PARTool
Design Description

Version 1.1

Doc. No.:

Revision History

Date	Version	Description	Author
2011-10-26	0.9	Initial Draft	Matija Hanžić and Inderjeet Singh
2011-10-26	0.91	Minor corrections	Igor Rutkowski
2011-10-28	0.92	Minor corrections	Davor Perić
2011-10-29	0.93	System Architecture rewritten, GUI description removed from External Interfaces section	Igor Rutkowski
2011-10-29	0.94	Section 4 full and section 3.4 and 3.5 changed	Inderjeet Singh
2011-11-04	0.95	Final review, grammatical and spelling corrections	Davor Perić
2011-11-10	0.96	Project supervisor comments implemented	Inderjeet Singh Igor Rutkowski
2011-11-22	1.0	Version changed to 1.0	
2012-01-08	1.1	Minor changes	Davor Perić

Table of Contents

1. Introduction

1.1 Purpose of this document

The purpose of this document is to give an insight into the design of the profile analysis and reconciliation tool. It elaborates how this project will be realized and provides a more detailed view into its architecture.

1.2 Intended Audience

The intended audience for this document is:

- Project members,
- Project supervisor,
- Customer.

1.3 Scope

This document shows the system architecture of the project with a high-level explanation of the different components involved in the project and the relationships between components. This document is not intended to give very detailed information regarding individual components.

1.4 Definitions and acronyms

1.4.1 Definitions

Keyword	Definitions
Agent	Final user of the profile analysis and reconciliation tool.
Subscriber	User of telecom system whose personal data and CDR are stored in system.

1.4.2 Acronyms and abbreviations

Acronym or abbreviation	Definitions
PARTool	Profile Analysis and Reconciliation Tool
HTML	HyperText Markup Language
CSS	Cascading Style Sheets
GUI	Graphical User Interface
MVC	Model, view and controller architectural pattern
CDR	Call Detail Records
JS	JavaScript
JSF	JavaServer Faces
RDBMS	Relational database management system
AJAX	Asynchronous JavaScript and XML
JDBC	Java Database Connectivity
JPA	Java Persistence API
ORM	Object-Relational Mapping
JSON	JavaScript Object Notation
GSON	Java library that can be used to convert Java Objects into their JSON representation
HTTP	Hyper Text Transfer Protocol

1.5 References

[1]. Database Tables: Link Analysis Tool 00.RDC.1013-1 Rev. 0.1

2. External interfaces

No external interfaces will be used by our system.

3. Software architecture

3.1 Conceptual design

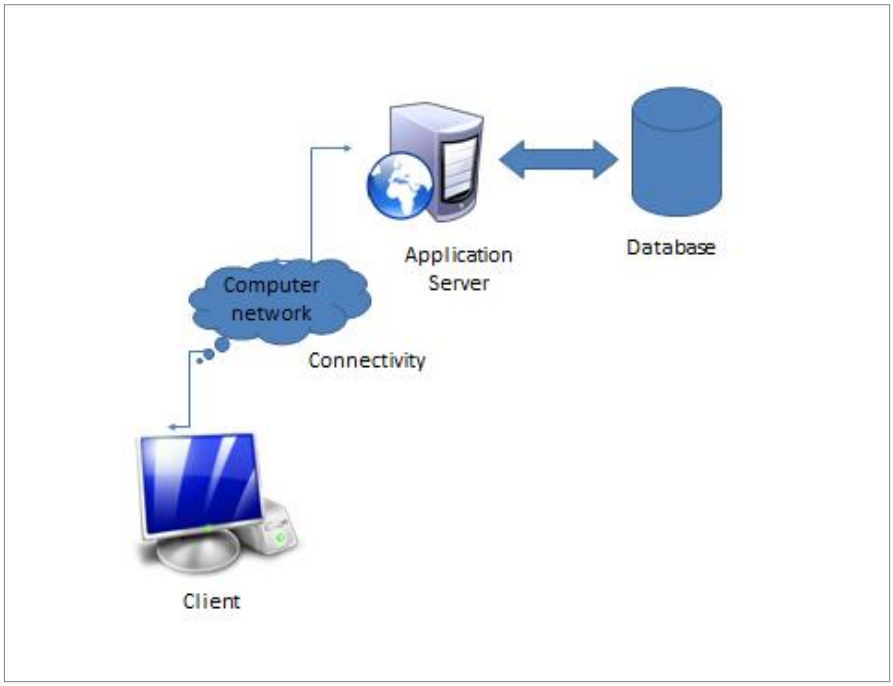

Figure 1 Conceptual design drawing

Fraud is one of the biggest problems for telecom service providers today. One of the techniques used to combat this is called “profile analysis”. Profiles are subscriber patterns of behavior created using information gathered as call details records (CDR-s).

The goal of this project is to develop a module of a fraud management system responsible for visualization of selected subscriber profiles. The tool should be easy to integrate and flexible enough to be extended easily later. It has to be user-friendly to agents and require no training. Because of the private nature of the information, agents are required to authenticate before using the tool.

The system utilizes a client-server architecture. Our application will be hosted on an application server accessible over a computer network (LAN or Internet). On the agent side a modern web browser is required, which simplifies deployment. Only the data needed for the visualization is sent to the agent, raw CDRs will be processed on server side.

Used technologies, frameworks and libraries have been chosen in order to make development process fast without sacrificing the quality or reliability of the final product.

3.2 System Architecture in General

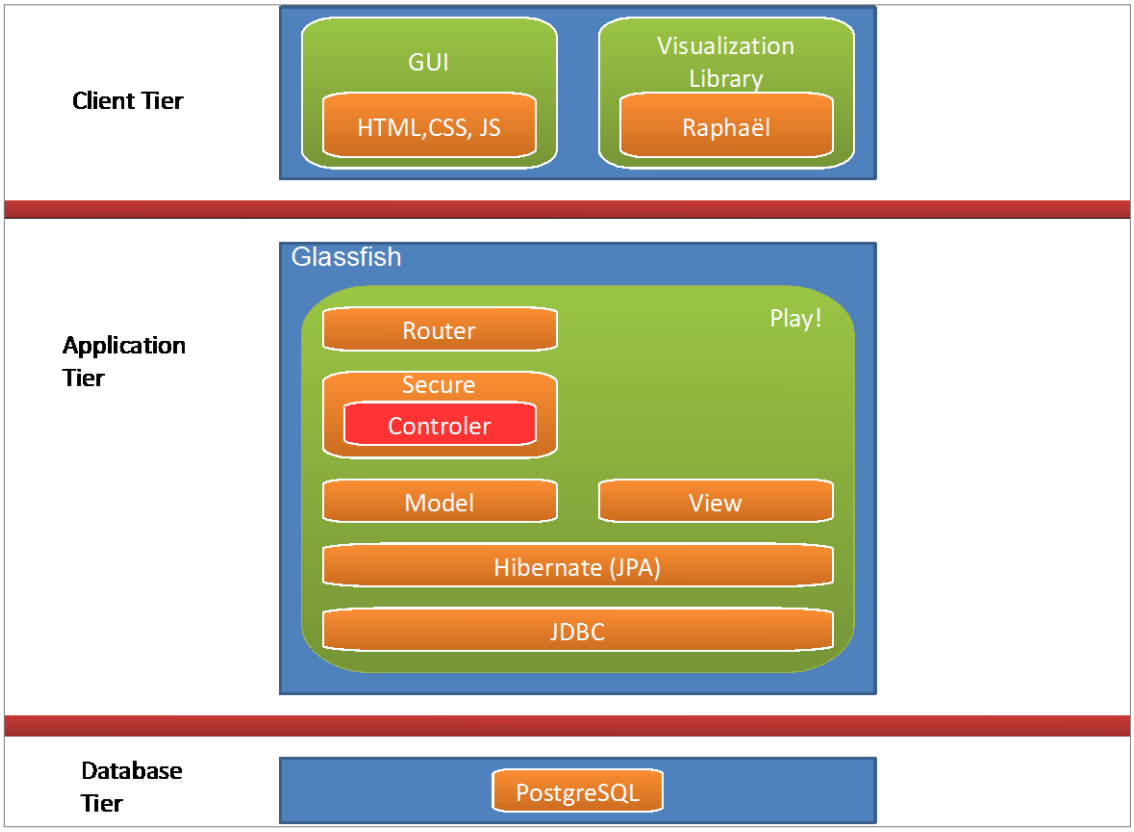

Figure 2 System Architecture schematics

The system will be divided into 3 tiers (as shown in figure 2.). The database tier contains a PostgreSQL database as specified by the customer.

Above the database tier is the application tier where the business logic resides. The server-side application runs on Glassfish – an open source web application server. Glassfishs ease of use gave it competitive edge.

The application makes use of the Play! web application framework following the MVC architectural pattern. Models represent the data stored in Database and are retrieved through JPA (framework for ORM) specification which is implemented by Hibernate. Play provides some extensions to this database paradigm. Hibernate itself makes use of the JDBC API. Views will render the user interface in HTML format or provide data in JSON format. Controllers respond to events such as agent's HTTP or AJAX requests, finalization of data processing and others. The Router module of Play! will get the HTTP request from the Agent and forward it to one of the controllers. Each controller is enclosed by a security module verifying whether an agent has already authenticated. If not, the request will be redirected to login page. Play! framework has been used instead of the full J2EE framework because it suits smaller projects better as it focuses on increasing productivity.

The top tier is the client which will be responsible for getting input from the agent and displaying the result. The GUI will be created by using HTML/CSS and the JavaScript language. Raphaël.js will be used to create the charts – it has been chosen due to its ease of use, small size and high speed.

The GUI consists of web pages that are accessible in all the major browsers. These pages will be created by the use of HTML/CSS and JavaScript.

Agents will have to authenticate first with proper credentials on the login page (see Figure 3.). The main page will contain two panels (aligned vertically) with tabs visualizing user profiles as charts.

The first tab will be a form for searching subscribers whose profiles will displayed containing checkboxes, radio-buttons and other fields modifying type of data to be displayed. Each row in visualization will represent day and each column – one hour (as shown in Figure 4.).

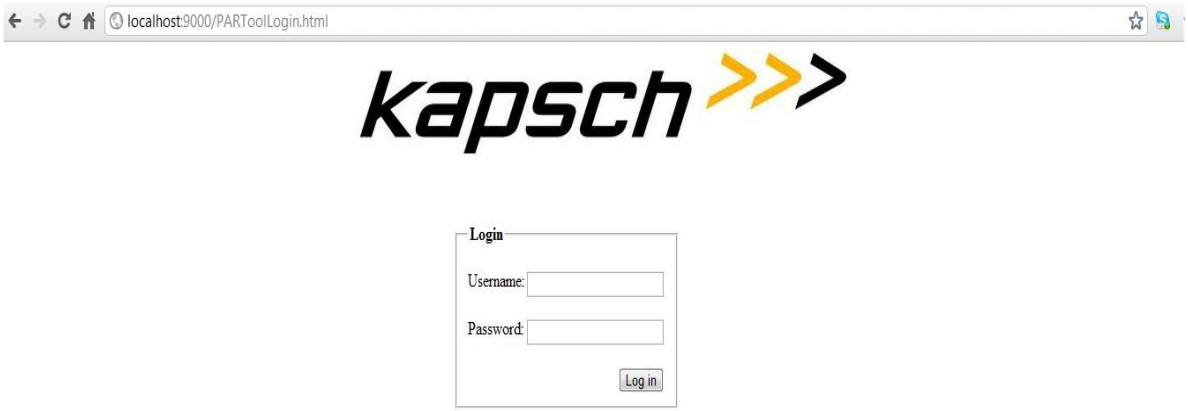

Figure 3 Login screen

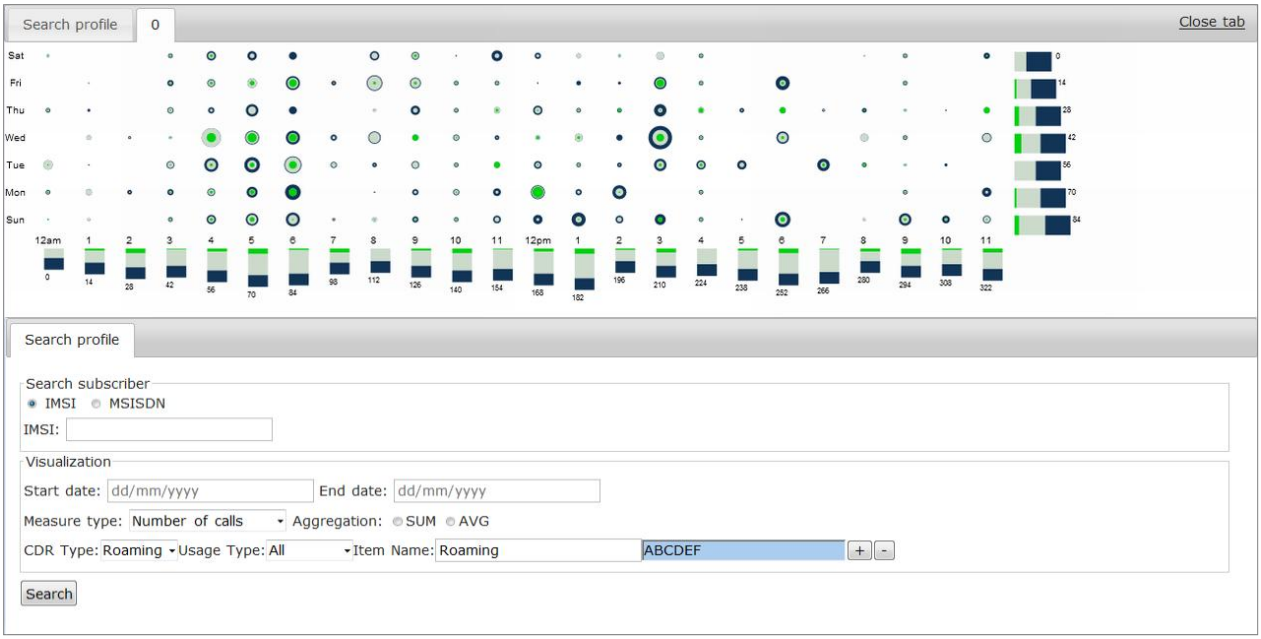

Figure 4: Main Page

Search tab

The search tab contains two sections, labeled "Search subscriber" and "Visualization". The first section contains radio buttons and one text field labeled IMSI. Radio buttons allow agent to choose if search will be based on the IMSI or on the MSISDN number. Text field is required in both cases. Without filling it Agent cannot proceed and it shall be validated if it contains exactly 15 digits.

The visualization section allows to select the parameters required to create the visualization of a profile. The start and end dates can be selected through the a popup calendar. The measure type input makes it possible to choose what kind of data will be shown – rated amount, duration of calls etc. and aggregation radio buttons enables the agent to select between "SUM" and "AVG".

The last part of the form is a list of the items to be displayed. Items can be added or removed using "+" and "–" buttons. Through CDR Type agent can select types of call visible e.g. voice, roaming, SMS and GRRS and Usage Type makes it possible to select the type of usage like On-net call, Domestic or international. Each item can be given a name using Item Name field which will then be used in the visualization.

3.3 System Specification

The specification is a mixture of the specification given by customer and input from team members. We have decided on using HTML/CSS and JavaScript for client side. We have also chosen lightweight web framework Play! over technologies such as JSF.

Glassfish will be the web server running on an Linux machine. The RDBMS will be PostgreSQL as specified by the customer. A short summary of the specification is given below:

1. GUI: HTML/CSS and JavaScript
2. Visualization: Raphaël.js library
3. Web Framework: Play! framework based on MVC.
4. Web Server: Glassfish on UNIX machine
5. Database: PostgreSQL

3.4 Error handling

Error handling on the server side will be done through the Play! web framework, which provides special folder (app/views/errors) for the error pages corresponding to HTTP error codes. Each controller can redirect user requests to special pages by using functions such as error(), forbidden() notfound() etc. All these function will be defined in the controller and called accordingly.

User input will be first validated on the user side (date format, whether data in IMSI field is numbers only etc.) and then on the server side (if such an IMSI number is registered at all).

Error	Action
Page doesn't exist	Show 404 error page
Coding error	Show 500, oops an error occur
Wrong username or password	Message : Incorrect user or password
Connection fail to database	Alert message and alert user and admin
Field not selected or filled	Select field or field is required
User input not valid	Show Pop-up window with error message.

3.5 Data Validation

There are two areas where data validation is needed:

- login page,
- search for subscriber tab on main page.

Both of them are not very complicated. We have decided to put a set of error message designated for search function (see section 3.4) while login page validation errors will be shown in a popup window.

The user name and password will be first validated on the agent's side to check if they contain no invalid characters. If correct credentials are entered the agent will be redirected to the main site. The login form

has been prepared using Secure module of Play! framework.

To validate user input on the search subscribers tab we also use Play! provided validation classes and methods. The main validation class is play.data.validation.Validation, which maintains a collection of play.data.validation.Errors objects. It consist of two part “key” and “message”, key is for determining the data element which caused error while message is to give descriptive detail. For example to validate name and age we need to put validate method with the parameter

Default messages are available in conf/message directory, which the developer can change according to the needs and use it in required place.

4. Detailed software design

4.1 Database schema

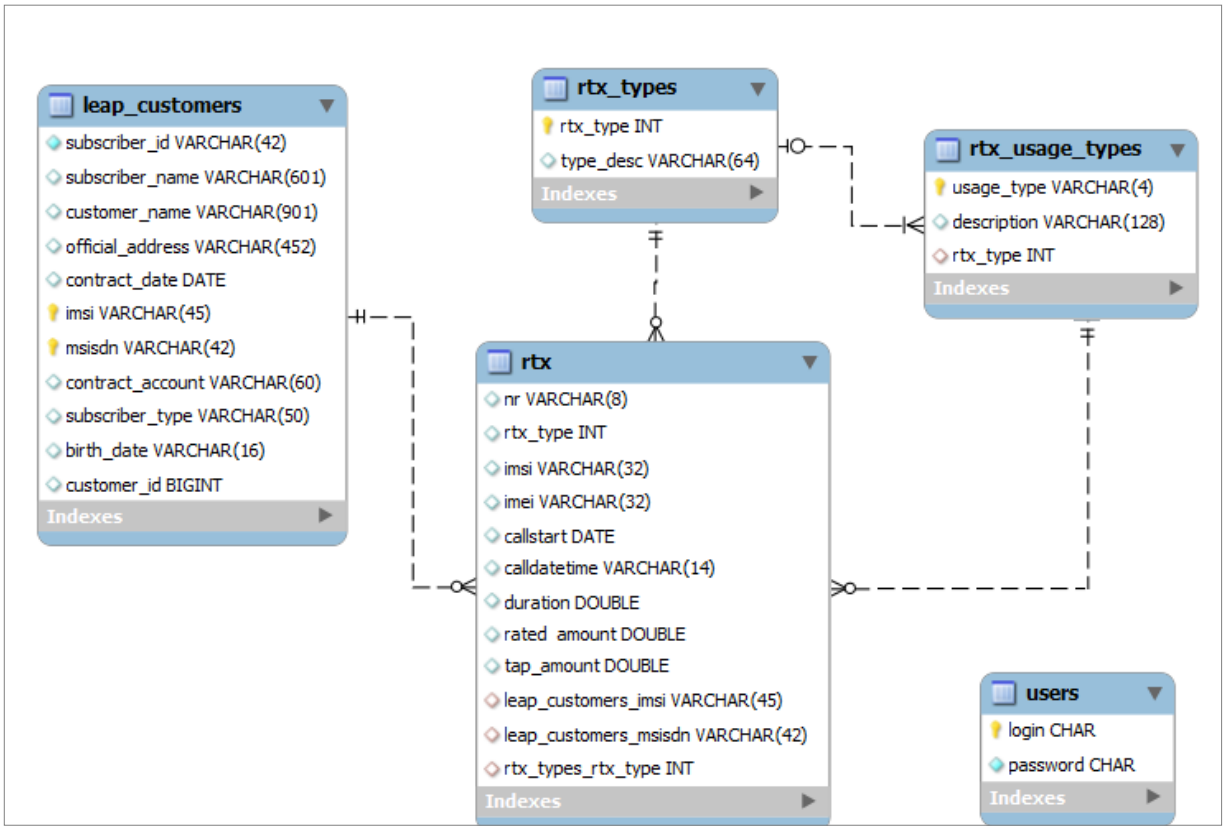

Figure 5 Database schema

The simplified schema of Client's provided database is shown in figure 5. – only fields useful to this project are shown. It has 5 major tables of which 4 are interconnected to each other and store information about both the subscriber and call detail records. Detailed description of each field can be found in Database Tables Link Analysis Tool document and will not be duplicated here.

The users table is independent from other 4 tables and is used for the authentication of Agents. It stores only the credentials, and has no relation to other data.

The table leap_customer contains all information related to subscriber personal data such as name, phone number etc. The primary key of table is subscriber_id. This table has one-to-many relationship with the rtx table holding the information of call detail records.

The tables rtx_types and rtx_usage_types store information about the CDR types and their specific usage. Some examples of CDR types are call, sms, mms, gprs. For the CDR type call there can be multiple usage types, such as international call, call to paid number, national call.

4.2 Class Diagram

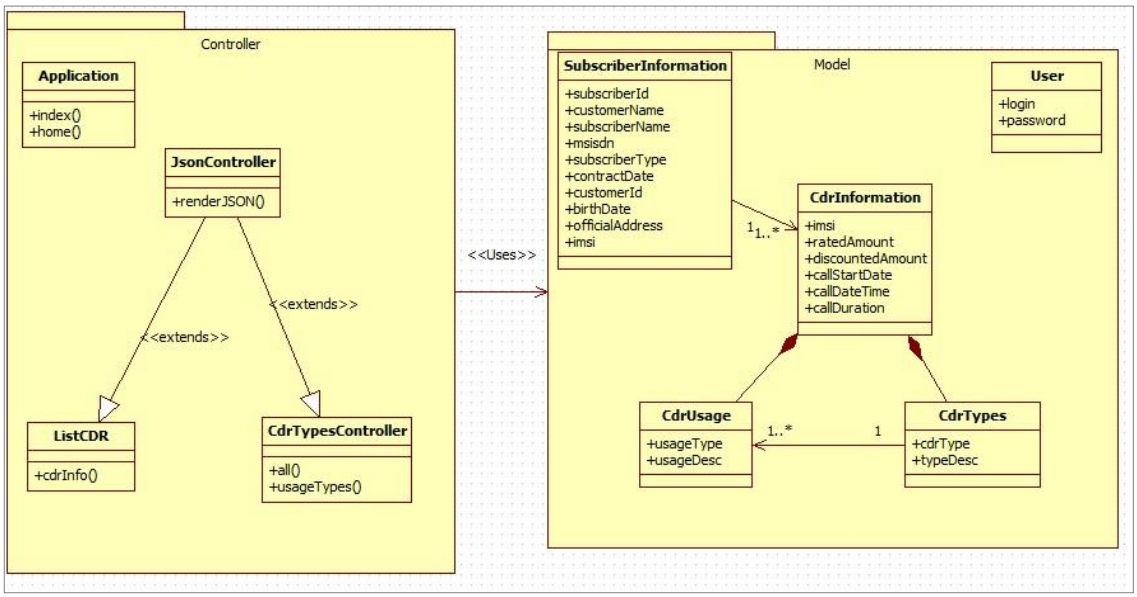

Figure 6 Class diagram

The class diagram (figure 6.) has been divided into two packages corresponding to the MVC support provided by the framework. Representation of classes of the model and controller layers of the application, connections and dependencies are visible. The controller package responds to events and processes them. It may also apply changes to model objects. The model classes represent data stored in RDBMS tables.

The controller package has 4 classes. The application class (renders main site) extends the controller class provided by the Play! framework and is independent from the other four present in package. JsonController is extended by CdrTypesController and ListCDR (listing CDR types and CDRs, respectively). JsonController should be extended by each controller that needs to render data in JSON format.

The user class is used for the authentication, it contains only two fields login and password (as it is object mapping of table users). It is independent from all other classes. SubscriberInformation class contains all the fields related to basic information of subscribers like subscriberName, subscriberId etc. This class has one-to-many relation to CdrInformation class.

In this package there are three classes which are for call detail records and are connected with each other through different relations. CdrInformation is responsible for dealing with specific data about call detail records. The other two classes CdrUsage and CdrTypes have composition relation with CdrInformation class. The CdrTypes class has data about different types of transaction that is subset of CdrInformation class. Similarly CdrUsage class has composition relation with CdrInformation based on usage types of transactions. The relation from CdrTypes to CdrUsage is one-to-many showing the type of transaction present in CdrTypes. The package has a generalization connection with the class Controller package.

4.3 Communication Overview

For this project an efficient communication schema decreasing the response time is one of the key goals. To provide the agent with a more responsive interface and to avoid long page load times during which an agent cannot do anything, it has been decided that all subscriber data will be requested using AJAX. For each subscriber's data there will be separate AJAX requests to the server. When the agent makes an initial request to access the main PARTool page the server will check to see if he is logged in and redirect him to a login page if necessary.

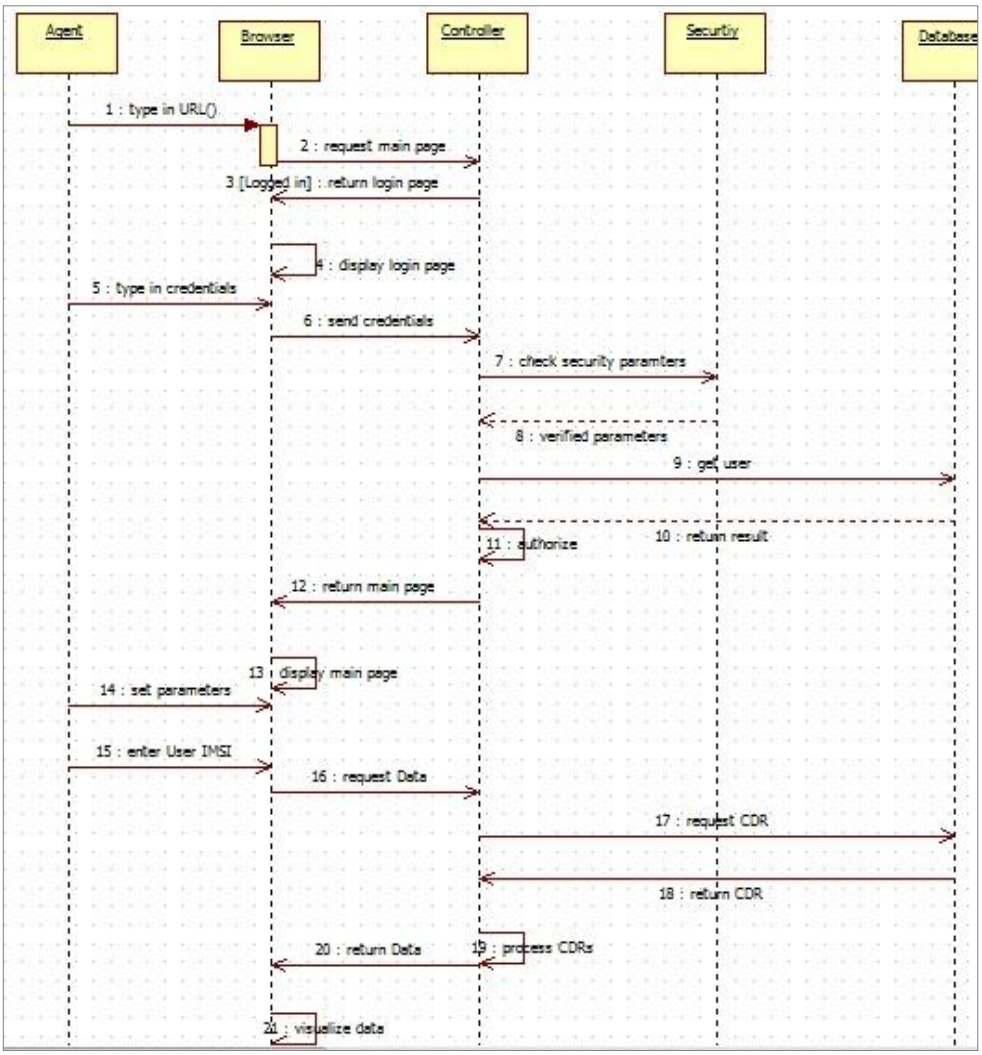

Figure 7 Sequence diagram

Figure 7 shows a sequence diagram which visualizes application flow considering specific use cases. It contains five main objects that will be used in the application. The agent will be responsible for giving input by using a browser, and then the browser will send the request to the controller who is core of the project. The controller will in turn process the request and send it either to security object or will query the database. Security module is for authentication which is provided by play! web framework. In both use cases the web server needs access to database and browser is used to display the processed information.

5. Approvals

Name	Title	Date yyyy-mm-dd	**Signature**
Marin Orlić	Supervisor		