyjen mainosten tunnisteet, esittävän pelin tunniste, päätelaitteen laitealusta, päätelaitteen uniikki tunniste ja päätelaitteen sijainnin maakoodi. Klikkausta ilmoittaessa parametrit ovat muuten samat, mutta vain yhden klikatun mainoksen tunniste annetaan. Annettujen parametrien perusteella verkkorajapinta luo tietokannan tauluihin tarvittavat tietueet.

2.4 Päätelaitekomponentti

Päätelaitekomponentti toteutetaan Unity 3D -ympäristössä ja siitä luodaan paketti, joka on helppo yhdistää osaksi Unityllä toteutettua peliä. Se sisältää käyttöliittymäobjektin mainosvalikosta, johon haetaan pelin käynnistyessä verkkorajapinnan kautta verkoston mainoksia esitettäväksi. Käyttäjä aktivoi mainosvalikon painikkeen avulla ja samalla verkkorajapinnalle lähetetään ilmoitus valikossa esitetyistä mainoksista. Valikossa mainoksista esitetään niiden mainoskuvat sekä lyhyt mainosteksti. Jos käyttäjä klikkaa jotain esitetyistä mainoksista, lähetetään verkkorajapinnalle ilmoitus ja avataan selaimeen mainostettavan pelin sivu sovelluskaupassa.

Komponentti selvittää mainoksia noutaessaan sen IP-osoitteen perusteella laitteen sijainnin ja tallentaa tämän sijainnin maakoodin muistiin myöhemmin käytettäväksi verkkorajapinnalle tehtävissä ilmoituksissa. Mikäli sijaintia ei kyetä selvittämään, ei maakoodia käytetä. Yhteysongelman tai internetyhteyden puuttuessa ei myöskään mainoksia kyetä noutamaan verkkorajapinnalta, jolloin käytetään vaihtoehtoista mainosta valikossa.

3 VALITUT TEKNOLOGIAT

3.1 MySQL

Päädyttiin valitsemaan MySQL-järjestelmä relaatiotietokantaohjelmistoksi, sillä se on yksi suosituimpia ratkaisuja maailmalla ja avoimen lähdekoodin ohjelmistona se on myös todella kustannustehokas. MySQL on suosituin tietokantaratkaisu PHP:tä käytettäessä ja yhdessä niillä on mahdollista toteuttaa järjestelmäriippumattomia sovelluksia (2). Internet on myös täynnä erilaisia oppaita ja käyttäjäkokemuksia MySQL:stä, joita on helppo hyödyntää. (3.)

MySQL on järjestelmä relaatiotietokantojen hallinnointiin. Relaatiotietokannassa tietokokoelma järjestellään tauluihin, jotka ovat yhteydessä toisiinsa muodostaen tietokannan kaavan. Näin data saadaan hajautettua siten, että sitä olisi mahdollisimman nopea hakea ja käsitellä. Tietokannan rakenne muodostetaan ensin, jonka jälkeen sitä käytetään sille asetettujen sääntöjen ja taulujen välisten suhteiden mukaisesti. MySQL-tietokantoja hallitaan standardoidulla SQL-kyselykielellä. (3.)

3.2 Yii

Yii on ilmainen avoimen lähdekoodin PHP-sovelluskehys verkkosovelluksien kehittämiseen. Se on rakennettu tehostamaan verkkosovelluksien kehitystä ja auttamaan monimutkaistenkin sovellusten ylläpidossa. Yii on todella suorituskykyinen ja se on erityisesti suunnattu erilaisten yrityshankkeiden toteuttamiseen, mutta sopii silti kaiken kokoisiin projekteihin. (4.)

Yiin keskeisimpiin ominaisuuksiin kuuluvat mm. MVC-arkkitehtuuri (model-view-controller), tietokannan käsittelyobjektit (DAO), verkkokaavakkeiden validointi, AJAX-kykyiset pienoisohjelmat, roolipohjainen käyttöoikeuksien hallinta (RBAC) ja automaattinen ohjelmakoodin generointi tietokannan CRUD-operaatioita (create-read-update-delete) varten. MVC-arkkitehtuurin avulla sovellus on helppo jakaa pienempiin loogisiin palasiin, jolloin esimerkiksi sovelluksen käyttöliittymää ja tietokannan hallintaa voidaan kehittää toisistaan erillään ja riippumattomina. Ohjelmakoodin generointi CRUD-operaatioita varten puolestaan no-

peuttaa huomattavasti sovelluksen alkuvaiheilla. Sen avulla valmiin tietokannan pohjalta voidaan luoda nopeasti valmiita luokkia tietokantaobjekteja varten, joilla hallitaan tietokannan tauluja ohjelmakoodista. Lisäksi samalla voidaan generoida ohjelmakoodia käyttöliittymä varten ja siten tietokannan taululle pystytään todella nopeasti toteuttamaan hallinnallisia toiminnallisuuksia. (5.)

Valmiin sovelluskehyksen valinta oli selvä tätä projektia ajatellen, sillä ilman sitä järjestelmän verkkosovellusten toteuttaminen olisi vienyt huomattavasti enemmän aikaa kuin tälle työlle oli varattuna. Yii valikoitui sen tehokkuuden vuoksi ja siksi, että siitä oli aikaisempaa kokemusta. Näin ollen ei tarvinnut varata aikaa uuden sovelluskehyksen opettelulle.

3.3 Tietokannat.fi

Tietokannat.fi on ilmainen postinumerorajapinta, jonka avulla voidaan suomalaisesta postinumerosta selvittää sen paikkakunta ja lääni. Tietokannat.fi-verkkosovellus käyttää suomenkielisen Wikipedian tarjoamia postinumerotietoja. Palautettava paikkakuntatieto on mahdollista saada suomeksi ja ruotsiksi joko XML- tai JSON-tiedostomuodossa. (6.)

Koska haluttiin toteuttaa pelinkehittäjän osoitetietojen syöttäminen mahdollisimman yksinkertaiseksi, päädyttiin paikkakunnasta vaatimaan vain tieto postinumerosta. Käyttäjän tarvitsee syöttää vain postinumero, jonka jälkeen järjestelmä hakee sen perusteella postinumerorajapinnasta paikkakunnan. Näin ollen myöskään järjestelmän omaan tietokantaan ei tarvitse paikkakuntatietoa tallentaa.

3.4 Unity 3D

Järjestelmän HTTP-protokollaa noudattava verkkorajapinta mahdollistaa yhteyden mainostietokantaan järjestelmäriippumattomasti. Frozen Vision on kuitenkin erikoistunut mobiilipeleihin ja heidän ensisijainen työkalunsa niitä kehittäessä on Unity 3D. Siten tätä järjestelmää ajatellen oli selvää, että päätelaitekomponentti tultaisiin toteuttamaan samassa kehitysympäristössä.

Unity 3D on monipuolinen ja kattava kehitysympäristö järjestelmäriippumattomien 3D- ja 2D-pelien kehittämiseen. Tällä hetkellä sillä on mahdollista kehittää pelejä 21 julkaisualustalle, joita ovat mm. PC, uuden sukupolven pelikonsolit ja yleisimmät mobiilialustat. Unity 3D:n suuri etu on myös sen kauppapaikka Unity Asset Store, jonka kautta kehittäjät voivat ostaa ja myydä valmiita pelikomponentteja. (7.)

Unity on todella suosittu eikä ainoastaan siksi, että työkaluista on ilmaisversio tarjolla. Sillä on kattava dokumentaatio ja yhteisö, joiden avulla ongelmien ratkominen ja nopea kehitys on todella helppoa.

3.5 Telize

Telize tarjoaa REST-arkkitehtuuria noudattavan verkkorajapinnan IP-osoitteen ja sen sijaintitiedon selvittämiseen. Rajapinnan käyttäminen on kaikille vapaata, eikä se sisällä rajoitteita kutsujen toteuttamiselle. (8.) Tämän palvelun avulla päätelaitteen sijainti pysytään selvittämään mainosten esitys- ja klikkaustietoja varten käyttämättä sen GPS-paikanninta.

Mobiilisovellusten on ilmoitettava käyttäjälle, jos ne tarvitsevat käyttöluvan GPS-paikantimen käyttöön. Näin ollen on mahdollista, että GPS-paikantimen käyttöluvan pyytäminen mobiilipelille voi aiheuttaa epäilyksiä käyttäjälle pelin luotettavuudesta. Lisäksi IP-osoitteeseen perustuva paikantaminen on nopeampaa, koska tarkkaa sijaintitietoa ei tarvita.

3.6 SimpleJSON

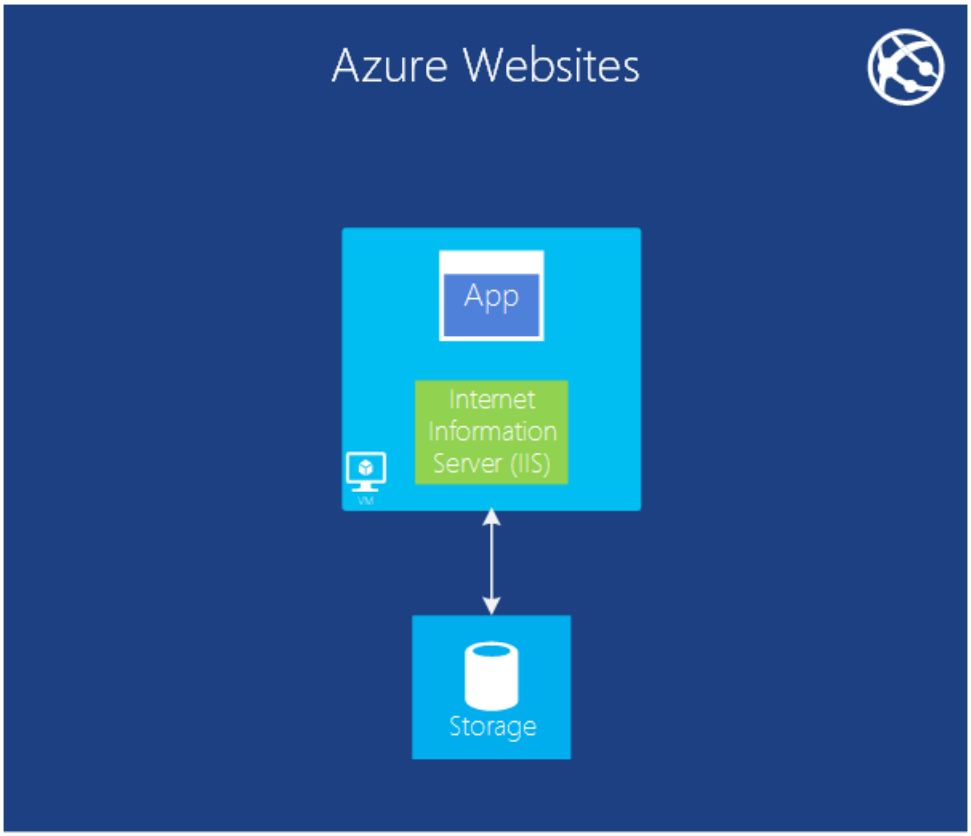
Verkkorajapintojen tarjoamien JSON-muotoisten vastauksien käsittelemiseen Unity 3D -ympäristössä tarvittiin ulkopuolinen luokkakirjasto. SimpleJSON valikoitui, koska se tukee C#-ohjelmointikieltä ja koska sen dokumentaation käyttöesimerkit tekivät sen käyttöönotosta todella helppoa. Yksinkertaisuudessaan SimpleJSON tarjosi kaikki tarvitut ominaisuudet JSON-tietueen purkamiseen käytettäväksi ohjelmakoodissa, muttei mitään ylimääräistä. Näin ollen päätelaitekomponentti ei turhaan sisällä laajaa luokkakirjastoa, jonka ominaisuuksista vain osa olisi käytössä. (9.)

3.7 Mircrosoft Azure

Projektin nopean etenemisen johdosta päädyttiin tutkimaan myös mahdollisuutta järjestelmän käyttöönotolle pilvipalvelussa. BizSpark-jäsenenä Frozen Visionilla on ilmainen pääsy Microsoftin Azure -pilvipalveluun, joten se oli jälleen helppo valinta (10). Lisäksi Azure tukee PHP:llä ja MySQL:llä toteutettuja verkkosovelluksia.

Microsoft Azure tarjoaa asiakkailleen sekä IaaS- (Infrastructure as a Service) että PaaS-tyyppisiä (Platform as a Service) palveluita eli hallintaa vaativia palvelinkeskus- ja hallintaa vaatimattomia julkaisualustapalveluita. Esimerkiksi IaaS-palveluna Azuressa voi ajaa ja hallita omia virtuaalikoneita, kun taas PaaS-palveluna on mahdollista julkaista verkkosovelluksia suoraan pilvipalveluun. Näiden lisäksi Azure tarjoaa myös palveluja mm. tiedon hallintaan, sovellusten väliseen kommunikointiin ja tunnistautumiseen. (11.)

Tässä työssä oltiin kiinnostuneita tietokannan ja kahden verkkosovelluksen käyttöönotosta pilvipalvelussa. Verkkosovelluksia varten käyttöön valikoitui Azure Websites -palvelu, jonka avulla valmiit PHP:llä ja MySQL:llä toteutetut verkkosovellukset voidaan suoraan ottaa käyttöön pilvipalvelussa. Azure Websites on käytännössä Azuren hallinnoima virtuaalikone tai joukko virtuaalikoneita, joissa verkkosovellusta ajetaan Microsoftin IIS-palvelimella (kuva 2). MySQL-palvelimen tarjoaa Microsoftin kumppani ClearDB kolmannen osapuolen palveluna. (12.)



KUVA 2. Azure Websites (12.)

Azure Websites mahdollistaa myös useiden instanssien ajamisen samasta sovelluksesta. Tästä johtuen mainoskuvien hallinnointiin keskitetysti tarvittiin oma palvelunsa, eikä niitä voinut tallentaa enää hallintasovelluksen kanssa samalle palvelimelle. Tätä varten valittiin Azure Storage -palvelu, joka mahdollistaa suurien binääriobjektien keskitetyn hallinnan sovellusten välillä, kunhan Azure SDK -sovelluskehitystyökalut ovat sisällytettynä sovellukseen. (12.)

4 TOTEUTUS

4.1 Tietokanta

Tietokannan kaava suunniteltiin ja toteutettiin MySQL Workbench -ohjelmistossa graafisella ER-kaaviotyökalulla. Kaava otettiin käyttöön kehitysvaiheessa ottamalla yhteys kehitysympäristön MySQL-palvelimelle ja siirtämällä kaava sinne. Testausvaiheessa kaava siirrettiin Azuren tarjoaman tietokantapalvelun MySQL-palvelimelle samalla toimenpiteellä. Varsinainen siirto toteutettiin siten, että ohjelmiston vientitoiminnolla luotiin kyselykielinen komentosarja kaavan muodostamiseen, jonka sitten ohjelmisto ajoi tietokantapalvelimen komentorivillä.

Pelinkehittäjän tietoja ei tarvinnut normalisoida useaan tauluun, sillä kaikki sen ominaisuudet ovat sille hyvin keskeisiä, eikä saman tiedon kertautuminen ole kovinkaan todennäköistä. Pelinkehittäjät päätettiin erotella toisistaan automaattisesti kasvatettavalla uniikilla kokonaislukutunnisteella, vaikka pelinkehittäjän uniikki nimikin olisi sen mahdollistanut. Tähän päädyttiin siksi, että hallintasovelluksessa käytetyn sovelluskehyksen olisi mahdollista generoida valmista ohjelmakoodia tietokantataulun CRUD-operaatioihin. (Kuva 3, taulu developer.)

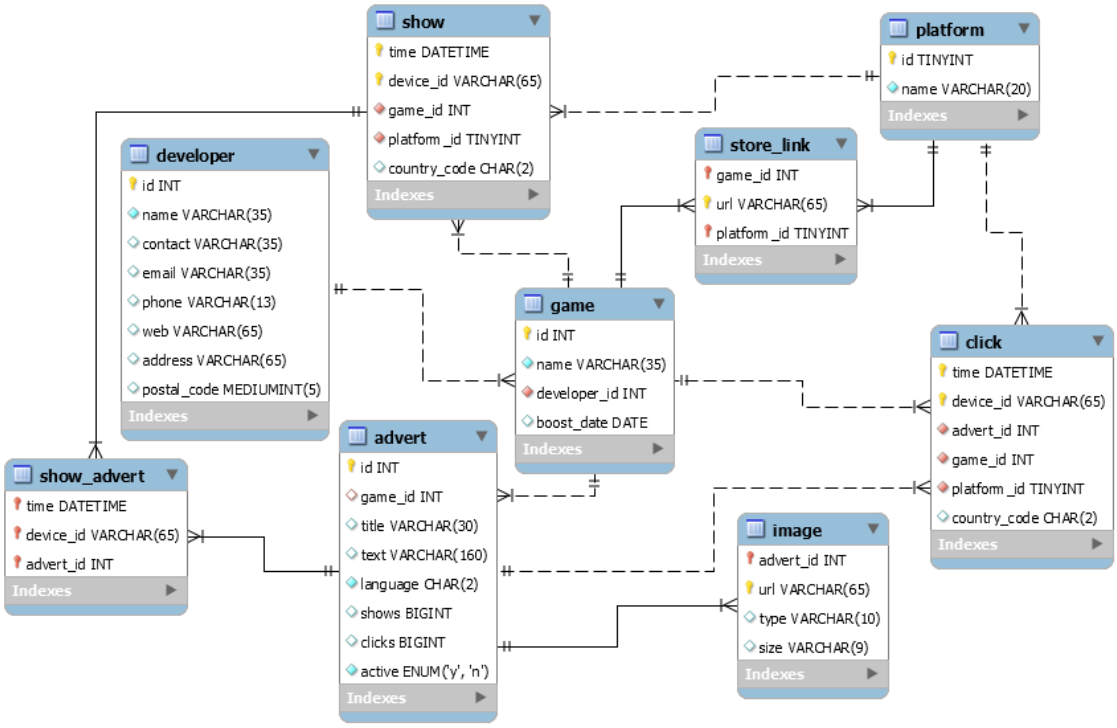
Peliin liittyvä tietokantataulu päätettiin normalisoida useampaan tauluun. Tämä siksi, että pelillä on ennalta määräämätön määrä laitealustoja. Pelin ja laitealustan välinen monen suhde moneen -yhteys toteutettiinkin niitä yhdistävällä taululla, jossa viitteet pelistä ja laitealustasta yhdistyvät ominaisuuteen sovelluskauppalinkistä. Sovelluskauppalinkit yhdistettiin peleihin mainoksien sijaan, sillä pelillä voi olla monta mainosta, jotka johtavat samaan sovelluskauppaan, ja siten se tieto olisi todennäköisestä kertautunut. Game-taulun pääavaimeksi valittiin myös kokonaisluku pelin uniikin nimen sijaan ja developer_id-ominaisuudella peli yhdistetään sen kehittäjään. (Kuva 3, taulut game, store_link ja platform.)

Mainoksista eroteltiin kuvan ominaisuudet omaksi taulukseen, sillä se mahdollistaa tulevaisuudessa useamman kuvan liittämisen mainokseen. Kuvan tiedot yhdistettiin mainokseen kuvan kokonaislukutunnisteen avulla. Mainoksen peli ja sitä kautta sen kehittäjä ja sovelluskauppalinkit yhdistettiin mainokseen sen

game_id-ominaisuudella toteutetulla yhteydellä game-tauluun. (Kuva 3, taulut advert ja image.)

Mainoksiin kohdistuneita klikkauksia ei ollut tarvetta erotella erillisellä pääavain-tunnisteella, sillä klikkauksia ei ole mahdollista hallita hallintasovelluksen graafisen käyttöliittymän kautta. Sen sijaan pääavaimena click-taulussa käytettiin tietoa klikkauksen ajankohdasta ja laitteen uniikista tunnisteesta, jotka yhdessä muodostavat jokaiselle klikkaukselle uniikin avaimen. Klikkaus yhdistettiin klikattuun mainokseen viitteellä advert-taulun tunnisteeseen. Peliin josta klikkaus tapahtui viitattiin game-taulun tunnisteella ja laitealustaan viitteellä platform-tauluun. (Kuva 3, taulu click.)

Mainoksiin kohdistuneiden esityksien toteuttamista lähestyttiin klikkauksien tapaan. Koska kuitenkin yhdellä esityskerralla saatetaan näyttää useampaa eri mainosta, oli esityksen ja mainoksen välille luotava taulu monen suhde moneen -yhteyttä varten. Tämän taulun avulla yhdistettiin show-taulun pääavaimena toimivat ajankohta ja laitteen uniikki tunniste jokaiseen esityskerralla näytettyyn mainokseen advert-taulun tunnisteen avulla. (Kuva 3, taulut show ja show_advert.)



KUVA 3. ER-kaavio tietokannan kaavasta

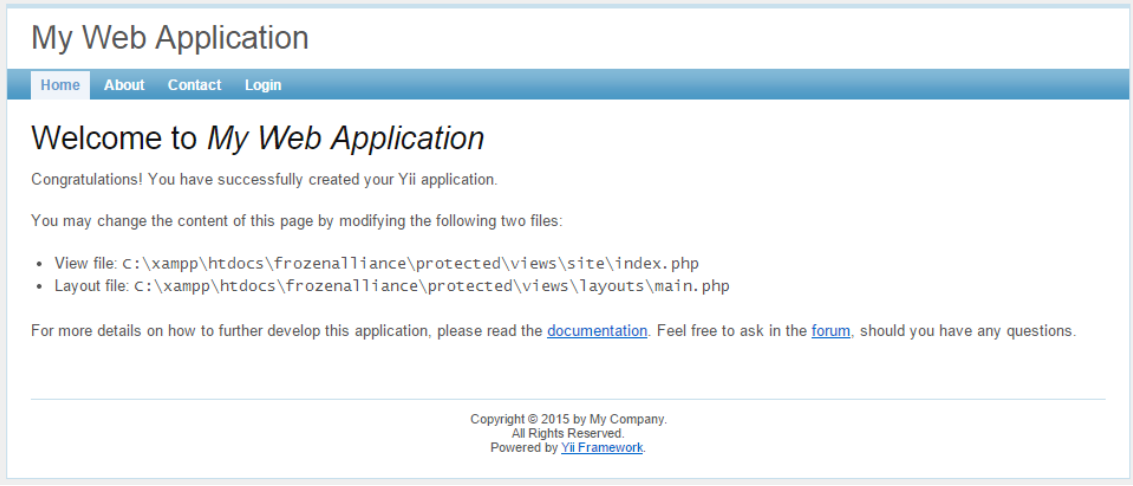
4.2 Hallintasovellus

Hallintasovelluksen kehitys alkoi Yii-sovelluskehyksen asentamisella kehitysympäristöön. Pakattu sovelluskehys ladattiin yiiframework.com -sivustolta, jonka jälkeen sen purettiin haluttuun tiedostopolkuun. Tämän jälkeen kehitysympäristön Apache-verkkopalvelin konfiguroitiin .htaccess-tiedostolla, joka estää palvelinta palvelemasta muita kuin haluttuja tiedostoja ja uudelleenohjaa palvelimelle osoitetut kutsut joko fyysisiin tiedostoihin tai sovelluksen esilatausohjelmaan (kuva 4).

```
RewriteEngine on

# prevent httpd from serving dotfiles (.htaccess, .svn, .git, etc.)
RedirectMatch 403 /\..*$
# if a directory or a file exists, use it directly
RewriteCond %{REQUEST_FILENAME} !-f
RewriteCond %{REQUEST_FILENAME} !-d
# otherwise forward it to index.php
RewriteRule . index.php
```

KUVA 4. Apache-konfiguraatio

Kun sovelluskehys oli asennettu kehitysympäristöön, generoitiin pohjasovellus sovelluskehyksen työkaluja apuna käyttäen. Tämä tapahtui siten, että ajettiin komentorivillä yiic.php -komentosarja ja välitettiin sille parametreinä haluttu sovelluksen nimi ja tiedostopolku. Tämä loi pohjasovelluksen, jonka ympärille hallintasovelluksen toiminnallisuudet alettiin rakentamaan (kuva 5).



KUVA 5. Pohjasovellus

4.2.1 Tunnistautuminen ja ulkoasu

Tunnistautuminen ja sivurakenne oli jo toteutettu pohjasovelluksessa. Sitä täytyi kuitenkin muuttaa siten, ettei sovelluksessa olisi muita kuin kirjautumisnäkymä, johon käyttäjällä olisi pääsy ennen tunnistautumista. Sovelluksen olisi siis tarkistettava jokaisen siihen kohdistuneen kutsun kohdalla, onko käyttäjä jo kirjautunut, ja jos ei, niin käyttäjältä pitää vaatia tunnistautumista.

Pohjasovelluksen valmiin UserIdentity-luokan authenticate-metodiin asetettiin sovelluksen kovakoodatut käyttäjätunnukset tunnistautumista varten. Tämän jälkeen kirjoitettiin Larry Ullmanin ohjeen pohjalta RequireLogin-luokka käyttäytymismallia varten, joka vaatisi tunnistautumattomalta käyttäjältä sovellukseen kirjautumisen ja ohjaisi kaikki tämän kutsun kirjautumissivulle (13). Käyttäytymismalli otettiin käyttöön sovelluksen konfiguraatio tiedostossa ja asetettiin kirjautumissivu (kuva 6). Lopuksi pohjasovelluksen alkuperäisestä kirjautumisnäkymästä poistettiin vielä vaatimus kuvavarmennuksesta.

```
// behaviors
'behaviors' => array(
    'onBeginRequest' => array(
        'class' => 'application.components.RequireLogin'
    )
),

// application components
'components'=>array(
    'user'=>array(
        // enable cookie-based authentication
        'allowAutoLogin'=>false,
        'loginUrl' => array('site/login'),
    ),
```

KUVA 6. Tunnistautumisen konfigurointi

Sovelluksen ulkoasua muutettiin sopivammaksi muokkaamalla HTML-elementtejä sen näkymätiedostoista. Sovelluksen otsake muutettiin vastaamaan sen oikeaa nimeä ja turhat sivut kuten About ja Contact poistettiin. Päänäkymän navigaatiopalkkiin lisättiin linkit tuleville Adverts-, Games- ja Developers-sivuille. (Kuva 8.)

KUVA 8. Hallintasovelluksen etusivu

4.2.2 Pelinkehittäjät

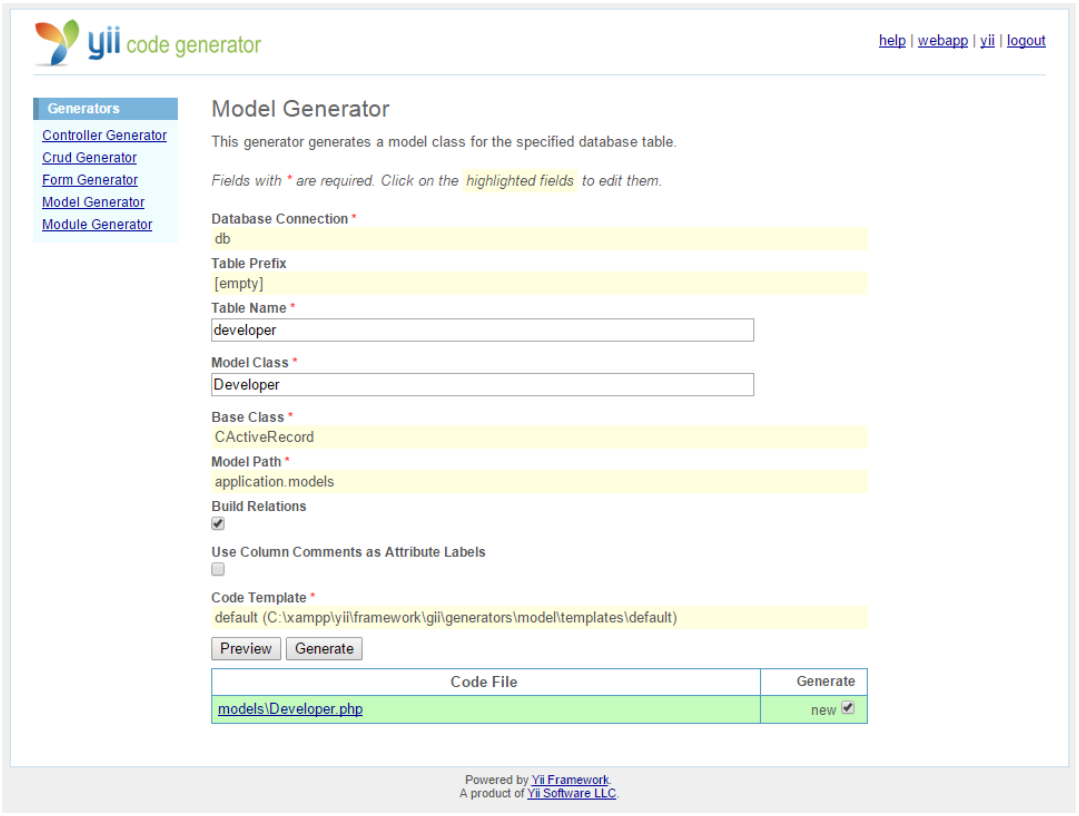
Pelinkehittäjien hallinnointia varten täytyi ensin toteuttaa MVC-arkkitehtuurin mukainen model-luokka tietokantataulun käsittelyyn. Tämä tapahtui Yiin ohjelmakoodingenerointityökalulla Giillä. Gii otettiin sovelluksessa käyttöön poistamalla kommentointi sen konfiguraatiosta sovelluksen konfiguraatiotiedostossa ja asettamalla työkalulle salasana (kuva 9).

```
'modules'=>array(
    // uncomment the following to enable the Gii tool
    'gii'=>array(
        'class'=>'system.gii.GiiModule',
        'password'=>'frozen123!',
        // If removed, Gii defaults to localhost only. Edit carefully to taste.
        'ipFilters'=>array('127.0.0.1','::1'),
    ),
),
```

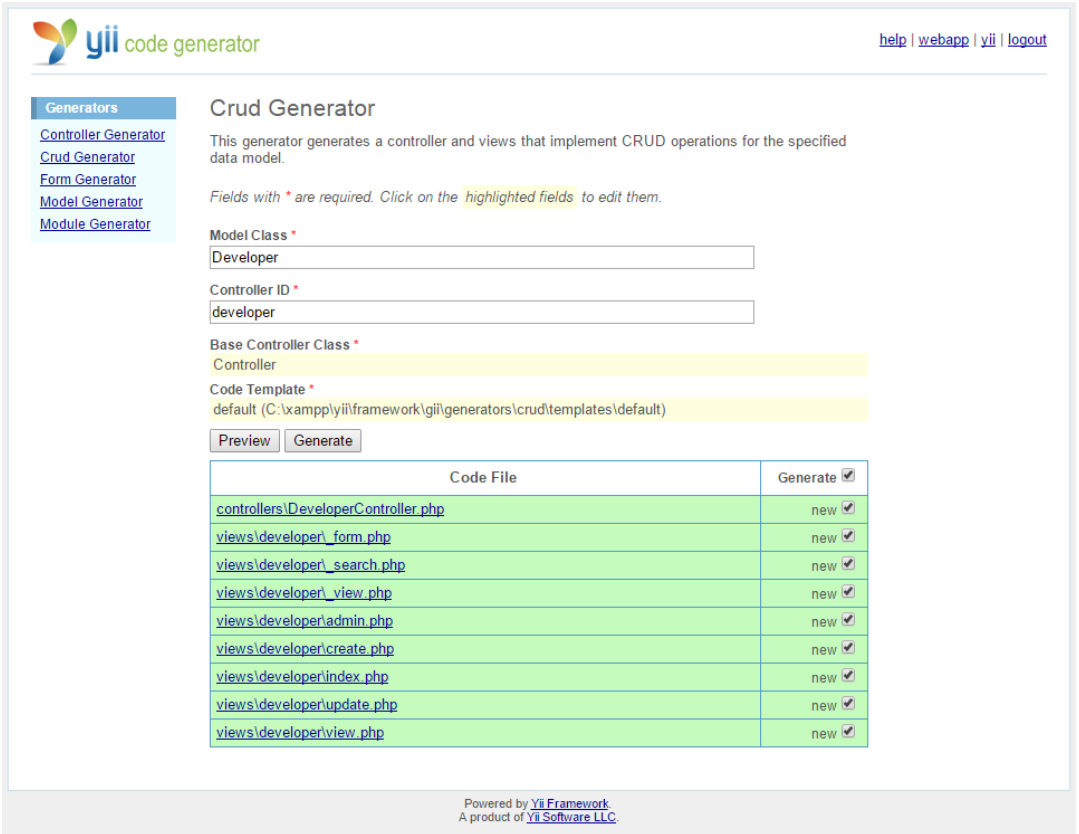
KUVA 9. Gii:n konfiguraatio

Tämän jälkeen Gii:tä käytettiin paikallisesti URL-osoitteesta http://127.0.0.1/frozenalliance/gii. Gii:n Model Generator -työkalulla uusi model-luokka luotiin syöttämällä tietokannan taulun nimi ja klikkaamalla Preview-painiketta (kuva 10). Tämän jälkeen Gii loi esikatselun model-luokasta, joka sitten generoitiin klikkaamalla Generate-painiketta. Tämä luokka sisälsi kaiken perustoiminnallisuuden developer-taulun käsittelyyn ohjelmakoodista.

Kun MVC-arkkitehtuurin mukainen model-luokka oli toteutettu, tarvittiin vielä controller-luokka ja näkymätiedostot tietokantataulun CRUD-operaatioihin käyttöliittymästä. Gii:n Crud Generator -työkalulla luodun model-luokan perusteella generoitiin nämä tiedostot (kuva 11).



KUVA 10. Model Generator



KUVA 11. Crud Generator

Koska pelinkehittäjien hallinnointi käytännössä vaati vain mahdollisuuden käsitellä developer-tietokantataulua käyttöliittymästä, oli sen toiminnallisuus jo toteutettu. Vielä kuitenkin vaadittiin paikkakuntatiedon esittäminen käyttäjän syöttämän postinumeron perusteella ja pieniä ulkoasumuutoksia näkymätiedostoihin.

Paikkakunnalle lisättiin oma tekstikenttä Developers-sivun näkymiin ja toteutettiin yhteys postinumerorajapintaan. Ensin kirjoitettiin model-luokkaan metodi, jolla annettujen parametrien mukaisesti haettiin rajapinnalta paikkakuntatieto JSON-tiedostosta ja palautettiin se (kuva 12). Tämän jälkeen kirjoitettiin luontinäkymän näkymätiedostoon jQueryllä toteutettu JavaScript-funktio, joka AJAX:lla controller-luokan kautta kutsuu model-luokan rajapintafunktiota, kun postinumero on syötetty ja asettaa palautetun paikkakuntatiedon sille tarkoitettuun kenttään. (Kuva 13).

```php
public function getPlace($postcode = null, $type='json', $lang='fin')
{
    if($postcode == null)
        $postcode = $this->postal_code;

    $url = Yii::app()->params['postCodeAPI'] . '&postcode=' . $postcode . '&type=' . $type . '&lang=' . $lang;
    $result = file_get_contents($url);

    $resultJSON = json_decode($result, true);

    if(isset($resultJSON['postoffice']))
    {
        $this->place = $resultJSON['postoffice'];
        return $this->place;
    }
    else
        return "";
}
```

KUVA 12. GetPlace-funktio

```javascript
<script type="text/javascript">
    function postCodeChanged()
    {
        var value = $('#postCodeField').val();
        if(value.length == 5)
        {
            $.ajax({
                  type: "GET",
                  url:'<?php echo Yii::app()->createUrl('developer/ajaxCity'); ?>' + "?code=" + value,
                }).done(function( place ) {
                    $("#placeField").val(place);
                  });
        }
    }
</script>
```

KUVA 13. PostCodeChanged-funktio

4.2.3 Pelit

Pelien hallinnointia varten täytyi luoda model-luokat game-, platform- sekä store_link-tietokantatauluille, jotta pelien lisäksi käyttöliittymästä voitaisiin hallita myös peleihin liittyviä sovelluskauppalinkkejä. Tämän lisäksi tarvittiin controller-luokka ja näkymätiedostot peleille, jotka generoitiin model-luokkien kanssa Gii-työkalulla samalla toimenpinteellä kuin pelinkehittäjällekin.

Pelien hallinnointi vaati jo hieman muutoksia Gii:llä toteutettuihin CRUD-operaatioihin. Pelin kehittäjän tiedon syöttäminen muutettiin tekstikentästä alasvetovalikoksi, jonka vaihtoehdot haettiin developer-tietokantataulusta. Näin uudelle pelille valittiin kehittäjä aikaisemmin järjestelmään luoduista pelinkehittäjistä. Tehostuspäivämäärän valinta toteutettiin Yii:n Zii-työkalulla, joka annettujen parametrien perusteella generoi JavaScript-ohjelmakoodia päivämäärävalitsimen toteuttamiseksi (kuva 14). Luontinäkymään lisättiin myös tekstikentät sovelluskauppalinkeille, joille peliä tallentaessa luotiin controller-luokassa tietokantaobjektilla rivit store_link-tauluun.

```
<div class="row">
<?php echo $form->labelEx($model,'boost_date'); ?>
<?php $this->widget('zii.widgets.jui.CJuiDatePicker', array(
        'model' => $model,
        'attribute' => 'boost_date',
        'language' => 'en',
        'options'=>array(
            'showOn' => 'both',             // also opens with a button
            'dateFormat' => 'yy-mm-dd',
            'showOtherMonths' => true,      // show dates in other months
            'selectOtherMonths' => true,    // can seelect dates in other months
            'changeYear' => true,           // can change year
            'changeMonth' => true,
            'firstDay' => 1,
        ),
        'htmlOptions' => array(
            'size' => '10',
            'maxlength' => '10',
        ),
    ));
?>
<?php echo $form->error($model,'boost_date'); ?>
</div>
<br><br>
```

KUVA 14. Päivämäärävalitsin

Myös Gii:n generoimaa pelien hallintanäkymää muutettiin (kuva 15). Lajitteluriville hallintataulukkoon vaihdettiin pelinkehittäjän kohdalle tekstikentän sijaan alasvetovalikko, jonka sisällöksi haettiin developer-taulusta kaikki järjestelmän pelinkehittäjät ja käytettiin sitä taulukon suodattimena. Tämä siksi, että alkuperäisessä tekstikentässä olisi pitänyt käyttää suodattimena pelinkehittäjän tunnistetta eikä nimeä. Lisäksi alasvetovalikko on helppokäyttöisempi. Muutos toteutettiin jälleen Zii-työkalun parametreillä, jotka generoivat JavaScript- ja HTML-ohjelmakoodin taulukkoa varten (kuva 16).

KUVA 15. Pelien hallintanäkymä

```php
<?php $this->widget('zii.widgets.grid.CGridView', array(
    'id'=>'game-grid',
    'dataProvider'=>$model->search(),
    'filter'=>$model,
    'columns'=>array(
        array(
            'name'=>'id',
            'value'=>'$data->id',
            'htmlOptions' => array('style' => 'width: 50px;'),
            'filterHtmlOptions' => array('style' => 'width: 50px;'),
        ),
        'name',
        array(
            'name'=>'developer_id',
            'filter'=>$devsList,
            'value'=>'Developer::Model()->FindByPk($data->developer_id)->name',
        ),
        //'boost_date',
        array(
            'class'=>'CButtonColumn',
        ),
    ),
)); ?>
```

KUVA 16. Pelien hallintanäkymän pienoisohjelman ohjelmakoodi

4.2.4 Mainokset

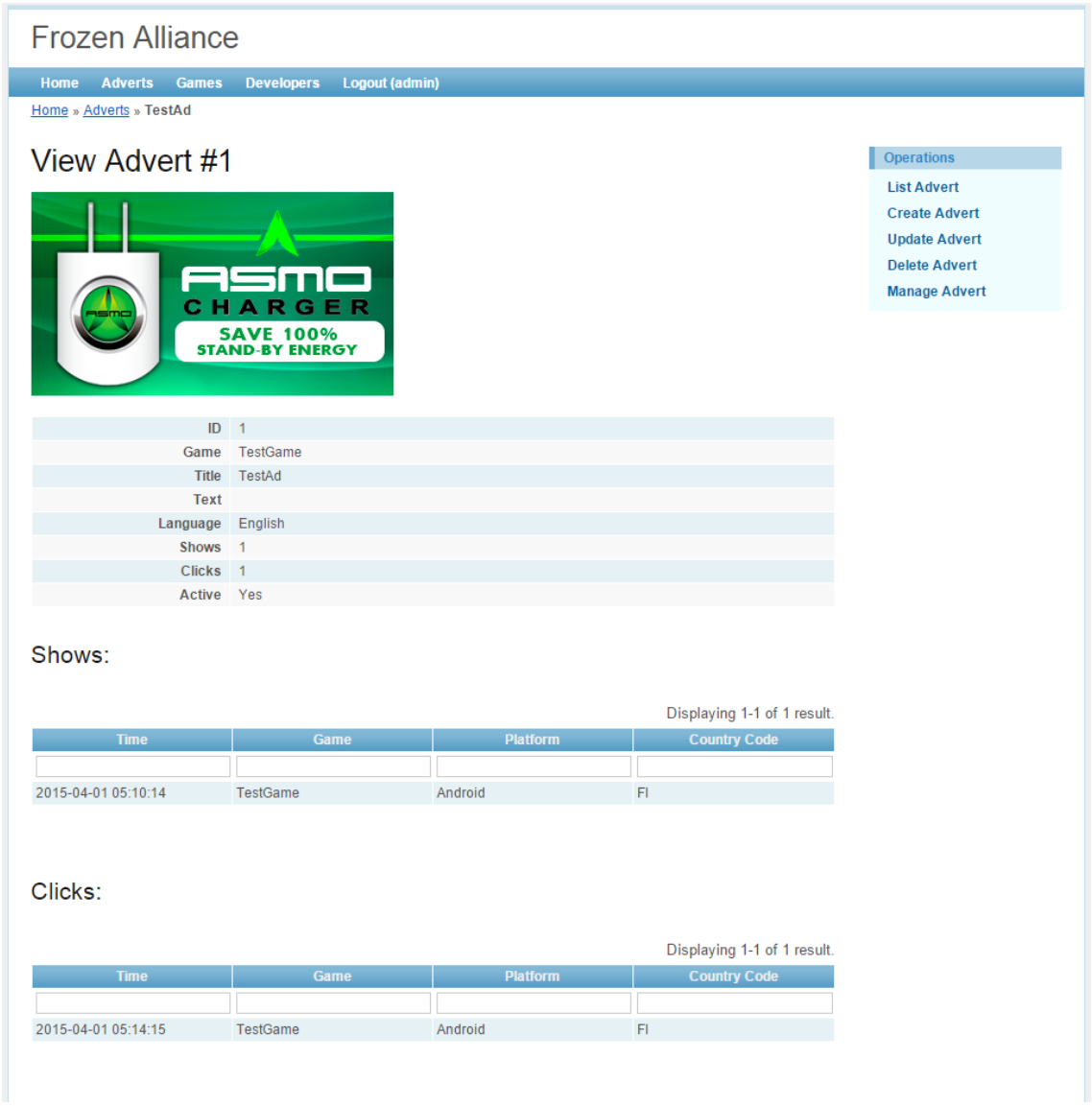
Mainosten hallinnointi aloitettiin jälleen pohjaohjelmakoodin generoimisella. Tällä kertaa tarvittiin model-luokat advert-, click-, image-, show- ja show_advert-tietokantatauluille. Niitä tarvittiin, koska mainoksista piti pystyä tallentamaan myös tiedot niihin liitetyistä kuvista ja lisäksi mainoksista piti pystyä esittämään tietoa esitys- ja klikkauskerroista. Näiden lisäksi Gii:llä generoitiin tarvittava mainoksen controller-luokka ja näkymätiedostot CRUD-operaatioita varten.

Mainoksen luontinäkymään pelitieto vaihdettiin jälleen alasvetovalikoksi, josta käyttäjä valitsee järjestelmään aikaisemmin luodun pelin. Otsake ja mainosteksti jätettiin tekstikentiksi, mutta aktiivisuuden tilalle ja kielelle asetettiin myös alasvetovalikot. Mainoksen kielelle asetettiin yleisimmät vaihtoehdot. Kuvan asettamista varten näkymään lisättiin tiedostovalitsin (kuva 17). Tiedostovalitsimen kautta mainoskuva ladattiin palvelimen väliaikaistiedostoihin, joista se controller-luokassa tallennettiin pysyvästi palvelimelle mainosta tallentaessa. Samalla tallennettiin image-tietokantatauluun tiedot kuvasta.

```
<div class="row">
<?php
    echo $form->labelEx($imageModel, 'imageFile');
    echo $form->fileField($imageModel, 'imageFile');
    echo $form->error($imageModel, 'imageFile');
?>
```

KUVA 17. Tiedostovalitsin

Tarkastelunäkymä koki suurimman muutoksen alkuperäiseen generoituun näkymään verrattuna (kuva 18). Image-taulusta haettiin tieto mainoskuvan sijainnista palvelimella ja sitä käytettiin kuvan esittämiseen tarkastelunäkymässä. Lisäksi näkymään lisättiin taulukoidut tiedot mainokseen kohdistuneista esityksistä ja klikkauksista. Tämä toteutettiin siten, että controller-luokassa luotiin tietokantaobjektit esitys- ja klikkaustiedoille, joiden perusteella Zii-työkalulla toteutettiin taulukkonäkymät niille (kuva 19).

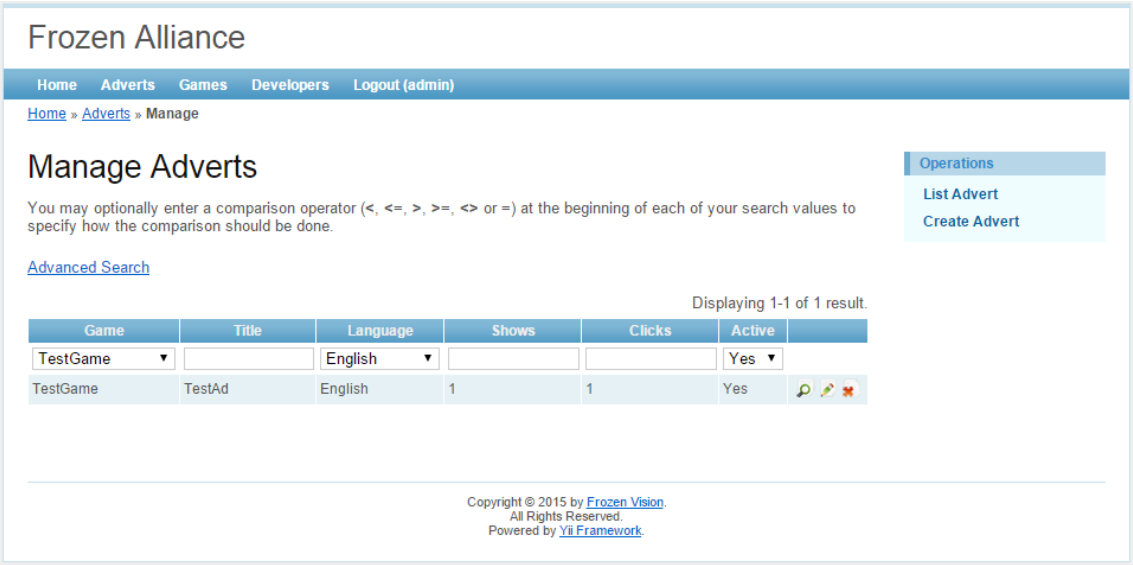


KUVA 18. Mainoksen tarkastelunäkymä

```
<br><br>
<h2>Shows:</h2>
<?php $this->widget('zii.widgets.grid.CGridView', array(
    'id'=>'show-grid',
    'dataProvider'=>$showModel->searchByAdvertId(),
    'filter'=>$showModel,
    'columns'=>array(
        'time',
        array(
            'name'=>'game_id',
            'value'=>function($data, $row) use ($gamesList){return $gamesList[$data->game_id];},
        ),
        array(
            'name'=>'platform_id',
            'value'=>function($data, $row) use ($platformList){return $platformList[$data->platform_id];},
        ),
        'country_code',
    ),
)); ?>
<br><br>
<h2>Clicks:</h2>
<?php $this->widget('zii.widgets.grid.CGridView', array(
    'id'=>'click-grid',
    'dataProvider'=>$clickModel->searchByAdvertId($model->id),
    'filter'=>$clickModel,
    'columns'=>array(
        'time',
        array(
            'name'=>'game_id',
            'value'=>function($data, $row) use ($gamesList){return $gamesList[$data->game_id];},
        ),
        array(
            'name'=>'platform_id',
            'value'=>function($data, $row) use ($platformList){return $platformList[$data->platform_id];},
        ),
        'country_code',
    ),
)); ?>
<br><br>
```

KUVA 19. Mainoksen esitys- ja klikkaustaulukot

Lopuksi muokattiin vielä hallintanäkymää, sillä mainosten pelien mukaan lajittelu olisi pitänyt toteuttaa pelin tunnisteella. Tämän helpottamiseksi taulukon lajitteluriville vaihdettiin jälleen alasvetovalikko pelien suodattamista varten (kuva 20). Samalla helppokäyttöisyyden vuoksi vaihdettiin alasvetovalikot myös kielen ja aktiivisuuden tilan suodattamiselle. Tämä toteutettiin jälleen Zii-työkalulla, joka generoi taulukon annettujen parametrien mukaisesti (kuva 21).



KUVA 20. Mainosten hallintanäkymä

```php
<?php $this->widget('zii.widgets.grid.CGridView', array(
    'id'=>'advert-grid',
    'dataProvider'=>$model->search(),
    'filter'=>$model,
    'columns'=>array(
        array(
            'name'=>'game_id',
            'filter'=>$gamesList,
            'value'=>function($data, $row) use ($gamesList){return $gamesList[$data->game_id];},
            'htmlOptions' => array('style' => 'width: 120px;'),
            'filterHtmlOptions' => array('style' => 'width: 120px;'),
        ),
        'title',
        array(
            'name'=>'language',
            'filter'=>$languageCodes,
            'value'=>function($data, $row) use ($languageCodes){return $languageCodes[$data->language];},
            'htmlOptions' => array('style' => 'width: 100px;'),
            'filterHtmlOptions' => array('style' => 'width: 100px;'),
        ),
        'shows',
        'clicks',
        array(
            'name'=>'active',
            'filter'=>$activeArr,
            'value'=>function($data, $row) use ($activeArr){return $activeArr[$data->active];},
            'htmlOptions' => array('style' => 'width: 50px;'),
            'filterHtmlOptions' => array('style' => 'width: 50px;'),
        ),
        array(
            'class'=>'CButtonColumn',
        ),
    ),
)); ?>
```

KUVA 21. Mainosten hallintanäkymän pienoisohjelman ohjelmakoodi

4.3 Verkkorajapinta

Verkkorajapinta on hallintasovelluksen tapaan verkkosovellus, mutta sille ei etusivua lukuun ottamatta toteutettu graafista käyttöliittymää. Se toteutettiin hal-

linta sovelluksen pohjalta, sillä sille oli jo valmiiksi generoitu MVC-arkkitehtuurin mukaiset model-luokat tietokantataulujen hallinnointiin. Hallintasovelluksesta tehtiin kopio, josta poistettiin sille aikaisemmin luodut controller-luokat ja näkymätiedostot. Lisäksi tunnistautumista varten toteutettu ohjelmakoodi poistettiin, konfiguraatiotiedosto korjattiin vastaamaan uutta sovellusta ja etusivun otsake vaihdettiin. Päänäkymästä poistettiin myös linkit, jotka johtivat poistetuille hallintasivuille. Tämän jälkeen luotiin uusi controller-luokka, johon rajapinnan toiminnallisuus toteutettiin.

4.3.1 Advert-kutsu

Ensimmäisenä toteutettiin advert-kutsu uuteen controller-luokkaan, joka HTTP-protokollan GET-metodin parametrien mukaisesti hakisi tarvittavat mainostiedot tietokannasta. Kutsusta tarkistetaan, että laitealustan ja pelin tunnisteet on syötetty ja niitä vastaavat tietueet löytyvät tietokannasta. Jos kutsussa ei ole mainosten lukumäärää määriteltynä, haetaan vain yksi mainos, mutta muuten lukumäärän mukainen määrä.

Mainostietueet haetaan niihin liittyvien kuva-, peli- ja sovelluskauppalinkkitietojen kanssa. Tämä tapahtuu niin, että aluksi valikoidaan aktiivisia mainoksia, joille löytyy halutun laitealustan mukaisia sovelluskauppalinkkejä, jotka eivät kuulu annetulle pelille ja joiden tehostuspäivämäärä on voimassa (kuva 22). Mikäli näin ei saada valikoitua haluttua määrää mainoksia, toteutetaan sama niin, että valikoidaankin mainoksia, joiden tehostuspäivämäärä on jo umpeutunut.

```
$model = Advert::model()->with(array(
    'images'=>array('joinType'=>'INNER JOIN', 'together'=>true),
    'game'=>array('joinType'=>'INNER JOIN','together'=>true),
    'storeLinks'=>array('joinType'=>'INNER JOIN','together'=>true)))->findAll(array(
            'condition'=>'storeLinks.platform_id=:platform_id AND t.active=:active
                          AND t.game_id!=:game_id AND game.boost_date >= CURDATE()',
            'params' => array(':platform_id'=>$_GET['platform_id'], ':active'=>'y',
                              ':game_id'=>$_GET['game_id']),
            'order' => 't.shows ASC',
            'limit' => $_GET['count']));
```

KUVA 22. Mainosten hakeminen tietokannasta

Haetuista mainostietueista koostetaan taulukko, jonka pohjalta PHP:n json_encode-funktiolla muotoillaan kutsuun vastaukseksi JSON-tiedosto (kuva 23).


```
[
  {
    "advert": {
      "id": "11",
      "game_id": "11",
      "title": "Neppis",
      "text": "Download Neppis!!",
      "language": "en",
      "shows": "10",
      "clicks": "4",
      "active": "y"
    },
    "image": {
      "advert_id": "11",
      "url": "54e34069b3d6b.png",
      "type": "png",
      "size": "320x180"
    },
    "storeLink": {
      "game_id": "11",
      "url": "test-link-aasd",
      "platform_id": "1"
    }
  }
]
```


KUVA 23. Esimerkki advert-kutsun vastauksesta JSON-muodossa

4.3.2 Show-kutsu

Show-kutsu toteutettiin ottamaan POST-metodin mukaisesti vastaan tietoja esitetyiden mainosten tunnisteista, esittäneen pelin tunnisteesta, laitealustasta, laitteen uniikista tunnisteesta ja maakoodista. Kutsu palauttaa virheen, jos näitä tietoja ei ole annettu, lukuun ottamatta maakoodia, joka ei ole pakollinen tieto.

Mikäli annettuja tunnisteita vastaavat tiedot löytyvät tietokannasta lukuun ottamatta laitteen tunnistetta, kirjataan esitykselle Greenwichin mukainen aika ja se tallennetaan uudeksi riviksi show-tietokantatauluun. Jokaista mainoksen tunnistetta kohden tämän jälkeen luodaan oma show_advert-tietue, jolla esitetyt mainokset yhdistetään esityskertaan. Vastaukseksi kutsulle annetaan joko OK-koodi tai sitten virhekoodi ja virheilmoitus sen mukaan, onko esityksen ilmoittaminen onnistunut vai ei.

4.3.3 Click-kutsu

Click-kutsu toteutettiin show-kutsun tapaan ottamaan parametrejä vastaan POST-metodin mukaisesti. Se kuitenkin ottaa vastaan vain yhden klikatun mainoksen tunnisteen. Mikäli mainoksen, pelin ja laitealustan tunnisteiden mukaiset tietueet löytyvät tietokannasta ja laitteen uniikki tunniste on syötetty, kirjataan klikkauksen tiedot tietokantaobjektiin (kuva 24). Tämän jälkeen klikkaukselle asetetaan Greenwichin mukainen aika ja se tallennetaan tietokantaan. Vastauksena klikkauksen onnistumiselle palautetaan jälleen koodi tapahtuman onnistumisesta ja tarvittaessa virheilmoitus.

```
// Try to assign POST values to attributes
foreach($_POST as $var=>$value) {
    // Does the model have this attribute? If not raise an error
    if($model->hasAttribute($var))
        $model->$var = $value;
    else
        $this->_sendResponse(500, sprintf('Parameter %s is not allowed for Click', $var));
        }

$model->time = gmdate('Y-m-d H:i:s');
```

KUVA 24. Klikkaustietojen asettaminen tietokantaobjektiin

4.4 Päätelaitekomponentti

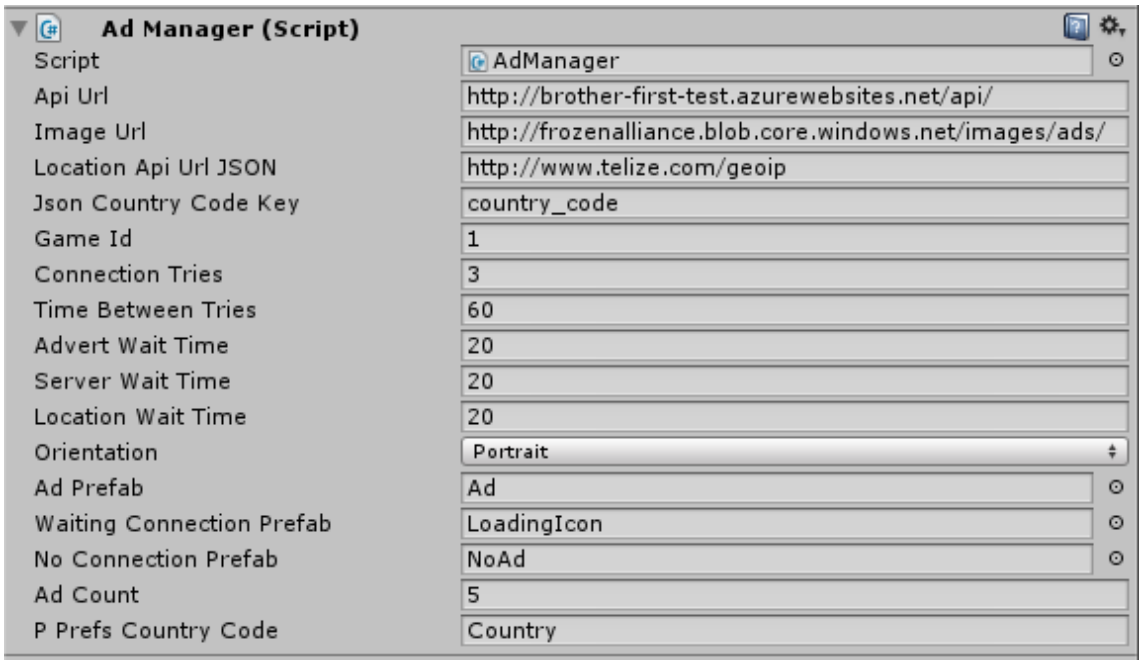
Unity 3D -kehitysympäristöön toteutettiin AdManager-peliobjekti suorittamaan mainosten hallinnointia pelin taustalla sekä Unityn uuden graafisen käyttöliittymämallin mukainen Canvas-objekti, jonka alle mainoskäyttöliittymä rakennettiin (kuva 25). AdManager-objektiin yhdistettiin C#-ohjelmointikielellä toteutettu komentosarja, joka noudattaa Singleton-olionluontimallia. Tämä peliobjekti toteutettiin siten, että se on tarkoitus asettaa pelin ensimmäiseen kohtaukseen (Scene), josta se ilmentää itsensä pelin käynnistyessä ja estää itsensä tuhoamisen siirryttäessä kohtauksesta toiseen. AdCanvasPortait- tai AdCanvasLandscape-käyttöliittymäobjekti puolestaan toteutettiin asetettavaksi pelin päävalikkoon pelin orientaation mukaan.

```
private static AdManager _instance;

public static AdManager instance
{
    get
    {
        if(_instance == null)
        {
            _instance = GameObject.FindObjectOfType<AdManager>();
            DontDestroyOnLoad(_instance.gameObject);
        }

        return _instance;
    }
}

void Awake ()
{
    if(_instance == null)
    {
        _instance = this;
        DontDestroyOnLoad(this);
    }
    else
    {
        if(this != _instance)
            Destroy(this.gameObject);
    }
}
```

KUVA 25. Toteutus Singleton-olionluontimallin mukaisesti

4.4.1 AdManager

AdManager-luokalle toteutettiin julkisia ominaisuuksia, joiden avulla pelinkehittäjä voi editorin kautta asettaa mm. käytettyjen rajapintojen URL-osoitteet, pelin tunnisteen rajapintakutsuja varten sekä pelin orientaation (kuva 26). Kun AdManager-peliobjekti ensimmäisen kerran ilmentyy, se aloittaa tausta-ajon mainosten ja laitteen sijainnin noutamiseksi ohjelmistorajapintoja käyttäen. Olio kutsuu rajapintoja Unityn WWW-luokan avulla, joka palauttaa haetun sisällön merkkijonomuuttujaan (kuva 27). Tästä tekstimuotoisesta vastauksesta sitten jäsennellään SimpleJSON-luokkakirjaston avulla tietueet ohjelmakoodissa käytettäväksi.



KUVA 26. AdManager-luokan ominaisuudet Unityn editorissa

```
IEnumerator GetAdverts ()
{
    string advertUrl = apiUrl + "advert/platform_id/" + platformId + "/count/" + adCount + "/game_id/" + gameId;
    WWW www = new WWW(advertUrl);

    float elapsedTime = 0.0f;

    while (!www.isDone)
    {
        elapsedTime += Time.deltaTime;

        if (elapsedTime >= serverWaitTime) break;

        yield return null;
    }

    if (!www.isDone || !string.IsNullOrEmpty(www.error))
    {
        Debug.LogError(string.Format("Server connection fail!\n{0}", www.error));
        yield break;
    }

    string response = www.text;

    var N = JSON.Parse(response);
```

KUVA 27. Advert-kutsun toteuttaminen WWW-luokan avulla

Kun peli siirtyy päävalikkoon eli samaan kohtaukseen, johon mainospaneelin käyttöliittymäobjekti on asetettu, kutsuu se AdManager-oliota, joka aloittaa taus-ta-ajon mainosten lisäämiseksi mainosvalikkoon. Aikaisemmin noudettujen mainostietojen perusteella olio hakee jälleen WWW-luokan avulla jokaiselle mainokselle kuvan ja ilmentää mainosvalikkoon jokaiselle mainokselle uuden käyttöliittymäobjektin. Tämä vähentää latausaikaa tai jättää sen kokonaan pois, kun käyttäjä klikkaa mainosvalikon esiin.

Samalla kun mainosvalikko esitetään, kutsutaan käyttöliittymästä jälleen AdManager-objektia, joka aloittaa rajapintakutsun tausta-ajon mainosten esittämistä varten. Tällä kertaa käytetään Unityn WWWForm-luokkaa, joka eroaa WWW-luokasta siten, että sillä voidaan toteuttaa POST-metodin mukaisia kutsuja. WWWForm-luokan oliolle asetetaan muuttujat noudettujen mainosten tunnisteille, pelinkehittäjän asettaman pelin tunnisteelle, ohjelmakoodissa selvitetylle laitealustan tunnisteelle ja laitteen uniikille tunnisteelle. Mikäli pelin käynnistyksessä on onnistettu noutamaan laitteen IP-osoitteen perusteella sen sijainnin maakoodi, myös se asetetaan osaksi olion tietuetta. (Kuva 28).

```
IEnumerator ShowCoroutine()
{
    WWWForm form = new WWWForm();

    string advert_ids = "";

    foreach(Advert a in advertList)
    {
        advert_ids = advert_ids + a.id + ",";
    }

    form.AddField("advert_ids", advert_ids);
    form.AddField("game_id", gameId);
    form.AddField("platform_id", platformId);
    form.AddField("device_id", deviceId);

    string cc = PlayerPrefs.GetString(pPrefsCountryCode);
    if(cc != "" && cc != null)
        form.AddField("country_code", cc);

    WWW www = new WWW(apiUrl + "show/", form);
    yield return www;

    if (!string.IsNullOrEmpty(www.error))
        Debug.LogError(www.error);
}
```

KUVA 28. Show-kutsun toteuttaminen WWWForm-luokan avulla

Jos käyttäjä klikkaa jotain mainosta, avataan selaimeen verkkorajapinnalta noudettu ja mainosobjektille asetettu URL-osoite. Tämän jälkeen kutsutaan AdManager-objektia ja välitetään sille klikatun mainoksen tunniste. AdManager aloittaa tausta-ajon klikkauskutsua varten verkkorajapinnalle, johon käytetään myös WWWForm-luokkaa esityskutsun tapaan, mutta tällä kertaa vain yhden klikatun mainoksen tunniste asetetaan olion tietueeseen.

4.4.2 Graafinen käyttöliittymä

Käyttöliittymästä toteutettiin kaksi lähes samankaltaista Unityn uuden käyttöliittymämallin mukaista Canvas-objektia, joista toinen on tarkoitettu pysty- (Portrait) ja toinen vaakaorientaatiolle (Landscape). Molemmat sisältävät samat lapsiobjektit ja toiminnallisuuden. Ensin päävalikossa on näkyvillä vain mainosvalikon avauspainike, jonka klikkaus asettaa mainosvalikon esiin (kuva 30).

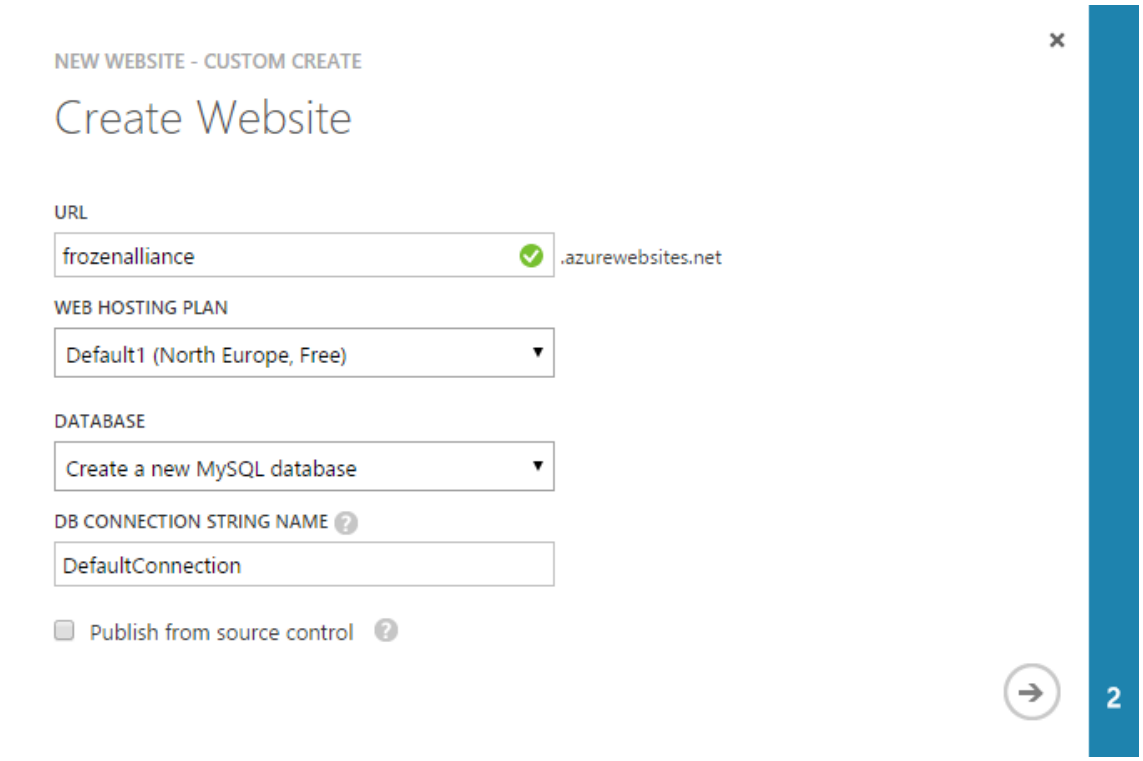
Mainosvalikko kehitettiin Unityn Panel-objektista, jonka lapsiobjekteina ovat sen otsake ja mainoslista. Otsakkeeseen graafikko toteutti Frozen Alliance -logotekstin ja sen alle asetettiin tekstiobjekti mainosvalikon kuvausta varten. Mainoslista toteutettiin myös Panel-objektilla, mutta lisäksi sille asetettiin Scroll Rect -komponentti, jonka avulla toteutettiin listan vierittäminen ylös ja alas. Mainoslistan lapsiobjektiksi lisättiin vielä Ads-objekti, jolle asetettiin Vertical Layout Gourp -komponentti. Tämän komponentin avulla Ads-objektin alle ilmennettävät mainosobjektit järjestetään pystysuunnassa. (Kuva 29.)

KUVA 29. Käyttöliittymäobjektien hierarkia

KUVA 30. Mainosvalikon graafinen käyttöliittymä

4.5 Pilvipalvelujen käyttöönotto

Hallintasovelluksen ja verkkorajapinnan käyttöönottoa varten Azure-pilvipalvelussa molemmille luotiin verkkosivuinstanssit Azuren portaalista. Hallintasovelluksen verkkosivuinstanssia luodessa asetettiin valinta uuden MySQL-tietokannan luontia varten (kuva 31). Näin luotu tietokanta asetettiin myös verkkorajapinnan käytettäväksi sen verkkosivuinstanssia luotaessa.



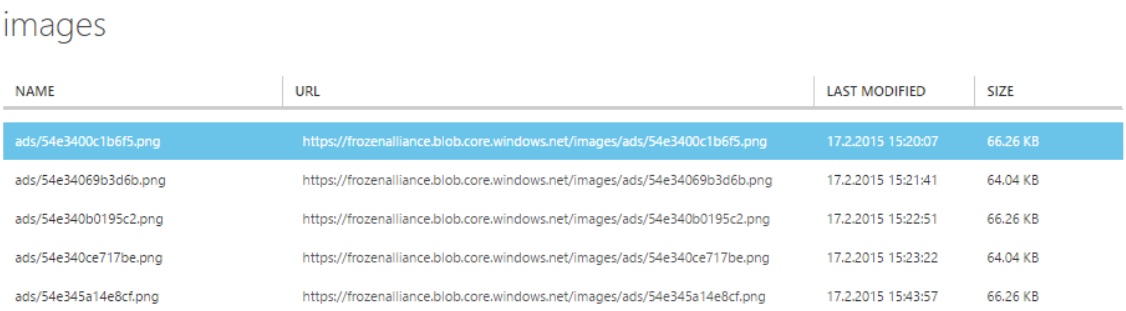
KUVA 31. Uuden verkkosivuinstanssin luonti MySQL-tietokannan kanssa

Kun verkkosivuinstanssit oli luotu, asetettiin ne käyttämään Git-versionhallinta-ohjelmistoa sovellusten siirtämistä varten. Tämän jälkeen molemmista verkkosivuinstansseista kloonattiin Gitin avulla toistaiseksi tyhjät projektikansiot kehitysympäristöön, joihin siirrettiin sovellusten lähdekoodit. Koska Azure palvelee verkkosivuja Microsoftin IIS-palvelinohjelmistolla, jouduttiin sovellusten juureen vielä lisäämään web.config-tiedosto, jolla palvelin konfiguroitiin toimimaan oikein Yii-verkkosovelluksen kanssa (kuva 32). Lopuksi versionhallintaohjelmiston Push-toiminnolla sovellusten lähdekoodit siirrettiin Azureen.

```
<system.webServer>

<directoryBrowse enabled="false" />

    <rewrite>
        <rules>
        <rule name="Hide Yii Index" stopProcessing="true">
            <match url="." ignoreCase="false" />
            <conditions>
            <add input="{REQUEST_FILENAME}" matchType="IsFile" ignoreCase="false" negate="true" />
            <add input="{REQUEST_FILENAME}" matchType="IsDirectory" ignoreCase="false" negate="true" />
            </conditions>
                <action type="Rewrite" url="index.php" appendQueryString="true" />
        </rule>
        </rules>
    </rewrite>

</system.webServer>
```

KUVA 32. IIS-palvelimen konfiguraatio Yii-verkkosovellusta varten

Azuren Storage-palvelun käyttöönottoa mainoskuvien tallentamista varten täytyi Azuren portaalista luoda uusi säilytystili. Tämän jälkeen hallintasovelluksessa täytyi ottaa Azure SDK-sovelluskehitystyökalut käyttöön. Azure SDK sisällytettiin hallintasovelluksen lähdekoodiin Giuliano Iacobellin tekemän Yii-lisäosan avulla (14). Lisäosa otettiin käyttöön lisäämällä sille määrittelyt hallintasovelluksen konfiguraatiotiedostoon. Kun Azure SDK oli käytettävissä, muutettiin hallintasovelluksen ohjelmakoodia siten, että se tallentaa mainoskuvat uuden säilytystilin binääriobjekteille tarkoitetun osion images-säiliöön ja hakee ne sieltä (kuva 33). Tämän jälkeen määriteltiin vielä päätelaitekomponentille kuvien sijainti säilytystilin URL-osoitteella.



KUVA 33. Images-säiliö Azuren portaalissa

4.6 Viimeistely

Kun prototyyppijärjestelmä oli valmis, siirryttiin testaamaan sitä. Tarkoituksena oli käytännössä toteuttaa integraatiotestausta järjestelmän eri komponenttien välillä ja todeta kaikkien toteutettujen ominaisuuksien toimivuus. Järjestelmälle ei kuitenkaan aikataulusyistä ehditty toteuttamaan rasitustestausta eikä sen suorituskykyä pystytty mittaamaan, mutta se ei toisaalta myöskään ollut tämän työn tarkoitus.

Testaus toteutettiin kolmella eri mobiililaitteella siten, että palvelinympäristö toimi Azuressa, ja päätelaitekomponenttia ajettiin kahdessa eri pelisovelluksessa. Toinen oli kehityksen aikana toteutettu esimerkkisovellus, joka sisälsi vain päätelaitekomponentin ja toinen oli Frozen Visionilla kehityksessä ollut mobiilipeli. Kaikki toteutetut ominaisuudet todettiin testauksen aikana toimiviksi.

Kun järjestelmä oli testattu, kirjoitetiin hallintasovelluksen käyttöä ja päätelaite-komponentin käyttöönottoa varten lyhyet ohjekirjat englanniksi. Nämä ohjeet ovat tämän työn liitteenä.

5 YHTEENVETO

5.1 Työn onnistuminen

Järjestelmän laajuus ja sen kehitykselle varattu aikataulu aiheuttivat sen, että projektin tavoitteet asetettiin alkuun lievästi alakanttiin. Työn toteutus lähti kuitenkin etenemään huomattavasti odotettua nopeammin, mikä osittain johtui siitä, että osasta valituista teknologioista oli aiempaa kokemusta. Tämä oli tietysti tietoinen valinta, sillä näillä valinnoilla piti myös kyetä toteuttamaan toiminnallisuuksia ja arkkitehtuuri, joista puolestaan ei ollut aiempaa kokemusta.

Työn hyvä eteneminen mahdollisti sen, että loppuvaiheessa kyettiin määrittelemään ja lisäämään järjestelmän sujuvuuden kannalta olennaisia toiminnallisuuksia. Järjestelmän käyttöönoton mahdollisuuksia tulevaisuudessa pystyttiin laajemmin tutkimaan ja myös sen jatkokehityksen suunnittelulle jäi aikaa. Näin ollen tämän työn tuloksien pohjalta on hyvä lähteä tulevaisuudessa rakentamaan varsinaista järjestelmää.

Kaiken kaikkiaan tämä projekti onnistui jopa odotettua paremmin. Toki on asioita, jotka olisi voinut ottaa paremmin huomioon, kuten järjestelmän monipuolinen testaaminen, mutta toisaalta työn tuloksena on kuitenkin prototyyppi. Tämän järjestelmän tarkoituksena ei ole tulla laajamittaiseen käyttöön, vaan se toimii pohjatyönä tulevaa varten.

5.2 Järjestelmän jatkokehitys

Tätä prototyyppijärjestelmää kehitettäessä ei pystytty järjestelmän laajuuden vuoksi juurikaan keskittymään tietoturvallisuuteen tai suorituskykyyn. Niin hallintasovellus kuin verkkorajapintakin varsinaisessa järjestelmässä vaatisivat tietoturvallisen tavan tunnistautumiselle. Hallintasovelluksen tapauksessa järkevin ratkaisu luultavasti olisi claims-pohjaisen tunnistautumisen toteuttaminen, jonkun ulkopuolisen palveluntarjoajan hallitessa käyttäjien tietoja. Näin tässä järjestelmässä ei tarvitsisi huolehtia käyttäjätunnuksien ylläpidosta ja niiden turvallisuudesta.

Suorituskykyyn varsinkin verkkorajapinnan osalta olisi syytä kiinnittää huomiota, sillä järjestelmän tullessa laajamittaiseen käyttöön olisi sen kyettävä toimimaan suuren rasituksen alla. Parasta tapaa toteuttaa vastaavanlainen verkkorajapinta olisi hyvä tutkia, jotta sitä voitaisiin kehittää vielä paljon suorituskykyisemmäksi ja helpommin skaalautuvaksi. Suorituskyvyn kannalta olisi myös hyvä tutustua paremmin eri pilvipalveluiden tarjoamiin palveluihin.

On myös järjestelmän kannalta tärkeitä ominaisuuksia, joita ei tässä työssä toteutettu. Esimerkiksi hallintasovellus vaatisi useamman eri käyttäjätason ja mainosstatistiikan seuraamista varten tarvittaisiin lisää erilaisia näkymiä. Päätelaitekomponentti Unity-ympäristössä puolestaan on hyvin avoin ja pelinkehittäjä pääsee käsiksi sen lähdekoodiin. Mahdollisuutta päätelaitekomponentin sulkemiseen ajonaikaiseen kirjastoon voisi myös siis tutkia.

LÄHTEET

1. Suvanto, Soile 2014. Fingersoftin mäkiautopeliä on ladattu jo yli 170 miljoonaa kertaa. Yle Uutiset. Saatavissa: http://yle.fi/uutiset/fingersoftin_makiautopelia_on_ladattu_jo_yli_170_miljoonaa_kertaa/7168989. Hakupäivä 24.2.2015.

2. PHP MySQL Database. W3schools.com. Saatavissa: http://www.w3schools.com/php/php_mysql_intro.asp. Hakupäivä 27.2.2015.

3. What is MySQL? MySQL. Saatavissa: http://dev.mysql.com/doc/refman/5.1/en/what-is-mysql.html. Hakupäivä 27.2.2015.

4. About Yii. Yii PHP Framewok. Saatavissa: http://www.yiiframework.com/about/. Hakupäivä 11.3.2015.

5. Features of Yii. Yii PHP Framework. Saatavissa: http://www.yiiframework.com/features/. Hakupäivä 11.3.2015.

6. Ilmainen postinumerorajapinta. Tietokannat.fi Suomi Oy. Saatavissa: http://www.tietokannat.fi/postinumerot.php. Hakupäivä 11.3.2015.

7. The Best Development Platform For Creating Games. Unity. Saatavissa: http://unity3d.com/unity. Hakupäivä 12.3.2015.

8. JSON IP and GeoIP REST API. Telize. Saatavissa: http://www.telize.com/. Hakupäivä 12.3.2015.

9. SimpleJSON. Unify Community. Saatavissa: http://wiki.unity3d.com/index.php/SimpleJSON. Hakupäivä 23.3.2015.

10. Startups: Get FREE Microsoft Azure cloud services. BizSpark. Saatavissa: http://www.microsoft.com/bizspark/. Hakupäivä 12.3.2015.

11. What is Microsoft Azure? Microsoft Azure. Saatavissa: http://azure.microsoft.com/fi-fi/overview/what-is-azure/. Hakupäivä 12.3.2015.

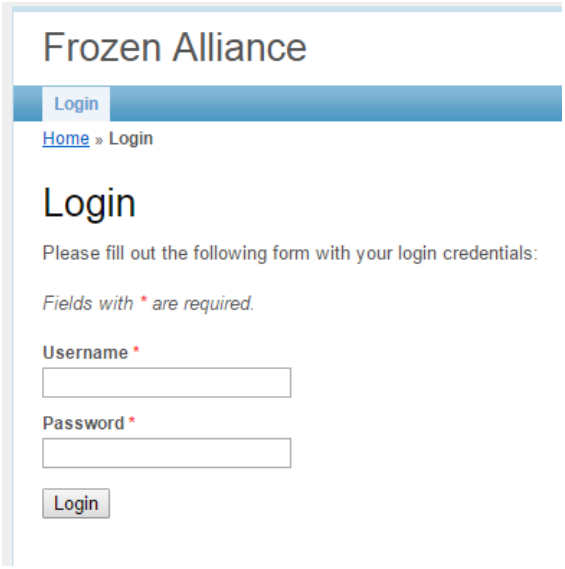
12. Boucher, Rob 2014. Introducing Microsoft Azure. Microsoft Azure. Saatavissa: http://azure.microsoft.com/en-us/documentation/articles/fundamentals-introduction-to-azure/. Hakupäivä 12.3.2015.

13. Ullman, Larry 2010. Forcing Login for All Pages in Yii. Saatavissa: http://www.larryullman.com/2010/07/20/forcing-login-for-all-pages-in-yii/. Hakupäivä 17.3.2015.

14. Iacobelli, Giuliano 2013. Yii-azure. Saatavissa: https://github.com/Giuliano84/yii-azure. Hakupäivä 24.3.2015.

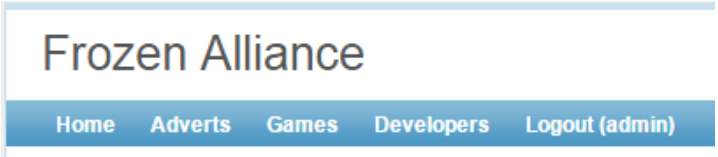
1. START

1.1. Login



Type your username and password to the appropriate input fields and then press Enter or click *Login*. If you type them incorrectly the application will give a warning and you can try again.

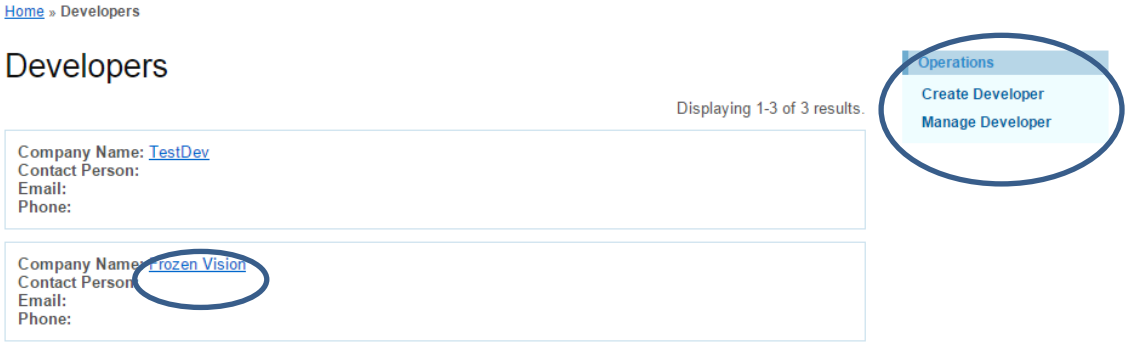
1.2. Navigation



After login you can navigate the pages via the navigation bar. The *Home* button will take you to the front page with basic about information. *Adverts*, *Games* and *Developers* will take you to the corresponding management pages. With the *Logout* button you can logout the current user.

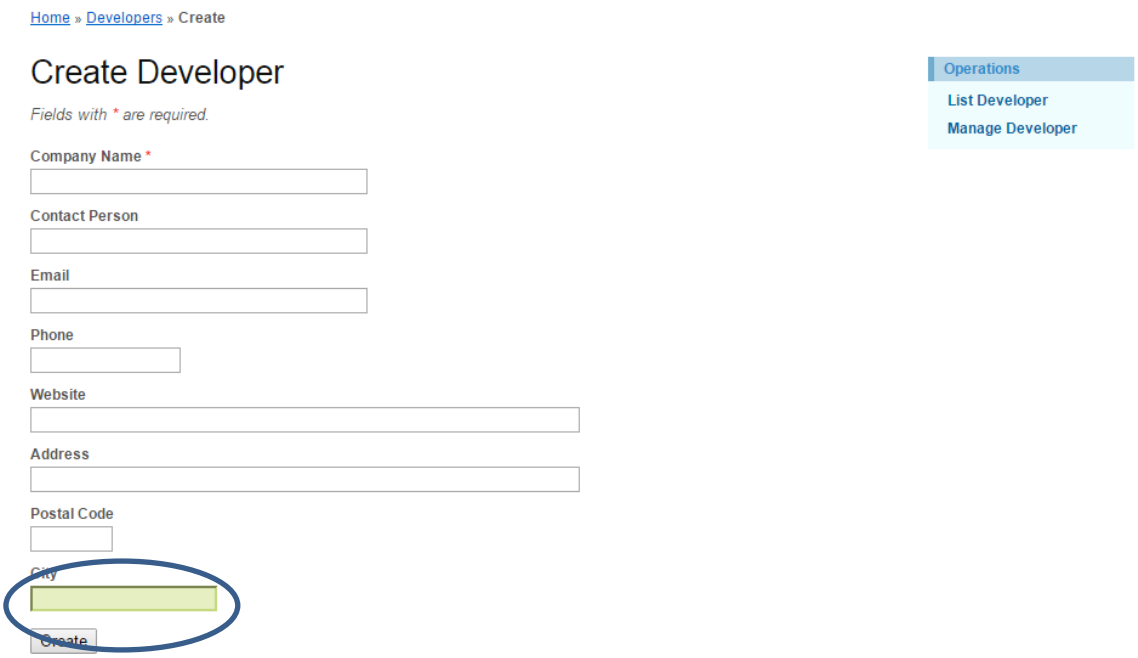
2. DEVELOPERS

2.1. List



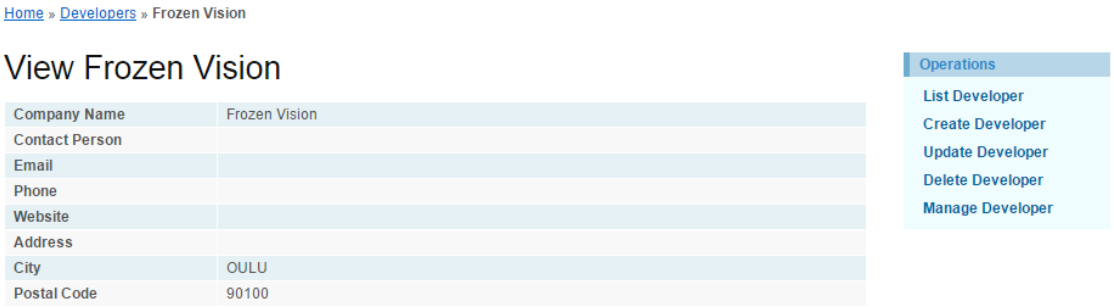
In the Developers index page you can see all the existing developers listed with some of their basic info. You can view a specific developer by clicking the *Company Name*. From the *Operations* panel you can access the *Create* and *Manage* pages.

2.2. Create



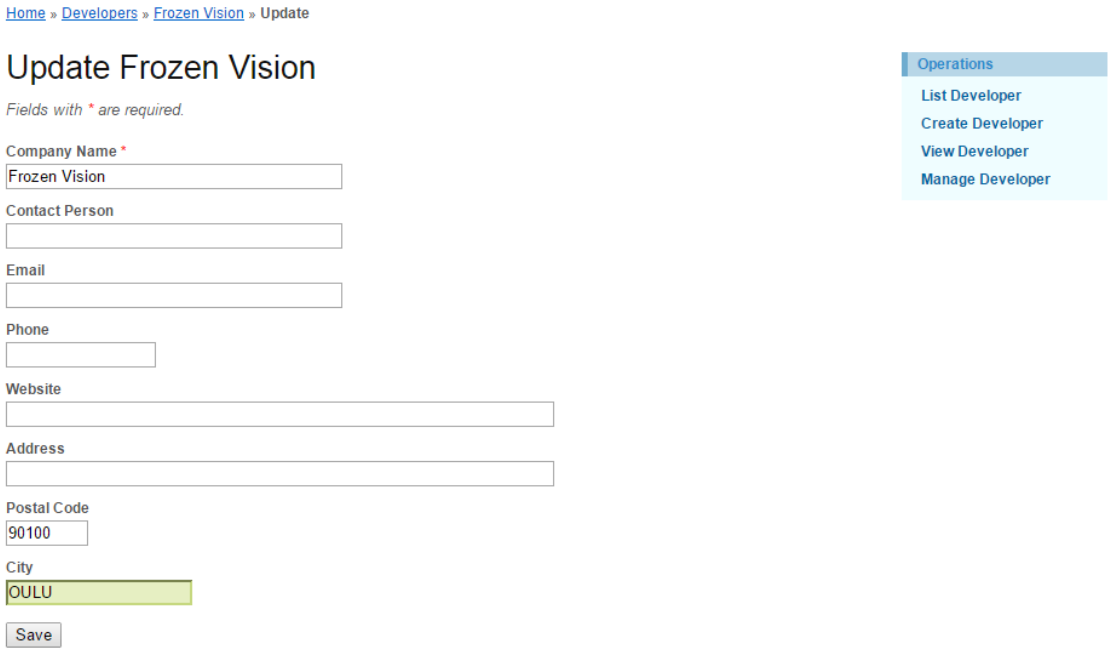
In the Developers create page you can input information for a new developer and create it in to the system. The developer is created by typing in the wanted information and then either by pressing Enter or clicking *Create*. *Company Name* is required and it has to be unique but all the other information are optional. *City* field is automatically updated according to the *Postal Code* field. You can navigate to the index and the management pages via the *Operations* panel.

2.3. View



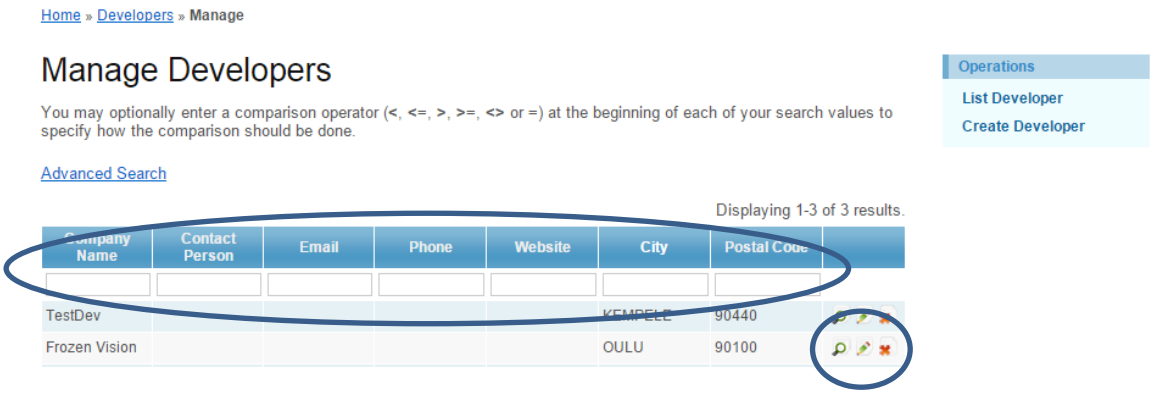
From the developer view page you can inspect all the developer information in a table format. Then you can use the *Operations* panel to navigate to developer index and manage pages, update the viewed developer, delete it or create a new one. If you choose to delete the developer the application will prompt a confirmation dialog.

2.4. Update



In the update page you can change the previously set developer information the same way as in the create page. From the *Operations* panel you can navigate to the index and management pages, get back to the view page of the developer or create a new one.

2.5. Manage

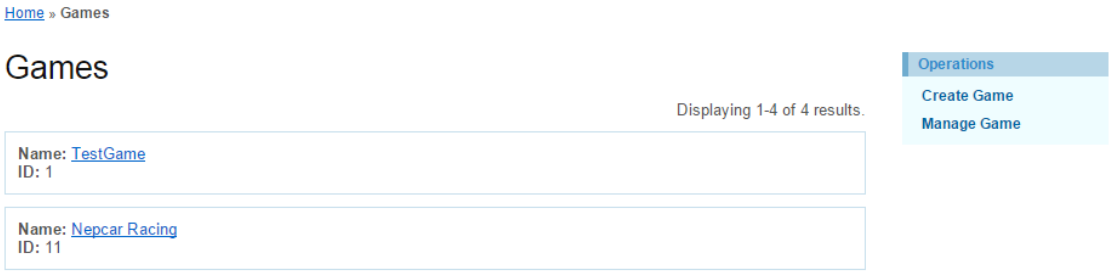


In the manage page you can list all the existing developers with filtering or sorting them by the attributes. For example by typing ”T” under the *Company Name* will filter all the companies with ”T” in their name or by clicking *Contact Person* will sort the developers by their contact persons. By clicking

the icons next to the developers you can either view, update or delete the corresponding developer. Delete option will prompt a confirmation dialog.

3. GAMES

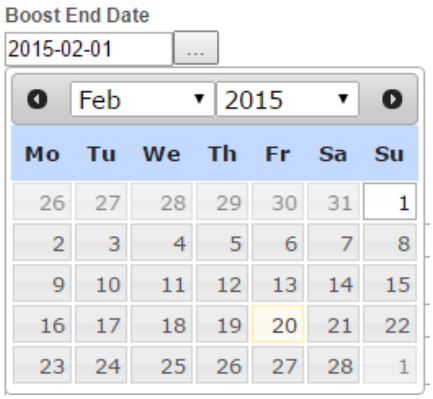
3.1. List



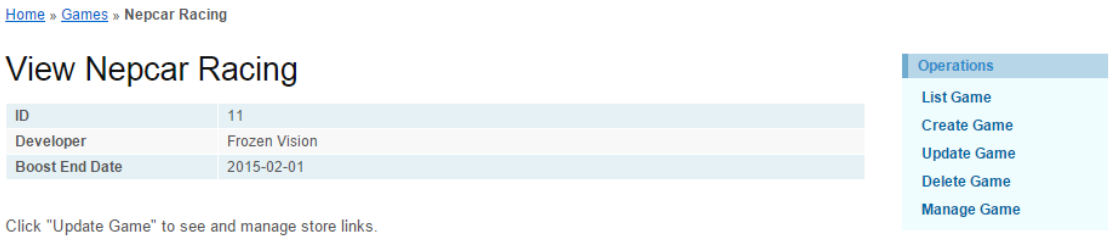
In the Games index page you can see all the existing games listed with their IDs. This ID is used with the API calls so the developer needs it for the Unity plugin configuration. You can view a specific game by clicking the *Name*. From the *Operations* panel you can access the *Create* and *Manage* pages.

3.2. Create

In the Games create page you can input information for a new game and create it in to the system. The game is created by typing in the name, choosing *Developer* from the drop-down list and then either by pressing Enter or clicking *Create*. *Name* and *Developer* are required and the name has to be unique. *Store Links* and the *Boost End Date* are optional and the boost date is set by a date picker associated with the input field. You can navigate to the index and the management pages via the *Operations* panel.

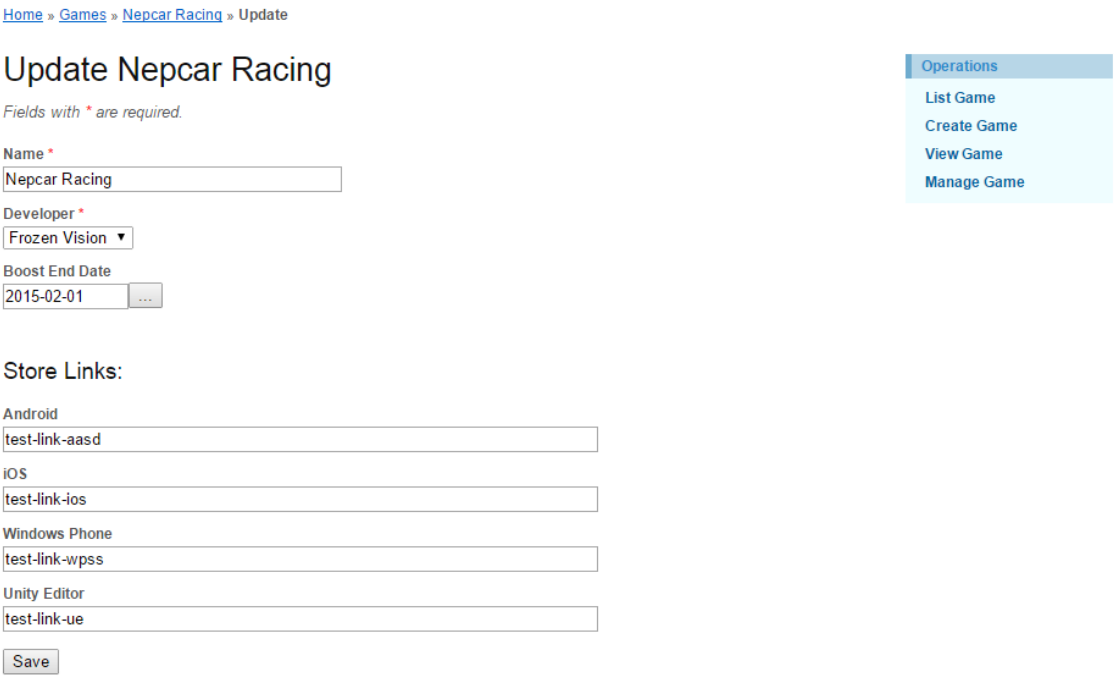


3.3. View



From the game view page you can inspect the game in a table format. Then you can use the *Operations* panel to navigate to game index and manage pages, update the viewed game, delete it or create a new one. If you choose to delete the game the application will prompt a confirmation dialog.

3.4. Update



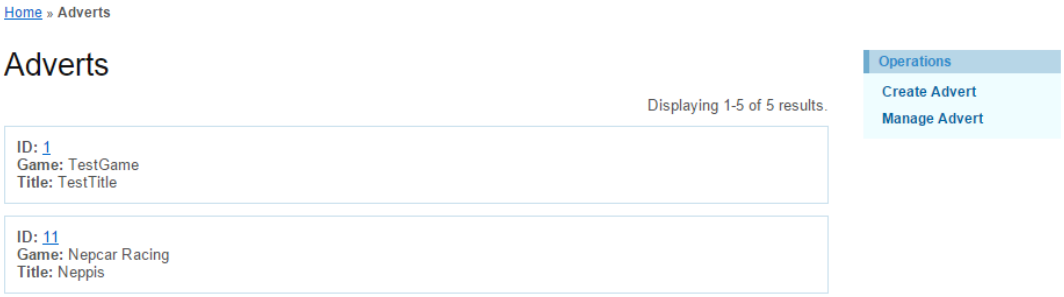
In the update page you can change the previously set game information the same way as in the create page. From the *Operations* panel you can navigate to the index and management pages, get back to the view page of the game or create a new one. By clearing a store link input field you delete it.

3.5. Manage

In the manage page you can list all the existing games with filtering or sorting them by their attributes in the same way as with the developers. *Developer* property has a drop-down list from which you can choose an existing developer as a filter. By clicking the icons next to the games you can either view, update or delete the corresponding game. Delete option will prompt a confirmation dialog.

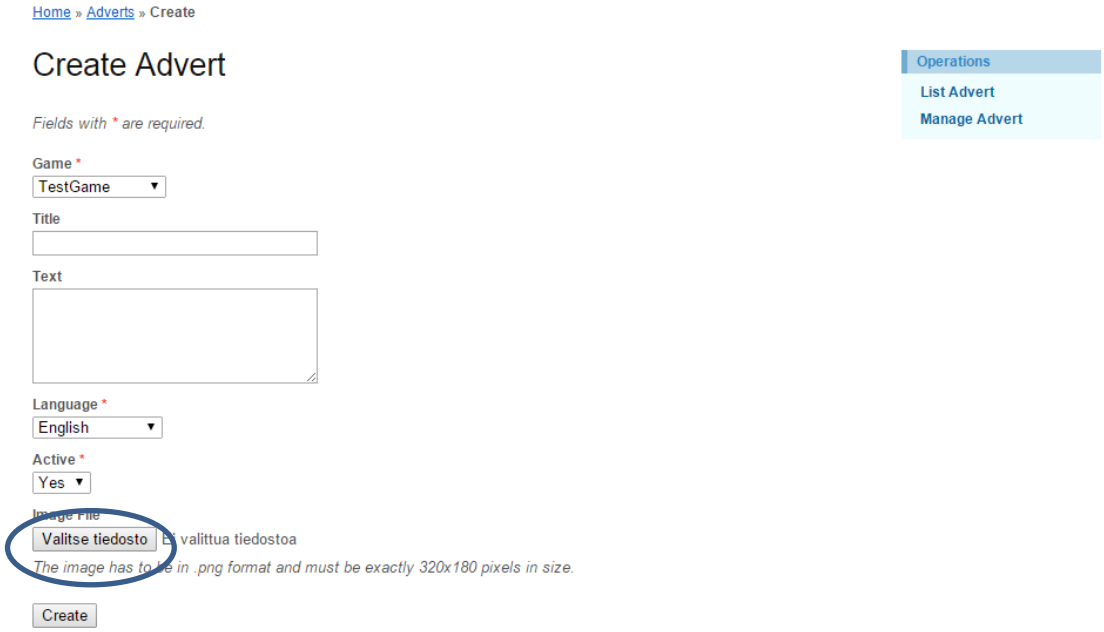
4. ADVERTS

4.1. List



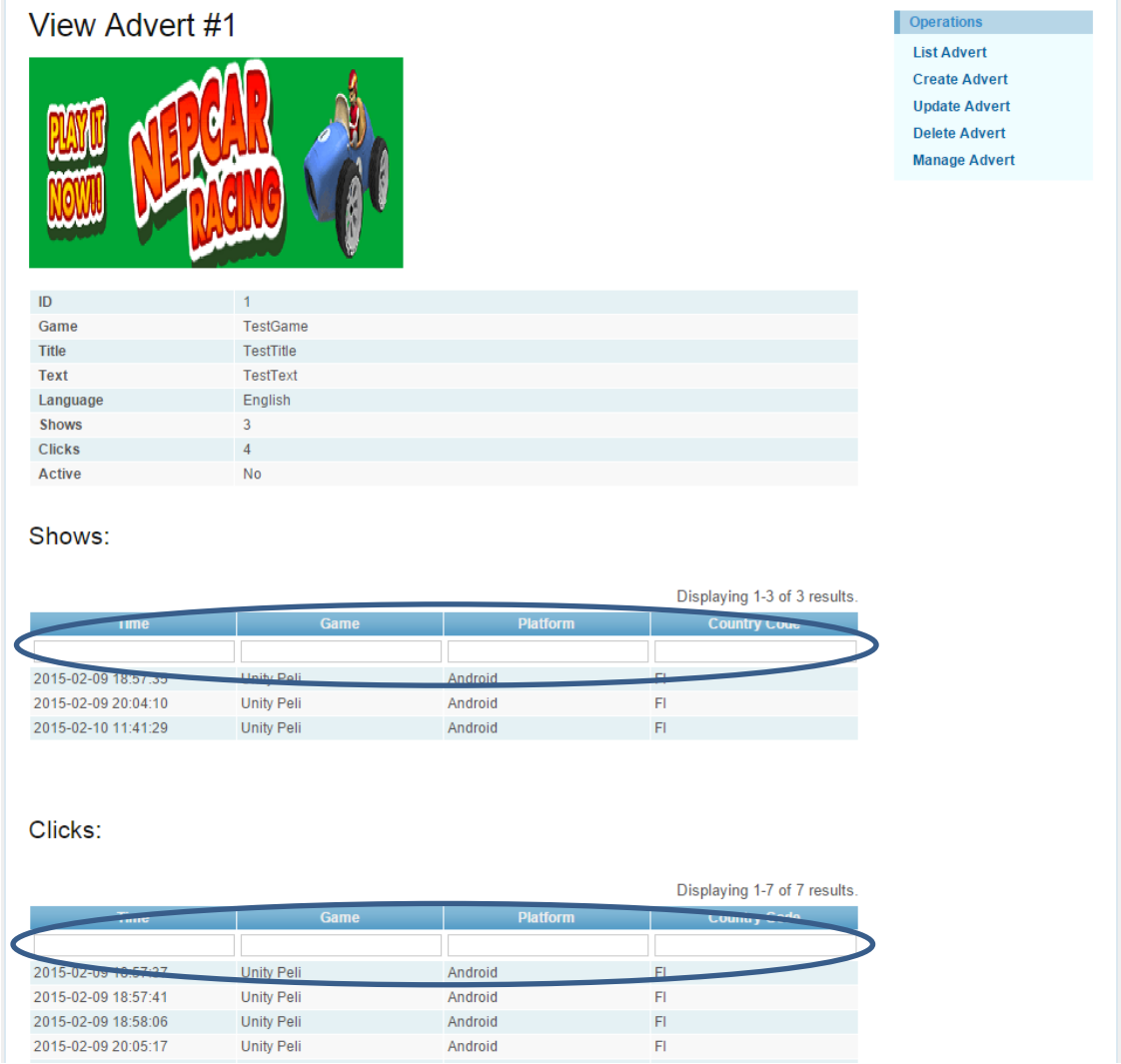
In the Adverts index page you can see all the existing adverts listed with some of their info. You can view a specific advert by clicking the *ID*. From the *Operations* panel you can access the *Create* and *Manage* pages.

4.2. Create



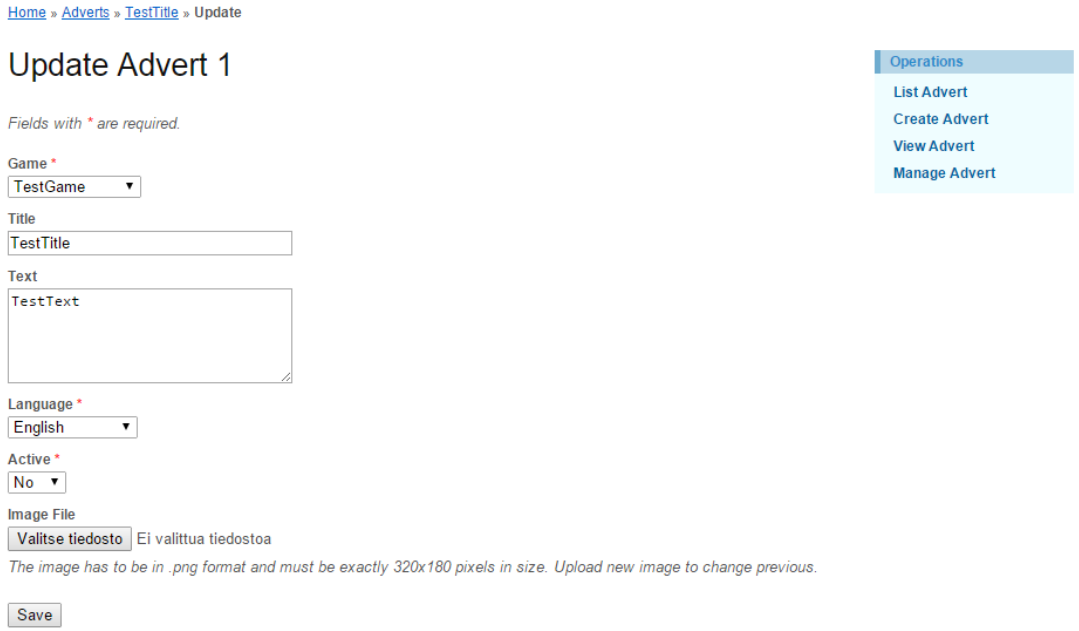
In the Adverts create page you can input information for a new advert and create it in to the system. The advert is created by typing in the wanted information, choosing *Game*, *Language* and *Active* state from the drop-down lists, by choosing the image with the file dialog and then either by pressing Enter or clicking *Create*. *Game* and *Image File* are required. If *Active* state is set to *Yes* all the other adverts of that game are set to *No*. You can navigate to the index and the management pages via the *Operations* panel.

4.3. View



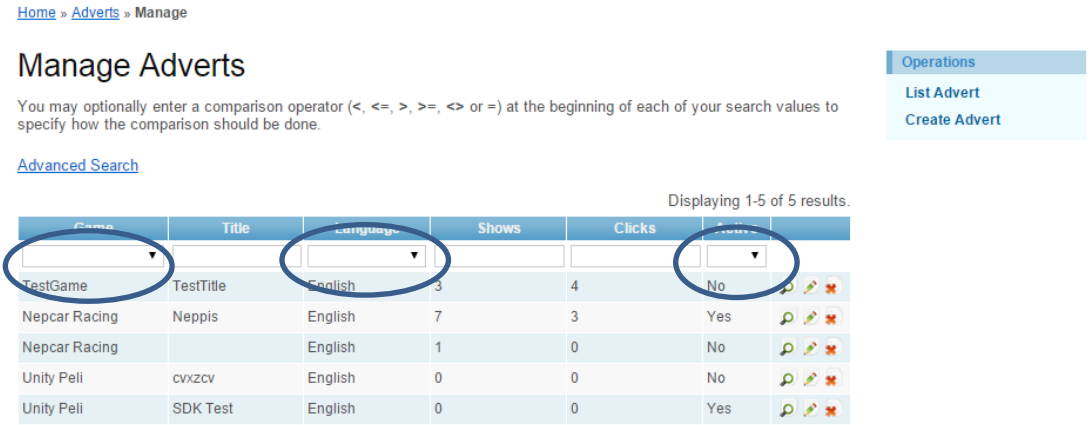
From the advert view page you can inspect the advert info and its *Shows* and *Clicks*. *Shows* and *Clicks* can be filtered and sorted with their properties. Then you can use the *Operations* panel to navigate to game index and manage pages, update the viewed advert, delete it or create a new one. If you choose to delete the advert the application will prompt a confirmation dialog.

4.4. Update



In the update page you can change the previously set advert information the same way as in the create page. To change the image upload a new one. From the *Operations* panel you can navigate to the index and management pages, get back to the view page of the advert or create a new one.

4.5. Manage

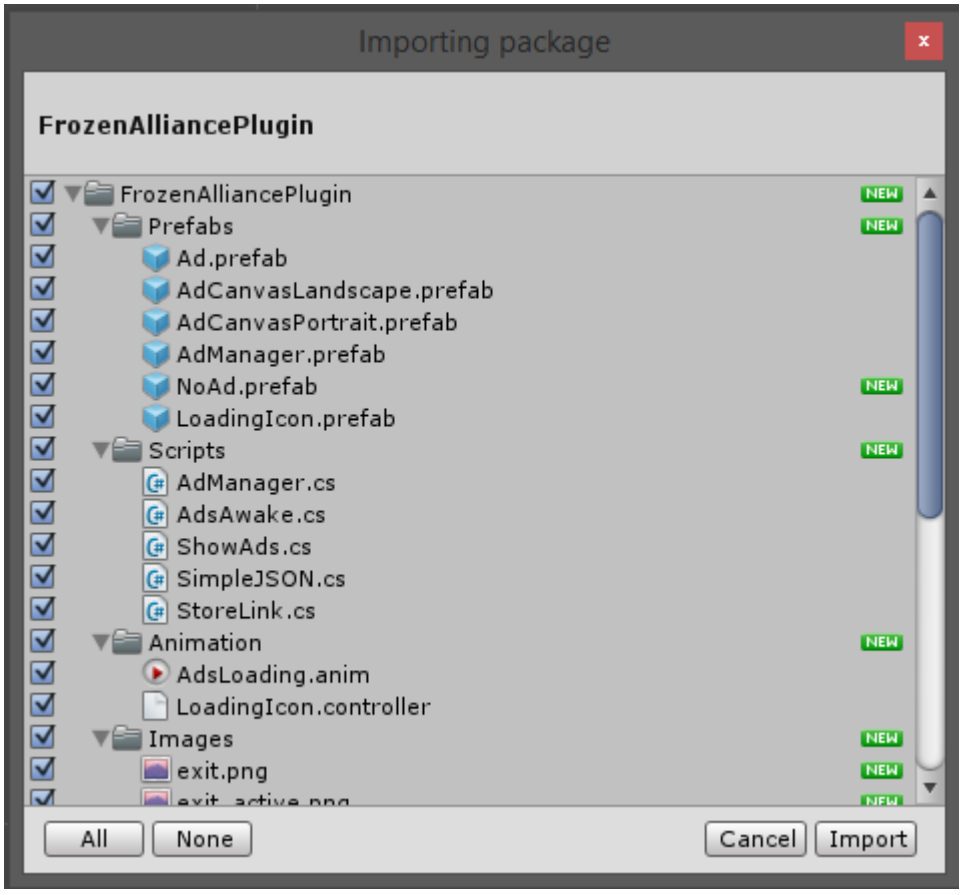


In the manage page you can list all the existing adverts with filtering and sorting them by their attributes in the same way as with developers and games. *Game* property has a drop-down list from which you can choose an

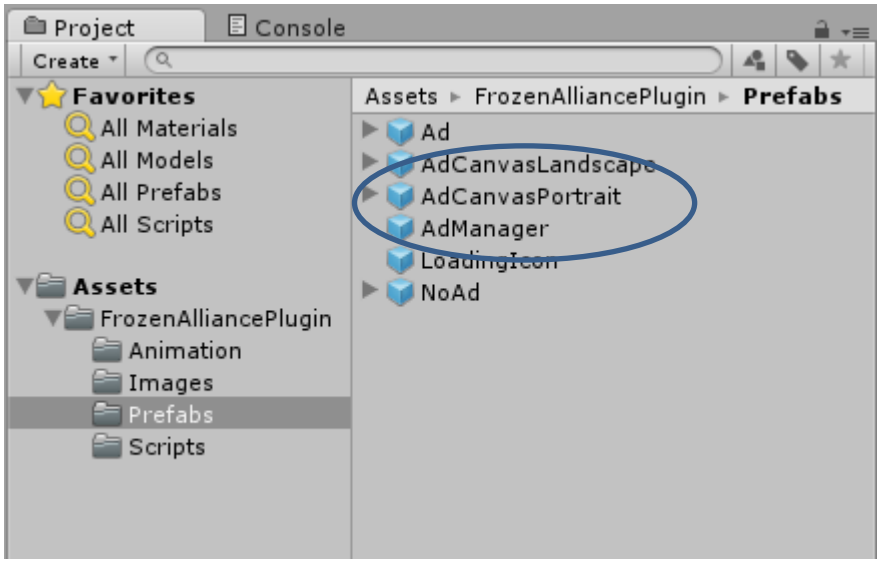
existing game as a filter and *Language* and *Active* have drop-down lists to choose filters for them. By clicking the icons next to the adverts you can either view, update or delete the corresponding advert. Delete option will prompt a confirmation dialog.

1. Import asset package

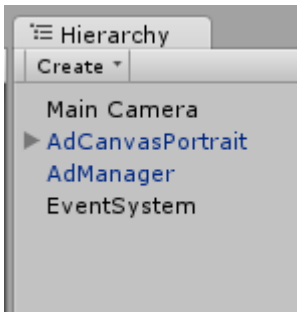
Frozen Vision will provide the necessary Unity 3D asset package to be used here. In Unity explorer click *Assets->Import Package->Custom Package…* and select the provided .unitypackage file from your computer. In the opening dialog check that every item is selected and then click *Import*.



2. Add prefabs to the hierarchy



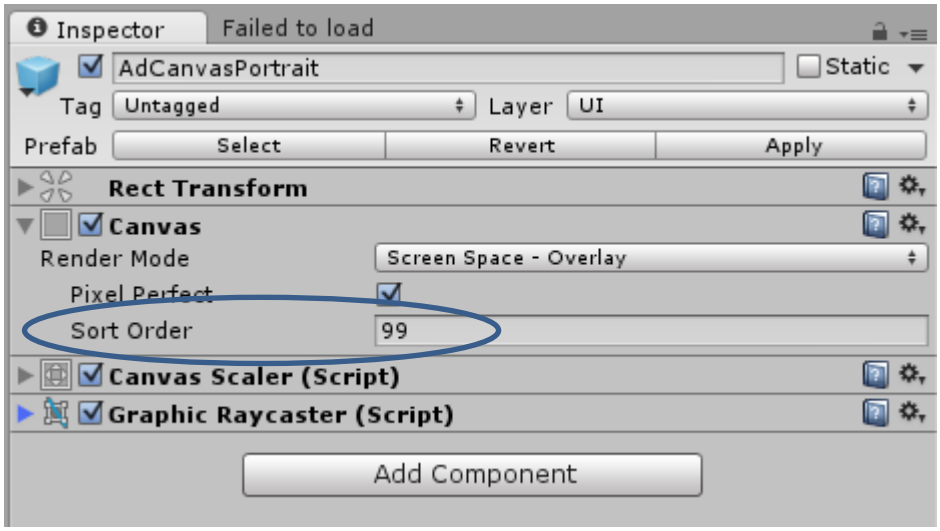
From your projects *Assets* directory under *FrozenAlliancePlugin/Prefabs* select either *AdCanvasLandscape* or *AdCanvasPortrait* prefab appropriate for your project and place it in the hierarchy of your main menu scene. Then place the *AdManager* prefab in the very first scene of your project (*could be same as the main menu*).



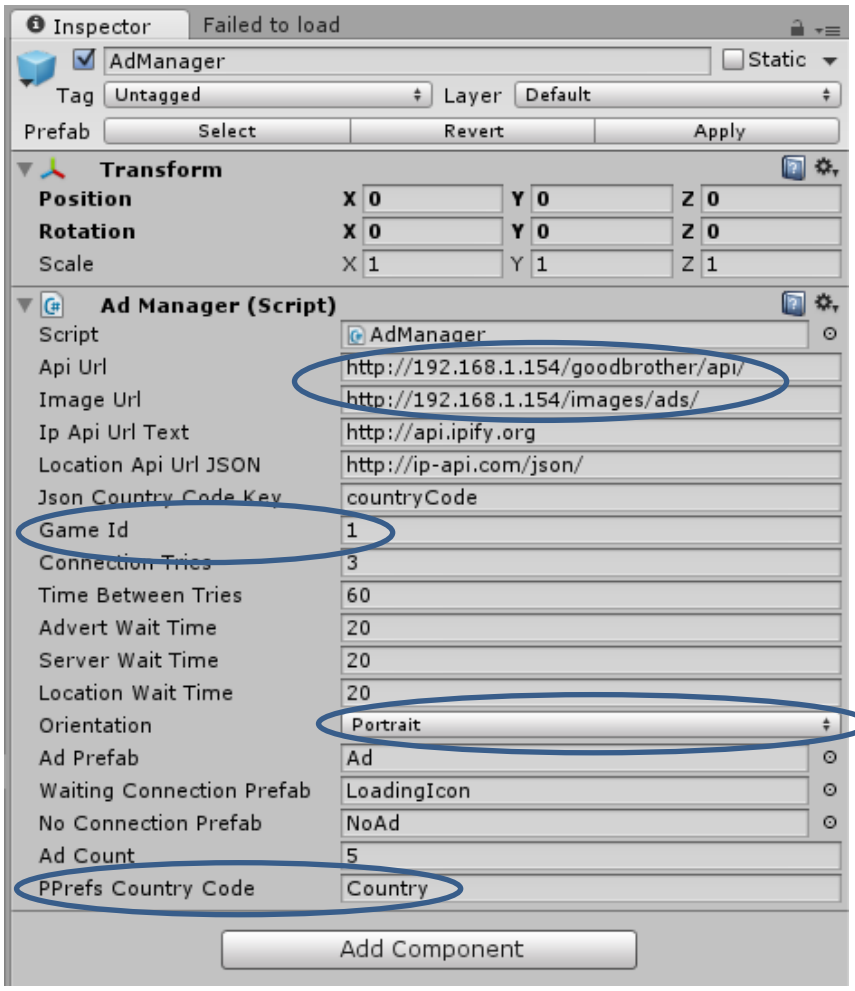
In the main menu scene check that you have *EventSystem* object in your hierarchy. If not click *Create->UI->EventSystem*.

3. Configure

Go to your main menu scene and select the *AdCanvas* object from the hierarchy. Check that the *Sort Order* property is set higher than in your other Canvas objects. This way the ad panel is always drawn last.



In the first scene of your project select *AdManager* object from the hierarchy. Set the *Api Url* and the *Image Url* properties with the information provided by Frozen Vision. Set the *Game Id* given to you accordingly. Choose the right *Orientation* for your project. Check that the *PPrefs Country Code* differs from the keys you use in your projects PlayerPrefs.


Inspector
Failed to load
AdManager
Static
Tag
Untagged
Layer
Default
Prefab
Select
Revert
Apply
Transform
Position
Rotation
Scale
Ad Manager (Script)
Script
AdManager
Api Url
http://192.168.1.154/goodbrother/api/
Image Url
http://192.168.1.154/images/ads/
Ip Api Url Text
http://api.ipify.org
Location Api Url JSON
http://ip-api.com/json/
Json Country Code Key
countryCode
Game Id
1
Connection Tries
3
Time Between Tries
60
Advert Wait Time
20
Server Wait Time
20
Location Wait Time
20
Orientation
Portrait
Ad Prefab
Ad
Waiting Connection Prefab
LoadingIcon
No Connection Prefab
NoAd
Ad Count
5
PPrefs Country Code
Country
Add Component