Risto Turtiainen

Entropia-tietokantajärjestelmä

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tietotekniikka

Insinöörityö

4.10.2016

Metropolia

Tekijä(t) Otsikko Sivumäärä Aika	Risto Turtiainen Entropia-tietokantajärjestelmä 47 sivua + 7 liitettä 4.10.2016
Tutkinto	Insinööri (AMK)
Koulutusohjelma	Tietotekniikka
Suuntautumisvaihtoehto	Ohjelmistotekniikka
Ohjaaja(t)	Lehtori Simo Silander

Insinöörityön tavoitteena oli käsitellä MMORPG-tietokonepelin sisältämää datan määrää ja rakentaa tämän datan varaan järjestelmä, joka esittää kyseistä dataa ja luo datan pohjalta arvioita, jotka pohjautuvat käyttäjän pelihahmon taitoihin ja siihen, mitä hänen tulisi tehdä pelissä.

Insinöörityössä luotiin MySQL-tietokanta, Spring-pohjainen verkkosovellus ja JFrame-pohjainen käyttöliittymäsovellus. Tietokantayhteyksissä käytettiin Hibernatea molempien sovellusten osalta. Verkkosovellukseen toteutettiin myös REST-rajapinta Jacksonia ja Jerseytä käyttäen. Verkkosovelluksen tiedon esitykseen käytettiin Javascript-kirjastoja DataTables ja BootStrap. Swing-pohjaisen käyttöliittymäsovelluksen tärkein ominaisuus oli suuren ja monimuotoisen tietomäärän vienti tietokantaan. Käyttöliittymäsovelluksella voi viedä joko XML- tai CSV-pohjaista tietoa tietokantaan, ja tämä tieto varmistetaan tietokanta-yhteensopivaksi XML-schema-tiedostolla. Sovellusten kehittämiseen käytettiin myös paikallista MySQL-tietokantaa.

Spring-pohjaisella verkkosovelluksella pystytään esittämään käyttäjälle arvioita käyttäjän esittämien tietojen mukaan tietokannasta löytyvän datan perusteella ja esittämään tietokannan sisältämää dataa, mutta sovelluksesta jäi uupumaan verkkokauppatyyppinen ominaisuus ja sisäänkirjautumismahdollisuus, jotka ovat osittaisesti tietokannassa toteutettu. Ominaisuudet jouduttiin jättämään insinöörityöstä pois aikarajoitteen takia, mutta ne tullaan toteuttamaan sovellukseen myöhemmin.

Swing-pohjaista käyttöliittymäsovellusta ei ole julkaistu, mutta Spring-pohjainen verkkosovellus on esillä kirjoittajan omalla palvelimella osoitteessa http://risto.turtiainen.eu:8080/MavenEntropiaWeb.

Avainsanat	MySQL, tietokanta, Spring, Rest, Jersey, Jackson, verkkosovellus, Swing, käyttöliittymäsovellus, DataTables, BootStrap

Author(s) Title Number of Pages Date	Risto Turtiainen Entropia Database System 47 pages + 7 appendices 4 October 2016
Degree	Bachelor of Engineering
Degree Programme	Information technology
Specialisation option	Software developing
Instructor(s)	Senior lecturer Simo Silander

The aim of this thesis was to handle the amount of data which is contained in MMORPG computer game and build System on this data which shows the data on web application and create estimations from the data and player character to the user.

MySQL database, Spring-based web application, and JFrame-based user interface application were created for the thesis. Hibernate was used for database connections in both applications. For the web application, a REST-interface was also created using Jersey and Jackson. Data displaying in web application was done by DataTables JavaScript library and BootStrap. Swing-based user interface application's main feature was importing a large and complex dataset to the database. The user interface application can export either XML or CSV-based data to the database and this information will be ensured to be database compatible with XML schema file. Also, local MySQL database was used for the development of applications.

Spring-based web application can display estimations for the user based on information he has submitted and the data from the database. Application also presents the data contained in the database for the user. As there was no time left, login feature and web shop were not implemented to the application, even though database has small implementation for them. These features will be implemented in future.

Swing-based user interface application has not been published, but the Spring web application has been deployed on writer’s own server at: http://risto.turtiainen.eu:8080/MavenEntropiaWeb.

Keywords	MySQL, database, Spring, Rest, Jersey, Jackson, web application, Swing, UI application, DataTables, BootStrap

Sisällys

Lyhenteet

1 Johdanto 1

2 Entropia Universe 1

3 Järjestelmän käyttötarkoitus 6

4 Tietokannan esittely 8

4.1 Rakenteen kuvaus 9
4.2 Taulujen yhteydet 10
4.3 Tietokannassa käytetyt teknologiat 11

5 Järjestelmän ylläpitäjän käyttöliittymäsovellus 11

5.1 Käytetyt teknologiat 12
5.1.1 Swing 12
5.1.2 JPA 12
5.1.3 Hibernate 13
5.1.4 JaxB 13
5.1.5 SAXParser 13
5.2 Sovelluksen elinkaari 14
5.2.1 Sovelluksen käynnistyminen 15
5.2.2 Tietokantakyselyiden luonti 16
5.2.3 Tiedon esitys 20
5.2.4 Tietokannan alustaminen uudelleen 23
5.2.5 Tiedon vienti kantaan 23
5.2.6 CSV-tiedon vienti tietokantaan 24
5.2.7 XML-tiedon vienti tietokantaan 27

6 Verkkosovellus 29

6.1 Käytetyt teknologiat 30
6.1.1 Maven 30
6.1.2 Spring 30
6.1.3 Hibernate 32
6.1.4 JAX-RS 32

6.1.5 Jersey 32
6.1.6 Jersey-spring 32
6.1.7 Jackson 33
6.1.8 Log4j 33
6.1.9 Ajax 33
6.1.10 BootStrap 33
6.1.11 JQuery 34
6.2 Sovelluksen elämänkaari ja rakenne 34
6.2.1 Sovelluksen käynnistyminen 35
6.2.2 Kyselyn luonti etusivun syöttölomakkeen kautta 36
6.2.3 Tiedon esittäminen Mob- ja Item-sivuilla 42

7 Järjestelmän toteutetut ja toteuttamattomat ominaisuudet 44

7.1 Muokkausmahdollisuus admin-käyttöliittymässä useampaan tauluun 44
7.2 Admin-käyttöliittymästä tiedon ulosvienti ominaisuus 44
7.3 Verkkosovelluksen kirjautumisominaisuus 45
7.4 Verkkosovelluksen verkkokauppapuoli 45
7.5 REST-rajapinnan laajennus kaikkiin tauluihin 45
7.6 Verkkosovelluksessa aseiden ja haarniskojen hakeminen tietokannasta suoraan 45
7.7 Järjestelmän jatkokehitys 46

8 Yhteenveto 46

Lähteet 47

Liitteet

Liite 1. Hibernate-konfiguraatiotiedosto hibernate.cfg.xml

Liite 2. Spring-konfiguraatiotiedosto web.xml

Liite 3. Spring-konfiguraatiotiedosto applicationContext.xml

Liite 4. Spring-konfiguraatiotiedosto dispatcher-servlet.xml

Liite 5. XML:n validointi schema-tiedosto xmlValidation.xsd

Liite 6. Admin-käyttöliittymäsovelluksen luokkakaavio

Liite 7. Verkkosovelluksen luokkakaavio

Lyhenteet

Admin	Järjestelmän ylläpitäjä, jolla on oikeus lisätä ja poistaa tietueita tietokannassa.
Entropia Universe	MMORPG-tietokonepeli, jossa pelaajat seikkailevat Scifi-maailmankaikkeudessa.
Jackson	Java-pohjainen XML- ja JSON- dataparseri.
Jersey	Java-pohjainen REST-rajapinnan toteutuskehys.
JSON	eli JavaScript Object Notation on avoimen standardin tiedostomuoto tiedonvälitykseen.
MMORPG	Massive multiplayer online role playing game on palvelimelle toteutettu tietokonepeli, jossa kaikki pelaajat pelaavat samalla palvelimella ja seikkailevat samassa maailmassa yhdessä.
Mob	Pelin sisällä oleva vihollinen.
Spring	Java-pohjainen verkkosovelluskehys.
Swing	Graafisen käyttöliittymän luontiin tarkoitettu kirjasto Javalle.
Stats	Pelaajan hahmon taidot, jotka kuvaavat hahmon kehitystä.
Enmatter	Pelissä maata kaivettaessa löytyvä jalostettava materiaali esim. öljy.
Item	Pelin mikä tahansa tavara.
TTValue	Pelin tavaroiden alkuperäinen arvo, jonka pelin tekijät ovat määritelleet tavaralle.
Loot	Pelin vihollisilta saatava tavaramäärä.

PED	Project Entropia Dollars eli PED on Entropia Universen sisäinen rahayksikkö, joka on suoraan verrannollinen Amerikan dollarin kurssiin suhteessa 1:10.
XML	Tiedon merkintäkieli, jolla tiedon merkitys on kuvattavissa tiedon sekaan.

1 Johdanto

Insinöörityön tarkoituksena oli luoda tietokantajärjestelmä, jossa hyödynnettäisiin dataa Entropia Universe MMORPG-pelistä. Tarkoituksena oli tutustua MMORPG-pelin sisältämään datamäärään ja hyödyntää tätä dataa verkkosovelluksessa. Innostus järjestelmän luontiin tuli kirjoittajan halusta tutustua MMORPG-tietokantaan, koska hän on kiinnostunut kehittämään oman MMORPG-pelin. Järjestelmä sisälsi Swingillä luodun graafisen adminin tietokannanhallintasovelluksen, Spring-pohjaisen verkkosovelluksen, joka sisälsi REST-rajapinnan JSON-datan hakemiseen, ja MySQL-tietokannan, jota molemmat sovellukset käyttivät. Verkkosovellus esittää pelin dataa käyttäjälle ja antaa hänelle arvioita hänen Entropia Universe -hahmon taitojen perusteella. Arviot sisältävät tiedon niistä vihollisista, joita pelaajan kannattaa metsästää Entropia Universe -pelissä. Swing-pohjainen adminin tietokannan hallintasovellus luotiin suurten ja monimuotoisten data määrien vientiä varten järjestelmään. Sovellus myös hahmottaa dataa paremmin ja dataa voi hallita sovelluksen kautta. Työ tehtiin kirjoittajan oman mielenkiinnon vuoksi aiheeseen ja työ toteutettiin kirjoittajan omasta aloitteesta. Insinöörityössä tehdyt sovellukset ovat käytössä työn kirjoittajalla. Sekä verkkosovellus että tietokanta ovat kirjoittajan omalla palvelimella toiminnassa.

2 Entropia Universe

Entropia Universe on massiivinen monen pelaajan verkkoroolipeli (MMORPG, Massively Multiplayer Online Role-playing Game), joka tarkoittaa verkon kautta monin pelattavia tietokoneroolipelejä, joissa pelaaja luo itselleen kuvitteellisen pelaajahahmon ja tämän jälkeen tutkii pelimaailmaa vuoro vaikuttaen muiden pelaajien kanssa. Entropia Universe pyrkii olemaan scifi-virtuaalitodellisuus, jossa pelaaja pystyy elämään scifi-hahmon elämään vieraassa ympäristössä. Entropia Universen galaksi sisältää kuusi planeettaa, pari kuuta ja useita avaruuskeskuksia, joita pelaaja voi tutkia. Jokainen planeetta tarjoaa erilaisen maailman ja oman talouden pelaajan tutkittavaksi. Hahmon luotuaan pelaaja ei saa juuri mitään pelimaailmassa, vaan joutuu pärjäämään pelkän PED-kortin ja Vibrant Sweat Extractor Mk 1:sen kanssa. PED-kortti on pankkikorttityyppinen esine, jolle tallennetaan kaikki rahat Entropiassa, ja Vibrant Sweat Extractor Mk 1 on työkalu, jolla pelaaja voi hankkia vihollisesta hikeä (sweat), jota hän voi myydä muille pelaajille hankkiakseen pelin aloitusta varten rahaa. Muut pelaajat voivat jalostaa eläinten hien ja Force Nexus -

nimisen enmatterin Light Mind Essenceksi, jota pelaajat käyttävät teleportaamiseen pelimaailmassa. Pelaaja myös saa taitoja (skills) hien keräämisestä ja pelaajan hahmo saa vähitellen taitoja kaikesta, mitä hän tekee Entropiassa. Taitojen kehittäminen on pääidea Entropiassa: mitä enemmän hahmolta löytyy taitoja, sitä parempi hahmo on ja sitä paremmin hahmo pärjää maailmassa.

Kun pelaaja on hankkinut alustavia taitoja hikeä keräämällä ja tekemällä hahmoille tarkoitetut aloitustehtävät, pelaaja voi harjoittaa jotakin ammattia Entropia Universessä. Ammatteja voi harrastaa vapaasti, mutta tietyt ammatit vaativat tiettyjä ennakkovaatimuksia. Ammatteja Entropiassa on karkealla jaolla: metsästäjä (hunter), kaivaja (miner), käsityöläinen (crafter) ja kauppias (trader). Ammattiin ei tarvitse lukkiutua, vaan pelaaja voi harrastaa kaikkia ammatteja vaihtelevasti ja tietyssä määrin tarvitseekin, mutta on hyvä keskittyä yhteen ammattiin etenkin pelaamisen alkuvaiheessa.

Kuva 1. Entropia Universe -pelaaja metsästämässä. Vihollisen (Atrox) päällä näkyy pelaajan tekemä vahinko (56,9 ja 46,9), ja oikeassa alakulmassa pelaajan HP (elämä) mittarin päällä näkyy vihollisen tekemä vahinko (1.0)

Metsästäjän ammatissa pelaaja hankkii aseen, ammuksia ja sopivan haarniskan ja niitä käyttäen metsästää pelimaailmasta löytyviä vihollisia, kuten kuvassa 1. Vihollisilta metsästäjä saa tavaroita (lootia), kuten kuvassa 2, jonka hän myy muille pelaajille joko suoraan tai Auction Housen eli pelin sisäisen huutokaupan kautta.

Kuva 2. Entropia Universen pelaaja on saanut metsästäessä tavaroita (loot) vihollilselta. Saadut tavarat näkyvät kuvan vasemmassa ylälaidassa.

Jos metsästäjä laittaa saadut tavarat myyntiin huutokauppaan, hänen tulee maksaa siitä huutokauppa maksu, joka vaihtelee tavaran hinnan mukaan 0,5 PED:stä ylöspäin. PED eli Project Entropia Dollar on Entropia Universen sisäinen rahayksikkö. Se on suoraan verrannollinen Amerikan dollariin suhteessa 1:10. Metsästäjä pyrkii metsästämään vihollisia, joilta saa mahdollisimman arvokasta tavaraa muiden pelaajien silmissä. Entropian tavaroilla on perusarvo (TTValue eli Trade Terminal Value), jonka pelin tekijät ovat jokaiselle tavaralle määrittäneet. Pelaajat maksavat tavaroista niiden tarpeen ja harvinaisuuden mukaan ylimääräistä, jolloin tavaroille syntyy markkinalisäarvo eli MarkUp. Markkinalisäarvoa voi olla kahta eri tyyppiä. Tavaroilla, jotka eivät ole kulutustavaroita ja eivät hajoa, esimerkiksi aseet, omaavat markkinalisäarvon, joka määritellään PED:issä, esimerkiksi +100 PED. Tällöin jos tavaran hinta on 100 PED ja markkinalisäarvon +100 PED, tulee tavaran hinnaksi 200 PED. Muiden tavaroiden markkinalisäarvo määritellään prosenteissa esimerkiksi 120 % eli jos tavaran perusarvon ollessa 5 PED:iä on kaupan hinta markkinalisäarvo mukaan lukien 6 PED:iä. Metsästäjät pyrkivät hankkimaan harvinaisimpia tavaroita vihollisilta saadakseen mahdollisimman hyvät markkinalisäarvot tuotteilleen. Insinöörityössä toteutettu järjestelmä käsittelee pääsääntöisesti metsästäjän ammattia ja pyrkii antamaan metsästäjälle vinkkejä siitä, mitä vihollisia hänen tulisi metsästää parhaiden markkinalisäarvo tavaroiden saamiseksi sen hetkisillä taidoillaan, aseillaan ja haarniskoillaan. Metsästäjän taitojen kehittyessä hän voi hankkia parempia

aseita ja parempia haarniskoita, jotka sallivat pelaajan metsästää hankalampia ja harvinaisempia vihollisia.

Pelaajan ollessa kaivaja hän tekee hyvin samanlaista asiaa kuin metsästäjä. Hän pyrkii löytämään mineraaleja ja materiaaleja pelimaailmasta ja myymään niitä mahdollisimman hyvään hintaan muille pelaajille. Mineraalit ovat pelimaailmasta löytyviä malmeja ja materiaalit ovat luonnosta löytyviä käsityöhön hyödynnettäviä nesteitä tai tavaroita, joita ovat esimerkiksi juuret ja öljy.

Kuva 3. Pelaaja kaivamassa mineraaleja. Vasemmassa laidassa näkyy pelaajan käyttämä etsin (finder) ja keskellä alareunaa näkyy finderin havainnollistava käyttöliittymä.

Pelaaja käyttää etsintälaitetta eli finderia, jolla hän ampuu maahan syvyyspommin (mining probe) kuten kuvassa 3. Se paljastaa räjähdettyään, löysikö pelaaja maasta mineraaleja tai materiaaleja, kuten kuvassa 4.

Kuva 4. Pelaaja löysi pienen (Small (V)) kätkön Narcanisum Stonea. Kätköä kuvastaa kaivausmerkki, josta pelaaja voi kaivaa extractorilla eli kaivurilla.

Kun pelaajan taidot kehittyvät, hän pystyy käyttämään hyötysuhteeltaan, etäisyydeltään ja syvyydeltään parempia etsimiä, jolloin pelaaja voi löytää harvinaisempia ja arvokkaampia mineraaleja tai materiaaleja. Jos pelaaja löysi jotakin, hän voi kaivaa löytönsä extractorilla eli kaivajalla, kuten kuvassa 5.

Kuva 5. Pelaaja on kaivannut extractorilla eli kaivurilla Narcanisum Stonea, jonka arvo on 1,52 PED:iä ja niitä on 19 kappaletta.

Trader-ammatti eli kauppias on haastava ammatti, eikä se sovellu vasta pelin aloittaneelle pelaajalle. Trader-ammatti vaatii suuren rahallisen aloitussumman, koska tavoitteena on haalia muilta pelaajilta heidän metsästämiä tai hankkimia tavaroita halvalla ja myydä niitä eteenpäin tehden voittoa. Kauppiaat myös hyödyntävät paljon pelin sisäistä huutokauppaa ostaen sieltä halvalla sulkeutuvat huutokauppakohteet. He hankkivat tarpeeksi suuria tavaramääriä, että ne ovat järkeviä myydä huutokaupassa. Kauppiaat pidetään yleensä piheinä ja huijareina, koska suurin osa heidän ostohinnoista on huonoja, mutta heitä tarvitaan pelissä, jottei pelin talous romahtaisi ja etteivät tavaroiden hinnat laskisi liian alhaisiksi. Kauppiaat yleensä myös välittävät Entropia Universessä muille pelaajille asuntoja, joihin pelaajat voivat sisustaa oman huoneistonsa. Kauppiaat myös hankkivat itselleen pelistä kaupan, johon he voivat laittaa esineitä esille ja myyntiin. Näin kauppiaat välttävät huutokaupanmaksut. Uutena ominaisuutena Entropia Universessä on oman kodin rakentaminen, ja jää nähtäväksi, mikä on kauppiaiden suunnitelma tämän ominaisuuden kohdalla.

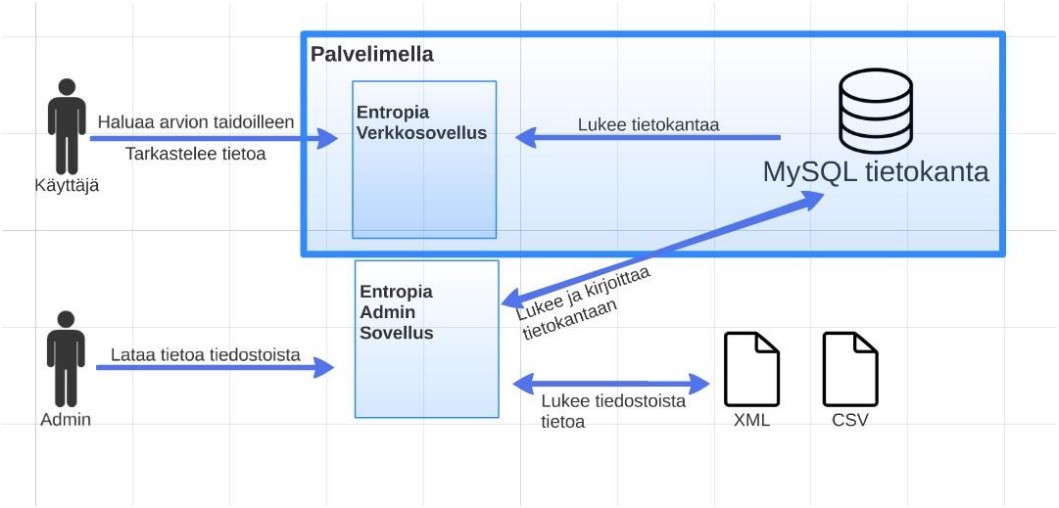
Käsityöläinen eli crafter hyödyntää kaivajan ja metsästäjän hankkimia tavaroita. Hän luo onnistumisen ja epäonnistumisen kautta tuotteita useista eri tavaroista, joita hän hankkii itse, tai muilta pelaajilta, huutokaupasta tai rakentaa itse. Käsityöläinen voi luoda tavaroita useassa eri tuoteryhmästä, jotka ovat haarniskat, lisävarusteet, rakennusosat, huonekalut, vaateet, työkalut, aseet ja kulkuneuvot. Rakentaessaan tuotteita hänen käyttämänsä piirustus (blueprint) paranee ajan myötä, ja hänen onnistumisen mahdollisuudet kasvavat sekä piirustuksen että hänen omien taitojensa mukaan. Kun pelaaja rakentaa tuotteita hänellä on mahdollisuus saada jokin uusi piirustus. Uuden piirustuksen taso riippuu pelaajan käyttämän piirustuksen tasosta. Pelaaja voi rakentaa haastavampia piirustuksia taitojensa kehittyessä.

Pelistä löytyy muitakin ammatteja, mutta tässä ovat tärkeimmät pelin kulun kannalta. Ammatit tukeutuvat toisiinsa ja tasapainottavat pelin talouden.

3 Järjestelmän käyttötarkoitus

Kuten edellä esitettiin, Entropia Universessä metsästäjän ammatti pohjautuu parhaiden vihollisten löytämiseen ja niiden markkinaosuutta sisältävän tavaran saantiin. Tämä järjestelmä on luotu helpottamaan näiden vihollisten nimeämistä. Järjestelmään on koottu

kaikki Entropia Universen tavarat, viholliset ja niiden välinen yhteys, mikä vihollinen pudottaa mitäkin tavaroita.



Kuva 6. Järjestelmän käyttäjät ja heidän tapansa kommunikoida järjestelmän kanssa.

Kuten kuvassa 6 näkyy, järjestelmällä on kaksi pääsääntöistä käyttäjäryhmää, admin sekä tavallinen käyttäjä. Admin vastaa ylläpidosta ja järjestelmään tiedon tuonnista admin käyttöliittymän kautta. Käyttäjälle esitetään hänen syöttämiensä taitoarvojen mukaan arvio siitä, mitä kaikkia vihollisia hän voi metsästää entropiassa ja arvio kymmenestä parhaasta vihollisesta, joita hän voi metsästää. Kaikkien esitettyjen vihollisten tiedot myös esitellään taulukoissa ja kymmenen parhaan vihollisen taulukossa esitetään myös modaalissa vihollisten pudottamat tavarat. Verkkosovellus esittää myös tietoa vihollisista ja tavaroista REST-rajapinnan kautta käyttäjälle. Lisää käyttötapauksista on oheisessa taulukossa 1.

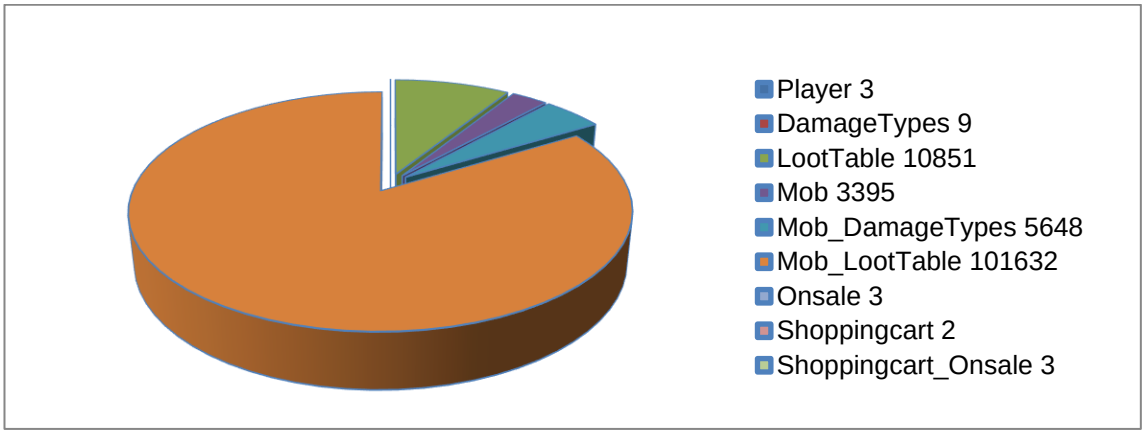
Taulukko 1. Järjestelmän käyttötapaukset

Rooli	**Käyttötapaus**	**Kuvaus**
Admin	Haluaa lisätä tietoa	Tiedon lisääminen tapahtuu admin-sovelluksen importin kautta. Adminille tarjotaan tiedostoselaaja, jossa hän voi valita useamman tai yhden tiedoston ladattavaksi järjestelmään. Tiedosto voi olla joko XML- tai CSV-tyyppistä dataa. Ohjelma tarkistaa datan XML-schema-tiedostoa vasten datan tunnistamisen jälkeen. Jos tiedossa ilmentyy jo tietokannasta löytyvää tietoa, sovellus kysyy käyttäjältä, korvataanko kaksoiskappale, korvataanko kaikki kaksoiskappaleet vai ohitetaanko tämä kaksoiskappale.
Admin	Käyttäjä datan muokkaus ja poisto	Admin käyttöliittymä mahdollistaa käyttäjä datan muokkaamisen Player-välilehden alta. Admin voi syöttää pelaajan etunimen, lempinimen tai sukunimen ja hakea, lisätä tai poistaa kyseisen pelaajan.

Käyttäjä	Haluaa nähdä Mob-dataa	Mob tietoa käyttäjä pääsee näkemään avaamalla verkkosovelluksessa Mob sivun. Siellä käyttäjä voi tarkastella kaikkien tietokannan sisältämien vihollisten tietoja taulukosta.
Käyttäjä	Haluaa nähdä LootTable-dataa	LootTable-tieto eli pelin sisältämien kaikkien tavaroiden tieto löytyy sovelluksen Items-välilehdestä. Tieto esitetään taulukossa käyttäjälle.
Käyttäjä	Pyytää sovellukselta arvion taitojensa pohjalta	Verkkosovelluksen etusivulta löytyy lomake, jonka tiedot täyttämällä käyttäjä voi hakea taitojansa vastaavia vihollisia, jotka ovat arvioitu niiden pudottamien tavaroiden arvon mukaan parhaimmuusjärjestykseen. Sovellus palauttaa 10 parasta vihollista ja esittää ne parhaimmuusjärjestyksessä taulukossa. Kun 10 parhaan vihollisen taulukon riviä painetaan, sovellus esittää vihollisten pudottamat tavarat modaalissa. Sovellus myös esittää kaikki viholliset, joita pelaaja voi metsästää taitojensa perusteella erillisessä taulukossa.
Käyttäjä	Tarkastelee arvion mukaisten vihollisten tavaroita	Käyttäjä syöttää tiedot lomakkeeseen ja lähettää sen. Sovellus näyttää 10 parasta vihollista taulukossa. Kymmenen parhaan arvion saaneen vihollisen pudottamia tavaroita käyttäjä voi tarkastella klikkaamalla taulukon riviä, jolloin sivulle avautuu modaali, jossa taulukossa esitellään kaikki vihollisen pudottamat tavarat.

4 Tietokannan esittely

Tietokanta sisältää 6 taulua ja 3 Hibernaten luomaa välitaulua. Kuvassa 7 näkyvät kaikki tietokantataulut ja niiden sisältämä tiedon määrä. Tärkein osa tietokantaa on sen sisältämät viholliset (Mob), tavarat (LootTable), niiden välinen yhteys (Mob_LootTable-välitaulu), vihollisten tekemät vahinkotyypit (DamageTypes) ja vihollisten vahinkotyyppimäärittely (Mob_DamageTypes-välitaulu). Näillä viidellä taululla voidaan esittää arvioita käyttäjälle, ja ne sisältävät suurimman osan järjestelmän datasta, Mob_LootTable sisältää 101 632 tietuetta, LootTable sisältää 10 851 tietuetta, Mob sisältää 3 395 tietuetta, DamageTypes sisältää 9 tietuetta ja Mob_DamageTypes sisältää 5648 tietuetta. Kuvassa 7 näkyy vain 4 suurinta taulua, koska taulujen sisältämän datan ero on niin suuri.

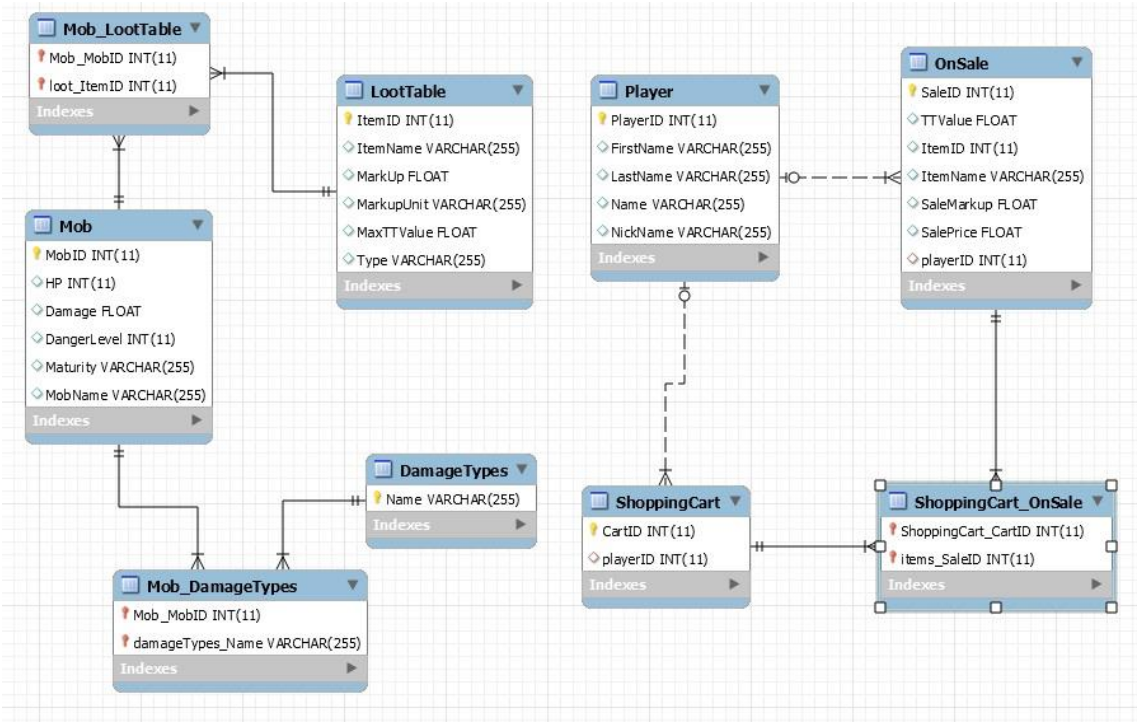


Kuva 7. Tietokannan taulujen datan määrä ilmaistuna riveinä.

Muut taulut tietokannassa on tarkoitettu pelaajien välisen verkkokaupan ominaisuuksia varten, joita ei vielä ole toteutettu verkkosovelluksessa. Kaikkia tauluja voi tarkastella ja hallinnoida admin-käyttöliittymäsovelluksessa.

4.1 Rakenteen kuvaus

Tietokanta koostuu kahdesta erillisestä kokonaisuudesta, jotka on tarkoitus yhdistää tulevaisuudessa. Kuvassa 8 näkyy vasemmalla verkkosovelluksen käyttämä Mob-, LootTable- ja DamageTypes-kokonaisuus ja oikealla näkyy käyttäjä rekisteri sekä verkkokauppakokonaisuus. Suurin osa taulujen välisistä liitoksista on toteutettu ManyToMany-yhteyksillä.



Kuva 8. Tietokannan rakenne.

Tietokantataulut listattuina ja niiden tarkoitus:

- Mob: sisältää tiedon Entropia Universessä esiintyvästä vihollisesta. Sisältää tiedon vihollisen id:stä, elämän määrästä, tuottamasta vahingosta, vaarallisuustasosta, iästä ja vihollisen nimestä.

- LootTable: sisältää tiedon Entropia Univesestä löytyvästä tavarasta. Sisältää tiedon tavaran id:stä, nimestä, markkinalisäarvosta, markkinalisäarvon yksiköstä, tavaran lähtöarvosta sekä tavaran tyypistä.

- DamageTypes: sisältää tiedon Entropia Universen vihollisten vahinko tyypistä. Viholliset voivat aiheuttaa yhdeksää eri tyyppiä, esim. polttavaa, kylmää, myrkyllistä, viiltävää, pistävää tai törmäävää vahinkoa. DamageTypes taulun avaimena toimii vahinko tyypin englanninkielinen nimi.

- Mob_LootTable: ManyToMany-taulu, joka sisältää id:t, joilla voidaan yhdistää eri viholliset heidän pudottamiinsa tavaroihin. Moni eri vihollinen voi pudottaa samoja tavaroita.

- Mob_DamageTypes: ManyToMany taulu, joka sisältää id:t joilla voidaan yhdistää viholliset heidän aiheuttamiinsa vahinkotyyppeihin. Vihollinen voi aiheuttaa montaa eri vahinkotyyppiä.

- Player: sisältää sovellukseen tulevaisuudessa kirjautuvan pelaajan tiedot. Tietoja hyödynnettäisiin myös verkkokaupassa, jossa pelaajat voisivat myydä tavaroita toisilleen. Taulu sisältää pelaajan hahmon nimen kolmessa eri osassa, joita ovat etunimi, lempinimi, sukunimi sekä järjestelmän sisäinen ID.

- OnSale: sisältää tiedon pelaajien myyntiin laittamista tavaroista. Vierasavaimena toimii pelaajan ID. Myynnissä saattaa olla useampi kappale samaa tavaraa kasassa. Taulu sisältää myynti tarjouksen ID:n, myynnissä olevien tavaroiden perus hinnan, tavaroiden ID:n, tavaroiden nimen, myynnin markkinalisäarvon, myynnin kokonais hinnan sekä myyvän pelaajan ID:n.

- ShoppingCart: taulu sisältää ostavan pelaajan ID:n ja hänen ostoskärrynsä ID:n

- ShoppingCart_OnSale: ManyToMany välitaulu, joka sisältää myynnissä olevan tuotteen ID:n sekä ostoskärryn ID:n, johon se on valittu.

4.2 Taulujen yhteydet

Kuten edellä mainittiin, suurin osa taulujen yhteyksistä on toteutettu Hibernaten tarjoamalla ManyToMany-yhteydellä. Esimerkiksi Mob-taulun ja LootTable-taulun yhdistäminen toisiinsa ManyToMany-yhteydellä mahdollistaa, että moni vihollinen voi pudottaa montaa tavaraa. Jokaisella vihollisella tulee olla lista tavaroita, joita ne voivat pudottaa.

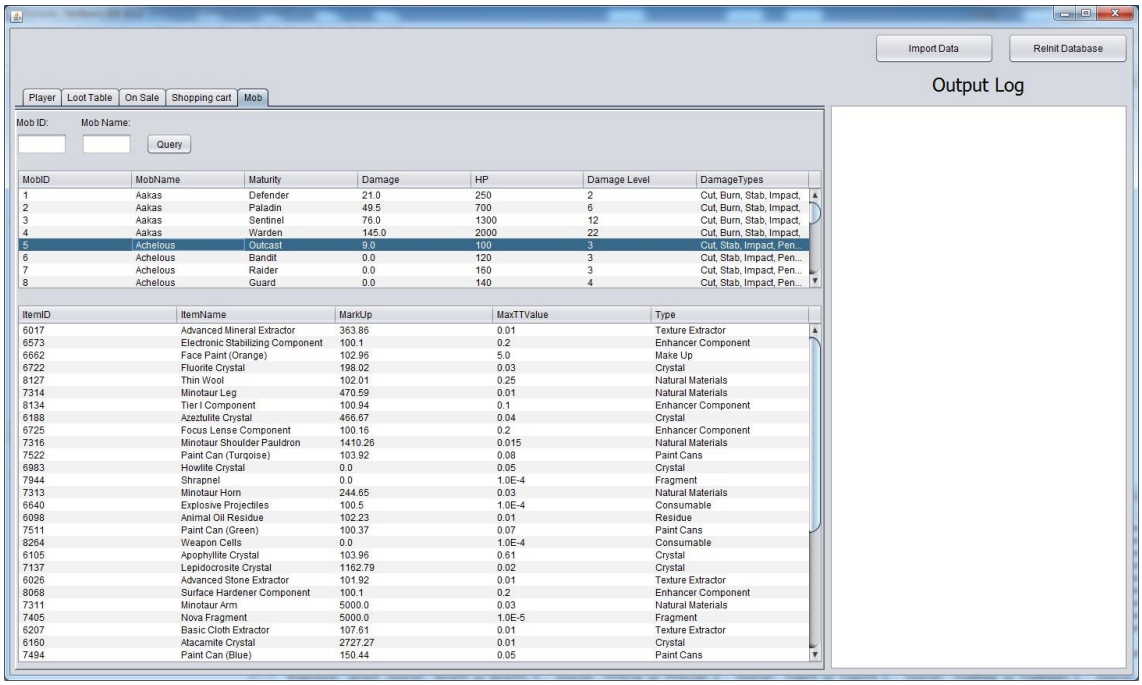
ManyToMany-yhteys myös nopeuttaa sovelluksen toimintaa. Mob- ja DamageTypes-taulujen välillä ei tarvitsisi olla ManyToMany-yhteyttä, koska vahinkotyyppejä pelissä on vain 9 kappaletta, mutta relaatio on toteutettu toimivasti sovelluksissa. Player taulun playerID on vieras avaimena ShoppingCart sekä OnSale tauluissa ja toimii viitteenä ostoskärryn omistajaan tai myyjän id:hen. OnSale-taulussa määritellyt tuotteet voidaan kerätä ID:n mukaan ostoskärryyn ShoppingCartin ID:n mukaan välitauluun ShoppinCart_OnSale.

4.3 Tietokannassa käytetyt teknologiat

Tietokanta perustettiin MySQL-pohjaiseksi, koska se toimii hyvin yhteydessä Hibernaten kanssa ja kirjoittajalla oli siitä eniten kokemusta koulutuksen kautta. MySQL:ää myös suositellaan verkkopohjaisten sovellusten kanssa. MySQL tukee hyvin myös kookkaita tietokantoja ja MMORPG-pelin tietokanta sisältää hyvinkin paljon tietoa.

5 Järjestelmän ylläpitäjän käyttöliittymäsovellus

Käyttöliittymäsovellus helpottaa järjestelmän ylläpitäjän työtä. Sovelluksella voi viedä dataa tietokantaan ja tarkastella kaikkia eri datatyyppejä, sekä muokata osaa datasta. Sovellus toteutettiin Javan Swing-käyttöliittymäkirjastolla, ja se tarjosi kaikki tarvittavat osat sovelluksen luontiin. Sovellus esittää viiden taulun datan välilehdillä, jotka näkyvät kuvan 9 vasemmassa ylälaidassa. DamageTypes-taulukon dataa ei ole tarpeen esittää, sillä se ei ole muuttuvaa ja sisältää vain 9 eri vahinkotyyppiä, joita viholliset voivat aiheuttaa. Sovelluksen oikeassa yläkulmassa on kaksi painiketta: Import Data ja Reinit Database. Import Data on tarkoitettu datan tuontiin tietokantaan joko XML- tai CSV-muodossa ja Reinit Database alustaa tietokannan uudelleen.



Kuva 9. Swingillä luotu ylläpitäjän käyttöliittymäsovellus

5.1 Käytetyt teknologiat

5.1.1 Swing

Swing on täysin Java-pohjainen graafinen käyttöliittymäkirjasto, joka ei tukeudu käyttöjärjestelmän omaan grafiikkaan. Swingin kehitti Netscape vuonna 1996, ja se liitettiin osaksi Javaa versiossa 1.2. Swing alun perin kehitettiin, jotta kehittäjillä olisi vahva Java-pohjainen frontend-kehityskirjasto käytössään, joka olisi toteutettu model-pohjaista ohjelmointia ajatellen. Swing sai suosiota lightweight-komponenttien eli käyttöjärjestelmän grafiikkaan tukeutumattomien elementtien käytössä ja helposta sovelluksen ulkoasun mukauttamisesta ilman, että toteutusta tarvitsisi muuttaa. [1.]

5.1.2 JPA

JPA eli Java Persistence API on ohjelmointirajapinta, joka määrittää tiedon relaatioiden esittämisessä hyödynnettävät annotaatiot. Se määriteltiin vuonna 2006 Java Community Processin toimesta, jotta tietokantojen entityjen määrittely saataisiin standardisoitua. Hibernate käyttää JPA-annotaatioita entityjensä määrittelyyn.

5.1.3 Hibernate

Hibernate on Java-pohjainen annotaatioita käyttävä tietokannan hallintakirjasto. Hibernaten kehittivät vuonna 2001 Gavin King ja hänen kollegansa yrityksessä Cirrus Technologies. Alkuperäinen tavoite oli luoda vaihtoehto silloiselle EJB2-tyyppisille entityille ja tarjota parempi persistence-tuki. Hibernatessa tietokannan taulut kuvataan Java-pohjaisina olioina, Hibernate-entityinä, ja näiden entityjen ilmentymät ovat tietokantataulun rivejä. Taulujen rivien ominaisuudet ja relaatiot kuvataan JPA:n tarjoamilla annotaatioilla. Taulut voidaan myös kuvata XML-tiedostoina ja Hibernate voi luoda XML-tiedostoista alustavan toteutuksen entity-olioista. Hibernate voi myös tarkastaa tietokantaan tulevaa dataa schema-tiedostoa vasten. Kyselyt tietokantaan suoritetaan Hibernaten omalla kyselykielellä HQL:llä eli Hibernate Query Languagella. Se on SQL:n inspiroima kyselykieli, jolla voidaan hakea dataa Hibernaten entityistä. Kyselyn lopputuloksena syntyy helposti hallinnoitava joukko haettuja olioita eli result set.

5.1.4 JaxB

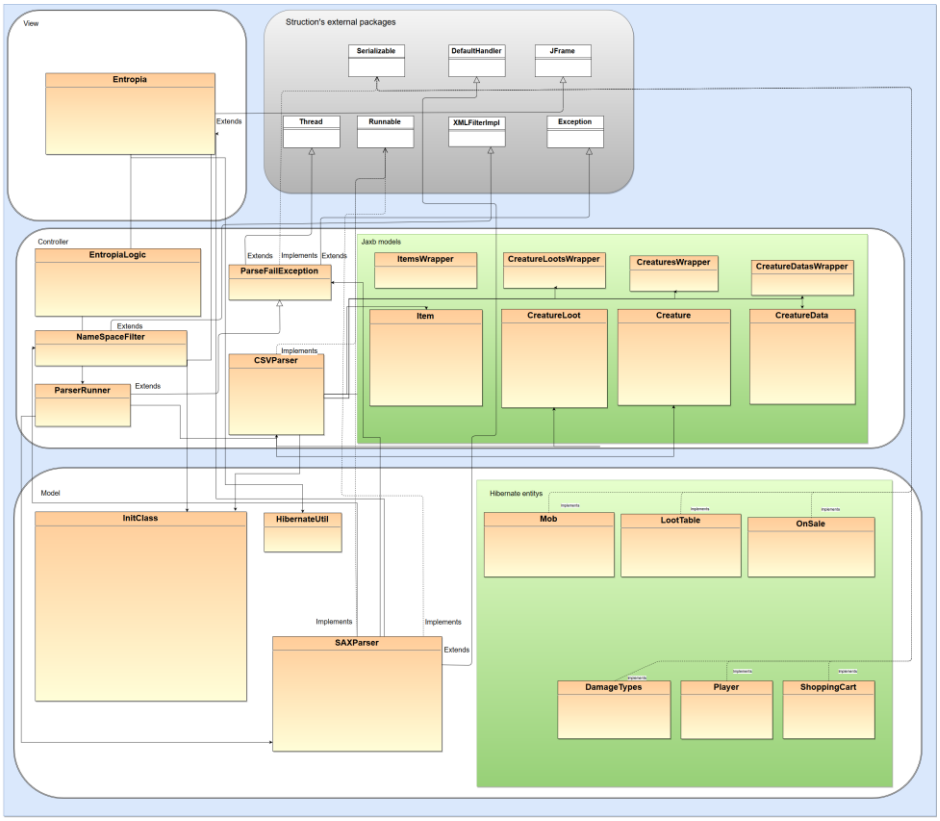
JaxB on Java-pohjainen kirjasto xml-annotoitujen olioiden kääntämiseksi XML:ksi ja XML:n kääntämiseksi olioiksi. XML on yksi tavallisimmista tietomuodoista, joissa tietoa siirretään järjestelmien välillä tai käsitellään sovelluksen sisällä. JaxB mahdollistaa tiedon muuntamisen olioiksi, joita voidaan hyödyntää muissa sovelluksen osissa esim. tässä tapauksessa Hibernate voi hyödyntää käännettyjä objekteja. JaxB:n kehitti Java Community Process vuonna 2006.

5.1.5 SAXParser

SAXParser on XML-jäsennin, joka toteuttaa SAX-rajapinnan. SAX eli Simple Api for XML määrittää SAXParserin tapahtumapohjaisen tietovirran käsittelyn. SAXParserille määritellään jokaista XML-tagiä varten oma toteuttava metodi. Kun SAXParser käy läpi XML-tiedostoa ja kohtaa jonkin seuraavista tapahtumista: tekstisolmun, elementtisolmun, prosessiohjeen tai kommentin, SAXParser kutsuu sen toteuttavaa metodia. SAXParser mahdollistaa XML-tiedon muuntamisen objekteiksi ja XML-tiedoston tarkistamisen schema-tiedostoa vasten.

5.2 Sovelluksen elinkaari

Sovellus koostuu 6 luokasta, 6 Hibernaten tietokanta-entitystä ja 8 JaxB model-luokasta. Sovellusen luokkakaavio näkyy kuvassa 10. JFramen implementoivana pääluokkana toimii Entropia-luokka. Sovelluksen käynnistyessä tämä luokka luo graafisen toteutuksen sovellukselle ja toimii käyttöliittymän toimien hallinnoijana. InitClass toimii tietokannan alustajana, jolla voi tehdä kevyttä testidataa sisältävän tietokannan. HibernateUtil luo ja hallinnoi Hibernaten tietokantayhteyksiä ja käyttää konfigurointiin hibernate.cfg.xml-tiedostoa, joka on liitteessä 1. Sovelluksen logiikasta vastaa EntropiaLogic luokka, joka välittää käyttöliittymän pyyntöjä suorittaviin rakenteisiin. Se myös toteuttaa helpoimman tietokanta haut. Kun sovelluksen kautta ladataan yksi tai useampi XML- tai CSV-tiedosto, niiden tietokantaan viennistä huolehtii ParserRunner. ParserRunner-luokka käynnistää erillisissä säikeissä CSVParsereita ja SaxParsereita riippuen viedyn tiedoston muodosta. CSVParser kääntää CSV-tiedostot XML:ksi JaxB:n modeleita käyttäen ja niiden wrappereitä hyödyntäen, jonka jälkeen muodostetut XML-tiedostot välitetään SAXParserille. SAXParser tarkistaa XML-tiedostot xmlValidation.xsd-shcema-tiedostoa vasten, joka löytyy liitteestä 5 ja vie korrektissa muodossa olevan tiedon tietokantaan. Schemasta löytyy neljälle eri tiedostotyypille korrekti XML-muoto. Eri tiedostoja, joita järjestelmään voidaan viedä ovat MobLevelData, eli vihollisten tiedot, MobDamageTypes, eli vihollisten tekemät vahinkotyypit, Items, eli pelin tavarat, ja CreatureLoots eli tiedostot, jotka määrittelevät vihollisten pudottamat tavarat.



Kuva 10. Järjestelmän ylläpitäjän tietokannan hallintasovelluksen luokkakaavio. Tarkempi luokkakaavio on liitteessä 6.

5.2.1 Sovelluksen käynnistyminen

Sovelluksen käynnistyessä kuvassa 11 näkyvä pääluokka luo uuden ilmentymän JFramen implementoivasta Entropia-luokasta.

```
899     public static void main(String args[]) {
            java.awt.EventQueue.invokeLater(new Runnable() {
901             @Override
                public void run() {
903                 entropiaUI = new Entropia();
904                 entropiaUI.setVisible(true);
905             }
906         });
907     }
```

Kuva 11. JFramen main-metodi, joka luo uuden ilmentymän Entropia-luokasta ja piirtää sen käyttäjälle

Kun Entropia-luokasta tehdään ilmentymä, sen kuvassa 12 esitetty konstruktori luo sovellusikkunan sisältämät komponentit funktiolla initComponents. Konstruktorissa myös

luodaan ilmentymä sovelluksen logiikan toteuttavasta EntropiaLogic-luokasta, jolle välitetään tieto Entropia-luokasta. Konstruktorifunktio luo ilmentymän InitClassista myöhempää käyttöä varten, jos käyttäjä haluaa alustaa tietokannan uudelleen käyttöliittymän kautta. Konstruktori myös ajaa kaikki perus-HQL-kyselyt tietokantaan tiedon esitystä varten sovelluksessa. Lopputuloksena on kuvan 9 mukainen sovellusikkuna, jossa HQL-kyselyiden tulos on esitetty eri välilehtien taulukoissa.

```
70      public Entropia() {
71          entropiaLogic = new EntropiaLogic(this);
72          initComponents();
73          runQueryBasedOnAll();
74          runQueryBasedOnAllInMob();
75          runQueryBasedOnAllInLootTable();
76          runQueryBasedOnAllInOnSale();
77      }
```

Kuva 12. Entropia-luokan konstruktorifunktio.

5.2.2 Tietokantakyselyiden luonti

Sovelluksen käynnistyessä ja käyttäjän päivittäessä taulukot sovellus hakee tietoa tietokannasta HQL-kyselyillä. JFrame seuraa nappien komentoja actionListerer-funktioilla, jollainen on esitetty kuvassa 13.

```
848     private void MobQueryButtonActionPerformed(java.awt.event.ActionEvent evt) {
849         if (!MobName.getText().trim().equals("")) {
850             runQueryBasedOnMobName();
851         } else if (!MobID.getText().trim().equals("")) {
852             runQueryBasedOnMobID();
853         } else {
854             runQueryBasedOnAllInMob();
855         }
856     }
```

Kuva 13. MobQueryButtonActionPerformed ajaa kyselyn syöttölomakkeen tietojen mukaan.

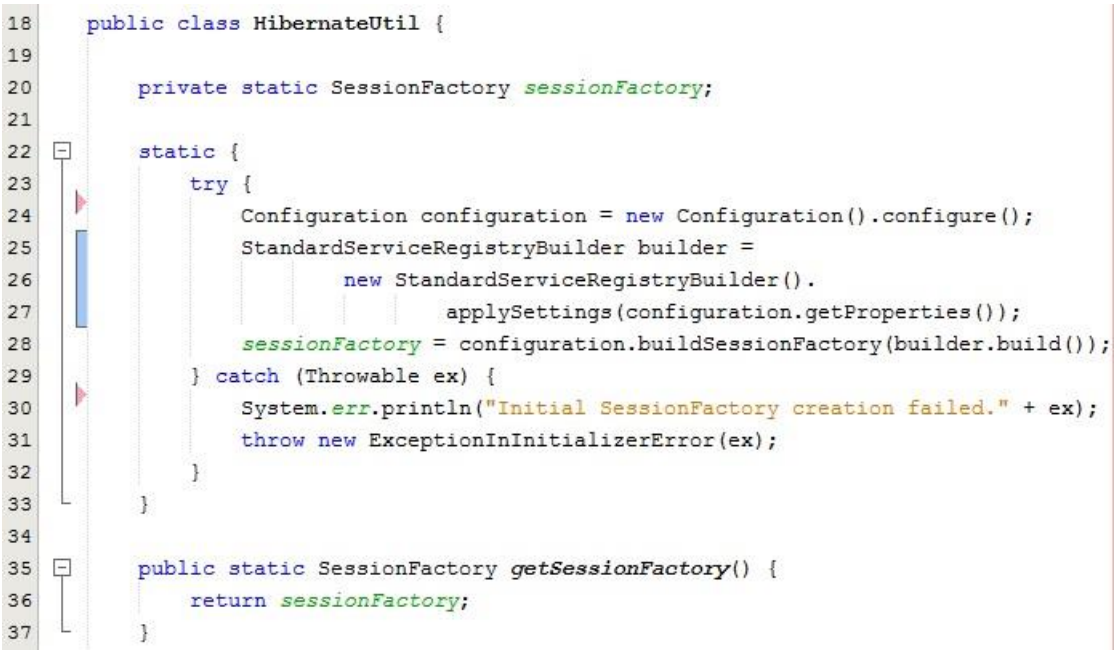
ActionListenerit kutsuvat apumetodeja, jotka hakevat käyttöliittymästä tarpeellisten kenttien arvot ja luovat kyselyn käyttäjän syötteistä ja valmiista kyselypohjista. Kysely välitetään EntropiaLogic-luokan metodille executeStringHQLQuery. ExecuteStringHQLQuery hakee session Hibernatea hallinnoivalta HibernateUtililta ja tekee tietokanta transaktion

aloituksen, kyselyn ajamisen ja datan välittämisen takaisin käyttöliittymään, jossa displayResult-funktio esittää datan taulukoissa. Kyselyä ei ajeta ennen kuin transaktio on onnistunut.

```
47      public List executeStringHQLQuery(String hql) {
48          List resultList = new ArrayList<>();
49          try {
50              Session session = HibernateUtil.getSessionFactory().openSession();
51              session.beginTransaction();
52              Query q = session.createQuery(hql);
53              resultList = q.list();
54              session.getTransaction().commit();
55          } catch (HibernateException he) {
                he.printStackTrace();
57          }
58          return resultList;
59      }
```

Kuva 14. HQL-lauseet suorittava executeStringHQLQuery-funktio, joka palauttaa tuloksen Entropia-käyttöliittymäluokalle

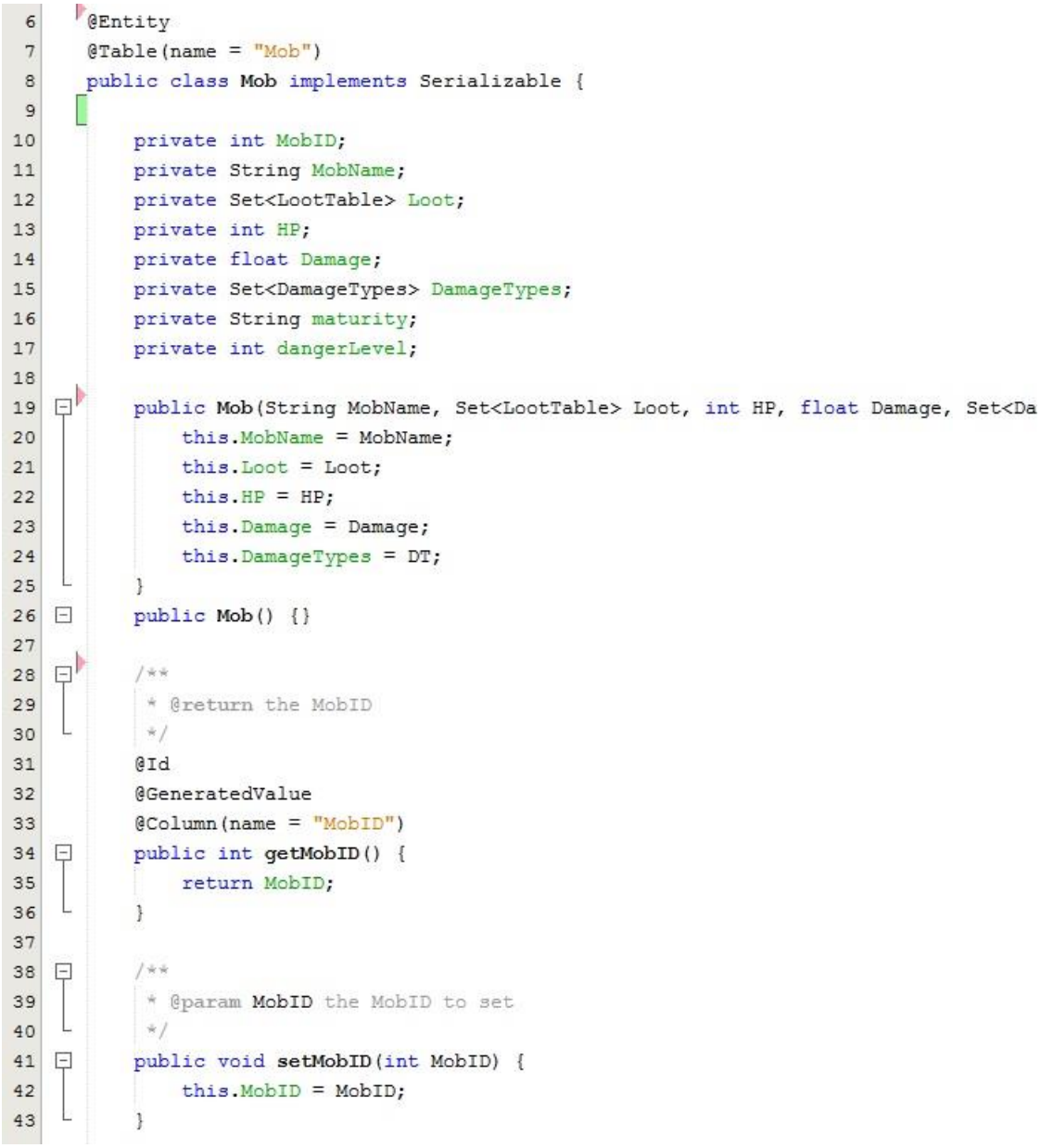
executeStringHQLQuery hakee Hibernaten session singletonina toteutetulta HibernateUtililta. HibernateUtil luo sessionFactoryn, kun HibernateUtil luodaan ensimmäisen kerran. HibernateUtil sisältää myös kaksi apufunktiota, joilla voidaan muuttaa session hibernate.hbm2ddl.auto-asetusta, joka määrittää, päivitetäänkö jo olemassa olevaa tietokantaa vai alustetaanko tietokanta uudestaan uusilla tauluilla.



```
18  public class HibernateUtil {
19
20      private static SessionFactory sessionFactory;
21
22      static {
23          try {
24              Configuration configuration = new Configuration().configure();
25              StandardServiceRegistryBuilder builder =
26                      new StandardServiceRegistryBuilder().
27                              applySettings(configuration.getProperties());
28              sessionFactory = configuration.buildSessionFactory(builder.build());
29          } catch (Throwable ex) {
30              System.err.println("Initial SessionFactory creation failed." + ex);
31              throw new ExceptionInInitializerError(ex);
32          }
33      }
34
35      public static SessionFactory getSessionFactory() {
36          return sessionFactory;
37      }
```

Kuva 15. HibernateUtil luo hibernate.cfg.xml-tiedoston mukaisen sessionFactory-ilmentymän Hinernatesta

SessionFactory saa asetuksensa joko Hibernaten määrittelemältä Configurations-luokalta, tai kuten tässä tapauksessa, hibernate.cfg.xml-tiedostolta, joka on liitteessä 1. hibernate.cfg.xml-tiedostossa esitetään tietokantayhteyden luova driver, yhteystyyppi, yhteyden url-osoite ja yhteyteen tarvittava käyttäjätunnus sekä salasana. Liitteen tiedostossa on myös esitelty lokaalin testiympäristön tietokantayhteys, joka normaalisti kommentoitaisiin pois, mutta on tässä näkyvissä tekstin selvyyden takia. Tämän lisäksi Hibernatelle voi määritellä vapaaehtoisia asetuksia, kuten tässä on määritelty hibernate.hbm2ddl.auto-asetus, jolla määritetään, luodaanko tietokanta aina uudelleen tai päivitetäänkö uudella tiedolla, ja hibernate.show_sql-asetus, joka määrää, näytetäänkö SQL-lauseita sovelluksen lokissa. Mapping-osassa Hibernatelle määritellään, mitkä luokat ovat tietokannan entity-luokkia. Entity-luokat ovat kuvan 16 esimerkin mukaisia tavallisia olioita, joihin on Javan JPA:n määrittelemillä annotaatioilla tehty merkintöjä Hibernatelle.



```
@Entity
@Table(name = "Mob")
public class Mob implements Serializable {

    private int MobID;
    private String MobName;
    private Set<LootTable> Loot;
    private int HP;
    private float Damage;
    private Set<DamageTypes> DamageTypes;
    private String maturity;
    private int dangerLevel;

    public Mob(String MobName, Set<LootTable> Loot, int HP, float Damage, Set<Da
        this.MobName = MobName;
        this.Loot = Loot;
        this.HP = HP;
        this.Damage = Damage;
        this.DamageTypes = DT;
    }
    public Mob() {}

    /**
     * @return the MobID
     */
    @Id
    @GeneratedValue
    @Column(name = "MobID")
    public int getMobID() {
        return MobID;
    }

    /**
     * @param MobID the MobID to set
     */
    public void setMobID(int MobID) {
        this.MobID = MobID;
    }
```

Kuva 16. Hibernaten käyttämä Mob-entity. Hibernate tunnistaa taulun ominaisuudet annotaatioista.

Entity-luokat määrittelevät Hibernatelle luotavat tietokantataulut ja luokissa olevat annotaatiot määrittävät tietokantataulun attribuutit eli sarakkeet. Kuvassa 16 @Entity määrittää luokan entityksi, ja @id määrää tietokantataulun pääavaimeksi MobID-muuttujan. @GeneratedValue määrittää MobID:n automaattisesti luoduksi juoksevaksi luvuksi ja @Column määrittää tietokantataulun attribuutin nimeksi MobID. Annotaatiot voivat määrittää myös tietokanta taulujen yhteyksiä esimerkiksi kuvassa 17 on kuvattu monen suhde moneen Mob-taulun ja LootTable-taulun välillä. Hibernatessa monen suhde moneen luo välitaulun, jossa pääavainten avulla yhdistetään kyseiset taulujen rivit toisiinsa.

ManyToMany mahdollistaa tietokannassa esimerkiksi vihollisten pudottamien kaikkien tavaroiden listauksen.

```
63      @ManyToMany()
64      @Column(name = "Loot")
65      public Set<LootTable> getLoot() {
66          return Loot;
67      }
```

Kuva 17. Mob entityssä määritelty joukko (set) LootTable-olioita. Yhteyksenä on monen suhde moneen.

5.2.3 Tiedon esitys

Kun EntropiaLogic palauttaa resultSetin kyselyn suorittamisen jälkeen, sen ottaa vastaan displayResult-funktio Entropia-käyttöliittymässä, joka näkyy kuvassa 18. Funktio tunnistaa listassa ensimmäisenä olevan objektin luokan ja hakee sitä vastaavan taulukon käyttöliittymästä. Se luo uuden DefaultTableModelin, jolle se tekee otsikkotiedot ja muuttaa objektit taulukkoon meneviksi riveiksi.

```
187     private void displayResult(List resultList) {
            Vector<String> tableHeaders = new Vector<>();
            Vector tableData = new Vector();
190         if (!resultList.isEmpty()) {
191             if (resultList.get(0).getClass().equals(Player.class)) {
192                 tableHeaders.add("PlayerId");
193                 tableHeaders.add("FirstName");
194                 tableHeaders.add("NickName");
195                 tableHeaders.add("LastName");
196
                    for (Object o : resultList) {
198                     Player player = (Player) o;
                        Vector<Object> oneRow = new Vector<>();
200                     oneRow.add(player.getPlayerID());
201                     oneRow.add(player.getFirstName());
202                     oneRow.add(player.getNickName());
203                     oneRow.add(player.getLastName());
204                     tableData.add(oneRow);
205                 }
206                 resultTable.setModel(new DefaultTableModel(tableData, tableHeaders));
207             } else if (resultList.get(0).getClass().equals(Mob.class)) {
208                 tableHeaders.add("MobID");
209                 tableHeaders.add("MobName");
210                 tableHeaders.add("Maturity");
211                 tableHeaders.add("Damage");
212                 tableHeaders.add("HP");
213                 tableHeaders.add("Damage Level");
214                 tableHeaders.add("DamageTypes");
215
                    for (Object o : resultList) {
217                     Mob mob = (Mob) o;
```

Kuva 18. Hibernate-kyselyn tuloksen esittämän displayResult-funktion osa.

Tämän lisäksi on erikoistapaus, kun käyttöliittymässä painetaan vihollistaulun riviä, jolloin suoritetaan findMobLootTable-funktio. Kuvan 19 funktio hakee painetun rivin vihollisen ID:n ja suorittaa tällä haun executeMobHQLQuery, joka palauttaa vihollisobjektin ja siihen ManyToMany-annotaatiolla liitetyt tavarat. Tämä tieto palautetaan findMobLootTable-funktiolle, ja se asettaa tiedot vihollisen tavaroista käyttöliittymässä olevaan taulukkoon, joka on vihollistaulun alapuolella. Vihollisen pudottamat tavarat saadaan getLoot-funktiolla

```
80      private void findMobLootTable(int mobID) {
81          List resultList;
82          resultList = EntropiaLogic.executeMobHQLQuery(mobID);
83          if (resultList != null) {
                Vector<String> tableHeaders = new Vector<>();
                Vector tableData = new Vector();
86              tableHeaders.add("ItemID");
87              tableHeaders.add("ItemName");
88              tableHeaders.add("MarkUp");
89              tableHeaders.add("MaxTTValue");
90              tableHeaders.add("Type");
                for (Object o : resultList) {
92                  Mob mob = (Mob) o;
93                  Iterator iter = mob.getLoot().iterator();
94                  while (iter.hasNext()) {
                        Vector<Object> oneRow = new Vector<>();
96                      LootTable lootTable = (LootTable) iter.next();
97                      oneRow.add(lootTable.getItemID());
98                      oneRow.add(lootTable.getItemName());
99                      oneRow.add(lootTable.getMarkup());
100                     oneRow.add(lootTable.getMaxTTValue());
101                     oneRow.add(lootTable.getType());
102                     tableData.add(oneRow);
103                 }
104             }
105             MobLootResultTable.setModel(
106                     new DefaultTableModel(tableData, tableHeaders));
107         }
108     }
```

Kuva 19. Vihollisten tavarat taulukossa esittävä findMobLootTable-funktio

ExecuteMobHQLQuery tekee kyselyn, joka hakee tiettyä ID:tä vastaavan vihollisen tietokannasta. Kuvassa 20 näkyy, kuinka kysely voidaan muodostaa sessioon HQL-lauseella, johon saa asetettua kaksoispistettä käyttäen parametrejä. Tässä tapauksessa vihollisen ID asetetaan mobID-muuttujaan.

```
26  public static List executeMobHQLQuery(int param) {
27      List resultList = null;
28      try {
29          Session session = HibernateUtil.getSessionFactory().openSession();
30          session.beginTransaction();
31          Query q = session.createQuery("from Mob a where a.mobID = :mobID");
32          q.setParameter("mobID", param);
33          resultList = q.list();
34          session.getTransaction().commit();
35      } catch (HibernateException he) {
            he.printStackTrace();
37      }
38      return resultList;
39  }
```

Kuva 20. executeMobHQLQuery hakee yhden vihollisen kaikki tiedot

5.2.4 Tietokannan alustaminen uudelleen

Tietokannan uudelleen alustamisen voi käynnistää sovelluksen oikeassa yläkulmassa olevalla Reinit Database -painikkeella. Komento välitetään EntropiaLogic-kontrollerille, joka vaihtaa HibernateUtilia kutsuen Hibernaten hibernate.hbm2ddl.auto-asetuksen updatesta createen, jolloin tietokantataulut luodaan uudelleen. InitClass tämän jälkeen luo uudet esiintymät tietokantatauluista, jolloin ne pyyhkivät vanhat esiintymät pois. InitClass myös lisää tauluihin testausta varten testidataa.

5.2.5 Tiedon vienti kantaan

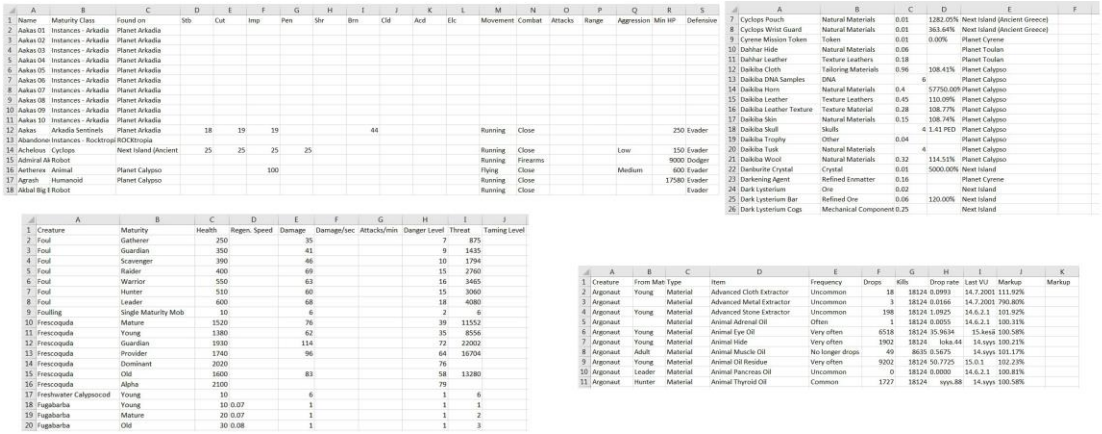
Tiedon vienti kantaan tapahtuu Import Data -painikkeella sovelluksen oikeassa yläkulmassa. Kun painiketta painetaan, käyttöliittymä avaa file explorerin adminin käyttöön. Hän voi valita XML- ja CSV-tiedostoja vietäväksi tietokantaan. Jokainen tiedosto tutkitaan erikseen ja tiedostopäätteen mukaan sovellus käynnistää joko CSVParserin tai SAXParserin tiedoston lukemista varten. Parsereiden käynnistämisestä vastaa kuvan 21 ParserRunner, jonka EntropiaLogic käynnistää. ParserRunner käynnistää kaikkia tiedostoja varten oman parserin, joista jokainen on oma säikeensä, jotta käyttöliittymän toiminta ei pysähdy ja jää odottamaan parsereita. Jokaiselle parserille välitetään tieto Entropia-käyttöliittymästä, jotta ne voivat välittää käyttöliittymän lokiin väliaikatietoja toiminnastaan.

```
25      @Override
        public void run() {
27          Thread thread;
28          String fileEnding;
29          for (File f : files) {
30              fileEnding = f.getName().substring(f.getName().indexOf('.'));
31              switch (fileEnding.toLowerCase()) {
32                  case ".csv":
33                      try {
34                          CSVParser p = new CSVParser(entropiaUI, f);
35                          thread = new Thread(p);
36                          thread.start();
37                          thread.join();
38                      } catch (InterruptedException ex) {
39                          Logger.getLogger(ParserRunner.class.getName()).log(Level.SEVERE, null, ex);
40                      }
41                      break;
42                  case ".xml":
43                      try {
44                          SaxParser sp = new SaxParser(entropiaUI, f.getPath());
45                          sp.init();
46                          thread = new Thread(sp);
47                          thread.start();
48                          thread.join();
49                          break;
                        } catch (Exception ex) {
51                          Logger.getLogger(ParserRunner.class.getName()).log(Level.SEVERE, null, ex);
52                          System.exit(2);
53                      }
54                  default:
55                      entropiaUI.writeToOutputLog("File Ending not regocnized!");
56                      break;
57              }
58          }
59          entropiaUI.writeToOutputLog("Finished importing files");
60      }
```

Kuva 21. ParserRunnerin run-metodi ajetaan, kun se käynnistetään. Funktio käynnistää säikeitä toteuttavia parsereita valittujen tiedostojen tiedostopäätteiden mukaan.

5.2.6 CSV-tiedon vienti tietokantaan

Kun ParserRunner huomaa tiedoston olevan CSV-tiedosto, se käynnistää CSVParserin ja välittää tiedoston sille. CSVParser luo uuden CSVReaderin, jolla se alkaa lukea tiedostoa rivi kerrallaan. Ensimmäisen rivin eli otsikoiden mukaan CSVParser tunnistaa, mitä neljää eri tyyppiä tiedosto on. Tiedosto voi olla MobLevelDataa, joka sisältää perustiedot vihollisesta, MobDamageTypes-tietoa, joka sisältää vihollisten tekemät vahinkotyypit, Item tietoa, joka sisältää kaikkien pelin tavaroiden tiedot, tai CreatureLoots, joka yhdistää viholliset niiden pudottamiin tavaroihin.



Name	Maturity Class	Found on	Stb	Cut	Imp	Pen	Sbr	Brn	Cld	Acd	Elc	Movement	Combat	Attacks	Range	Aggression	Min HP	Defensive
Aakas 01	Instances - Arkadia	Planet Arkadia																
Aakas 02	Instances - Arkadia	Planet Arkadia																
Aakas 03	Instances - Arkadia	Planet Arkadia																
Aakas 04	Instances - Arkadia	Planet Arkadia																
Aakas 05	Instances - Arkadia	Planet Arkadia																
Aakas 06	Instances - Arkadia	Planet Arkadia																
Aakas 07	Instances - Arkadia	Planet Arkadia																
Aakas 08	Instances - Arkadia	Planet Arkadia																
Aakas 09	Instances - Arkadia	Planet Arkadia																
Aakas 10	Instances - Arkadia	Planet Arkadia																
Aakas	Arkadia Sentinels	Planet Arkadia	18	19	19			44				Running	Close				250	Evader
Abandone	Instances - Rocktropi	ROCKtropia																
Achelous	Cyclops	Next Island (Ancient	25	25	25	25						Running	Close			Low	150	Evader
Admiral Ak	Robot											Running	Firearms				9000	Dodger
Aetherea	Animal	Planet Calypso			100							Flying	Close			Medium	600	Evader
Agrash	Humanoid	Planet Calypso										Running	Close				17580	Evader
Akbal Big E	Robot											Running	Close					Evader

Cyclops Pouch	Natural Materials	0.01	1282.05%	Next Island (Ancient Greece)
Cyclops Wrist Guard	Natural Materials	0.01	363.64%	Next Island (Ancient Greece)
Cyrene Mission Token	Token	0.01	0.00%	Planet Cyrene
Dahlar Hide	Natural Materials	0.06		Planet Toulan
Dahlar Leather	Texture Leathers	0.18		Planet Toulan
Daikiba Cloth	Tailoring Materials	0.96	108.41%	Planet Calypso
Daikiba DNA Samples	DNA	6		Planet Calypso
Daikiba Horn	Natural Materials	0.4	57750.00%	Planet Calypso
Daikiba Leather	Texture Leathers	0.45	110.09%	Planet Calypso
Daikiba Leather Texture	Texture Material	0.28	108.77%	Planet Calypso
Daikiba Skin	Natural Materials	0.15	108.74%	Planet Calypso
Daikiba Skull	Skulls		4: 1.41 PED	Planet Calypso
Daikiba Trophy	Other	0.04		Planet Calypso
Daikiba Tusk	Natural Materials		4	Planet Calypso
Daikiba Wool	Natural Materials	0.32	114.51%	Planet Calypso
Danburite Crystal	Crystal	0.01	5000.00%	Next Island
Darkening Agent	Refined Enmatter	0.16		Planet Cyrene
Dark Lysterium	Ore	0.02		Next Island
Dark Lysterium Bar	Refined Ore	0.06	120.00%	Next Island
Dark Lysterium Cogs	Mechanical Component	0.25		Next Island

Creature	Maturity	Health	Regen. Speed	Damage	Damage/sec	Attacks/min	Danger Level	Threat	Taming Level
Foul	Gatherer	250		35			7	875	
Foul	Guardian	350		41			9	1435	
Foul	Scavenger	390		46			10	1794	
Foul	Raider	400		69			15	2760	
Foul	Warrior	550		63			16	3465	
Foul	Hunter	510		60			15	3060	
Foul	Leader	600		68			18	4080	
Foulling	Single Maturity Mob	10		6			2	6	
Frescoquda	Mature	1520		76			39	11552	
Frescoquda	Young	1380		62			35	8556	
Frescoquda	Guardian	1930		114			72	22002	
Frescoquda	Provider	1740		96			64	16704	
Frescoquda	Dominant	2020					76		
Frescoquda	Old	1600		83			58	13280	
Frescoquda	Alpha	2100					79		
Freshwater Calypsocod	Young	10		6			1	6	
Fugabarba	Young	10	0.07	1			1	1	
Fugabarba	Mature	20	0.07	1			1	2	
Fugabarba	Old	30	0.08	1			1	3	

Creature	From Mat	Type	Item	Frequency	Drops	Kills	Drop rate	Last VU	Markup	Markup
Argonaut	Young	Material	Advanced Cloth Extractor	Uncommon	18	18124	0.0993	14.7.2001	111.92%	
Argonaut		Material	Advanced Metal Extractor	Uncommon	3	18124	0.0166	14.7.2001	790.80%	
Argonaut	Young	Material	Advanced Stone Extractor	Uncommon	198	18124	1.0925	14.6.2.1	101.92%	
Argonaut		Material	Animal Adrenal Oil	Often	1	18124	0.0055	14.6.2.1	100.31%	
Argonaut	Young	Material	Animal Eye Oil	Very often	6518	18124	35.9634	15.kesä	100.58%	
Argonaut	Young	Material	Animal Hide	Very often	1902	18124	kika.44	14.syys	100.21%	
Argonaut	Adult	Material	Animal Muscle Oil	No longer drops	49	8635	0.5675	14.syys	101.17%	
Argonaut	Young	Material	Animal Oil Residue	Very often	9202	18124	50.7725	15.0.1	102.23%	
Argonaut	Leader	Material	Animal Pancreas Oil	Uncommon	0	18124	0.0000	14.6.2.1	100.81%	
Argonaut	Hunter	Material	Animal Thyroid Oil	Common	1727	18124	wys.88	14.syys	100.58%	

Kuva 22. 4 eri CSV-tietotyyppiä ylhäältä vasemmalta lähtien lueteltuna: MobDamageTypes, Items, MobLevelData ja CreatureLoots

Jokaiselle eri tietotyypille on oma toteuttava funktio, joka muuntaa CSV-tiedon JaxB-olioiksi, jotka ovat annotoitu XML:ksi muuntamista varten. Kuvassa 23 näkyy Creature-luokka, johon voidaan kääntää MobLevelData sekä MobDamageTypes-tieto. MobDamageTypesejä varten määritellään Wrapper-annotaatiolla, koska DamageType-tietoa voi olla usea kappale. DamageTypet siis laitetaan DamageTypes-wrapperin sisälle. Samanlaiset wrapperit on määritelty kaikille tiedostoille, koska ne listamaisesti sisältävät useita esiintymiä samasta tiedosta.

```
20  @XmlRootElement(name = "Creatures")
21  @XmlAccessorType(XmlAccessType.FIELD)
22  @XmlType(propOrder = {"mobName", "HP", "damage", "damageTypes"})
23  public class Creature {
24
25      private String mobName;
26      private int HP;
27      private float damage;
28      @XmlElementWrapper(name="damageTypes")
29      @XmlElement(name="damageType")
30      private Set<String> damageTypes;
31
32      /**
33       * @return the MobName
34       */
35      public String getMobName() {
36          return mobName;
37      }
38
39      /**
40       * @param mobName the MobName to set
41       */
42      public void setMobName(String mobName) {
43          this.mobName = mobName;
44      }
45
46      /**
47       * @return the HP
48       */
49      public int getHP() {
50          return HP;
51      }
```

Kuva 23. Creature-luokka, jota käytetään CSV:n kääntämiseksi XML:ksi. JaxB-annotaatiot määräävät, kuinka luokka käännetään.

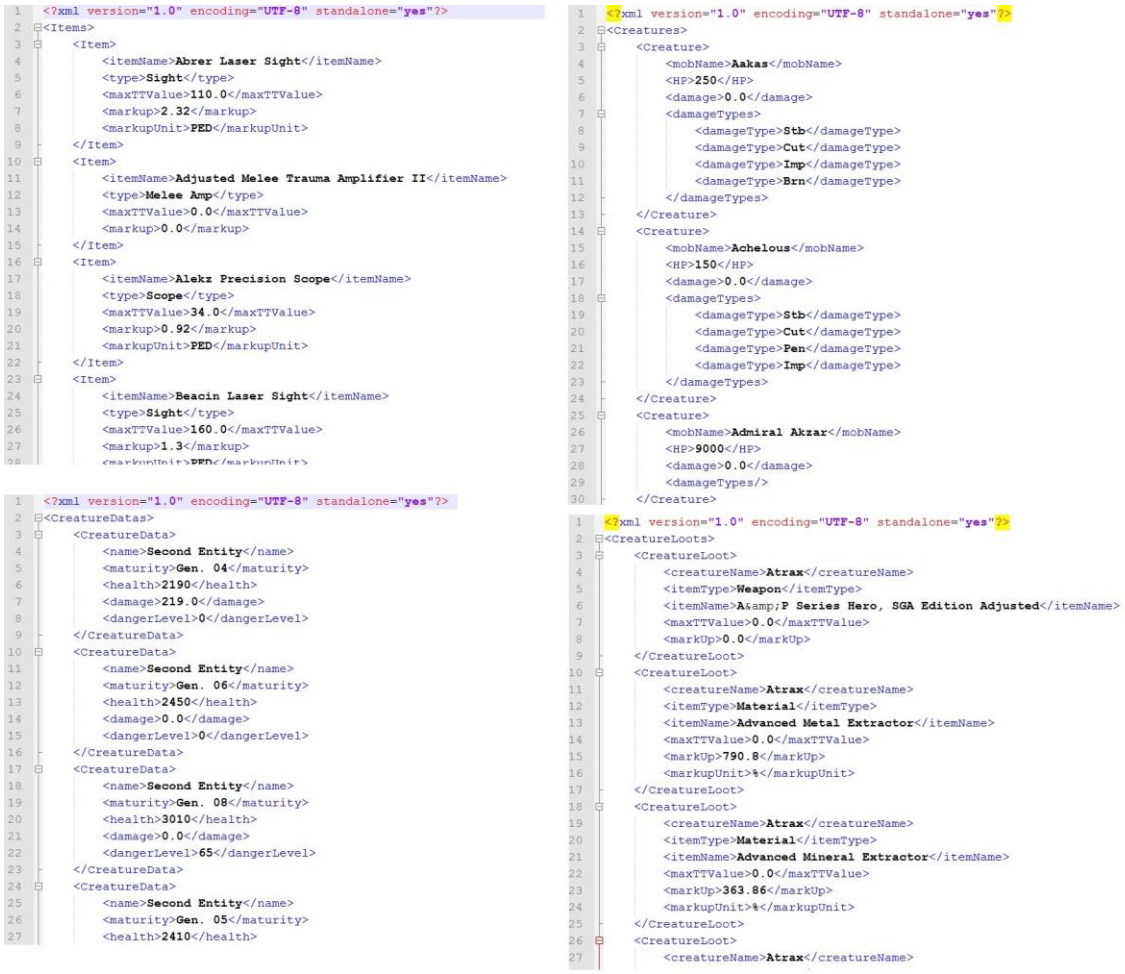
Kun CSV-tieto on luettu JaxB-luokiksi, oliot käännetään kuvan 24 funktiolla jaxbObjectToXML. Funktio valitsee oikean wrapper-luokan aikaisemmin tunnistetun tiedostomuodon mukaan ja asettaa objektilistan tähän wrapperiin. Tämän jälkeen JaxB:n Marshallerista luodaan uusi ilmentymä ja sille asetetaan, missä formaatissa tieto halutaan viedä tiedostoon, määritellään käsiteltävä tieto ja tiedosto, johon Marshaller tulostaa lopputuloksen. Marshaller on luokka, joka vastaa serialisoitavien Java objectien kääntämisestä XML-muotoon. Lopputuloksena on XML-tiedosto, joka sisältää kaiken halutun tiedon alkuperäisestä CSV-tiedostosta. Kun XML-tiedosto on muodostettu, välitetään se SAX-Parserille, jonka toimintaperiaate selostetaan seuraavassa luvussa.

```
296     private void jaxbObjectToXML(ArrayList objects) {
297         try {
            JAXBContext context = null;
            Marshaller m = null;
300         switch (CSVdataType) {
301             case "creature":
302                 context = JAXBContext.newInstance(CreaturesWrapper.class);
303                 CreaturesWrapper cw = new CreaturesWrapper();
304                 cw.setCreatures(objects);
305                 m = context.createMarshaller();
306                 m.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT, true);
307                 m.marshal(cw, new File("./convertedXML/" +
308                         fileName.substring(0, fileName.indexOf(".csv")) + ".xml"));
309                 break;
310             case "item":
311                 context = JAXBContext.newInstance(ItemsWrapper.class);
312                 ItemsWrapper iw = new ItemsWrapper();
313                 iw.setItems(objects);
314                 m = context.createMarshaller();
315                 m.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT, true);
316
317                 m.marshal(iw, new File("./convertedXML/" +
318                         fileName.substring(0, fileName.indexOf(".csv")) + ".xml"));
319                 break;
320             case "creatureLoot":
321                 context = JAXBContext.newInstance(CreatureLootsWrapper.class);
322                 CreatureLootsWrapper alw = new CreatureLootsWrapper();
323                 alw.setCreatureLoots(objects);
324                 m = context.createMarshaller();
```

Kuva 24. Funktio jaxbObjectToXML kääntää listassa olevat objectit XML-tiedostoksi Marshalleria käyttäen

5.2.7 XML-tiedon vienti tietokantaan

Kuvassa 25 on esitetty kaikki eri XML-tiedostotyypit, joita järjestelmään voidaan ladata. Tietotyyppejä ovat Items, joka kuvaa kaikki järjestelmään vietävät tavarat; Creatures, joka kuvaa pelin vihollisten perustiedot ja vahinkotyypit; CreatureData tieto, joka kuvaa tarkempaa tietoa vihollisista, ja CreatureLoot, joka kuvaa kaikki vihollisten pudottamat tavarat.



Kuva 25. 4 eri XML-tietotyyppiä ylhäältä vasemmalta lähtien lueteltuna, joita ovat Items, Creatures, CreatureDatas ja CreatureLoots

ParserRunnerin havaitessa XML-tiedoston se luo uuden SaxParserin ja initialisoi sen. Kuvan 26 mukainen Init-funktio luo uuden SchemaFactoryn, jolle annetaan schema-tiedosto XML:n tarkistusta varten. Schema-tiedosto on liitteessä 5. Kun tiedosto on tarkistettu schemaa vasten, aloitetaan parserin toiminta kutsumalla funktiota SAXParserFactory.newInstance().newSAXParser().parse(). SaxParseri on tapahtumapohjainen eli, kun XML-tiedosto alkaa tai loppuu, ja aina kun tiedostossa alkaa tai loppuu XML-elementti, niin tapahtuu funktiokutsu. Näissä funktioissa kehittäjä voi määritellä, mitä haluaa tapahtuvan kunkin tapahtuman aikana.

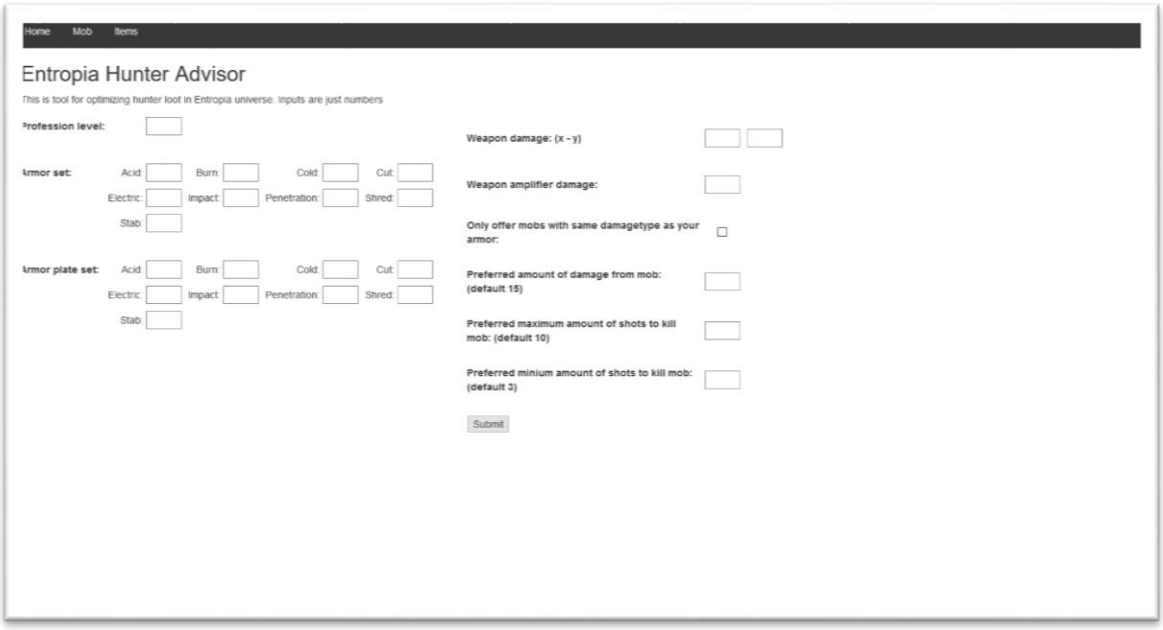
```
69  public void init() throws org.xml.sax.SAXException, IOException,
70          ParserConfigurationException, ParseFailException {
71      System.setProperty("javax.xml.parsers.SAXParserFactory",
72              "org.apache.xerces.jaxp.SAXParserFactoryImpl");
73      try {
74          // create a SchemaFactory capable of understanding WXS schemas
75          SchemaFactory factory = SchemaFactory.newInstance(
76                  XMLConstants.W3C_XML_SCHEMA_NS_URI);
77          // load a WXS schema, represented by a Schema instance
78          Source schemaFile = new StreamSource(
79                  new File("./src/entropia/util/xmlValidation.xsd"));
80          Schema schema = factory.newSchema(schemaFile);
81          Validator validator = schema.newValidator();
82          SAXSource source = new SAXSource(
83                  new NameSpaceFilter(XMLReaderFactory.createXMLReader()),
84                  new InputSource(new FileInputStream(path)));
85          // validate the DOM tree
86          try {
87              validator.validate(source, null);
88              parserFactory = SAXParserFactory.newInstance();
89              parser = parserFactory.newSAXParser();
90          } catch (SAXException e) {
91              System.err.print(e);
92              throw new ParseFailException("parseri ei initialisoitunut", e);
93          }
94      } catch (FactoryConfigurationError exception) {
            exception.printStackTrace();
96      }
97  }
```

Kuva 26. SaxParserin Init-funktio, joka tarkistaa tiedoston schemaa vasten

Funktioissa luodaan uusia Hibernate-entity luokkia, joihin elementtien sisältämä tieto siirretään. Kun entity saadaan valmiiksi, eli kun päästään tietyn elementin loppuun, entity tallennetaan tietokantaan. Tämä prosessi voidaan suorittaa kaikille neljälle eri tietotyypille.

6 Verkkosovellus

Sovellus luotiin näyttämään käyttäjälle tietokannan dataa ja luomaan arvioita käyttäjän hahmon taitojen pohjalta. Sovellus toteutettiin Springillä ja siihen luotiin REST-rajapinta Jerseytä ja Jacksonia käyttämällä, jotta tietokannan tietoja voidaan esittää sovelluksessa. REST-rajapinnan tietoja esitetään Mob- ja Items-välilehdillä, jotka ovat sivun ylälaidan header-osiossa. Käyttäjä voi syöttää järjestelmään hahmonsa tiedot kuvan 27 mukaisen etusivun lomakkeen avulla.



Kuva 27. Verkkosovelluksen etusivu ennen kyselyn suorittamista.

6.1 Käytetyt teknologiat

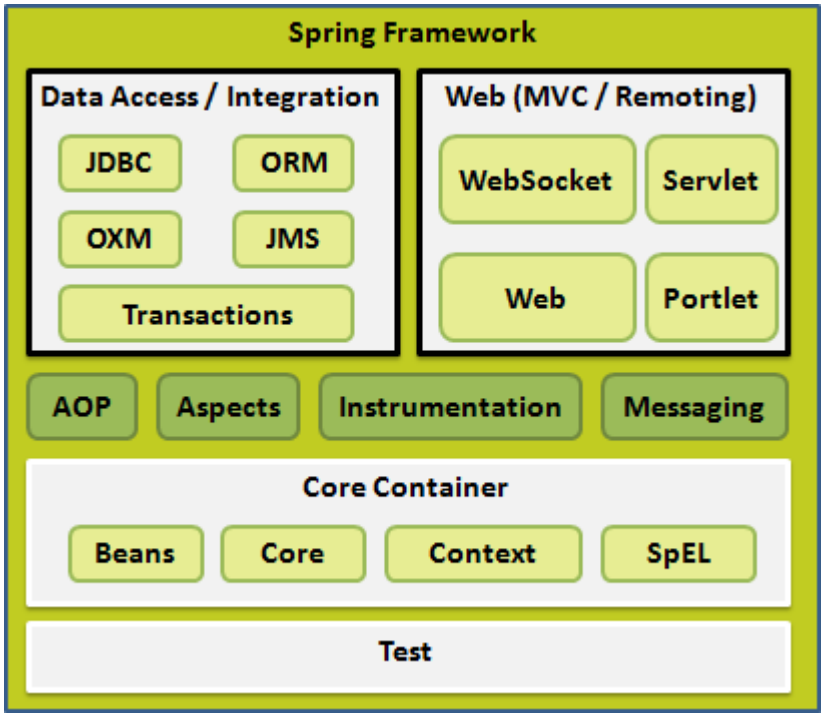
6.1.1 Maven

Maven on koontityökalu, jolla voidaan rakentaa sovellus sovelluskehittäjän määrittelemän pom.xml-tiedoston pohjalta. Pom.xml-tiedostossa voidaan määritellä kaikki sovelluksen tarvitsemat ulkoiset kirjastot, niiden versiot sekä sovelluksen rakentamisohjeistukset. Yleensä projektin rakentamiseen käytetään pluginia, joka hoitaa rakentamisen automaattisesti. Maven hoitaa kaikkien kirjastojen hakemisen omasta repositoriostaan, jossa se ylläpitää kirjastojen versiohistoriaa, ja tarpeen mukaan kirjastojen kääntämisen sovellusta varten. Maven ei tallenna kirjastoja projektiin vaan tallentaa kirjastot omaan kansioonsa käyttäjän kotikansioon. Tällöin vältetään kirjastojen joutuminen versionhallintaan, jolloin ne veisivät turhaa tilaa järjestelmän hallinnassa.

6.1.2 Spring

Spring on Java-pohjainen verkkosovellusten tekemiseen tarkoitettu sovelluskehys. Sen loi Rob Johnson vuonna 2002 samassa yhteydessä, kun hän julkaisi kirjansa *Expert*

One-on-One J2EE Design and Development. Tässä sovelluksessa käytössä oleva versio 4.3 julkaistiin vuonna 2016 kesäkuussa. Nykyinen Springin versio tukee annotaatiopohjaista konfigurointia projekteille eli Spingin käytössä olevat luokat voidaan merkitä Springin kontekstin määrittelemillä annotaatiolla. Nämä annotaatiot antavat ohjeita Springille, kuinka luokat tulisi yhdistää, alustaa ja mihin tarkoitukseen ne on luotu. Annotaatio-pohjaisen asetusten teon ohella on myös perinteinen XML-pohjainen asetusten teko. Tässä tavassa Springin asetukset määritellään dispatcher-servlet.xml-tiedostossa.



Kuva 28. Spring-sovelluskehyksen arkitehtuuri

Spring koostuu noin kahdestakymmenestä moduulista, jotka näkyvät kuvassa 28. Näistä sovelluksen tekijä voi määrittää, mitä sovelluksessaan tarvitsee. Core Container sisältää Springille välttämättömät moduulit: Core määrittää olennaisimman osan Springiä, Bean määrittää BeanFactoryn verkkosovelluksen papujen tekoa varten, Context rakentaa Coren päälle objektien hallinnointikerroksen ja SpEL mahdollistaa objektidiagrammien muokkaamisen ajonaikaisesti. Springin moduulien Data Access -puoli sisältää moduuleita datan käsittelyyn ja tietokantakyselyiden luontiin, ja Web-puoli sisältää työkaluja verkkosovelluksen luontiin.

6.1.3 Hibernate

Hibernaten toiminnallisuudesta ja alkuperästä on kirjoitettu tässä lopputyössä luvussa 5.1.3. Hibernatea käytettiin verkkosovelluksessa kaikkiin tietokantayhteyksiin ja tiedon hakuun. Verkkosovelluksessa oli käytössä täysin sama konfiguraatiotiedosto kuin admin käyttöliittymäsovelluksessa.

6.1.4 JAX-RS

JAX-RS eli Java API for RESTful Web Services on ohjelmointirajapinta, joka määrittää REST-arkitehtuurin toteuttavan verkkopalvelun rakenteen ja annotaatiot, joilla sovellusta voidaan rakentaa. JAX-RS:n kehitti Oracle Corporation vuonna 2013. JAX-RS 1.1 lisättiin Java EE 6:een, joten sitä ei tarvitse määrittää Java EE 6 -pohjaisiin projekteihin erikseen.

6.1.5 Jersey

Jersey on Java pohjainen RESTful web service, joka pohjautuu avoimeen lähdekoodiin. Jerseyllä voidaan luoda REST-palvelu, joko erilleen tai osaksi verkkosovellusta. Jersey antaa työkalut palvelun luontiin (jersey-core, jersey-server), palvelun kanssa kommunikointiin (jersey-client), JaxB-tuen ja JSON-tuen. Jerseylle on olemassa liitosmoduuli Spring-sovelluksiin nimeltään Jersey-spring.

6.1.6 Jersey-spring

Jersey-spring mahdollistaa Spring- ja Jersey-sovelluskehyksien yhteisen kommunikoinnin. Kun Jersey-spring-kirjasto on lisätty projektiin, voi Jersey hyödyntää Springin tarjoamat autowire-annotaatiot, jolloin Jersey voi käyttää Springin luomia ja alustamia objekteja Spring context- kerroksen kautta.

6.1.7 Jackson

Jackson on Java-pohjainen JSONin luontikirjasto. Se pyrki olemaan mahdollisimman kevyt, nopea, oikeellinen ja ergonominen kehittäjille ja sai suosionsa näiden ominaisuuksien pohjalta. Jacksonillä voidaan kääntää objekteja JSON-tiedoksi ja JSON:ia objekteiksi. Jacksonin kehitti suomalainen Tatu Saloranta vuonna 2007.

6.1.8 Log4j

Log4j on Java-pohjainen lokitiedostojen kirjoituskirjasto, jonka on tehnyt Ceki Gülcü vuonna 2001. Log4j tarjoaa sovelluskehittäjälle paremman lokin kirjoitustyökalun, jossa voi määritellä lokiin kirjoitukselle vakavuustason ja Log4j automaattisesti kirjaa ajankohdan sekä tapahtuman luokan lokiin. Log4j myös antaa mahdollisuuksia lokin ohjaamisesta erillisiin tiedostoihin.

6.1.9 Ajax

Ajax eli Asyncronous Javascript And XML on verkkosovelluskehityksen tekniikoita sisältämä Javascript-kirjasto. Ajax mahdollistaa verkkosovelluksessa suorituksen aikaisen tiedon hakemisen joko XML- tai JSON-pohjaisesti. Tekniikka lisää verkkosivuilla vuorovaikutteisuutta, nopeutta ja käytettävyyttä. Ajax kehitettiin vuonna 2005 monen eri tekijän toimesta, joista esimerkkeinä ovat Google ja Microsoft.

6.1.10 BootStrap

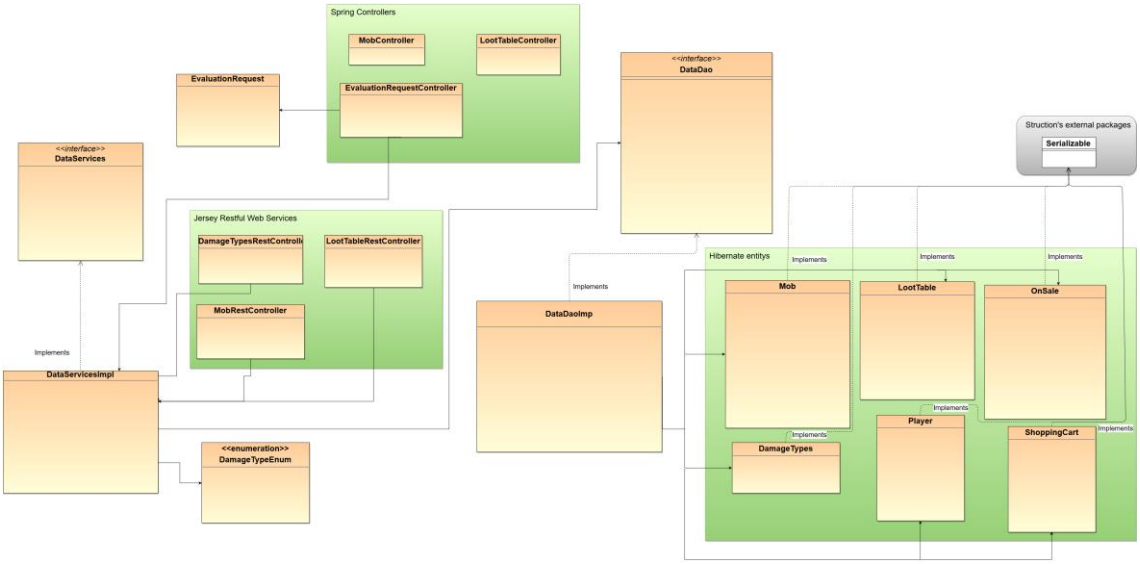
Bootstrap on avoimen lähdekoodin projekti, joka on GitHubin toisiksi eniten tähtiä saanut projekti. Sen loivat alun perin Mark Otto ja Jacob Thornton ollessaan töissä Twitterillä. Avoimen lähdekoodin projektiksi se julkaistiin vuonna 2011 ja sitä ylläpidetään ja kehitetään vielä aktiivisesti. BootStrap toimii verkkosivun ulkoasun yhdistäjänä ja antaa kehittäjälle tehokkaita työkaluja ja valmiita rakenteita verkkosivun kehittämiseen. BootStrap käyttää less-tyylitiedostoa ulkoasun luontiin. Erinäisiä teemoja on saatavilla BootStrappiin useita, ja kehittäjä voi tehdä sellaisen itsekin, jos osaamista less-ohjelmoinnista löytyy.

6.1.11 JQuery

JQuery on avoimen lähdekoodin Javascript-kirjasto, joka julkaistiin vuonna 2006. Kirjasto tarjoaa verkkosivun kehitykseen Javascript-pohjaisia edistyksellisiä metodeja, joilla voidaan esimerkiksi käsitellä toimintoja, luoda animaatioita, valita DOM-elementtejä ja toteuttaa Ajax-sovelluksia. JQuery on noussut maailman suosituimmaksi Javascript-kirjastoksi.

6.2 Sovelluksen elämänkaari ja rakenne

Sovelluksen rakenne on esitetty kuvan 29 luokkakaaviossa. Kuvasta näkyy, että sovellus koostuu kolmesta Springin controllerista, Jerseyn Restful-kontrollereista, DataServicestä sekä sen implementaatiosta, Hibernate entityistä, DataDaosta ja sen implementaatiosta. Näiden lisäksi sovelluksessa on apuluokka DamageTypeEnum, joka vastaa DamageTypes-taulun dataa, sekä EvaluationRequest, jota käytetään etusivun syöttölomakkeen tiedon välityksessä. Spring controllerit vastaavat käyttäjän url-pyyntöihin ja esittävät frontend-sivut käyttäjälle. Frontendin sisältämä javascript-koodi vastaa tiedon hausta REST-controllereilta AJAXia käyttäen. Arviot käyttäjälle laskee EvaluationRequestController, joka käyttää tiedon hakuun DataServiceImpl-luokkaa. DataServiceImpl-luokka käyttää tiedon hakuun tietokannasta DataDaoimpl-luokkaa, joka hyödyntää Hibernaten toiminnallisuutta. DataDao- ja DataService-rajapinnat ovat olemassa, jotta tarpeen vaatiessa tietokantayhteys voidaan korvata toisella tietokannalla, tai jos service halutaan vaihtaa toiseen implementaatioon.



Kuva 29. Verkkosovelluksen luokkakaavio. Tarkempi luokkakaavio liitteessä 7.

6.2.1 Sovelluksen käynnistyminen

Sovelluksen käynnistyessä Spring-sovelluskehys luo ilmentymän itsestään liitteen 2 web.xml-tiedoston mukaan. Tiedostossa on määritelty Springille olennaisimmat asetukset. Tiedosto sisältää servlet-määrittelyt, jotka vastaavat url-pyyntöihin. Servletejä tässä projektissa on dispatcher, joka vastaa kaikkiin *.htm-loppuisiin url-pyyntöihin, sekä Jersey-servlet, joka vastaa Restful-webservicestä ja vastaa /rest/-alkuisiin url-pyyntöihin. Log4j:n asetustiedosto määritellään myös web.xml-tiedostossa. Web.xml sisältää viitteen applicationContext.xml-tiedostoon, joka on liitteessä 3. Liitteen tiedoston ylimmät rivit määrittelevät Springille, mitä Schema-määritelmiä projektissa on käytössä. Näiden avulla voidaan luoda XML:n mukaisia asetus-tagejä projektin konfiguraatiotiedostossa. Tiedostossa on kerrottu Springille, mistä Hibernaten asetukset löytyvät: classpath:hibernate.cfg.xml, mistä Springin tulisi etsiä annotoituja luokkia, jotka ovat projektin juuressa olevassa paketissa: fi.metropolia.ristotu, ja että sovellus on annotaatiopohjainen: <context:annotation-config/>-tag. Kun Spring on luonut ilmentymän itsestään, se määrittää verkkosovellusosan liitteen 4 mukaisen dispatcher-servletin. Dispatcher-servlet.xml määrittää Springin käytössä olevat Beanit, osoitteet, joista ne vastaavat ja mitä näkymää ne käyttävät.

Kun Spring alustaa luokat, se lukee luokkien alussa olevat annotaatiot ja luo niiden mukaisen ilmentymän kyseisestä luokasta. Jos luokka sisältää konstruktorimetodin, jossa on @Autowired annotaatio, Spring etsii automaattisesti konstruktorin parametrinä olevat

luokat ja ylläpitää niistä vain yhden ilmentymän. Käyttämällä Spring-Jerseytä voidaan hyödyntää myös näitä annotaatioita Jerseyn Rest-kontrollereissa.

6.2.2 Kyselyn luonti etusivun syöttölomakkeen kautta

Sovelluksen etusivu näyttää hyvin yksinkertaiselta, mikä näkyi kuvassa 27. Etusivulla on vain navigointi-header ja syöttölomake käyttäjälle täytettäväksi. Syöttölomakkeen tiedoilla verkkosovellus luo arvion, mitä vihollisia käyttäjän kannattaisi metsästää pelissä. Etusivun syöttölomakkeen kenttien selitys:

- Profession level: Pelaajan hahmon taitotaso, jota hän käyttää vihollisten metsästykseen. Kentän tietoa ei hyödynnetä arvioiden tekemisessä. Se on käyttäjäseurantaa varten oleva kenttä.
- Armor set: Jokaista vahinkotyyppiä varten on yksi syöttökenttä. Pelaajan käyttämä haarniska voi puolustaa useaa eri vahinkotyyppiä vastaan ja tähän syötetään ne luvut.
- Armor plate set: Pelaajan haarniskaansa kiinnittämien lisäsuojusten antamat puolustusvahinkotyyppiluvut kirjataan näihin syöttökenttiin.
- Weapon damage: (x - y): Ensimmäiseen kenttään syötetään pelaajan käyttämän aseen pienin vahinkomäärä ja toiseen kenttään suurin.
- Weapon amplifier damage: Pelaajan aseeseensa liittämän vahvistimen lisäämä vahinkomäärä.
- Preferred amount of damage from mob: (default 15): Kuinka paljon pelaaja on valmis ottamaan keskimäärin vahinkoa viholliselta per lyönti. Alustava arvo on 15 vahinkoa.
- Preferred maximum amount of shots to kill mob: (default 10): Kuinka monta kertaa pelaajan tulee ampua vihollista maksimissaan, jotta se kuolee.
- Preferred minium amount of shots to kill mob: (default 3): Mikä on vähin määrä, mitä vihollista tulee ampua. Pelissä ei oli järkevää metsästää liian helppoja vihollisia.

Käyttäjän lähettäessä etusivun lomakkeen javascript-tiedosto main.js muuttaa kyselylomakkeen tiedot evaluationRequest-objektiksi ja lähettää tämän kuvan 30 EvaluationRequestController-luokalle Ajaxia käyttäen. Ajax-funktio hyödyntää JSON.stringify-metodia ja lähettää tiedon JSON-muodossa kontrollerille. Kontrolleri muuttaa saamansa JSONin Jacksonia käyttäen EvaluationRequest-olioksi ja hyödyntää tältä oliolta saamiaan tietoja.

```java
@Controller
public class EvaluationRequestController {

    @Autowired
    private DataServices dataServices;
    private static final Logger log =
            LoggerFactory.getLogger(EvaluationRequestController.class);

    @RequestMapping(value = "/evaluation.htm", method = RequestMethod.POST,
            consumes = "application/json; charset=UTF-8")
    @ResponseBody
    public String evaluation(@RequestBody final String evaRequest) throws IOException {
        ObjectMapper mapper = new ObjectMapper();
        EvaluationRequest evr = mapper.readValue(evaRequest, EvaluationRequest.class);
        int totaldamage = (evr.getWeaponMinDamage() + evr.getWeaponMaxDamage())
                / 2 + evr.getWeaponAmplifierDamage();
        log.info("totaldamage: " + totaldamage);
        Map<DamageTypeEnum, Integer> protectionValues = extractProtectionValues(evr);
        log.info("armor: " + protectionValues);
        List<Mob> ml = dataServices.getValidMobsForHunter(protectionValues,
                totaldamage, evr.isBooleanMobsWithSameDamagetype(),
                evr.getAmountOfDamageFromMob(), evr.getAmountOfShotsToKillMobMin(),
                evr.getAmountOfShotsToKillMobMax());
        TreeMap<Float, Mob> organisedMobsByMarkup = validateMobs(ml);
        ArrayList<Map<Float, Mob>> returnData = new ArrayList<>();
        Map<Float, Mob> bestTenMobs = new HashMap<>();
        int i = 0;
        for(Map.Entry<Float, Mob> entry : organisedMobsByMarkup.entrySet()){
            if(!entry.getKey().equals(Float.NaN)){
                if(i < 10){
                    entry.getValue().setLoot(entry.getValue().getLoot());
                    bestTenMobs.put(entry.getKey(), entry.getValue());
                    i++;
                }else{
                    break;
                }
            }
        }
        returnData.add(bestTenMobs);
        returnData.add(organisedMobsByMarkup);
        return mapper.writeValueAsString(returnData);
    }
```

Kuva 30. Etusivun syöttölomakkeen vastaanottava EvaluationRequestController.java

Käyttäjän syöttämistä tiedoista saadaan laskettua aseen keskimääräinen vahinko laskemalla aseen minimi- ja maksimivahinkojen keskiarvo ja lisäämällä siihen aseen vahvistimen aiheuttaman vahingon. Jos pelaaja on syöttänyt puolustusvahinkotyyppejä, ExtractProtectionValues-funktio hakee ne Mapiin. Tämän jälkeen DataServiceltä pyydetään lista kaikista valideista vihollisista, joita pelaaja voi metsästää. DataServicen palauttamat viholliset järjestetään kuvan 31 validateMobs-funktion toimesta TreeMapiin.

```
private TreeMap<Float, Mob> validateMobs(List<Mob> ml) {
    TreeMap<Float, Mob> bestMarkupMobs = new TreeMap<>(Collections.reverseOrder());
    for (Mob m : ml) {
        if(m.getLoot() != null){
            float totalMarkupEstimateForMob = 0.00f;
            int amountOfMarkups = 0;
            for (LootTable lt : m.getLoot()) {
                if (lt.getMarkupUnit() != null && lt.getMarkupUnit().equals("%")) {
                    amountOfMarkups++;
                    totalMarkupEstimateForMob += (lt.getMaxTTValue() * (lt.getMarkup()/100)) - lt.getMaxTTValue();
                }else if (lt.getMarkupUnit() != null && lt.getMarkupUnit().equals("PED")) {
                    amountOfMarkups++;
                    totalMarkupEstimateForMob += lt.getMarkup();
                }
            }
            totalMarkupEstimateForMob = totalMarkupEstimateForMob / amountOfMarkups;
            bestMarkupMobs.put(totalMarkupEstimateForMob, m);
        }
    }
    return bestMarkupMobs;
}
```

Kuva 31. ValidateMobs-funktio, joka määrittää, kuinka paljon markkinalisäarvoa viholliselta voi saada

ValidateMobs käy DataServicen palauttaman listan kaikki viholliset läpi ja laskee niiden pudottaman jokaisen esineen markkinalisäarvon ja tekee niiden perusteella keskiarvon, jonka mukaan se järjestää ne TreeMapiin. Vihollisen markkinalisäarvio tulee TreeMapissä avaimeksi, jolloin se järjestää viholliset automaattisesti. Markkinalisäarvo on Float-muuttuja kahdella desimaalilla, joten saman avaimen syntyminen TreeMappiin on hyvin epätodennäköistä. TreeMap on käänteisessä järjestyksessä, joten isoimman markkinalisäarvon omaavat viholliset ovat TreeMapin kärjessä. Tämän TreeMapin 10 parhaan markkinalisäarvon omaavaa vihollista esitetään käyttäjälle etusivulla ja niiden pudottamat tavaratkin esitellään taulukosta painamalla avautuvassa modaalissa. Myös lista kaikista vihollisista, joita pelaaja voi metsästää, esitetään alempana etusivulla taulukossa, mutta näiltä vihollisilta ei esitetä niiden pudottamia tavaroita.

DataServicen getValidMobsForHunter-funktio ottaa vastaan EvaluationRequestControllerilta Mapin käyttäjän puolustusvahinkotyyppejä, käyttäjän tekemän vahingon, boolean-arvon, halutaanko samoja vihollisia kuin mitä puolustusvahinkotyyppejä on, vihollisten tekemän vahingon määrän ja pelaajan tarvitsemien ammuntakertojen minimi- ja maksimimäärän. Jos pelaaja on syöttänyt puolustusarvoja ja pyydetään samoja vihollisia kuin pelaaja on syöttänyt puolustusarvoja, pyytää DataService DataDao:lta funktiolla getMobsWithDamageTypes() vihollisia, jotka tekevät tiettyjä pelaajan syöttämiä vahinkotyyppejä. Muutoin käytetään kriteerinä pelaajan tekemää vahinkoa ja pelaajan syöttämää ampumakertoja. Tällöin käytössä on DataDao:n getMobsWithLessHPThanButMoreThan-funktio. Kun DataDao on palauttanut listan vihollisista, tarkistetaan, onko käyttäjä

syöttänyt puolustusvahinkotyyppejä. Jos näin on, käytetään validateMobDamageToArmors-funktiota poistamaan kaikki viholliset listasta, jotka tekevät liikaa vahinkoa pelaajaan. Pelaajan lomakkeen kautta syöttämä amountOfDamageFromMob toimii validoijana vihollisten karsinnassa. Listan viholliset käydään läpi yksi kerrallaan ja vihollisilta tarkistetaan, tekevätkö ne samaa vahinkotyyppiä kuin pelaajan haarniskat puolustavat. Jos haarniska puolustaa samaa vahinkotyyppiä, poistetaan vahingosta haarniskan puolustama määrä. Näin saadaan lista vihollisista, jotka eivät tee pelaajaan liian paljon vahinkoa.

```
166     @Override
        public List<Mob> getValidMobsForHunter(Map<DamageTypeEnum, Integer>
168             protectionValues, int totaldamage, boolean mobsWithSameDamagetype,
169             int amountOfDamageFromMob, int amountOfShotsToKillMobMin,
170             int amountOfShotsToKillMobMax) {
171         List<Mob> ml;
172         if(checkIfProtectionValuesExist(protectionValues) && mobsWithSameDamagetype){
173             ml = dataDao.getMobsWithDamageTypes(protectionValues);
174         }else{
175             ml = dataDao.getMobsWithLessHPThanButMoreThan(totaldamage *
176                     amountOfShotsToKillMobMax, totaldamage * amountOfShotsToKillMobMin);
177         }
178         List<Mob> newml = new ArrayList();
179         if (checkIfProtectionValuesExist(protectionValues)) {
180             newml.addAll(validateMobDamageToArmors(protectionValues, ml,
181                     amountOfDamageFromMob));
182             return newml;
183         } else {
184             return ml;
185         }
186
187     }
188
189     private Collection<? extends Mob> validateMobDamageToArmors(
190             Map<DamageTypeEnum, Integer> protectionValues,
191             List<Mob> ml, int amountOfDamageFromMob) {
192         List<Mob> newml = new ArrayList<>();
193         ml.stream().forEach((m) -> {
194             int damageToPlayer = Math.round(m.getDamage());
195             if(Math.round(m.getDamage()) != 0){
197                 for (Map.Entry<DamageTypeEnum, Integer> entry:
                            protectionValues.entrySet()) {
199                     for (DamageTypes dt : m.getDamageTypes()) {
200                         if (dt.getName().equals(entry.getKey().toString())) {
201                             damageToPlayer -= entry.getValue();
202                         }
203                     }
204                     if (damageToPlayer < amountOfDamageFromMob) {
205                         newml.add(m);
206                     }
207                 }
208             });
209             return newml;
210     }
```

Kuva 32. EvaluationRequestControllerin kutsuma getValidMobsForHunter-funktio ja sen hyödyntämä ValidateMobDamageToArmors-funktio

Kuvassa 33 näkyvä DataDaon funktio getMobsWithLessHPThanButMoreThan saa parametrina kaksi lukua, jotka määrittelevät, miltä elämäpisteväliltä vihollisia haetaan. Viholliseen täytyy tehdä tietyn verran vahinkoa, jotta se kuolee, ja tämän määrittää elämäpisteet eli HP. Funktio hakee Hibernaten sessionFactoryltä session ja luo transaktion tietokantaan. Criterialla voidaan luoda tiettyyn luokkaan kriteeri, jonka mukaan tietueita voidaan karsia tuloksesta. Tässä tapauksessa luodaan kriteeri, joka hakee vihollisen HP-kentän perusteella kaikki viholliset, joiden HP ja damage ei ole 0 ja niiden HP on enemmän kuin moreThan-parametri ja vähemmän kuin lessThan-parametri. Kriteerin täyttävät viholliset lisätään mobList-muuttujaan, joka palautetaan kyselyn onnistuessa. Jos kysely epäonnistuu, transaktio perutaan ja sessio suljetaan.

```
354     @Override
        public List<Mob> getMobsWithLessHPThanButMoreThan(int lessThan, int moreThan) {
356         List<Mob> mobList = null;
357         try {
358             session = this.sessionFactory.openSession();
359             tx = session.beginTransaction();
360             Criteria cr = session.createCriteria(Mob.class);
361             cr.add(Restrictions.le("HP", lessThan));
362             cr.add(Restrictions.gt("HP", moreThan));
363             cr.add(Restrictions.ne("HP", 0));
364             cr.add(Restrictions.ne("damage", 0f));
365             mobList = cr.list();
366             tx.commit();
367         } catch (Exception e) {
368             log.error("getMobsWithLessHPThanButMoreThan Error: " + e);
369             tx.rollback();
370             session.close();
371         }
372         return mobList;
373     }
```

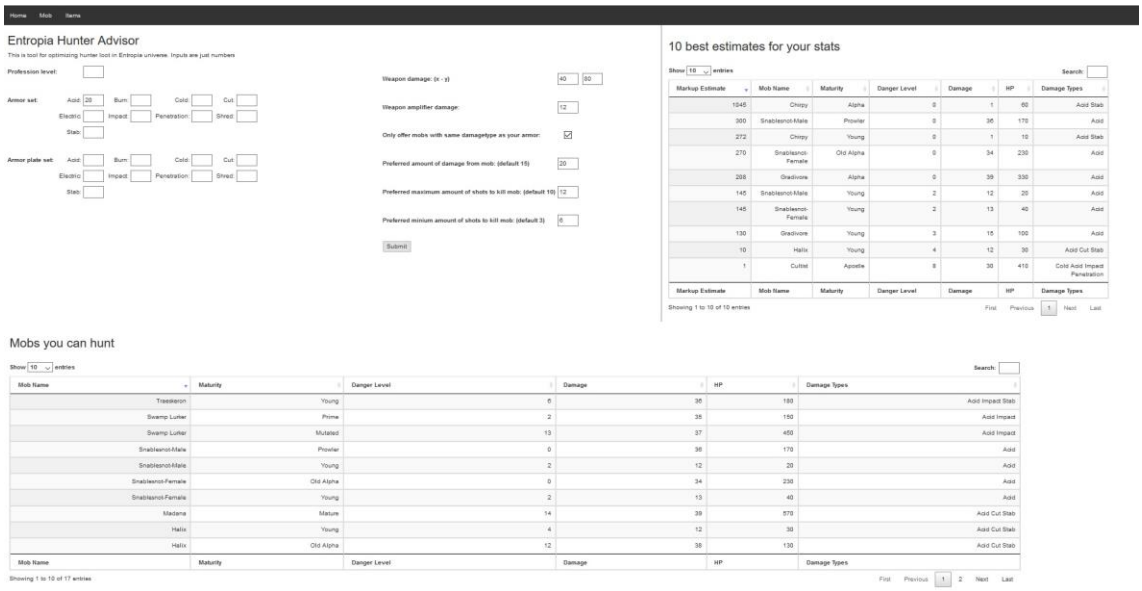
Kuva 33. getMobsWithLessHPThanButMoreThan-funktio hakee Hibernaten avulla tietokannasta vihollisia tietyltä elämäpisteväliltä

DataDaon kuvassa 34 näkyvä getMobsWithDamageTypes-funktio toimii hieman eri tavalla. Tässä funktiossa kriteerinä toimii vihollisen tekemät vahinkotyypit. Kriteerit tehdään alias taulua kohtaan, joka on damageTypes-taulu. Jokaisesta protectionValuesta mapissä luodaan kriteeri kyselyyn, joiden mukaan kyselyyn jätetään vain ne viholliset, jotka sisältävät nämä vahinkotyypit.

```
375     @Override
        public List<Mob> getMobsWithDamageTypes(Map<DamageTypeEnum, Integer> protectionValues) {
377         List<Mob> mobList = null;
378         try {
379             session = this.sessionFactory.openSession();
380             tx = session.beginTransaction();
381             Criteria cr = session.createCriteria(Mob.class);
382             Criterion[] ress = new Criterion[1];
383             cr.createAlias("damageTypes", "damageTypesAlias");
384             cr.add(Restrictions.ne("damage", 0f));
385             for (Map.Entry<DamageTypeEnum, Integer> entry : protectionValues.entrySet()) {
386                 Criterion res = Restrictions.eq("damageTypesAlias.name", entry.getKey().toString());
387                 if (ress.length == 1) {
388                     ress[0] = res;
389                 } else {
390                     ress = addToRess(ress, res);
391                 }
392             }
393             cr.add(Restrictions.or(ress));
394             mobList = cr.list();
395             System.out.println("damageTypes mobs: " + mobList);
396             tx.commit();
397         } catch (Exception e) {
398             log.error("getMobsWithDamageTypes Error: " + e);
399             tx.rollback();
400             session.close();
401         }
402         return mobList;
403     }
404
405     private Criterion[] addToRess(Criterion[] rest, Criterion res) {
406         Criterion[] temp = new Criterion[rest.length + 1];
            for (int i = 0; i < rest.length; i++) {
408             temp[i] = rest[i];
409         }
410         temp[rest.length] = res;
411         return temp;
412     }
413
```

Kuva 34. DataDaon getMobsWithDamageTypes-funktio ja sen apufunktio addToRess

Kun kaikki validointi ja arvioiden muodostus on suoritettu, palautetaan tieto vihollisista ja niiden pudottamista tavaroista frontendiin, jossa Ajax ottaa ne vastaan. Javascript esittää käyttäjälle tuloksen kahdessa eri taulukossa etusivulla ilman, että sivua tarvitsee ladata uudelleen. Taulukot on luotu käyttäen DataTables javascript -kirjastoa. Kymmenen parhaan vihollisen esittämä taulukko myös sisältää Javascriptin luoman ominaisuuden, että jos käyttäjä painaa taulukon riviä, avautuu BootStrapin sisältämä modaali, jossa on kolmas taulukko vihollisen pudottamista tavaroista. Etusivun loppunäkynä on kuvassa 35.



Kuva 35. Entropia verkkosovelluksen etusivu käyttäjän pyytämän arvion valmistumisen jälkeen

6.2.3 Tiedon esittäminen Mob- ja Item-sivuilla

Sovelluksen otsikkorivistä löytyy linkki Mob- ja Item-sivuille. Näitä sivuja hallinnoi Springin kontrollerit MobController ja LootTableController. Kontrollerit ovat hyvin yksinkertaiset, ja ne vain palauttavat näkymän mob.jsp tai loottable.jsp. Näillä sivuilla datan hakemisesta ja esittämisestä vastaa Javascript, Ajax ja DataTables. Ajax pyytää tiedon JSON-muodossa Jerseyn toteuttamilta REST-kontrollereilta. Esimerkiksi vihollistietoa pyydettäessä käytetään MobRestController-luokkaa, joka on osittain kuvassa 36. Jerseylle kerrotaan annotaatioilla luokan olevan kontrolleri ja sen vastaavan osoitteesta /mob. Luokassa on myös käytetty hyväksi Springin @Autowired-annotaatiota, jolla on yhdistetty DataService-luokan käyttöön. Rest-funktiot on määritelty annotaatiolla, joka märittää niiden toimintamallin, joko: GET, POST, PUT tai DELETE. GET on tarkoitettu tiedon hakuun, POST tiedon tallentamiseen, PUT tiedon päivittämiseen ja DELETE tiedon poistamiseen. Turvallisuussyistä sovellukseen on toteutettu vain erilaisia GET funktioita, mutta DataDao- sekä DataService-toteutukset löytyvät suurimmalle osalle dataa sekä poisto, päivitys, haku että lisäystoiminnoille.

```
30  @Controller
31  @Path("/mob")
32  public class MobRestController {
33
34      @Autowired
35      private DataServices dataServices;
36      private final ObjectMapper mapper;
37      private static final Logger log =
38              LoggerFactory.getLogger(MobRestController.class);
39
40      public MobRestController(){
41          log.info("MobRestController INIT");
42          mapper = new ObjectMapper();
43      }
44
45      /* Get a single objct in Json form in Spring Rest Services */
46      @GET
47      @Path("/get/{id}")
48      @Produces("application/json")
49      public Response getMob(@PathParam("id") String id)
50              throws JsonProcessingException {
51          Mob mob = null;
52          try {
53              mob = dataServices.getMobById(Integer.parseInt(id));
54
55          } catch (Exception e) {
57              e.printStackTrace();
58          }
59          if(mob == null){
60              return Response.status(Response.Status.NOT_FOUND)
61                      .entity("Mob Entity not found for ID: " + id).build();
62          }
63          return Response.ok(mapper.writeValueAsString(mob),
64                  MediaType.APPLICATION_JSON).build();
    }
```

Kuva 36. MobRestController-luokka, joka vastaa vihollisten palauttamisesta JSON-muodossa

Kun sovelluksen /mob/get/{id} url-osoitetta kutsutaan, getMob-funktio hakee DataServiceltä tietyn vihollisen, jonka id on urlissa annettu id. Jos vihollinen löytyi, käytetään Jacksonin mapperiä objektin kääntämiseksi JSON-tiedoksi. JSON-tieto paketoidaan responseen, johon saadaan lisäksi status-tieto kyselyn onnistumisesta. DataService hakee tiedon vihollisesta DataDao:sta, jossa haku on toteutettu hyvin samanlaisella funktiolla kuin aiemmin esitetty.

7 Järjestelmän toteutetut ja toteuttamattomat ominaisuudet

Sovellus toteutti kaikki siltä odotetut käyttötapaukset, mutta toteutettavia ominaisuuksia olisi ollut paljon lisää. Toteutettuja ominaisuuksia olivat tietokannan luonti ja datan vienti siihen, admin-käyttöliittymän datan esitys, admin-käyttöliittymän kautta datan vienti, admin-käyttöliittymän kautta tiedon muokkaus, admin-käyttöliittymän kautta tietokannan uudelleen alustaminen, verkkosovelluksessa arvioiden esitys käyttäjälle, verkkosovelluksessa tiedon esitys käyttäjälle ja REST-rajapintojen luonti tärkeimpiin tauluihin. Toteutettavia ominaisuuksia olisivat olleet useamman tietokantataulun suora muokkausmahdollisuus admin-käyttöliittymässä, admin-käyttöliittymästä tiedon ulosvienti ominaisuus, verkkosovelluksen kirjautumisominaisuus, verkkosovelluksen verkkokauppapuoli, REST-rajapinnan laajennus kaikkiin tauluihin ja verkkosovelluksessa aseiden ja haarniskojen tietojen hakeminen tietokannasta suoraan. Tärkeintä projektissa oli saada toimiva järjestelmä kasaan, jonka palaset toimisivat yhteistyössä keskenään luoden käyttäjälle arvioita. Ohessa on lyhyt selitys jokaisesta toteuttamattomasta ominaisuudessa järjestelmästä.

7.1 Muokkausmahdollisuus admin-käyttöliittymässä useampaan tauluun

Adminille olisi hyödyllistä omata muokkausmahdollisuus kaikkiin tauluihin ja kaikkiin tietueisiin. Ominaisuus jäi toteuttamatta järjestelmästä usean ManyToMany-liitoksen muokkauksen hankaluuden takia. Esimerkkinä tästä on taulu, joka esittää vihollisten tiedot yhdessä taulukossa ja sen alapuolella, kun taulun riviä painetaan, esitetään kaikki vihollisen pudottamat tavarat taulukossa. Näihin taulukoihin muokkausmahdollisuuden lisääminen olisi hyvin haastavaa mutta tarpeellista.

7.2 Admin-käyttöliittymästä tiedon ulosvienti ominaisuus

Käyttöliittymään olisi hyvä lisätä painike, jolla saisi ladattua kaikki tietokannan taulut XML-muodossa zip-paketoituina. Tämä mahdollistaisi tietokannan siirtämisen ja backup-toiminnon kaikelle datalle. Tieto pitäisi tuoda sovelluksesta schema-tiedoston määritelmien mukaisesti.

7.3 Verkkosovelluksen kirjautumisominaisuus

Kirjautuminen olisi tärkeä ominaisuus sovellukselle, jos haluttaisiin säilyttää käyttäjäkohtaista dataa ja arvioita käyttäjää varten. Verkkosovelluksen verkkokauppapuoli myös hyödyntäisi tätä tietoa. Mahdollinen Premium-kirjautuminen myös toteutettaisiin kirjautumista hyödyntäen. Premium voisi mahdollistaa laajemman datan tarkastelumahdollisuuden ja kattavammat arviot käyttäjälle.

7.4 Verkkosovelluksen verkkokauppapuoli

Tietokanta sisältääkin jo 4 taulua tätä ominaisuutta varten, mutta se jätettiin verkkosovelluksesta pois aikarajoitteiden takia. Tarkoituksena olisi luoda käyttäjille mahdollisuus mainostaa joko käyttäjien kaupoissa esillä olevia tuotteita tai sitten myydä omia varastossaan olevia tavaroita. Järjestelmä hyödyntäisi LootTable-tietokantataulun esineitä, mutta niitä voisi olla pelaajalla useiden tavaroiden pinoja myynnissä. Tämän lisäksi pelaajilla olisi henkilökohtainen ostoskori, johon kerätä tuotteita ja varata ne itselleen muilta pelaajilta. Kaupan käynti tapahtuisi Entropia Universe -pelin sisällä eikä verkkosovelluksessa.

7.5 REST-rajapinnan laajennus kaikkiin tauluihin

Tällä hetkellä REST-rajapinnat kattavat vain tärkeimmät tietokanta taulut, jotka ovat Mob, LootTable ja DamageTypes. Verkkokaupan toteuttamisen myötä REST-rajapintaan olisi hyvä lisätä ainakin OnSale REST -toteutus.

7.6 Verkkosovelluksessa aseiden ja haarniskojen hakeminen tietokannasta suoraan

Tietokantaan olisi tärkeätä lisätä 2 uutta taulua aseiden ja aseiden vahvistimien tietoja varten ja haarniskojen ja haarniskojen lisäsuojusten tarkempia tietoja varten. Tämän hetkinen LootTable-tietokantataulu ei sisällä tietoa aseiden vahingosta eikä haarniskojen puolustusarvoista. Jos nämä tarkemmat tiedot saataisiin tietokantaan, voitaisiin etusivun

syöttölomakkeessa pyytää käyttäjää vain valitsemaan aseensa ja haarniskansa valmiista listasta, joka saisi tiedot tietokannasta nimen perusteella kehittyneemmistä tauluista.

7.7 Järjestelmän jatkokehitys

Järjestelmää tullaan vielä jatkokehittämään ja ylläpitämään kirjoittajan vapaa-aikana mahdollisimman paljon. Arvioita käyttäjille luova logiikka vaatii hienosäätöä ja käyttäjä kokemusta parantuakseen. Edellä mainittu lista toteuttamattomista ominaisuuksista olisi tarkoitus vähitellen kehittää loppuun ja saada verkkosovelluksesta paljon monimuotoisempi ja rikkaampi kokemus. Kun sovellus on kehittynyt tähän pisteeseen, voisin harkita sovelluksen julkaisemista isommallekin yleisölle. Tällä hetkellä vain minä ja muutama tuttuni käyttää sovellusta pienmuotoisesti.

8 Yhteenveto

Insinöörityön aikana toteutettiin järjestelmä, joka koostui tietokannasta, verkkosovelluksesta ja admin-käyttöliittymäsovelluksesta. Verkkosovellus toteutettiin käyttäen Springiä ja siihen tehtiin REST-rajapinta käyttäen Jerseytä ja Jacksoniä. Järjestelmän tavoitteena oli viedä tietokantaan suuria määriä monimuotoista tietoa, hallinnoida tietokantaa, esittää tietokannan tietoa käyttäjälle ja luoda käyttäjälle arvioita hänen pelihahmonsa taitojen ja tietokannan tietojen pohjalta. Näihin tavoitteisiin insinöörityössä luotu järjestelmä kykeni hyvin. Järjestelmä toteutti siltä vaaditut toimet ja siihen tuli useita ominaisuuksia, jotka pitävät sen modernina vielä kauan.

Insinöörityönä järjestelmä oli hyvin onnistunut. Se sisälsi paljon moderneja sovelluskehyksiä, kirjastoja ja työkaluja. Projektin kautta opin paljon sovelluksen rakenteellisesta suunnittelusta, tietokannoista ja verkkosovelluskehityksestä. Järjestelmään jäi paljon kehitettävää ja sen kehittämistä on hyvä jatkaa lopputyön valmistumisenkin jälkeen.

Lähteet

1 Fowler, Amy. A Swing Architecture Overview. Verkkodokumentti. http://www.oracle.com/technetwork/java/architecture-142923.html (luettu 3.11.2016)

2 Red Hat Middleware, LLC. 2004. Hibernate Envers Reference Documentation. Verkkodokumentti. http://docs.jboss.org/envers/docs/ (luettu 4.6.2015).

3 Ort, Ed ja Mehta, Bhakti. 2003. Java Architecture for XML Binding (JAXB). Verkkodokumentti. http://www.oracle.com/technetwork/articles/javase/index-140168.html (luettu 3.11.2016)

4 Jackson Project Home @github https://github.com/FasterXML/jackson (luettu 3.11.2016)

5 About SAX. Sax projektin kotisivu. http://www.saxproject.org/ (luettu 10.11.2016)

Hibernate-konfiguraatiotiedosto hibernate.cfg.xml

Tässä tiedostossa määritellään Hibernateen liittyvät asetukset. Tiedostosta on poistettu käyttäjätunnukset ja salasanat.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-configuration PUBLIC
"-//Hibernate/Hibernate Configuration DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
  <session-factory>
    <!-- Main Server -->
    <property name="hibernate.dialect">org.hibernate.dialect.MySQLDialect</property>
    <property name="hibernate.connection.driver_class">com.mysql.jdbc.Driver</property>
    <property name="hibernate.connection.url"> jdbc:mysql://risto.turtiainen.eu:3306/Entro-
pia?zeroDateTimeBehavior=convertToNull </property>
    <property name="hibernate.connection.username"> </property>
    <property name="hibernate.connection.password"> </property>
    <!-- local test Server -->
    <property name="hibernate.dialect">org.hibernate.dialect.MySQLDialect</property>
    <property name="hibernate.connection.driver_class">com.mysql.jdbc.Driver</property>
    <property name="hibernate.connection.url"> jdbc:mysql://localhost:3306/entropia?zero-
DateTimeBehavior=convertToNull</property>
    <property name="hibernate.connection.username"> </property>
    <property name="hibernate.connection.password"> </property>
    <property name="hibernate.hbm2ddl.auto">update</property>
    <property name="hibernate.show_sql">true</property>
    <property name="hibernate.query.factory_class"> org.hibernate.hql.inter-
nal.ast.ASTQueryTranslatorFactory </property>
    <mapping class="entropia.entity.LootTable"/>
    <mapping class="entropia.entity.DamageTypes"/>
    <mapping class="entropia.entity.Mob"/>
    <mapping class="entropia.entity.OnSale"/>
    <mapping class="entropia.entity.Player"/>
    <mapping class="entropia.entity.ShoppingCart"/>
  </session-factory>
</hibernate-configuration>
```

Spring-konfiguraatiotiedosto web.xml

Web.xml-tiedostossa esitellään servletit ja niiden vastaamat url osoitteet.

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="3.0" xmlns="http://java.sun.com/xml/ns/javaee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLoca-
tion="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-
app_3_0.xsd">
   <context-param>
      <param-name>contextConfigLocation</param-name>
      <param-value>/WEB-INF/applicationContext.xml</param-value>
   </context-param>

   <!-- Jersey -->
   <servlet>
      <servlet-name>jersey-servlet</servlet-name>
      <servlet-class>
         com.sun.jersey.spi.spring.container.servlet.SpringServlet
      </servlet-class>
      <init-param>
         <param-name>com.sun.jersey.config.property.packages</param-name>
         <param-value>fi.metropolia.ristotu.controller</param-value>
      </init-param>
      <load-on-startup>1</load-on-startup>
   </servlet>

   <servlet-mapping>
      <servlet-name>jersey-servlet</servlet-name>
      <url-pattern>/rest/*</url-pattern>
   </servlet-mapping>

   <!-- Spring -->
   <servlet>
      <servlet-name>dispatcher</servlet-name>
```

```
        <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
        <load-on-startup>2</load-on-startup>
    </servlet>
    <servlet-mapping>
        <servlet-name>dispatcher</servlet-name>
        <url-pattern>*.htm</url-pattern>
    </servlet-mapping>
    <session-config>
        <session-timeout>
            30
        </session-timeout>
    </session-config>
    <welcome-file-list>
        <welcome-file>redirect.jsp</welcome-file>
    </welcome-file-list>

    <!--J4Log -->
    <context-param>
        <param-name>log4jConfigLocation</param-name>
        <param-value>/WEB-INF/classes/log4j.properties</param-value>
    </context-param>
    <listener>
        <listener-class>
            org.springframework.web.util.Log4jConfigListener
        </listener-class>
    </listener>
    <listener>
        <listener-class>org.springframework.web.context.ContextLoaderListener</lis-
tener-class>
    </listener>
</web-app>
```

Spring-konfiguraatiotiedosto applicationContext.xml

Springin käyttämä konfiguraatiotiedosto, jonka pohjalta Spring luo uuden ilmentymän.

```
<?xml version='1.0' encoding='UTF-8' ?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:aop="http://www.springframework.org/schema/aop"
    xmlns:tx="http://www.springframework.org/schema/tx"
    xmlns:context="http://www.springframework.org/schema/context"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-4.3.xsd
    http://www.springframework.org/schema/aop
    http://www.springframework.org/schema/aop/spring-aop-4.3.xsd
    http://www.springframework.org/schema/tx
    http://www.springframework.org/schema/tx/spring-tx-4.3.xsd
    http://www.springframework.org/schema/context
    http://www.springframework.org/schema/context/spring-context-4.3.xsd">

<!-- Hibernate session factory -->
<bean id="sessionFactory" class="org.springframework.orm.hibernate4.LocalSessionFacto-
ryBean">
<property name="configLocation" value="classpath:hibernate.cfg.xml"/>
</bean>

<context:component-scan base-package="fi.metropolia.ristotu"/>
<context:annotation-config/>

</beans>
```

Spring-konfiguraatiotiedosto dispatcher-servlet.xml

Springin käyttämä konfiguraatiotiedosto, jonka pohjalta Spring luo uuden verkkopalvelun ja määrittää sen asetukset.

```
<?xml version='1.0' encoding='UTF-8' ?>
<beans xmlns="http://www.springframework.org/schema/beans"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:aop="http://www.springframework.org/schema/aop"
      xmlns:tx="http://www.springframework.org/schema/tx"
      xmlns:context="http://www.springframework.org/schema/context"
      xmlns:p="http://www.springframework.org/schema/p"
      xsi:schemaLocation="
      http://www.springframework.org/schema/beans                    http://www.springframe-
work.org/schema/beans/spring-beans-4.3.xsd
      http://www.springframework.org/schema/aop                      http://www.springframe-
work.org/schema/aop/spring-aop-4.3.xsd
      http://www.springframework.org/schema/tx                       http://www.springframe-
work.org/schema/tx/spring-tx-4.3.xsd
      http://www.springframework.org/schema/context                  http://www.springframe-
work.org/schema/context/spring-context-4.3.xsd">
   <bean id="urlMapping" class="org.springframework.web.servlet.handler.SimpleUrlHandler-
Mapping">
       <property name="mappings">
          <props>
             <prop key="index.htm">indexController</prop>
             <prop key="evaluation.htm">evaluationRequestController</prop>
             <prop key="mob.htm">mobController</prop>
             <prop key="loottable.htm">lootTableController</prop>
          </props>
       </property>
   </bean>
   <bean id="viewResolver"
        class="org.springframework.web.servlet.view.InternalResourceViewResolver"
        p:prefix="/WEB-INF/jsp/"
        p:suffix=".jsp" />
   <bean name="indexController"
```

```
        class="org.springframework.web.servlet.mvc.ParameterizableViewController"
        p:viewName="index" />
    <bean name="evaluationRequestController" class="fi.metropolia.ristotu.controller.Evaluation-
RequestController"/>
    <bean name="mobController" class="fi.metropolia.ristotu.controller.MobController"/>
    <bean name="lootTableController" class="fi.metropolia.ristotu.controller.LootTableControl-
ler"/>
</beans>
```

XML:n validointi schema-tiedosto xmlValidation.xsd

Tiedostolla varmistetaan XML-tiedon oikeellinen muoto, jotta tieto voidaan viedä tietokantaan.

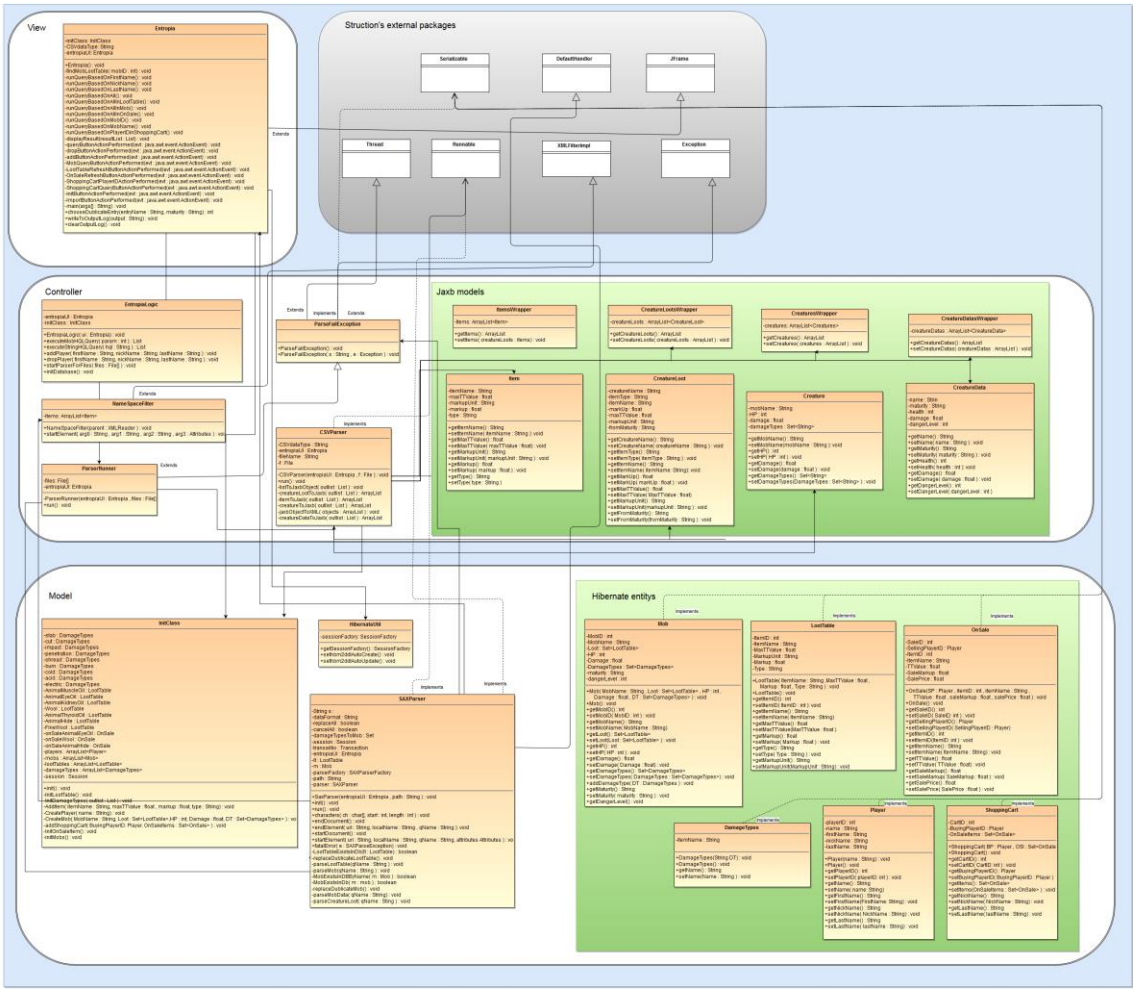
```
<?xml version="1.0" encoding = "UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
  <!-- <xs:element name="Items"/> -->

<!-- Items -->
   <xs:element name="Items" type = "ItemListType"/>
   <xs:complexType name="ItemListType">
      <xs:sequence>
         <xs:element name = "Item" type= "ItemType" maxOccurs="unbounded"/>
      </xs:sequence>
   </xs:complexType>
   <xs:complexType name = "ItemType">
      <xs:sequence>
         <xs:element name ="itemName" type = "xs:string" maxOccurs="1" minOccurs="0"/>
         <xs:element name ="type" type = "xs:string" maxOccurs="1" minOccurs="0"/>
         <xs:element name ="maxTTValue" type = "xs:string" maxOccurs="1" minOccurs="0"/>
         <xs:element name ="markup" type = "xs:string" maxOccurs="1" minOccurs="0"/>
         <xs:element name ="markupUnit" type = "xs:string" maxOccurs="1" minOccurs="0"/>
      </xs:sequence>
   </xs:complexType>

   <!-- CreatureDatas -->
   <xs:element name="CreatureDatas" type = "CreatureDataListType"/>
   <xs:complexType name="CreatureDataListType">
      <xs:sequence>
         <xs:element name = "CreatureData" type= "CreatureDataType" maxOccurs="un-
bounded"/>
      </xs:sequence>
   </xs:complexType>
   <xs:complexType name = "CreatureDataType">
      <xs:sequence>
         <xs:element name ="name" type = "xs:string" maxOccurs="1" minOccurs="0"/>
```

```
            <xs:element name ="maturity" type = "xs:string" maxOccurs="1" minOccurs="0"/>
            <xs:element name ="health" type = "xs:string" maxOccurs="1" minOccurs="0"/>
            <xs:element name ="damage" type = "xs:string" maxOccurs="1" minOccurs="0"/>
            <xs:element name ="dangerLevel" type = "xs:string" maxOccurs="1" minOccurs="0"/>
        </xs:sequence>
    </xs:complexType>

    <!-- Creatures -->
    <xs:element name="Creatures" type = "CreaturesListType"/>
    <xs:complexType name="CreaturesListType">
        <xs:sequence>
            <xs:element name = "Creature" type= "CreatureType" maxOccurs="unbounded"/>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name = "CreatureType">
        <xs:sequence>
            <xs:element name ="mobName" type = "xs:string" maxOccurs="1" minOccurs="0"/>
            <xs:element name ="HP" type = "xs:string" maxOccurs="1" minOccurs="0"/>
            <xs:element name ="damage" type = "xs:string" maxOccurs="1" minOccurs="0"/>
            <xs:element name ="damageTypes" type = "DamageTypesList" maxOccurs="1" minOccurs="0"/>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name = "DamageTypesList">
        <xs:sequence>
            <xs:element name ="damageType" type = "xs:string" minOccurs="0" maxOccurs="unbounded"/>
        </xs:sequence>
    </xs:complexType>

    <!-- CreatureLoots -->
    <xs:element name="CreatureLoots" type = "CreatureLootsListType"/>
    <xs:complexType name="CreatureLootsListType">
        <xs:sequence>
            <xs:element name = "CreatureLoot" type= "CreatureLootType" maxOccurs="unbounded"/>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name = "CreatureLootType">
```

```
        <xs:sequence>
            <xs:element name ="creatureName" type = "xs:string" maxOccurs="1" minOccurs="0"/>
            <xs:element name ="itemType" type = "xs:string" maxOccurs="1" minOccurs="0"/>
            <xs:element name ="itemName" type = "xs:string" maxOccurs="1" minOccurs="0"/>
            <xs:element name ="fromMaturity" type = "xs:string" maxOccurs="1" minOccurs="0"/>
            <xs:element name ="maxTTValue" type = "xs:string" maxOccurs="1" minOccurs="0"/>
            <xs:element name ="markUp" type = "xs:string" maxOccurs="1" minOccurs="0"/>
            <xs:element name ="markupUnit" type = "xs:string" maxOccurs="1" minOccurs="0"/>
        </xs:sequence>
    </xs:complexType>
</xs:schema>
```

Admin-käyttöliittymäsovelluksen luokkakaavio

Kuva sisältää tarkemman kuvauksen luokkien sisällöstä sovelluksessa.



Verkkosovelluksen luokkakaavio

Kuva sisältää tarkemman kuvauksen luokkien sisällöstä sovelluksessa.