OULUN SEUDUN
AMMATTIKORKEAKOULU

Jani Alatyppö

PILVIPALVELUN TOTEUTUS

PILVIPALVELUN TOTEUTUS

Jani Alatyppö
Opinnäytetyö
Syksy 2012
Tietotekniikan koulutusohjelma
Oulun seudun ammattikorkeakoulu

TIIVISTELMÄ

Oulun seudun ammattikorkeakoulu
Tietotekniikan koulutusohjelma, ohjelmistokehityksen suuntautumisvaihtoehto

---

Tekijä(t): Jani Alatyppö
Opinnäytetyön nimi: Pilvipalvelun toteutus
Työn ohjaaja(t): Eero Nousiainen
Työn valmistumislukukausi ja -vuosi: Syksy 2012 Sivumäärä: 40

Kuru Digital Creations Oy on uusi oululainen yritys, joka tuottaa tiedonkeruu- ja analysointisovelluksia useille urheilulajeille. Opinnäytetyön tavoitteena oli toteuttaa pilvipalvelu, jonka avulla saadaan ajantasaiset tiedot pelaajista ja joukkueista yrityksen kaikkien sovelluksien käyttöön.

Työn toteutuksessa päädyttiin käyttämään Qt-ohjelmistokehitysympäristöä ja sen mukana tulevaa SQLite-tietokantamoottoria. Näin saatiin rakennettua helposti käyttöönotettava kokonaisuus, joka toimii Windows- ja Linux-käyttöjärjestelmissä suoraan kääntämisen jälkeen.

Tuloksena saatiin toimiva palvelinohjelma ja siihen asiakasohjelma, jolla voidaan testata palvelimen toimivuus. Palvelinohjelmaa on helppo jatkokehittää lisäämällä siihen ominaisuuksia, jos niille on tarvetta.

Asiasanat: Qt, C++, SQL, salaus, TCP, pilvipalvelu

ABSTRACT

Oulu University of Applied Sciences
Information Technology, Software development

Author(s): Jani Alatyppö
Title of thesis: Creating a cloud service
Supervisor(s): Eero Nousiainen
Term and year when the thesis was submitted: Autumn 2012 Pages: 40

Kuru Digital Creations Oy is a new company in Oulu that makes software for gathering and analyzing information about many different sports.

Idea of this thesis was to create a player database that could be used to get information about teams and players to any of the company's program that needs them.

Qt SDK and SQLite were chosen to create this project, because this way it was possible to create a project that is easy to take into use in Windows or Linux straight after building.

Product of this thesis is a working server software and a client software that can be used to test server's functionality. Adding new functionality to server software can be done easily in the future if need be.

Keywords: Qt, C++, SQL, encryption, TCP, cloud service

SISÄLLYS

SANASTO

API – Application Programming Interface, ohjelmointirajapinta.

GUI – Graphical User Interface, käyttöliittymä.

JavaScript – Ohjelmointikieli, laajasti käytössä esim. web-sivuilla.

Qt – Ohjelmistokehitysympäristö.

Sprintti – Parin viikon pituinen jakso Scrum-ohjelmistokehitysmenetelmässä.

SQL – Structured Query Language, kieli jolla voidaan tehdä kyselyjä tietokantaan.

SQLite – Kevyt tietokantamoottori.

TCP – Transmission Control Protocol, yhteydellinen tiedonsiirtoprotokolla.

UDP – User Datagram Protocol, yhteydetön tiedonsiirtoprotokolla.

UTF-16 – Merkistökoodaus, jossa jokainen merkki tallennetaan 2 tavun mittaisena.

1 JOHDANTO

Työn tavoitteena oli toteuttaa pilvipalvelu Kuru Digital Creations Oy:lle. Ohjelman ideana on välittää tiedot pelaajista ja joukkueista helposti kaikille yrityksen ohjelmille. Tämä oli tärkeää, koska yrityksen kaikki sovellukset pitivät yllä omaa tietokantaa pelaajista.

Työn toteuttamisen apuna käytettiin ketterää ohjelmistokehitysmenetelmää nimeltä Scrum. Sen ansioista projekti saatiin jaettua pieniin toteutettaviin osiin, jolloin oli helppo seurata, mitä teki viimeksi ja mitä pitää tehdä seuraavaksi. Työ päätettiin toteuttaa kahdessa sprintissä. Ensimmäisessä suunniteltiin ohjelma ja toisessa toteutettiin.

Suunnitteluvaiheessa tehtiin vaatimuslista, valittiin käytettävät tekniikat, laadittiin luokkakaavio palvelinohjelmasta sekä suunniteltiin tietokanta. Toteutusvaiheessa käytiin Scrum-menetelmällä laadittuja tehtäviä yksi kerrallaan läpi, kunnes kaikki saatiin tehtyä.

2 PILVIPALVELUT

2.1 Pilvipalvelutyypit

Pilvipalvelu on uusi termi, jonka merkitys kasvaa ja muuttuu koko ajan. Jotkut analyytikot ja myyjät määrittelevät pilvipalvelun yksinkertaisesti virtuaaliseksi serveriksi, joka on tarjolla internetin välityksellä. Toiset taas ovat sitä mieltä, että kaikki, mikä on internetissä on, on osa ns. pilveä. (1.)

Pilvipalvelut voidaan jakaa useisiin eri tyyppeihin:

- Web-based cloud services, tarjoaa rajapinnan, jonka kautta ohjelmoijat voivat käyttää toiminnallisuutta internetin yli (2).
- SaaS (Software as a Service), tarjoaa yhden sovelluksen nettiselaimen kautta tuhansille asiakkaille (2). Sähköposti on hyvä esimerkki tästä (3).
- PaaS (Platform as a service), toinen SaaS variaatio, tarjoaa kehitysympäristön, joka pyörii tarjoajan palvelimilla (2).
- Managed services, käyttäjälle näkymätön ohjelma, kuten sähköpostin suodatin (2).
- IaaS (Infastructure as a service), vapaasti käytettävä virtaalinen palvelin, joka on vuokrattu tarjoajalta (3).

2.2 Google-karttojen ohjelmointirajapinta

Google Maps Image API on oiva esimerkki Web-based cloud servicestä. Rajapinta tekee staattisten karttojen upottamisesta sivulle erittäin helppoa ilman JavaScriptiä. Käyttö tapahtuu rakentamalla osoite, jossa kerrotaan palvelimelle tarvittavat tiedot kuvasta ja palvelin rakentaa näistä tiedoista kuvan. Esimerkiksi lisäämällä **<img src="http://maps.googleapis.com/maps/api/staticmap?center=-15.800513,-47.91378&zoom=11&size=200x200&sensor=false">** HTML-sivulle saataisiin kuvan 1 näköinen kartta näkyviin. (4.)

KUVA 1. Esimerkki Google Maps Image API:n tuotoksesta (4)

3 KETTERÄT OHJELMISTOKEHITYSMENETELMÄT

3.1 Ketterät kehitysmenetelmät yleisesti

Ketterä kehitysmenetelmä on yleinen termi, jota käytetään useille ohjelmistokehitysmenetelmille, jotka perustuvat saman prosessin toistamiseen ja toiminnallisuuden lisäämiseen pienissä erissä. Näistä kuuluisimpia ovat XP (Extreme Programming), Scrum, Crystal, DSDM (Dynamic Systems Development Method), Lean Development ja FDD (Feature-Driven Development). (5.)

Kaikki ketterät kehitysmenetelmät ovat yksilöllisiä tietyssä suuntauksessa, mutta kaikilla on yhteinen visio ja ydinarvot. Nämä ovat listattuna Agile Manifeston sivulla (http://www.agilemanifesto.org/). Ne kaikki sisällyttävät iteraatiot ja jatkuvan palautteen huomioon ottamisen ohjelmiston jalostukseen ja toimitukseen. Niihin kaikkiin sisältyy jatkuvaa suunnittelua, testausta, integrointia ja muita projektin ja ohjelmiston kehitysmuotoja. Ne ovat kaikki myös keveitä verrattuna perinteiseen vesiputousmalliin ja ehkä tärkeimpänä ne keskittyvät tehostamaan ihmisiä olemaan yhteistyössä ja tekemään päätöksiä nopeasti ja tehokkaasti. (5.)

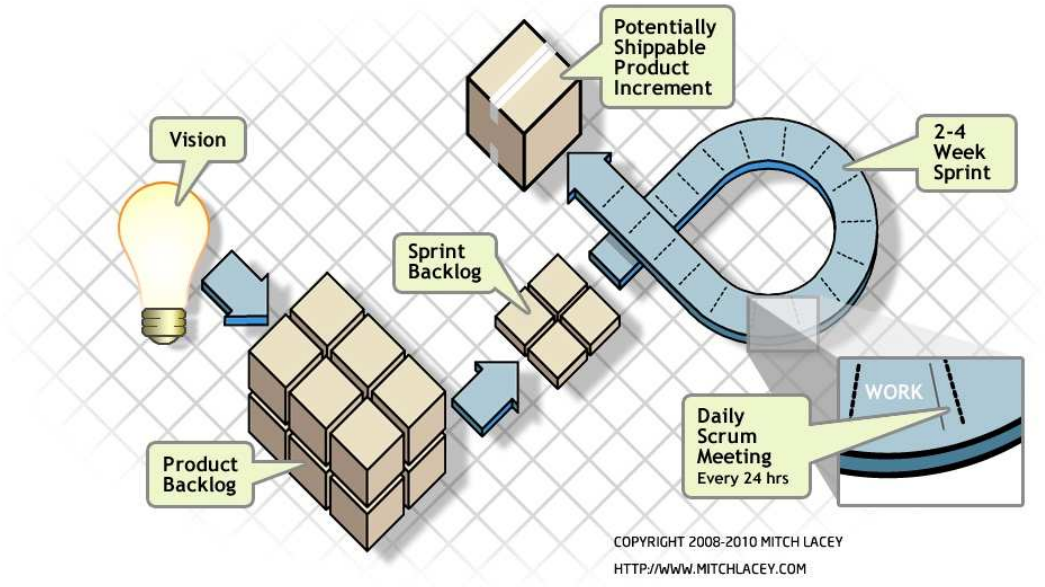
3.2 Scrum-menetelmä

Jeff Sutherland kehitti Scrum-menetelmän vuonna 1993. Nimen hän lainasi Takeuchin ja Nonakan vuonna 1986 tekemästä tutkielmasta, jossa he vertailivat hyvin suoriutuvia työryhmiä scrum-muodostelmaan, jota rugby-joukkueet käyttävät. Tutkielman julkaisi Harvard Business Review. (6.)

Scrum-menetelmän eri vaiheet ovat esiteltynä kuvassa 2 ja ovat seuraavat:

- Tuotteen omistaja (engl. product owner) tekee toivelistan prioriteettien mukaan eli tuotteen kehitysjonon (engl. product backlog, esimerkki kuvassa 3).

- Sprintin suunnittelun aikana työryhmä ottaa pienen palan tästä listasta ja tästä palasta tulee sprintin tehtävälista (engl. sprint backlog). Samalla päätetään, miten nämä palaset toteutetaan.
- Työryhmällä on sprintin pituuden verran aikaa toteuttaa edellisessä vaiheessa päätetyt tehtävät. Sprintin pituus on yleensä 2–4 viikkoa. Päiväpalaverissa (engl. daily Scrum) jaetaan ongelmat ja työn eteneminen työryhmän kanssa.
- Scrum-mestari pitää koko ajan huolen, että työryhmä on keskittynyt tavoitteeseen.
- Sprintin lopussa työn pitäisi olla potentiaalisesti asiakkaan käytettävissä.
- Sprintti lopetetaan esittelemällä tuotteen omistajalle, mitä sprintin aikana on saatu aikaan ja samalla etsitään keinoja parantaa tuotetta ja prosessia.
- Uuden sprintin alkaessa työryhmä valitsee uuden palan tuotteen kehitysjonosta ja aloittaa työn uudestaan. (6.)



KUVA 2. Tapahtumien kulku Scrum-menetelmässä (6)

	Item #	Description	Est	By
Very High				
	1	**Finish database versioning**	16	KH
	2	**Get rid of unneeded shared Java in database**	8	KH
	-	**Add licensing**	-	-
	3	Concurrent user licensing	16	TG
	4	Demo / Eval licensing	16	TG
		Analysis Manager		
	5	File formats we support are out of date	160	TG
	6	Round-trip Analyses	250	MC
High				
	-	**Enforce unique names**	-	-
	7	In main application	24	KH
	8	In import	24	AM
	-	**Admin Program**	-	-
	9	Delete users	4	JM
	-	**Analysis Manager**	-	-
	10	When items are removed from an analysis, they should show up again in the pick list in lower 1/2 of the analysis tab	8	TG
	-	**Query**	-	-
	11	Support for wildcards when searching	16	T&A
	12	Sorting of number attributes to handle negative numbers	16	T&A
	13	Horizontal scrolling	12	T&A
	-	**Population Genetics**	-	-
	14	Frequency Manager	400	T&M
	15	Query Tool	400	T&M
	16	Additional Editors (which ones)	240	T&M
	17	Study Variable Manager	240	T&M
	18	Haplotypes	320	T&M
	19	**Add icons for v1.1 or 2.0**	-	-
	-	**Pedigree Manager**	-	-
	20	Validate Derived kindred	4	KH
Medium				
	-	**Explorer**	-	-
	21	Launch tab synchronization (only show queries/analyses for logged in users)	8	T&A
	22	Delete settings (?)	4	T&A

KUVA 3. Esimerkki tuotteen kehitysjonosta (7)

Kierros toistuu niin kauan, että tuotteen kehitysjono on tyhjä, budjetti on loppunut tai takaraja tulee vastaan. Scrum-menetelmä kuitenkin varmistaa, että suurin osa tärkeimmästä työstä on tehty, kun projekti loppuu. (6.)

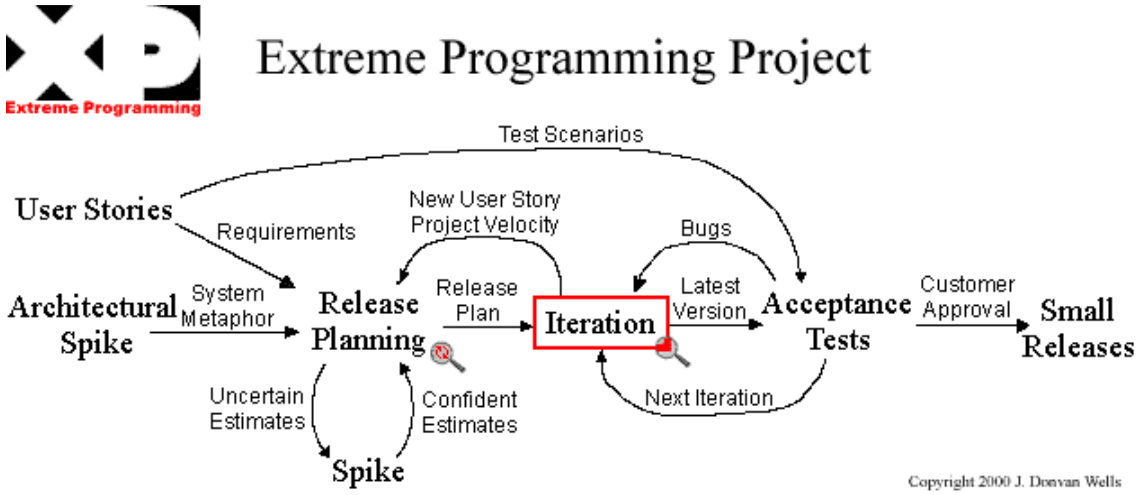
3.3 Extreme Programming -menetelmä

XP oli alun perin Kent Beckin kuvailema kehitysmenetelmä, joka on noussut yhdeksi suosituimmaksi ja kantaanottavimmaksi ketterän kehityksen menetelmäksi. XP on itsekurillinen lähestymistapa, jonka tavoitteena on toimittaa korkealaatuisia ohjelmia nopeasti ja jatkuvasti. Se vaatii asiakkaan läheistä osallistumista, tiheää palautteen vaihtoa, jatkuvaa testausta, jatkuvaa suunnittelua ja tiivistä tiimityötä, jotta voidaan toimittaa toimivia ohjelmia erittäin lyhyillä aikaväleillä, yleensä 1–3 viikon välein. (8.)

Alkuperäinen XP-menetelmä koostuu neljästä yksinkertaisesta arvosta, yksinkertaisuus, kommunikointi, palautteen anto ja rohkeus, sekä kahdestatoista hyväksi todetusta tavasta:

1. Suunnittelu
2. Pienet julkaisut
3. Asiakkaan hyväksyttämistestit
4. Yksinkertainen suunnitelma
5. Pariohjelmointi
6. Testiperusteinen eteneminen
7. Refaktorointi (koodin uudelleenkirjoittaminen)
8. Jatkuva integrointi
9. Kollektiivinen koodin omistus
10. Ohjelmointistandardit
11. Vertauskuva
12. Ylläpidettävä tahti. (8.)

Kuvassa 4 on Don Wellssin havainnollistama kaavio XP-prosessista.



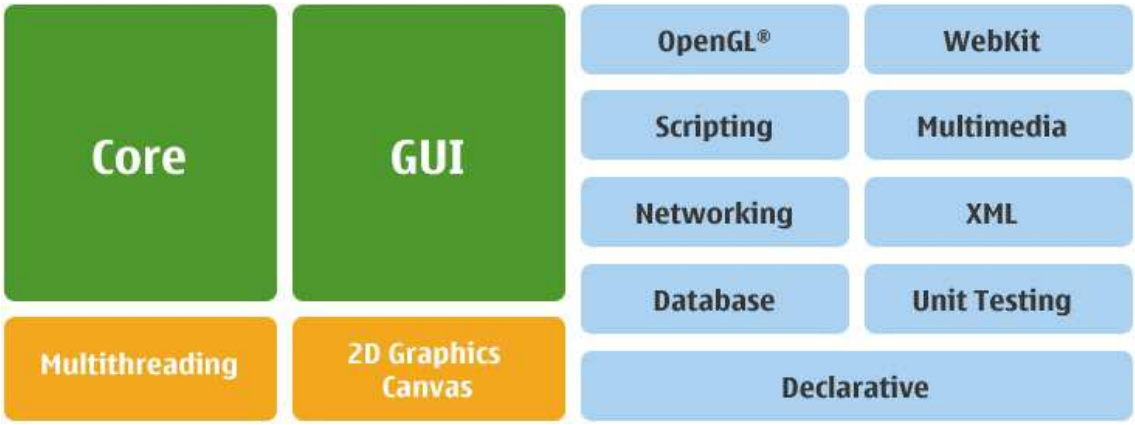
KUVA 4. Extreme Programming -prosessi (9)

4 QT-OHJELMISTOKEHITYSYMPÄRISTÖ

4.1 Yleistä Qt-ohjelmointiympäristöstä

Qt on alustariippumaton ohjelmistokehitysympäristö, joka toimii Windows-, Linux- ja Mac-käyttöjärjestelmissä. Kehityksen aloitti norjalainen firma Trolltech vuonna 1994. Nokia osti Trolltechin vuonna 2008. Nykyään Qt:tä kehitetään avoimena projektina nimellä Qt project ja kehitykseen voi osallistua kuka vain. (10.) Ylläpidosta vastaa lista henkilöitä, joista jokainen on ottanut pienen osan hoidettavaksi. Suurin osa näistä henkilöistä on töissä Nokialla (11).

Qt:n luokkakirjasto koostuu useista moduuleista (kuva 5), jotka yhdessä mahdollistavat monipuolisten ohjelmien toteuttamisen.



KUVA 5. Qt:n luokkakirjastot (12)

4.2 Ydinmoduuli

Ydinmoduuli eli Core sisältää yleiset luokat, joilla ohjelman runko rakennetaan. Se sisältää luokat mm. tiedostojen käsittelyyn, objektien hallintaan, säikeisiin ja signal-slot-kommunikointiin. (12.) Merkittävämpinä näistä voisi mainita QString-, QFile-, QByteArray-, QList- ja QObject-luokat.

QString-luokka pitää sisällään UTF-16 koodattua tekstiä, joten ongelmia esimerkiksi skandinaavisten merkkien kanssa ei juurikaan ole.

QFile-luokkaa käytetään lukemaan ja kirjoittamaan tiedostoja. Myös tässä luokassa on pyritty yhteensopivuuteen eri käyttöjärjestelmien välillä. Esimerkiksi polkujen kirjoittamisessa molemmat kauttaviivat toimivat samalla tavalla.

QByteArray-luokka on kuin taulukko, jonka kokorajoituksista ei tarvitse itse huolehtia. Tätä luokkaa voi käyttää lähes aina, kun käsittelee rakaa dataa. Esimerkiksi QFile-luokka voi lukea tiedoston suoraan QByteArrayksi ja tästä voidaan tehdä vaikkapa QString-olio erittäin helposti. QString-olioon voidaan lukea myös suoraan tiedostosta, mutta tämä vaatii uusien luokkien opettelun, eikä siihen yleensä ole tarvetta, koska QString- ja QByteArray-luokat tarjoavat jo tarpeeksi joustavan ratkaisun kahdestaan.

QList-luokka on Qt:n eniten käytetty yleinen säiliöluokka (engl. container class), joka sisältää automaattisesti koostaan huolehtivan listan, johon voi tallettaa minkä tahansa tyyppistä dataa. Datan tyyppi pitää kuitenkin päättää QList-olion määrittelyn yhteydessä. Sekalaisen tyyppistä dataa on myös mahdollista tallentaa esimerkiksi kierrättämällä se void-osoittimen tai QVariant-luokan kautta, mutta sille ei pitäisi olla tarvetta.

QObject-luokkaa käytetään signal-slot-kommunikoinnissa. Tällä mekaniikalla voidaan määritellä QObject-aliluokalle signaaleja, jotka taas voidaan yhdistää minkä tahansa QObject-aliluokan slot-funktioon.

4.3 Tietoliikennemoduuli

Tietoliikennemoduuli eli Network sisältää kaiken tarpeellisen tiedonsiirtoon TCP- ja UDP-yhteyksien avulla ja ylemmän tason FTP-, HTTP- ja DNS-protokolliin on myös tuki. (12.)

Yleisessä käytössä tärkeimmät luokat ovat QTcpSocket, QTcpServer, QUdpSocket ja QHostAddress. Näillä saadaan hoidettua lähes kaikki kommunikointi internetin välityksellä hyvin yksinkertaisesti.

QHostAddress-luokkaa käytetään abstractoimaan IP-osoitteet, koska Qt tukee IPv4 ja IPv6 protokollia.

4.4 Graafinen käyttöliittymämoduuli

Graafinen käyttöliittymämoduuli eli GUI tarjoaa toiminnallisuuden monipuolisten graafisten käyttöliittymien rakentamiseen. Se hyödyntää järjestelmän paikallisia grafiikkaominaisuuksia ja täten ottaa täyden hyödyn järjestelmän resursseista. (12.)

Kaikki moduulin tarjoamat valmiit käyttöliittymäkomponentit perivät QWidget-luokan, joka puolestaan perii QObject-luokan, joten kaikki käyttöliittymäkomponentit voivat lähettää ja vastaanottaa signaaleja.

Monimutkaiset käyttöliittymäikkunat saadaan rakennettua lisäämällä lapsi-ikkunoita (engl. child window) isä-ikkunaan (engl. parent window). Tähän on erittäin monipuolisesti valmiita luokkia, esimerkiksi QLabel, QLineEdit, QTextEdit, QTabWidget, QPushButton, QListWidget. Luokkien nimistä on helppo päätellä mitä niillä voi tehdä. Ikkunoiden koon muuttumiseen on myös helppo ratkaisu. Esimerkiksi jos käyttöliittymä rakennetaan QGridLayout-luokan sisälle, jolloin ikkunassa olevat komponentit liikkuvat itsestään suhteessa samalle kohdalle ja vaihtavat kokoa suhteessa alkuperäiseen kokoon ikkunan koon vaihtuessa.

4.5 Tietokantamoduuli

Tietokantamoduuli eli Database tarjoaa tuen kaikkiin yleisiin SQL-tietokantapalvelimiin samoja luokkia käyttämällä (12). Näistä luokista näkyvimmät ovat QSqlDatabase ja QSqlQuery, joilla otetaan yhteys tietokanta-palvelimeen ja suoritetaan kyselyjä.

Tietokannan ensimmäisen avauksen yhteydessä täytyy määritellä tietokannan tyyppi ja yhteydelle nimi, millä siihen viitataan jatkossa (kuva 6). Lisäksi tyypistä riippuen täytyy antaa joko tiedoston nimi tai palvelimen osoite ja kirjautumistiedot.

```
QSqlDatabase db = QSqlDatabase::addDatabase("QSQLITE", "yhteyden_nimi");
```

KUVA 6. Tietokannan avaus

Kuvassa 7 on esimerkki miten tietokantaa voidaan käyttää avauksen jälkeen.

```
QSqlDatabase db = QSqlDatabase::database("yhteyden_nimi");
QSqlQuery q(db);
q.exec("SELECT tieto1 FROM taulu1;");
while( q.next() )
{
    QString s_tieto = q.value(0).toString();
    kasittele(s_tieto);
}
```

KUVA 7. Tietokannan käyttö

4.6 Dynaaminen käyttöliittymämoduuli

Dynaaminen käyttöliittymämoduuli eli Declarative mahdollistaa dynaamisten ja kustomoitujen käyttöliittymien rakentamisen, sekä niiden animoinnin erittäin nopeasti JavaScript-kielellä. Esimerkiksi kaksi eriväristä suorakulmiota saataisiin ikkunaan kuvan 8 esimerkin mukaisesti.

```
1   import QtQuick 1.0
2
3   Rectangle {
4       id: root
5       width: 200
6       height: 200
7       color: "red"
8
9       signal mySignal(int value_out)
10
11      function myFunc(value_in)
12      {
13          console.log("MyFunc: " + value_in)
14          root.mySignal(value_in+1)
15      }
16
17      Rectangle {
18          anchors.left: root.right
19          width: root.width
20          height: root.height
21          color: "blue"
22      }
23  }
24
```

KUVA 8. Kahden suorakulmion piirtäminen ikkunaan

Yllä olevassa esimerkissä määritellään lisäksi ikkunalle signaali ja funktio, joita voidaan käyttää myös C++ koodin puolelta.

QML tarjoaa oletuksena vain perusosat käyttöliittymien tekoon, joten monimutkaisemmat täytyy rakentaa itse osa kerrallaan. Kaikki rakennetut osat ovat erittäin helposti uusiokäyttöisiä. Jos vaikkapa kuvan 8 esimerkki olisi talletettu tiedostoon MyComponent.qml, voitaisiin sitä käyttää muista QML-tiedostoissa kuvan 9 mukaisesti.

```
1   import QtQuick 1.0
2
3   Rectangle {
4       id: other_root
5       MyComponent {
6           id: mycomponent
7           x: 5
8           y: 5
9           onMySignal: {
10              console.log("mycomponent: got signal with value " + value_out)
11          }
12      }
13  }
14
```

KUVA 9. QML-komponentin uudelleen käyttäminen

5 TIETOKANNAT

SQL (Structured Query Language) on ohjelmointikieli, jota käytetään rakentamaan tekstipohjaisia kyselyitä, joissa kerrotaan tietokantamoottorille mitä tehdä. Tietokantamoottoreista on useita eri ratkaisuja, näistä suurin osa pyrkii noudattamaan SQL-perusteita, jotta ohjelmoijien ei tarvitse opetella omaa kieltä jokaista eri moottoria varten. Perustoiminnot kuten taulujen rakentaminen, tiedon lisääminen ja hakeminen onnistuvat kaikissa samalla tavalla.

5.1 SQL-syntaksi

Yleinen tapa on, että käskyt kirjoitetaan pölkkykirjaimilla, sarakkeiden nimet pienillä kirjaimilla ja lause lopetetaan puolipisteeseen. Nämä eivät kuitenkaan ole pakollista kaikissa toteutuksissa, esim. ”**CREATE TABLE taulu1 (tieto1 TEXT);**” ja ”**create table taulu1 (tieto1 TEXT)**” toimivat täysin samalla tavalla SQLitessä.

Yllä oleva kysely tekee taulun nimeltä ”taulu1”, jonka jokaisella rivillä on yksi sarake, mihin viitataan nimellä ”tieto1”. Yhteen tauluun voi määritellä sarakkeita lähes niin paljon kuin haluaa, mutta niiden liiallinen käyttö tekee tietokannan käytöstä hitaampaa ja monimutkaisempaa. Useamman sarakkeen määrittely tapahtuu erottamalla sarakkeiden määrittelyt pilkulla toisistaan, esim. ”**CREATE TABLE taulu2 (tieto1 TEXT, tieto2 INTEGER);**”.

SQL-kielen komennot voidaan jakaa kahteen osaan, DML (Data Manipulation Language) ja DDL (Data Definition Language). DML-komennoilla voidaan hakea, muokata, poistaa ja lisätä dataa tietokannasta. DML-komennoilla taas rakennetaan tietokanta. (13.)

DML-komentoja ovat SELECT, UPDATE, DELETE ja INSERT INTO (13.). Enimmäkseen näistä tarvitaan lähinnä SELECT- ja INSERT INTO -komentoja, joilla haetaan ja lisätään dataa, esim. ”**INSERT INTO taulu1 (tieto1) VALUES (”tekstiä”);**” ja ”**SELECT tieto1 FROM taulu1;**” lisäisi tauluun yhden rivin ja sen jälkeen hakisi kaikki tiedot sieltä.

5.2 SQLite-tietokantamoottori

SQLite on avoimen lähdekoodin palvelimeton tietokantamoottori, joka säilyttää tietokannan yhdessä tiedostossa. Tämän ansiosta tietokanta on käyttövalmiina samantien kun tiedosto avataan ja kaikki konfigurointi on vapaaehtoista. Kirjaston ideana on olla mahdollisimman yksinkertainen, pienikokoinen, nopea ja luotettava. Sen tarkoitus ei ole koskaan ollut syrjäyttää tietokantapalvelimia, vaan olla parempi vaihtoehto datan tallentamiseen levylle. (14.)

Kaikki ominaisuudet mukaan otettuna käännetyn kirjaston koko voi olla jopa vähemmän kuin 350 kB. Tietokantatiedosto on myös täysin alustariippumaton, sen voi kopioida vapaasti 32- ja 64-bittisten järjestelmien välillä ja järjestelmän tavujärjestyksellä ei myöskään ole väliä. (14.)

SQLiten ensimmäinen versio julkaistiin vuonna 2000, minkä jälkeen siihen on lisätty ominaisuuksia koko ajan (15.). Suurin osa SQL92-standardista on toteutettu, mutta joitakin ominaisuuksia puuttuu, kuten RIGHT JOIN, FULL OUTER JOIN, täydellinen ALTER TABLE -tuki, täydellinen trigger-tuki ja VIEWeihin kirjoittaminen. GRANT ja REVOKE on jätetty kokonaan pois moottorin rakenteen takia. (16.)

5.3 MySQL-tietokantapalvelin

MySQL on nykyään Oraclen kehittämä palvelimellinen SQL-tietokantamoottori. Ensimmäinen versio tuli vuonna 1995 ja sitä kehitetään jatkuvasti. Monet suuret www-sivut käyttävät sitä, kuten Wikipedia, Google, Facebook ja Twitter. Lisenssinä on avoimen lähdekoodin GPL-lisenssi, mutta myös maksullisia lisenssejä on saatavilla. (17.)

5.4 PostgreSQL-tietokantapalvelin

PostgreSQL on avoimen lähdekoodin tietokantapalvelin. Sitä on kehitetty aktiivisesti viimeiset 15 vuotta ja se noudattaa hyvin ANSI-SQL:2008-standardia. (18.)

PostgreSQL on ollut Linux-maailmassa erittäin hyvin vastaanotettu ja se on voittanut The Linux Journal Editorsin palkinnon parhaasta tietokannanhallintajärjestelmästä viidesti. (18.)

6 TIEDONSIIRTOPROTOKOLLAT

6.1 Tiedonsiirtoprotokolla TCP

TCP (Transmission Control Protocol) on yhteydellinen tiedonsiirtoprotokolla, joka automaattisesti huolehtii, että datajonot tulevat vastaanottajalle perille juuri samanlaisena kuin ne on lähetetty. Luotettavuutensa ansiosta se soveltuu lähes kaikkiin tarpeisiin. Ainoa ongelma on, että tietokoneiden täytyy muodostaa yhteys, ennen kuin dataa voidaan alkaa siirtämään. Tämä lisää viivettä tiedonsiirron alussa.

6.2 Tiedonsiirtoprotokolla UDP

UDP (User Datagram Protocol) on yhteydetön protokolla, toisin kuin TCP. Tämä vähentää viivettä nopeiden tiedonsiirtojen välillä, mikä ei olisi haitaksi tässä työssä. Ongelmaksi kuitenkin tulee paketin 65 527 tavun teoreettinen kokorajoitus. (19.)

Tämä raja on helppo testata käytännössä. Tässä työssä testaamiseen rakennettiin erilliset sovellukset datan lähettämiseen (kuva 10) ja vastaanottamiseen (kuva 11).

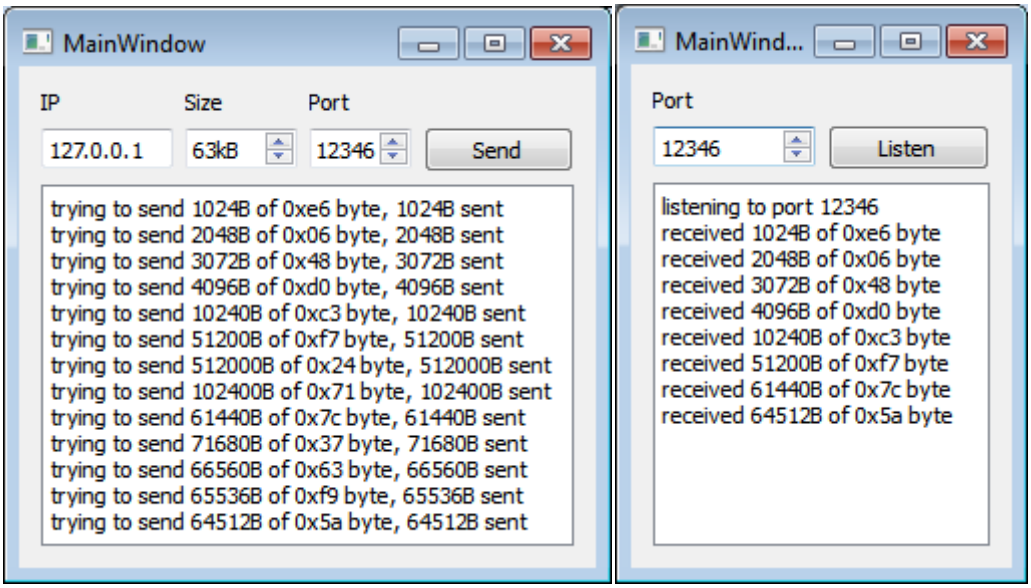
```
void MainWindow::on_pushButton_send_clicked()
{
    quint8 rand_char = qrand()%256;
    int size_to_send = ui->spinBox_size_kb->value()*1024;
    QByteArray to_send = QByteArray(size_to_send, rand_char);

    QString s_byte = QString(QByteArray((const char*)&rand_char, 1).toHex());

    int size_sent = m_socket->writeDatagram(to_send,
                    QHostAddress(ui->lineEdit_ip->text()), ui->spinBox_port->value());

    QString to_log = QString("trying to send %1B of 0x%2 byte, %3B sent")
            .arg(to_send.size()).arg(s_byte).arg(size_sent);
    ui->plainTextEdit_log->appendPlainText(to_log);

}
```

KUVA 10. Datan lähettäminen

```
void MainWindow::slot_socket_readyRead()
{
    while( m_socket->hasPendingDatagrams() )
    {
        QHostAddress sender;
        quint16 port;
        int pending_size = m_socket->pendingDatagramSize();
        QByteArray arr_data; arr_data.resize(pending_size);
        m_socket->readDatagram(arr_data.data(), pending_size, &sender, &port);

        QString s_byte = QString(arr_data.left(1).toHex());

        QString to_log = QString("received %1B of 0x%2 byte").arg(arr_data.length()).arg(s_byte);
        ui->plainTextEdit_log->appendPlainText(to_log);
    }
}
```

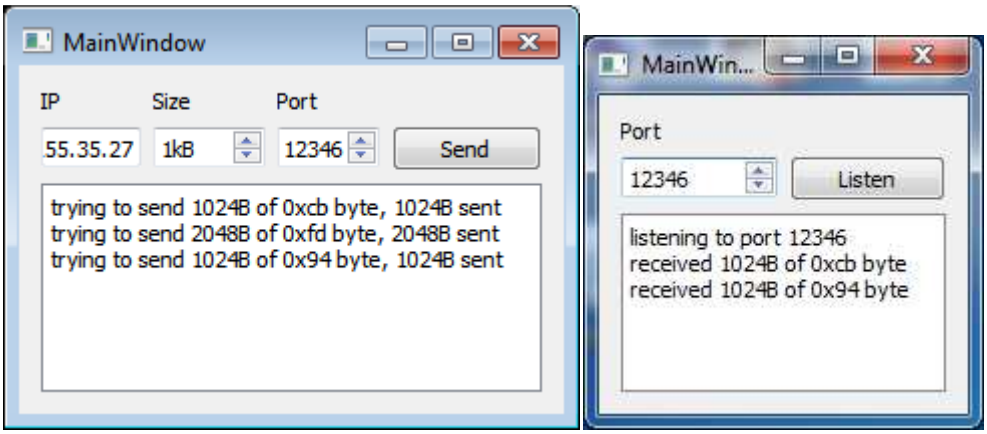
KUVA 11. Datan vastaanottaminen

Kuvassa 12 on esimerkki datan lähettämisestä lokaalisti, eli kun molemmat ohjelmat pyörivät samalla koneella. Testistä voidaan todeta, että 64 512 tavun paketti tuli perille, mutta 65 536 tavun paketti ei tullut perille, joten 65 527 tavun teoreettinen raja pitää paikkansa.



KUVA 12. Datan lähetys lokaalisti

Kuvassa 13 on vastaavanlainen esimerkki samoilla ohjelmilla. Ainoa ero on että data lähetetään internetin välityksellä toiselle koneelle.



KUVA 13. Datan lähetys internetin välityksellä

Tässä testissä tuli raja vastaan jo 1 kilotavun jälkeen, joten lähettäjän pitäisi pilkkoa data esimerkiksi 1 kilotavun kokoisiin paketteihin ja samalla huolehtia, että ne kaikki tulevat perille. Tämä on monimutkaista, koska paketit saattavat tulla perille eri järjestyksessä kuin missä ne on lähetetty ja osa voi kadota välistä tai jopa monistua (19).

UDP on kuitenkin käytössä monissa internetin tärkeimmissä osissa kuten DNS ja DHCP, joista etenkin DNS vaatii erittäin nopeat vasteajat, jotta netin selailu on tarpeeksi sulavaa.

7 PELAAJATIETOKANNAN SUUNNITTELU JA TOTEUTUS

Projekti jaettiin kahteen sprinttiin. Ensimmäisessä suunniteltiin ohjelma ja valittiin tekniikat, joita käytetään. Toisessa tehtiin toteutus ja testaus samanaikaisesti.

Työn toteutuksen aikana käytettiin Scrum-menetelmää Microsoft Excel -taulukkolaskentaohjelman avulla. Tämä osoittautui hyödylliseksi, vaikka oikean kokoista työryhmää ei ollut.

7.1 Suunnittelu

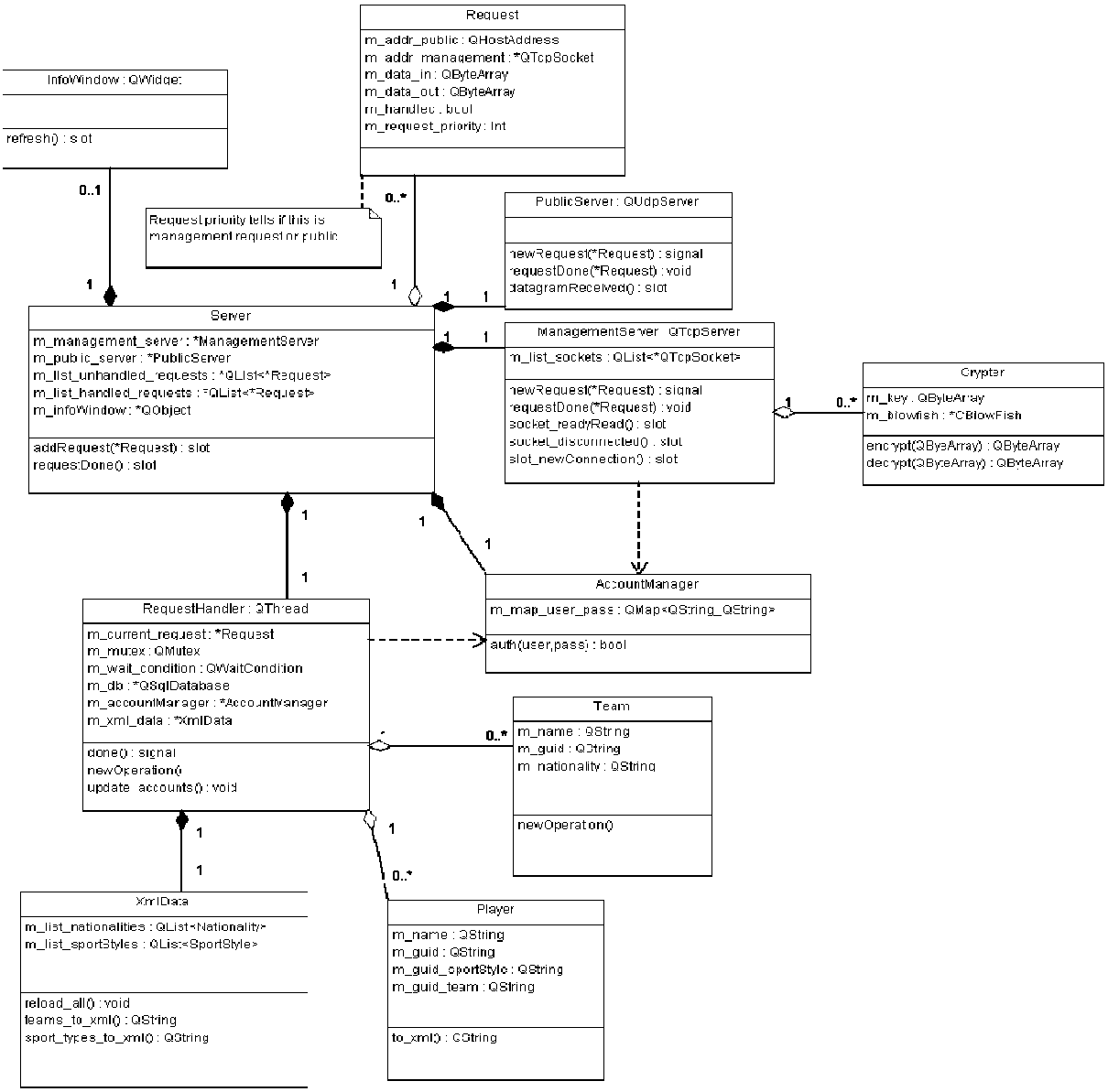
Suunnittelu tehtiin projektin ensimmäisessä sprintissä. Aivan aluksi listattiin vaatimusmäärittelyt, joiden perusteella voitiin alkaa miettiä, millaisilla komponenteilla tällainen järjestelmä saadaan toteutettua.

- Järjestelmästä saadaan haettua pelaajien ja joukkueiden tiedot.
- Järjestelmässä on hallintajärjestelmä, jolla voi muokata pelaajia ja joukkueita.
- Järjestelmän tulee toimia Linux-ympäristössä.
- Järjestelmä toteutetaan Qt-kielellä.

Heti suunnittelun alussa kävi ilmi, että tarvitaan SQL-tietokanta ja siihen salattu yhteys. Tietokannan suhteen päädyttiin nopeasti siihen valintaan, että aluksi tietokanta toteutetaan SQLiteä käyttäen ja myöhemmin voi vaihtaa MySQL- tai PostgreSQL-tietokantapalvelimeen, jos sille on tarvetta. Salaamisen toteutukseen päädyttiin käyttämään ennestään tuttua BlowFish-algoritmiä, koska sen tiedettiin toimivan moitteetta.

Ohjelman rakenteen suunnittelu aloitettiin etsimällä Linuxissa toimiva ohjelma luokkakaavion piirtämiseen. Ensimmäisenä katsottiin tuttua Boumlia, mutta ilmaisen version jakaminen oli lopetettu, eikä siitä ollut mitään versiota saatavilla. Nykyään siitä on saatavana maksullinen Qt versio.

Lopulta päädyttiin käyttämään avoimen lähdekoodin ArgoUML-ohjelmaa. Se on tehty Javalla, joten se toimii Windows- ja Linux-käyttöjärjestelmissä. Alkuperäinen suunniteltu luokkakaavio on esitetty kuvassa 14.

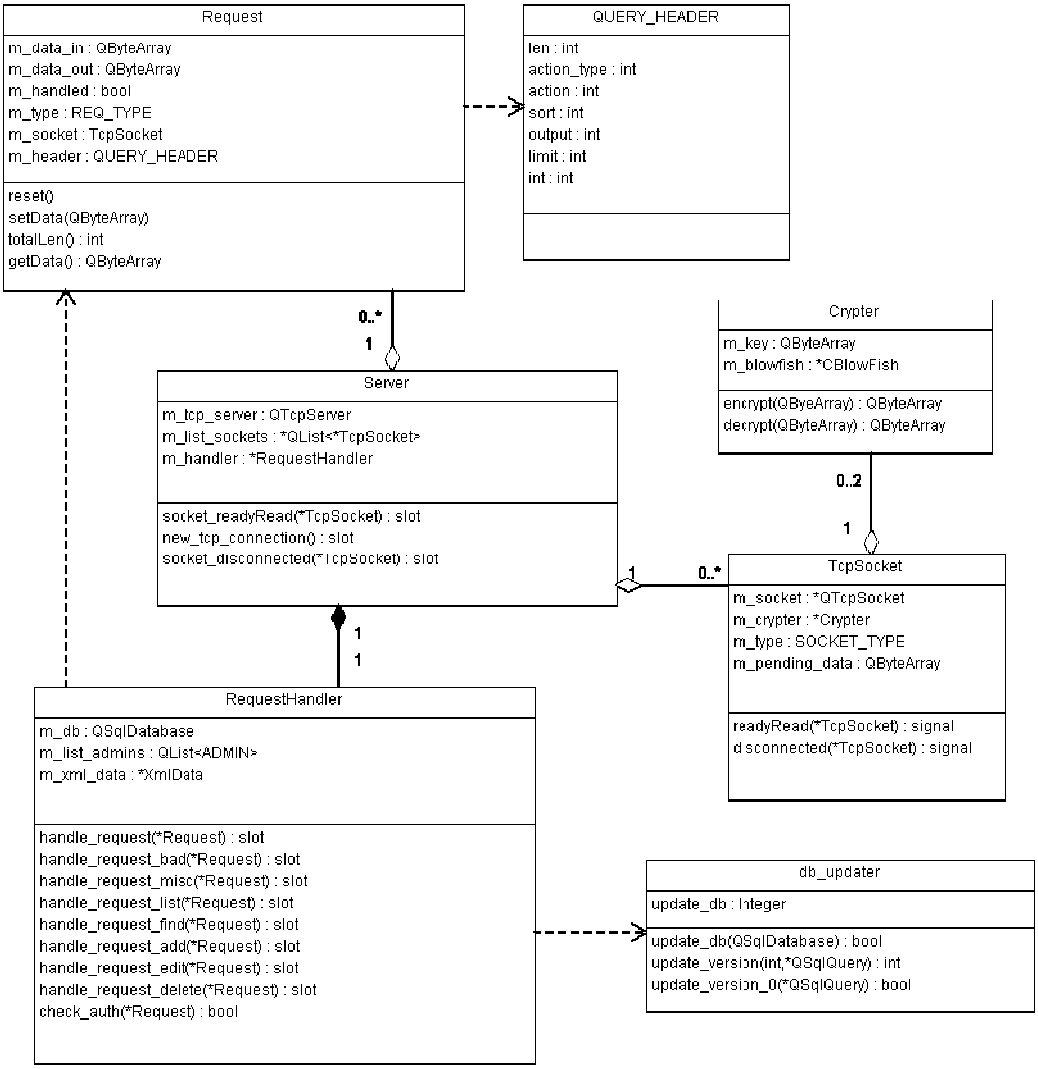


KUVA 14. Suunniteltu luokkakaavio

Tietokannan suunnittelu oli hyvin suoraviivaista: listattiin mitä datakenttiä tarvitaan ja päätettiin näille datoille sopiva tyyppi. Tekstipohjaisten datatyyppien käyttö on monesti hyvä valinta, jos valinnan kanssa on epävarmuutta.

7.2 Palvelinohjelma

Työn toteutus aloitettiin avaamalla suunnitteluvaiheessa tehty luokkakaavio (kuva 14). Ohjelman runkoa alettiin rakentaan tämän kaavion pohjalta. Puolessa välissä toteutusta huomattiin, että alkuperäinen suunnitelma olikin turhan monimutkainen ja siihen pitää tehdä muutoksia. RequestHandler-luokkaa ei toteutettu omana säikeenä, vaan Server-luokka kutsuu siltä suoraan käsittelyfunktiota. Säikeiden käyttö ei ole yleensä tarvittavaa Qt:n joustavan signal-slot-rakenteen ansiosta. PublicServer- ja ManagementServer-luokat yhdistettiin yhdeksi TCP-serveriksi, koska huomattiin, että salaus saattaisi olla hyvä vaihtoehto myös julkiselle puolelle. Ohjelmaan tulee myös vähemmän haaroja, kun tarvitsee käyttää vain yhtä TCP-serveriä. AccountManager-luokalle ei myöskään koettu tarpeelliseksi rakentaa omaa luokkaansa, vaan se yhdistettiin RequestHandler-luokkaan. Toteutunut luokkakaavio on esitelty kuvassa 15.



KUVA 15. Toteutuneen ohjelman luokkakaavio

Työssä tehtiin myös esimerkiksi pelaajien ja joukkueiden käsittelyyn omat luokat, joilla on helppo muuntaa olio tekstiksi lähetystä varten ja vastaanottaessa tekstistä takaisin olioksi. Näitä luokkia ei ole listattuna luokkakaaviossa.

7.3 Tietokanta

Tietokanta toteutettiin Qt:n tietokantamoduulia käyttäen. Moottoriksi valittiin SQLite sen helpon käyttöönoton, yksinkertaisuuden ja siirrettävyyden takia.

Tietokannan luonti tapahtuu RequestHandler-luokan luonnin yhteydessä, jolloin tarkistetaan tietokannan nykyinen versio ja päivitetään versio kerrallaan niin kauan, että se on uusimmassa versiossa tai tapahtuu virhe. Tämä logiikka on nähtävissä kuvissa 16 ja 17.

```
while( version < MAX_DB_VERSION )
{
    q.clear();
    int newver = update_version(version, &q);   //getting version again from db might be a good idea
    if( newver <= version )
    {
        qDebug() << Q_FUNC_INFO << "updating version" << version << "failed";
        q.clear();
        return false;
    }
    version = newver;
}
```

KUVA 16. Tietokannan luonti ja päivitys

Update_version-funktiossa on switch-case-rakenne, joka kutsuu seuraavan funktion riippuen versionumerosta. Kohtia on nollasta MAX_DB_VERSION-arvoon asti. Tietokannan versio haetaan aina ohjelman käynnistyksessä. Jos versiota ei löydy, asetetaan versio oletuksena nollaan, jolloin switch-case-rakenne osaa ohjata sen ensimmäiselle update_version-funktiolle.

```
int update_version(int version, QSqlQuery *q)
{
    switch(version)
    {
    case 0:
        if( update_version_0(q) ) version = 1;
        break;

    case 1:
        //latest version, no need to do anything
//        if( update_version_1(q) ) version = 2;
        break;

    default:
        //unknown version
        qDebug() << Q_FUNC_INFO << "unknown version" << version;
        return 0;
        break;
    }
    return version;
}
```

KUVA 17. Update_version-funktio

7.4 Tietoliikenne

Ohjelman kaikki tietoliikenne toteutettiin TCP-yhteyksillä. Yhteyksien käyttöön löytyy Qt:n tietoliikennemoduulista näppärät QTcpServer- ja QTcpSocket-luokat. Kuvassa 18 on esimerkki miten QTcpServerillä voidaan kuunnella tiettyä porttia kaikissa tarjolla olevissa IP-osoitteissa.

```
m_tcp_server = new QTcpServer(this);
if( !m_tcp_server->listen(QHostAddress::Any, defaults::tcp_port) )
{
    qCritical() << "cannot listen tcp port" << defaults::tcp_port << "error:" << m_tcp_server->errorString();
    exit(-1);
}

connect(m_tcp_server, SIGNAL(newConnection()), this, SLOT(slot_tcp_newConnection()));
```

KUVA 18. QTcpServerin käyttöönotto

QTcpServer lähettää newConnection()-signaalin aina, kun uusi yhteys luodaan. Tämä signaali käsitellään Server-luokan slot_tcp_newConnection-slotissa.

```
void Server::slot_tcp_newConnection()
{
    while( m_tcp_server->hasPendingConnections() )
    {
        QTcpSocket *soc = m_tcp_server->nextPendingConnection();

        soc->setSocketOption(QTcpSocket::LowDelayOption, 1);

        TcpSocket *tcpsoc = new TcpSocket(soc);
        m_list_sockets.append(tcpsoc);

        connect(tcpsoc, SIGNAL(disconnected(TcpSocket*)), this, SLOT(slot_socket_disconnected(TcpSocket*)));
        connect(tcpsoc, SIGNAL(readyRead(TcpSocket*)), this, SLOT(slot_socket_readyRead(TcpSocket*)));

        qDebug() << Q_FUNC_INFO << soc->peerAddress().toString();
    }
}
```

KUVA 19. TCP-yhteyksien hyväksyminen

Osoitin avattuun yhteyteen laitetaan talteen listaan, josta niitä on helppo käsitellä tarpeen tullen. Yhteyksiltä otetaan pois käytöstä Nagle-algoritmi, joka on oletuksena päällä. Tämä algoritmi välttää pienien pakettien lähettämistä keräämällä niitä pienen ajan ja sitten lähettämällä ne kaikki kerralla. Joissakin tapauksissa tämä saattaa vähentää tietoliikenteen määrää, mutta tässä työssä se hyvin luultavasti aiheuttaisi vain turhaa viivettä.

Datan lähettämisessä ja vastaanottamisessa käytettiin hyväksi osoitinta QUERY_HEADER-tietueeseen. Vastaanotettaessa datan alusta tehtiin osoitin tietueeseen, jolla pystyttiin sen jälkeen hakemaan tietueesta haluttu arvo. Lähettäessä tietueen arvot asetettiin oikeisin arvoihin, jonka jälkeen tehtiin char tyyppinen osoitin tietueen alusta ja talletettiin QByteArray-luokkaan tietueen koon verran dataa. Tämä on helppo ja yksinkertainen tapa välittää tietoa, mutta sen kanssa voi kuitenkin tulla helposti ongelmia eri kääntäjien välillä. Tästä voi lukea lisää esimerkiksi sivulta http://en.wikipedia.org/wiki/Data_structure_alignment.

7.5 Salaus

Salaus toteutettiin Bruce Schneierin vuonna 1993 kehittämällä Blowfish-algoritmillä, jonka tarkoitus oli syrjäyttää vanha DES (3). Yksi syy Blowfishin käyttöön oli sen integroinnin helppous verrattuna muihin algroritmeihin. Myös sen aiempi käyttö vaikutti valintaan.

Koska Blowfishin avaimen vaihtaminen on raskas operaatio (3). Päätettiin ohjelmassa tehdä oma olio jokaista avainta varten, jotta sitä ei tarvitse tehdä uudestaan joka kerta.

```
class Crypter
{
public:
    static QByteArray make_key(const QByteArray &input);    //makes max len key
    Crypter(const QByteArray &key);
    QByteArray Encrypt(const QByteArray &arr);
    QByteArray Decrypt(const QByteArray &arr);

private:
    Blowfish *m_blowfish;
};
```

KUVA 20. Crypter-luokan esittely.

Blowfish vaatii että salattavan datajonon pituus on jaollinen kahdeksalla. Tätä varten täytyy kaikki datajonot laajentaa sopivan kokoisiksi ja merkitä niihin miten paljon oikeata dataa on tai kuinka paljon dataa on laajennettu. Laajentamiseen käytettiin tässä työssä nolla-tavuja ja logiikka on esitelty kuvissa 21 ja 22.

```
QByteArray Crypter::Encrypt(const QByteArray &arr)
{
    QByteArray ret;
    int len = arr.length();
    ret.reserve(len+8);
    ret.append((char*)&len, sizeof(int));
    ret.append(arr);
    while(ret.count()%8 != 0)
        ret.append('\0');

    m_blowfish->Encrypt(ret.data(), ret.length());
    return ret;
}
```

KUVA 21. Datan kryptaus

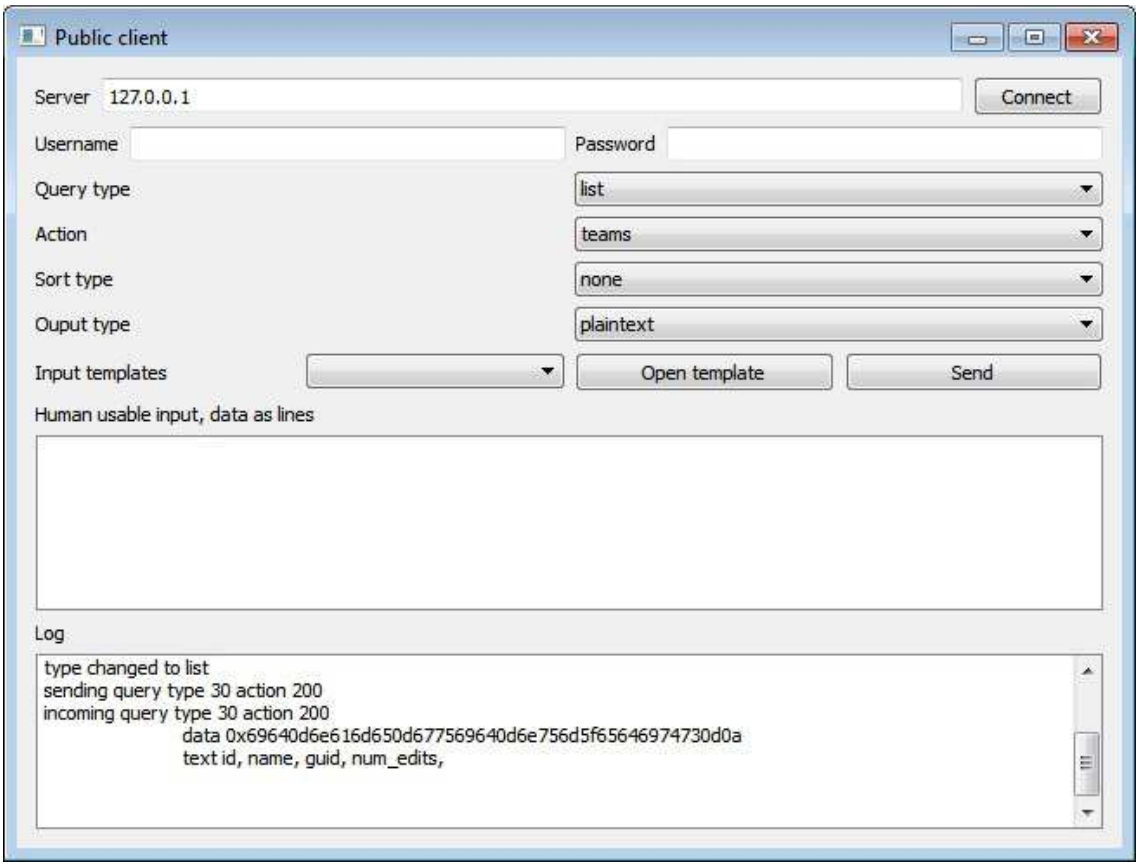
Kun salattaessa lisätään salatun tiedon alkuun kuinka paljon oikeaa dataa on, voidaan salaus purkaa joka kerta varmasti. Pelkästään nolla-tavujen poistaminen lopusta ei toimi, koska salattu datajono voi loppua nolla-tavuun.

```
QByteArray Crypter::Decrypt(const QByteArray &arr)
{
    QByteArray tmp;
    tmp.append(arr);
    m_blowfish->Decrypt(tmp.data(), tmp.length());
    int len = *( (int*)tmp.data() );

    if( len > arr.length()-4 || len < 0) return QByteArray();

    return tmp.mid(sizeof(int), len);
}
```

KUVA 22. Kryptatun datan purkaminen

7.6 Testaus

Palvelimen testausta varten rakennettiin asiakasohjelma (kuva 23), jolla voidaan lähettää palvelimelle pyyntöjä manuaalisesti.



KUVA 23. Testausohjelma

Ohjelmalla voidaan esimerkiksi kirjautua sisään, ottaa salaus käyttöön, lähettää kaikkia mahdollisia paketteja ja syöttää pakettiin menevä data manuaalisesti. Lisäksi joukkueiden, lajien, kansallisuuksien, pelaajien ja roolien lisäämiseen voi käyttää niille tarkoitettua pohjaa.

Käyttöliittymässä olevan Action-valitsimen alle täytyi saada uudet arvot, kun Query type -valinta vaihtui. Tätä varten rakennettiin logiikka, jonka avulla saatiin lista uusista arvoista Query typen perusteella.

QMap on Qt:n ydinmoduulin säiliöluokka, joka säilyttää sisällään listan avain-arvo-pareja. Tätä listaa voidaan myöhemmin käydä yksitellen läpi tai etsiä arvo avaimen perusteella.

Aluksi esiteltiin QMap-olio (kuva 24), jonka avulla saadaan yhdistettyä Query type toiseen QMap-olioon, jolla puolestaan saadaan yhdistettyä Action sitä vastaavaan numeeriseen arvoon, jota käytetään myöhemmin tiedonsiirrossa.

```
QMap<QString, QMap<int, QString>* > m_map_type_to_map;
```

KUVA 24. QMapin esittely

Tämän jälkeen rakennetaan QMap-olioita (kuva 25), joihin jokaiseen lisätään Actionin nimi ja sen numeerinen arvo. Nämä oliot lisätään ensimmäiseen m_map_type_to_map-olioon, josta niitä käytetään myöhemmin.

```
QMap<int, QString> *map_action_to_name;

map_action_to_name = new QMap<int, QString>();
map_action_to_name->insert(Request::ACTION_LIST_NATIONALITIES, "nationalities");
map_action_to_name->insert(Request::ACTION_LIST_PLAYERS, "players");
map_action_to_name->insert(Request::ACTION_LIST_ROLES, "roles");
map_action_to_name->insert(Request::ACTION_LIST_SPORTS, "sports");
map_action_to_name->insert(Request::ACTION_LIST_TEAMS, "teams");
m_map_type_to_map.insert("list", map_action_to_name);

map_action_to_name = new QMap<int, QString>();
map_action_to_name->insert(Request::ACTION_MISC_AUTH, "auth");
map_action_to_name->insert(Request::ACTION_MISC_AUTH_REPLY, "auth reply");
map_action_to_name->insert(Request::ACTION_MISC_ENCRYPT, "encrypt");
map_action_to_name->insert(Request::ACTION_MISC_GET_ADMIN_LIST, "get admin list");
map_action_to_name->insert(Request::ACTION_MISC_PING, "ping");
map_action_to_name->insert(Request::ACTION_MISC_PONG, "pong");
map_action_to_name->insert(Request::ACTION_MISC_UPDATE_PERMISSIONS, "update permissions");
m_map_type_to_map.insert("misc", map_action_to_name);
```

KUVA 25. Actionien listaus

Nyt m_map_type_to_map-olion avulla saadaan helposti haettua kaikki Actionit tietyn Action typen perusteella ja vaihtaa ne näkymään käyttöliittymässä (kuva 26).

```
void ClientWindow::on_comboBox_query_type_currentIndexChanged(const QString &arg1)
{
    ui->plainTextEdit_log->appendPlainText("type changed to " + arg1);
    ui->comboBox_action->clear();

    QStringList items;

    QMap<int, QString> *map_action_to_name = m_map_type_to_map.value(arg1, 0);
    if( map_action_to_name == 0) return;

    QMapIterator<int, QString> iter(*map_action_to_name);
    while(iter.hasNext() )
    {
        iter.next();
        items << iter.value();
    }

    ui->comboBox_action->addItems(items);
}
```

KUVA 26. Query typen vaihtumiseen reagointi

Actionin numeerinen arvo saadaan haettua pyynnön lähetysvaiheessa, kun käydään QMap-olion kaikki arvot läpi yksitellen. Tästä esimerkki kuvassa 27.

```
int ClientWindow::enum_val(const QString &type, const QString &action)
{
    QMap<int, QString> *map_action_to_name = m_map_type_to_map.value(type, 0);
    if( map_action_to_name == 0) return 0;

    QMapIterator<int, QString> iter(*map_action_to_name);
    while(iter.hasNext() )
    {
        iter.next();
        if( iter.value() == action ) return iter.key();
    }
}
```

KUVA 27. Actionin numeerisen arvon hakeminen

Vastaavanlainen funktio voitaisiin rakentaa, jos haluttaisiin Actionin numeerisesta arvosta sen nimi esimerkiksi näkymään lokissa.

8 YHTEENVETO

Opinnäytetyön tavoitteena oli suunnitella ja toteuttaa pelaajatietokanta Kuru Digital Creations Oy:lle. Ohjelman idea on tarjota tiedot pelaajista ja joukkueista kaikille yrityksen sovelluksille.

Työn toteutuksen aikana opin, että suunnitteluvaiheessa kannattaa ohjelman ensimmäisestä versiosta suunnitella mahdollisimman yksinkertainen, eikä koittaa saada kerralla täydellistä ohjelmaa. Suunnitellusta ohjelmasta olisi tullut liian monimutkainen ja sitä jouduttiin yksinkertaistamaan kovin ottein yhdistämällä luokkia ja poistamalla turhan monimutkaisia rakenteita. Toteutuksen aikana sai myös paljon kokemusta tietoliikenteen toteuttamisesta, tietokannoista, palvelin-ohjelman toteuttamisesta, salauksesta, tietueiden käsittelystä ja testausohjelman tekemistä. Näistä on varmasti hyötyä myöhemmin työelämässä.

Työn tuloksena syntyi toimiva, perustoiminnot hoitava ohjelma, johon on helppo lisätä ominaisuuksia tai jatkokehittää siitä isompi kokonaisuus.

LÄHTEET

1. What cloud computing really means | Cloud Computing - InfoWorld. Saatavissa: http://www.infoworld.com/d/cloud-computing/what-cloud-computing-really-means-031?page=0,0. Hakupäivä 25.5.2012.

2. 6 Types of Cloud Computing. Saatavissa: http://www.unitiv.com/it-solutions-blog/bid/77277/6-Types-of-Cloud-Computing. Hakupäivä 25.5.2012.

3. IaaS vs. PaaS vs. SaaS. Saatavissa: http://www.networkworld.com/news/2011/102511-tech-argument-iaas-paas-saas-252357.html Hakupäivä 16.9.2012.

4. Google Maps Image APIs - Google Maps Image APIs — Google Developers. Saatavissa: https://developers.google.com/maps/documentation/imageapis/ Hakupäivä 16.9.2012.

5. What Is Agile Software Development - Learn About Agile Software Development - VersionOne. Saatavissa: http://www.versionone.com/Agile101/Agile-Development-Overview/. Hakupäivä 24.5.2012.

6. Scrum Alliance - What Is Scrum? . Saatavissa: http://www.scrumalliance.org/pages/what_is_scrum. Hakupäivä 25.5.2012.

7. Simple Product Backlog Example | Agile Software Development. Saatavissa: http://agilesoftwaredevelopment.com/scrum/simple-product-backlog. Hakupäivä 25.5.2012.

8. Agile Lean Software Development - Scrum Methodology - Extreme Programming. Saatavissa: http://www.versionone.com/Agile101/Agile-

Development-Methodologies-Scrum-Kanban-Lean-XP/. Hakupäivä 24.5.2012.

9. XP flow Chart. Saatavissa: http://www.extremeprogramming.org/map/project.html. Hakupäivä 24.5.2012.

10. Who we are — Qt - A cross-platform application and UI framework. Saatavissa: http://qt.nokia.com/about/who-we-are hakupäivä 9.5.2012.

11. Maintainers |Qt Wiki | Qt Developer Network. Saatavissa: http://wiki.qt-project.org/Maintainers hakupäivä 9.5.2012.

12. Modular Class Library. Saatavissa: http://qt.nokia.com/products/library/modular-class-library hakupäivä 9.5.2012.

13. SQL Tutorial. Saatavissa: http://www.w3schools.com/sql/default.asp hakupäivä 10.5.2012.

14. About SQLite. Saatavissa: http://www.sqlite.org/about.html. Hakupäivä 23.5.2012.

15. Release history of SQLite. Saatavissa: http://www.sqlite.org/changes.html. Hakupäivä 23.5.2012.

16. SQL Features That SQLite Does Not Implement. Saatavissa: http://www.sqlite.org/omitted.html. Hakupäivä 23.5.2012.

17. MySQL - Wikipedia, the free encyclopedia. Saatavissa: http://en.wikipedia.org/wiki/MySQL hakupäivä 10.5.2012.

18. PostgreSQL: About. Saatavissa: http://www.postgresql.org/about/. Hakupäivä 23.5.2012.

19. User Datagram Protocol. Saatavissa: http://en.wikipedia.org/wiki/User_Datagram_Protocol. hakupäivä 12.4.2012.

20. Blowfish (cipher) - Wikipedia, the free encyclopedia. Saatavissa: http://en.wikipedia.org/wiki/Blowfish_%28cipher%29 hakupäivä 10.5.2012.