

OTP2-Raportti

Ryhmä 10: Jarno Sundström, Lauri Järvisalo, Joni Kokko, Mikael Engström

Tehtävä: League of legends tilastointi ohjelma

1. Asiakasvaatimukset

Asiakas asetti ryhmälle tehtäväksi tuottaa League of legends-pelin pelaajia avustavan ohjelman. Pelin päätyttyä on pelaajan pystyttävä näkemään käyttöliittymän kautta tietokantaan tallennetut tiedot kyseisestä pelistä, jonka jälkeen ohjelma käsittelee ne seuraavalla tavalla:

- Osa datasta menee tietokantaan laskurin kautta, jossa ohjelma laskee esimerkiksi paljonko kultaa, pelaaja on tienannut minuutissa
- Ohjelma osaa antaa pelaajalle rakentavaa palautetta jokaisesta pelatusta pelistä asianmukaisella tavalla
- Ohjelma tarjoaa pelaajalle mahdollisuuden tutkia pelihistoriaansa ja verrata omia tilastojaan esimerkiksi parempia pelaajia vastaan
- Ohjelman on pystyttävä noutamaan asiakkaan haluamia tietoja tietokannasta
- Ohjelma tulee lokalisoida jotta muita kieliä kuin englantia pääkielenään puhuvat ihmiset voivat käyttää ohjelmaa helposti
- Ohjelman tulee pystyä hakemaan tietoa Riot games:in tarjoamasta avoimesta rajapinnasta

Asiakas haluaa testattavissa olevan helppokäyttöisen käyttöliittymän, jonka kautta hän pystyy näkemään pelaamansa pelien tiedot pelattavista peleistä tietokantaan tallennettujen tietojen perusteella.

2 Suunnittelu

2.1 Tietokanta

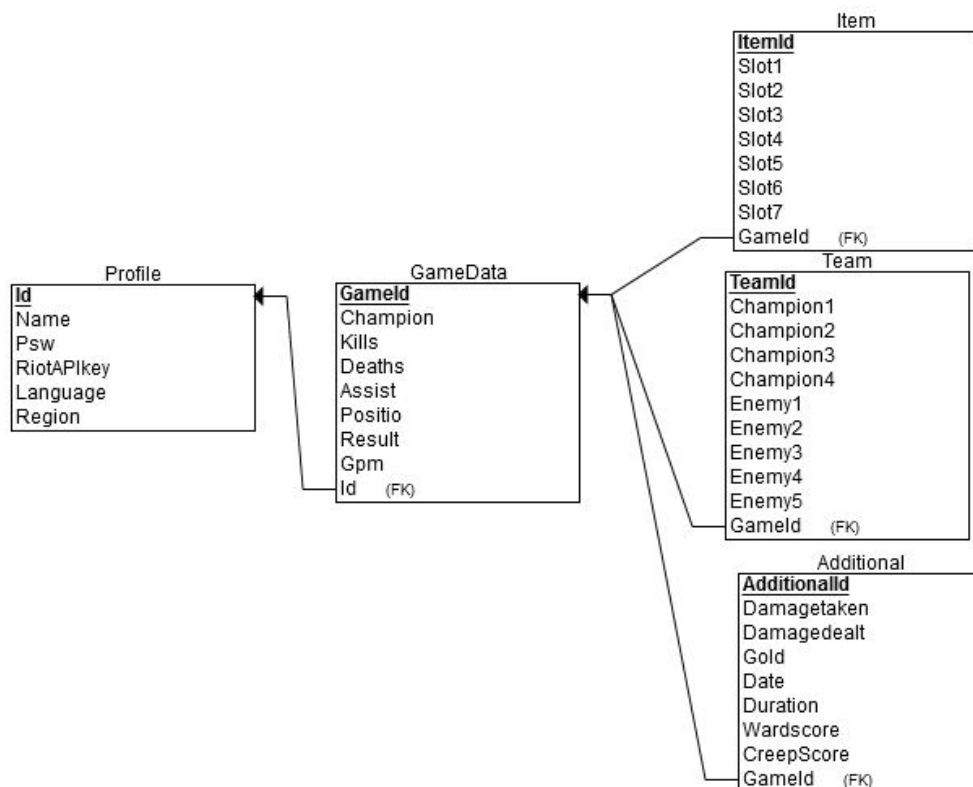
Ryhmä päätös kurssin OTP2 alussa ottaa käyttöönsä Riot gamesin tarjoaman avoimen rajapinnan (Riot API) tämän seurauksena vanha tietokantapohja oli riittämätön palvelemaan projektin tarpeita, sillä tiedon määrä kasvoi huomattavasti. Ryhmä päätti pitää oman tietokannan mukana sovelluksessa, koska se oli tehtävänannon mukaista ja toiseksi Riot API:n tutkimisen ja testien yhteydessä todettiin sen antavan tietoa suhteellisen hitaasti ulos, varsinkin kun suuria määriä pelejä haettiin. Ryhmä katsoi siis oman tietokannan olevan parempi ratkaisu kerran haetun tiedon palauttamiseen kunhan se on tallennettu tietokantaan. Edellisen jakson pohjana oli kaksi taulua sisältävä tietokanta, mistä toinen taulu palveli sisään kirjautumista ja toinen itse pelitietoja.

Ryhmä päätyi toteuttamaan uuden viisi taulua sisältävän tietokannan, josta SoftwareProfile ja Profile taulut jätettiin edelleen palvelemaan sisään kirjautumista. Tauluun lisättiin muutama uusi rivi API:n vaatimuksista johtuen. Uudesta rivistä esimerkkinä API-key, johon sidottiin käytettävä pääsyavain Riot API:n pääsemiseksi .

Ryhmä päätti jakaa pelitiedot neljä eri taulun välille. Nämä taulut ovat Gamedata, Item, Team ja Additional. Gamedata pitää edelleen sisällään perustiedot yhdestä pelistä, eli pelaajan saavuttamat tapot, kuolemat, avustukset, yms. Samalla tästä tehtiin ikään kuin yhdystaulu Profiiliin ja kaiken pelitiedon välille. Täten Gamedata sisältää ManytoOne liitoksen Profiilitauluun. Muut pelitietotaulut ovat OnetoOne liitoksella liitetty Gamedata-tiluun.

Pelitietoihin liittyvä seuraava taulu on Item-tilu. Tähän tauluun on tarkoituksena varastoida yksittäisen pelin kaikki pelaajan lopussa hallussa olleet tavarat. Ryhmän ajatuksena on, että tämä on toinen tietotaulu, joka tuodaan Gamedatan lisäksi näkyviin päänäkyvään.

Kaksi viimeistä taulua ovat Additional- ja Team-tilut. Näistä Team-tiluun on tarkoituksena merkata muut sankarit, pelaajan sankaria lukuun ottamatta, jotka olivat mukana kyseisessä ottelussa. Additional-tilu pitää sisällään enemmän pelaajan toimiiin liittyvään tarkempaa informaatiota, kuten pelaajan ansaitsema kulta, creep score, ward score, yms. pelin aikana. Additional-tilun tietoja on tarkoituksena käyttää pääasiassa analysoinnin yhteydessä. Lisäksi näiden kahden edellä mainitun taulun tiedot palvelee lisätieto-osiota peliä kohden meidän ohjelmassa. Kuvasta 1 selviää ryhmän tietokanta rakenne.



Kuva 1.

2.2 Käyttöliittymä

Käyttöliittymän suunnittelussa ryhmä halusi toteuttaa mahdollisimman helppokäyttöisen ja yksinkertaisen kokonaisuuden. Käyttäjän annetaan syöttää tietoa ohjelmaan vaiheittain ja monta eri ikkunaa hyödyntäen. Tällä tavalla uudet käyttäjät eivät kokisi uusille käyttäjille tyypillisiä ongelmia liian monimutkaisesta työympäristöstä johtuen.

Käyttöliittymän asiakasystävällisyyttä päätettiin viedä pidemmälle tarjoamalla käyttäjälle mahdollisuuden esimerkiksi varoitusteksteillä virheiden ennaltaehkäisemiseksi, sekä tekstikenttien pyyhkiminen napin painalluksella vaivattomamman käytön mahdollistamiseksi.

Myös värimaailman ja tyylin suunnittelua aloitettiin ja päätettiin, että käyttöliittymän tulisi olla mahdollisimman yksinkertainen. Yksinkertaisella värimaailmalla ja tyyllillä käyttöliittymä ei rasita käyttäjän silmiä ja tekee käyttäjäkokemuksesta miellyttävämmän.

OTP2 kurssilla ryhmä otti käyttöön Riotin tarjoaman avoimen rajapinnan jonka kautta tiedot saadaan pelien päätyttyä tallennettua tietokantaan, tämän seurauksena käyttöliittymään oli tehtävä muutoksia sillä käyttäjän syöttämät tiedot olivat tästä eteenpäin turhia. Käyttöliittymän vaatimuksiin lisättiin lokalisointi joka otettiin huomioon käyttöliittymän suunnittelussa.

3 Toteutus

3.1 Tietokanta

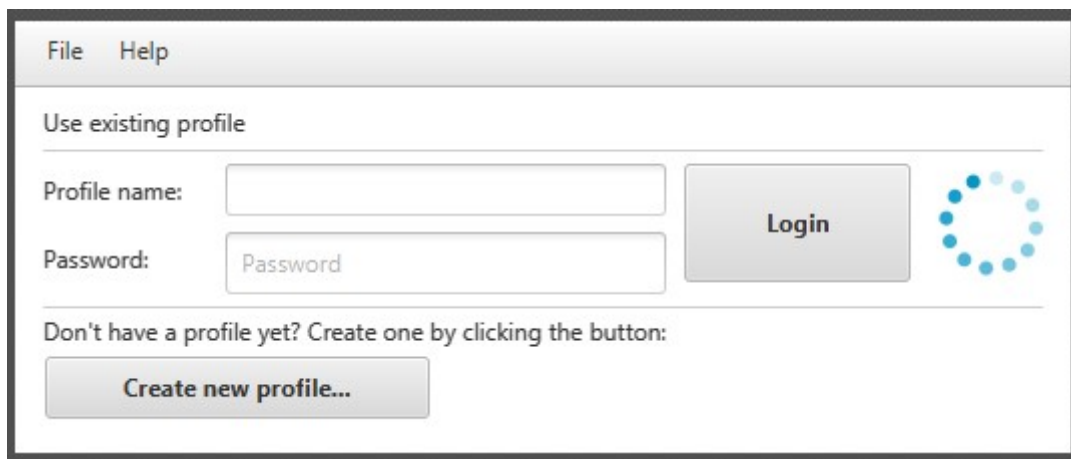
Ryhmä toteutti edellisellä periodilla tietokannan eCloud koneelle. Tätä samaa tietokantaa ryhmä käyttää tämän projektin aikana. Ryhmä käytti tietokannan osalta Hibernatea, eli tiedot tauluista ja niiden rakenteesta löytyy Model pakkauksesta omina luokkina. Jokaisen luokan kohdalla ryhmä määritteli sen tietokanta suunnitelman mukaiseksi. Edellisessä periodissa ryhmällä oli ongelmia saada linkitykset taulujen välillä toimimaan, mutta nyt linkitykset saatiin toimimaan taulujen välillä. Ryhmä antoi Hibernaten koota taulut ja niiden liitokset Hibernaten configuraatio XML:än olevan create ominaisuuden avulla. Lisäksi taulujen omat id arvojen luonti säilytettiin auto generoituna, jotta sekaannuksia ei tulisi.

Taulujen välisistä mappauksista huomioitavaa on se että Gamedatan ja SoftwareProfilen välisestä ManytoOne linkityksestä huolehtii Gamedata. Kaikki OnetoOne linkitykset mappaa aina Gamedatan viereinen taulu.

Loppujen lopuksi ryhmä onnistui toteuttamaan suunnitellun mukaisen tietokantapohjan. Eli viisi taulua sisältävän tosiinsa linkitetyn taulurivistön. Lisäksi koska käytämme Hibernatea työssä on luontevaa todeta, että tietokantametodit lisäys- ja hakumetodit toteuttavat osaltaan sql:ää. Ryhmä oli viime periodilla toteuttanut Model pakkauksen rajapinnan ja jatkoimme sen käyttämistä tällä toisella periodilla. Alkuperäinen suunnitelma oli DAO-mallin mukainen toteutus.

3.2 Käyttöliittymä

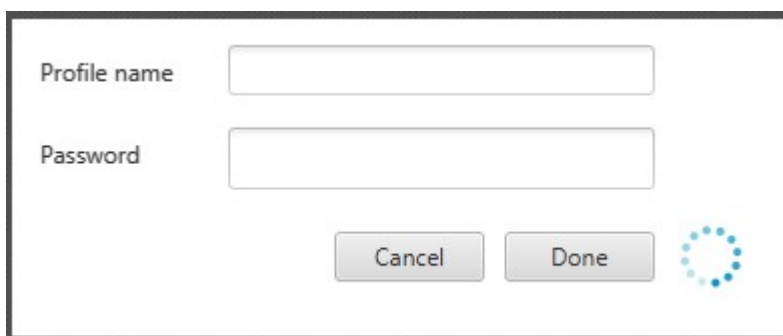
Ryhmä päätti toteuttaa käyttää käyttöliittymän rakentamiseen Scenebuilderiä. Ryhmä oli suunnitteluvaiheessa päättänyt tarvittavien ikkunoiden määrän (5) sekä niiden eri toiminnot. Ohjelman käynnistyessä ensimmäinen ikkuna on sisäänkirjautumista varten kuvassa 2.



Kuva 2.

Kuten kuva 2:stä ilmenee, käyttäjälle tarjotaan mahdollisuus kirjautua suoraan sisään tai luoda uusi käyttäjätili.

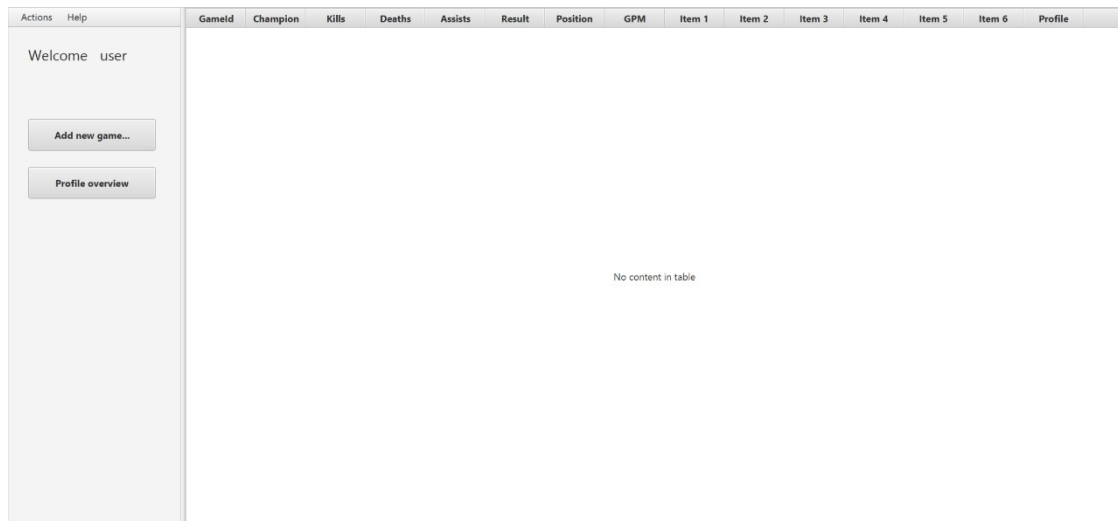
Käyttäjätilin luontia varten ryhmä toteutti kuvassa 3 näkyvän ikkunan.



Kuva 3.

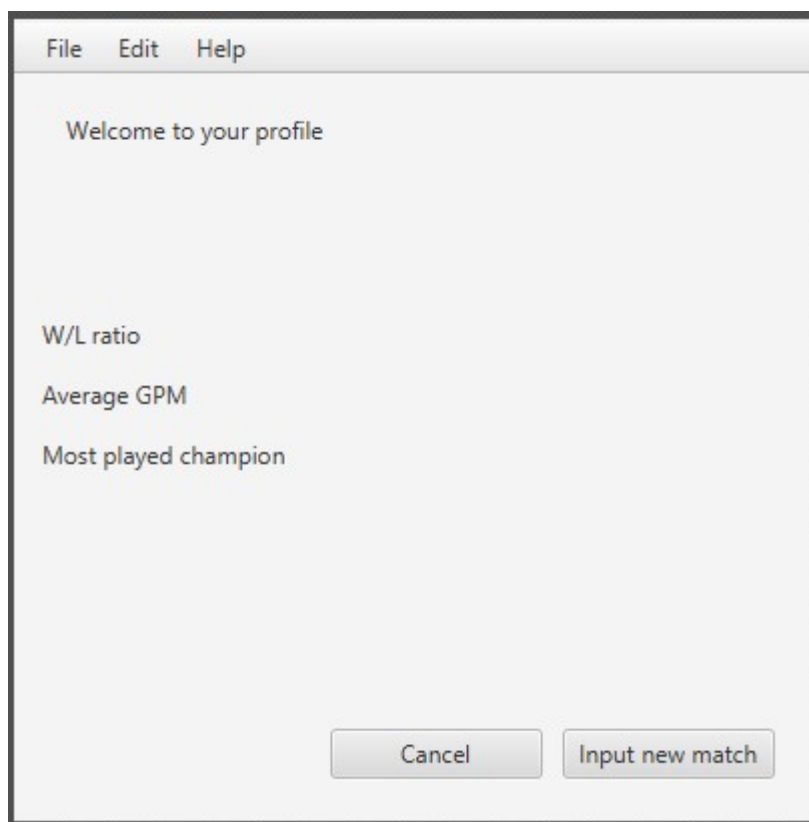
Kuvassa 2 käyttäjän luonti-ikkuna.

Sisäänkirjautumisen jälkeen käyttäjälle tulee näkyviin keskusikkuna, josta käyttäjä voi nähdä pelihistoriansa ja halutessaan lisätä uusia pelaamiansa pelejä tietokantaan tai vain tarkastella profiilinsa yksityiskohtia. Keskusikkuna näkyvissä kuvassa 4.



Kuva 4.

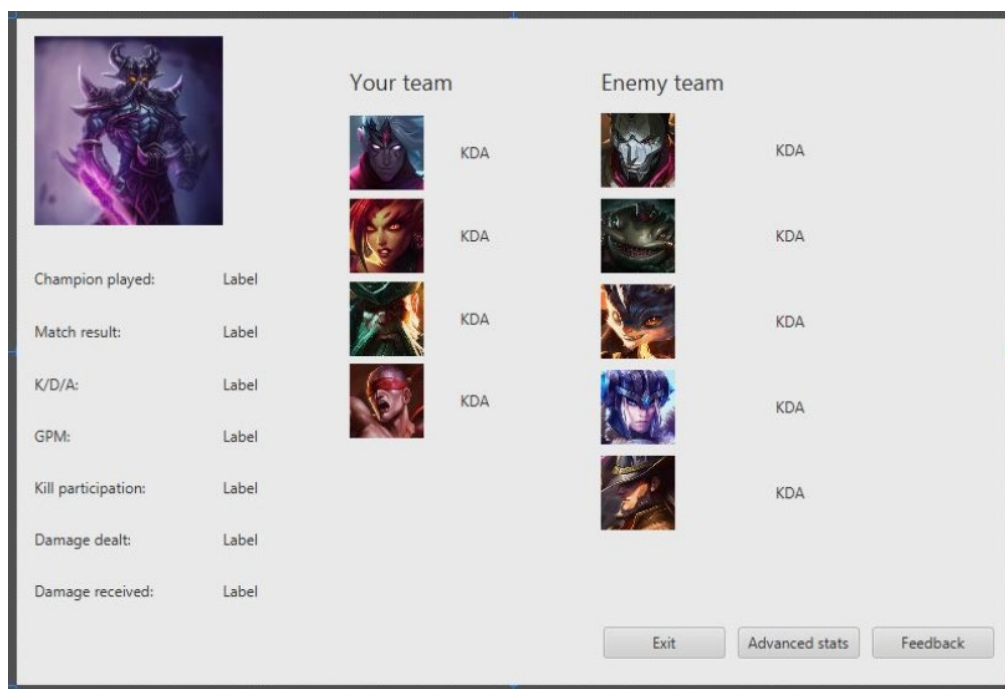
Painamalla 'Profile overview' -painiketta, pystyy käyttäjä näkemään oman profiilinsa, jossa tulee näkyä käyttäjälle tärkeää dataa peleihin liittyen. Kuvassa 5 profiili-ikkuna.



Kuva 5.

Kuvassa 5 käyttäjän oma profiili josta selviää W/L ratio (voitot / tappiot), average GPS (Paljonko kultaa tienattu per minuutti) sekä käyttäjän eniten pelaama hahmo.

Seuraavaksi ryhmä suunnitteli ikkunan josta käyttäjä näkisi yksittäisen pelin yleiset tiedot tulostettuna. Ikkunasta ilmenee pelissä pelattu hahmo, pelin lopputulos, pelaajan KDA (Tapot ja avustukset tappoihin jaettuna kuolemilla), pelaajan GPM (gold per minute eli paljonko kultaa pelaaja on tienannut per minuutti), CS score (Paljonko pelaaja on saanut farmattua kultaa antavia hahmoja), Wards placed (Näkyvyyttä antavien esineiden käyttö), Damage dealt (Paljonko vahinkoa pelaaja on tehnyt vastustajiin) sekä pelaajan rank (Tällä mitataan pelaajan tasoa muihin samaa peliä pelaavien pelaajien tasoon nähden). Kuvassa 6 näkyy kyseinen toteutus mallinnettuna.



Kuva 6.

Kuten kuvasta 6 ilmenee, käyttäjälle tarjotaan yksinkertainen näkymä, josta ilmenee kaikki oleelliset tiedot valitun pelin osalta.

Viimeisenä tarvittiin enää palaute ikkuna, josta käyttäjä saa heti palautetta pelatusta ottelusta datan analysoinnin jälkeen, esimerkiksi jos käyttäjä on kuollut liian monta kertaa pelissä sen pituuteen nähden, niin käyttöliittymä mainitsee asiasta palautteessa ja kertoo myös muun olennaisen datan laskutoimituksen jälkeen. Kuvassa 7 ryhmän toteutus palaute ikkunasta.

We have analyzed your in-game stats and have come up with the following conclusions

About KDA:

About warding:

About farming:

Score:

Kuva 7.

Kuten kuvasta 7 ilmenee, käyttöliittymä antaa palautteen teksti-ikkunoissa sekä lopuksi antaa käyttäjälle pisteet kyseisestä pelistä analytiikka työkalun avulla (0 matalin arvosana 10 korkein, tämän lisäksi tekstimuodossa miten pelaaja voi parantaa pelaamistaan jatkossa analyysin perusteella). Mukana navigointi takaisin profiiliin tai edelliseen ikkunaan painikkeiden kautta.

4 Analytiikka työkalu

Ohjelma analysoi pelaajan tiedot tietokannasta saamien tietojen perusteella seuraavasti: FeedBack-luokassa kaikki oleelliset tiedot yksittäisestä pelistä viedään luokan konstruktoriin, jonka jälkeen konstruktoria lasketaan pelaaja KDA (Tapot ja avustukset tappoihin jaettuna kuolemilla), GPM (kultaa tienattu minuutissa) sekä CS per/min (Kultaa antavien hahmojen tappojen määrä). Tietysti myös tärkeät yksittäiset tiedot pelistä haetaan tietokannasta kuten tapot, kuolemat ja ostetut wardit (näkyvyyttä antavat esineet).

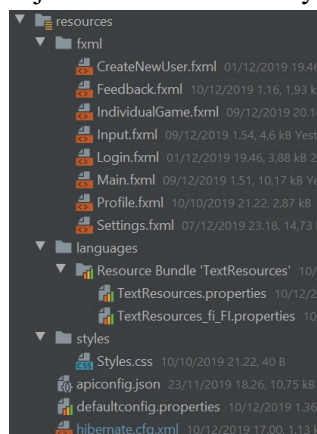
Tiedonhaun jälkeen kehiteltiin menetelmät joiden avulla yksittäistä peliä analysoidisiin kurssin rajoitteiden sallimissa rajoissa. Tällä kertaa päätimme että peliä analysoidaan peliin liittyvien perusasioiden pohjalta ja analyysi onkin tarkoitettu lähinnä aloittelijoita ja vähän edistyneempiä pelaajia varten. Esimerkkejä oleellisista metodeista käyttäjän näkökulmasta ovat `getWardFeedback()` sekä `getCSFeedback()`- metodit joiden avulla saadaan sekä palautetta käyttäjälle näistä yksittäisistä pelin osa-alueista sekä jaetaan yksittäisen pelin pisteytys sen mukaan kuinka hyvin pelaaja on nämä perusasiat hallinnut kyseisessä pelissä.

Luokan muut metodit ovat yksittäisten arvojen saamista varten tai niiden String muotoon muuttamista varten sillä useat arvot tuodaan Riotin avoimen rajapinnan avulla long muodossa. Yhteenvetona analytiikka työkalu siis vertaa pelaajan yksittäisen ottelun suoritusta tiettyihin Riotin sivuilta saatuihin keskiarvoihin ja tekee tämän perusteella johtopäätöksiä siitä mitä pelaajan pitäisi omassa pelissään kehittää tullakseen paremmaksi pelaajaksi.

Lisäksi mainittavaa on ohjelman pisteytys systeemi joka analysoi jo valmiiksi tehtyä analyysiä ja antaa pelaajalle pisteet pelistä sekä yleiskäsityksen pelatun pelin suorituksesta.

5 Lokalisaatio

Lokalisaatio luotiin kahdessa eri osassa, käyttäen kuitenkin samoja resurssitiedostoja. Ohjelmistomme oletuskieleksi valittiin englanti ja toiseksi suomi. Ensimmäinen osa oli käyttöliittymä ennen sisäänkirjautumista ja toinen sisäänkirjautumisen jälkeen. Tämä tehtiin, koska ohjelmiston profiileille pystyi valitsemaan oman oletuskielen, joka vaihdettiin mikäli se erosi sisäänkirjautumista edeltävästä oletuskielestä. Tällä tavoin tehtiin mahdolliseksi käyttää ohjelmaa monikielisessä ympäristössä. Kuvassa 8 ryhmän luokkarakennetta.



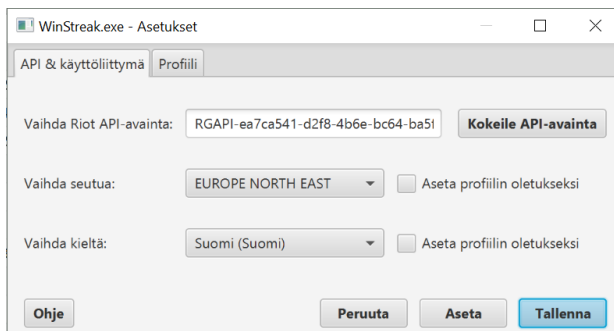
Kuva 8.

Yllä oleva kuva hahmottaa koodipuolen- ja alta löytyvät käyttäjäpuolen toteutusta. Oletuskieli luetaan defaultconfig.properties-tiedostosta ja käyttäjän kieli taasen tietokannassa olevasta defaultLanguage-kolumnista. Molemmat toteutukset käyttävät languages-hakemistossa sijaitsevia TextResources-tiedostoja. Kielivalikot luodaan ohjelmassa dynaamisesti, joten uusien kielten lisääminen vaatii ainoastaan kielitiedoston, joka on muotoa TextResources_<kieli>_<MAA>.properties.

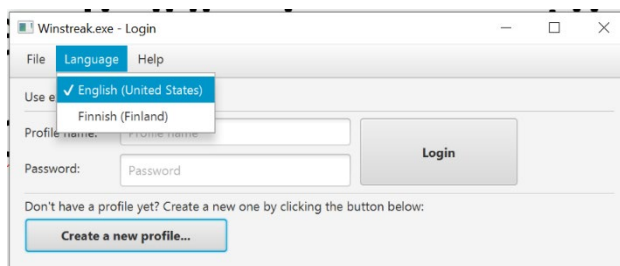
Toteutuksessa oli jonkin verran haastavuutta, erityisesti Sisäänkirjautumisikkunassa, jossa valikkotyypin takia ohjelman tuli myös osata ruksata oikea kieli ja poistaa ruksi toisesta, ottaen samalla huomioon kielten helpon lisäyksen ohjelmaan.

Kielten vaihto ohjelman ajon aikana onnistuu kätevästi ja tämä vaihtaa aivan kaiken toiselle kielelle, mukaan lukien kielivalikot ja kaikki muut taustalla pyörivät ikkunat. Kielen vaihto kesken operaation, esim. pelien haun aikana, vaihtaa myös kielen pelkästään operaation aikana näkyvissä teksteissä odotetulla tavalla.

Kuvissa 9 ja 10 ilmenee ryhmän kielivalikko.



Kuva 9



Kuva 10

6 Riot API

Ryhmä käytti pelaajan ottelutietojen haussa Riot games:in tarjoamaa avointa rajapintaa. Ryhmä sai käyttöönsä henkilökohtaisen API-avaimen, joka täytyy uusua vuorokauden välein ja jonka pyyntökapasiteetti on vain 100 pyyntöä kahdessa minuutissa. Suuria ottelumääriä haettaessa pyyntökapasiteetti täyttyi nopeasti, joten ryhmä rajoitti pyyntötiheyttä ohjelmakoodiin lisätyillä `sleep()` -metodeilla

Ryhmä käytti apunaan Meraki Analyticsin kehittämää Orianna Frameworkia, joka tarjoaa getterit ottelukohtaisille tiedoille, sekä kääntää rajapinnan palauttaman json-muotoisen tiedon String- ja long-muotoon. Rajapinnasta saadut ottelutiedot tallennettiin olioina kaksisuolotteiseen taulukkoon, josta tiedot lopulta vietiin tietokantaan

Tiedonhaku rajapinnasta osoittautui henkilökohtaisen API-avaimen pyyntörajoituksen takia erittäin hitaaksi, mutta ohjelman analyysien tekoon käyttämät tiedot ovat niin spesifejä, että ilman rajapinnan käyttöä ohjelman toiminnotkin olisivat hyvin pelkistettyjä.

7 Lopputulos

Ryhmä sai aikaan toimivan ohjelman, joka suoriutuu asiakkaan vaatimuksista kiitettävästi. Aikarajoitteista johtuen ulkoasu jäi hieman vaatimattomaksi mutta käyttöliittymä on selkeä ja toimii. Lokalisaatio on kunnossa ja tietokanta toimii moitteettomasti. Ryhmä teki myös jUnit testejä

tietokantapuolen operaatiolle, mitkä kaikki menevät läpi. Muutoin ryhmä testannut pitkälti manuaalisesti lopputuloksia, tästä näkyvänä jälkeenä muutamia testaustiedostoja järjestelmän paketeissa.