SE31520 Assignment: Car Insurance System

Edgar Ivanov
edi@aber.ac.uk
Department of Computer Science, Aberystwyth University

December 9, 2013

Contents

Introduction

The assignment task was to implement the prototype system which would allow a customer to request a price for the car insurance. We were required to write two applications: the first one is so-called "underwriter" application which represents insurance company and the second one is broker application which would usually collect the quotes from the different insurance companies. For this assignment, however, the task was slightly simplified and broker needed to collect the quote from one insurance underwriter only.

I have developed "broker" and "underwriter" applications using different technologies: for the first one PHP was used and ROR was utilized to develop the second one. Broker application communicates with the underwriter using HTTP protocol and does it in a RESTful way. It is a great example of the interoperability, when applications can communicate despite the fact that they are written in different programming languages and are running on the different platforms. In the following I will describe the designs of the broker and underwriter systems, the testing strategy that was used, followed by the self evaluation section and the conclusions I have come up with.

Architecture of the underwriter

The "Underwriter" application has been developed using Ruby on Rails and designed as a RESTful web service; it uses JSON for the representation of the content and data exchange. HTTP has been used for the communication and supports all the usual HTTP methods like GET, PUT, PATCH, POST, DELETE for the resource creation, deletion etc. In the beginning I tried to implement XML support for the representation exchange but faced some issues which I couldn't overcome. The further research conducted on the data formats like XML, JSON, YAML gave me the reasons to believe that JSON would be the best choice since it is lightweight, human-readable and it easy to implement JSON support on the broker side.

Figure 1 shows the database design used to store the data about the customers. "Users" table holds the customer's information like name, surname, DOB etc. "Vehicles" table provides us with information about the car: the registration number, the mileage, the car value. It is linked to the main users table by the user_id field and has one-to-one relationship. Since there may be a few incidents that would result in a claim I decided to store them in the separate "driver_history" table; this table is linked to the customer with the user_id field and holds the information about the incident's date, the value claimed etc. It has one-to-many relationship with the user's table. "Addresses" table contains the information about the user's address: the street name, the postcode, the country etc.; it has one-to-one relationship since each user can only have one address registered in the system. "Quotations"

table holds the quotes for each user and is linked to the user's table by user_id field.

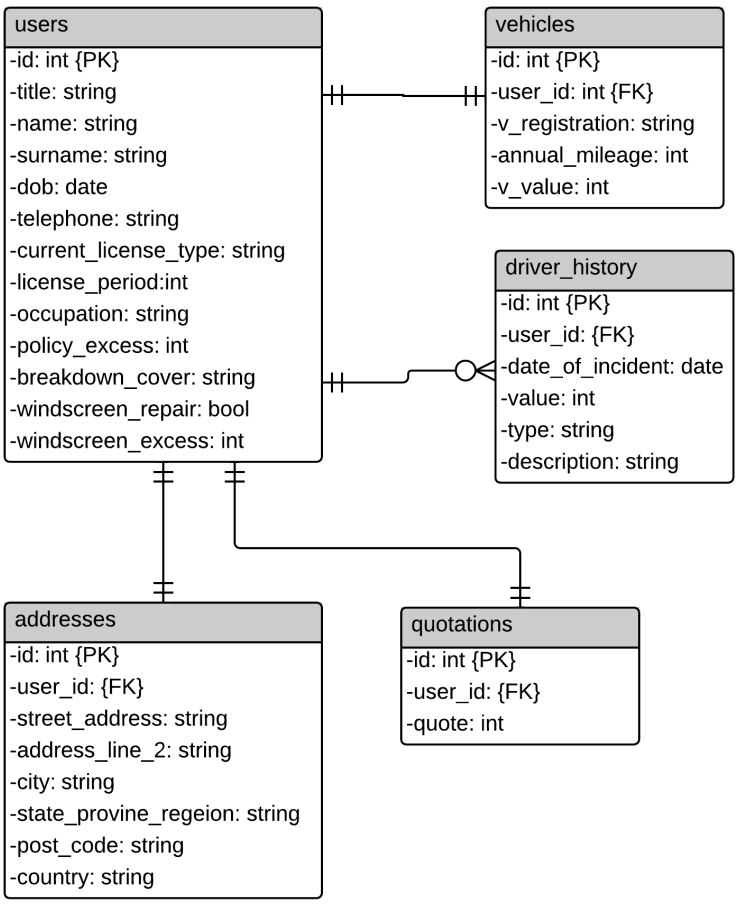

Figure 1: Database Design

On the figure 2 there is a class diagram of the underwriter application. The idea is to use it as a business-to-business application; instead of generating HTML pages the interaction is done using the data formatted in JSON and is expected to be used by automated clients. There is nothing under the "view" in class diagram since it only responds to the data encoded in JSON format (HTML support was left for the debugging purpose to see what is held in the database, but it would be removed in production mode). To get a quote broker submits PUT request to the */users/new.json* address with the user data, then users controller handles this request and sends back the ID assigned to the newly created user encoded as JSON. I used the ID field in "Users" table as the unique reference number for the later quote retrieval. ID fields are handled by the rails and are guaranteed to be unique across all the users: it is one of the useful ways to identify a user in the future when the quote needs to be redisplayed.

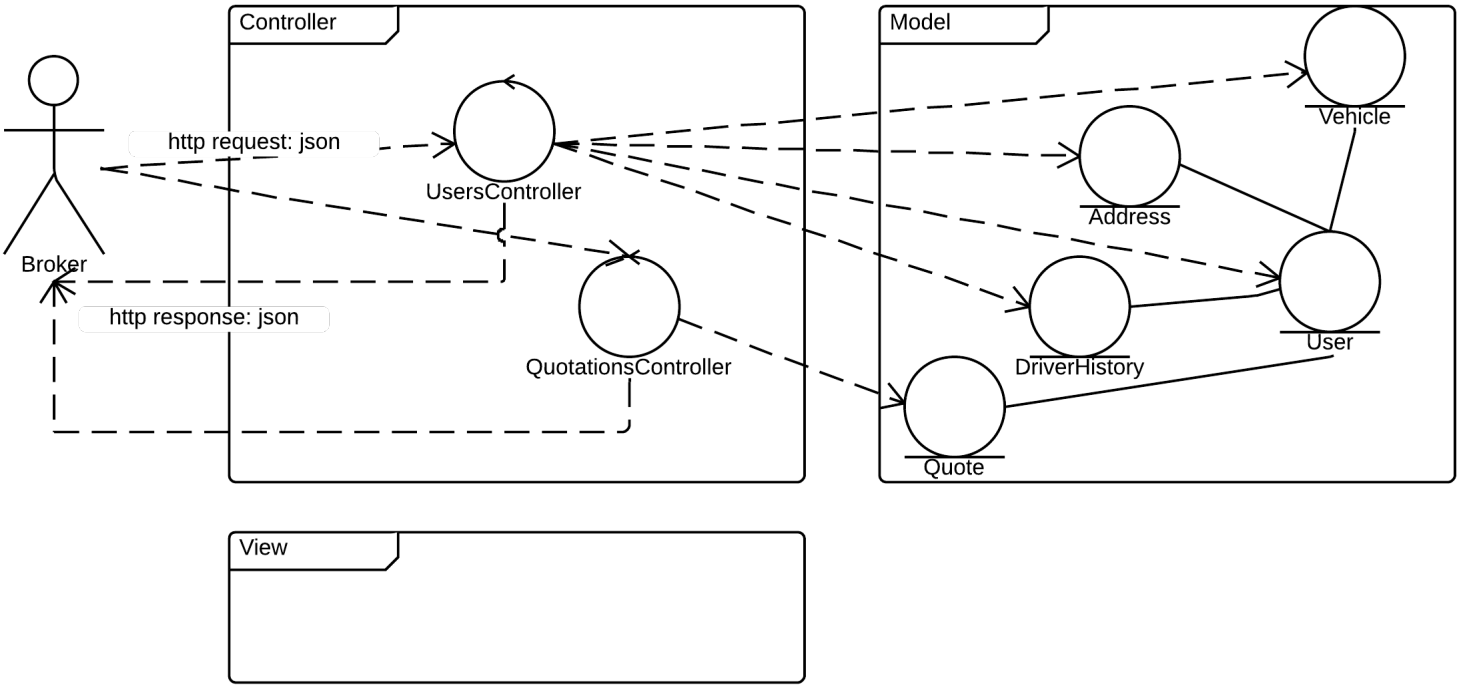

Figure 2: Class Diagram

Architecture of the RESTful broker

The broker application was written as a web based application with the PHP support. It also uses cURL command line tool; it seems to be the only way to send JSON encoded data to the specific URL in the PHP. cURL is produced by the cURL computer software project and allows data to be transferred using various protocols [1]. My server side scripting experience is limited to the use of SSI in HTML web pages. The fact that PHP is considered to be one of the most popular languages used for the server side coding [2] gave me the reasons to believe that it was worth to try to use it. I had no experience in writing PHP code; this assignment was a great opportunity to make the first steps and learn to use it. The online tutorials have helped me to understand the basic concepts and start implementing my broker application.

The broker web application provides the opportunity for the potential customer to request a quote premium from the underwriter application (use case diagram is presented on the figure 3). The customers interact with the broker only via HTTPS and the HTML formatted documents are transmitted to them. On the web page itself a customer needs to fill in and submit a form; on the next page a user is presented with the cost of the car insurance. This quote can be saved for the later retrieval; a customer is presented with the unique number which can be used to retrieve it. It also provides a customer with the link for a web page where the unique number can be entered and the quote will be redisplayed. The forms used on my web site were generated by the online form generator tool and modified accordingly to suite my needs. PHP is used to get the data from the forms and later on when building JSON objects containing customer data. With the help of cURL the broker application can use the standard HTTP methods to communicate with the underwrite application. To get a quotation the broker preforms the PUT request to the */users/new.json* address and receives ID number assigned to that user in the database. The broker then uses the ID received to retrieve the quotation from the underwriter using GET request and displays it to the user. On the figures 4, 5, 6, 7 and 8.

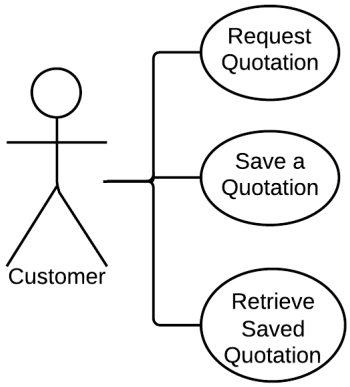

Figure 3: Use Case Diagram

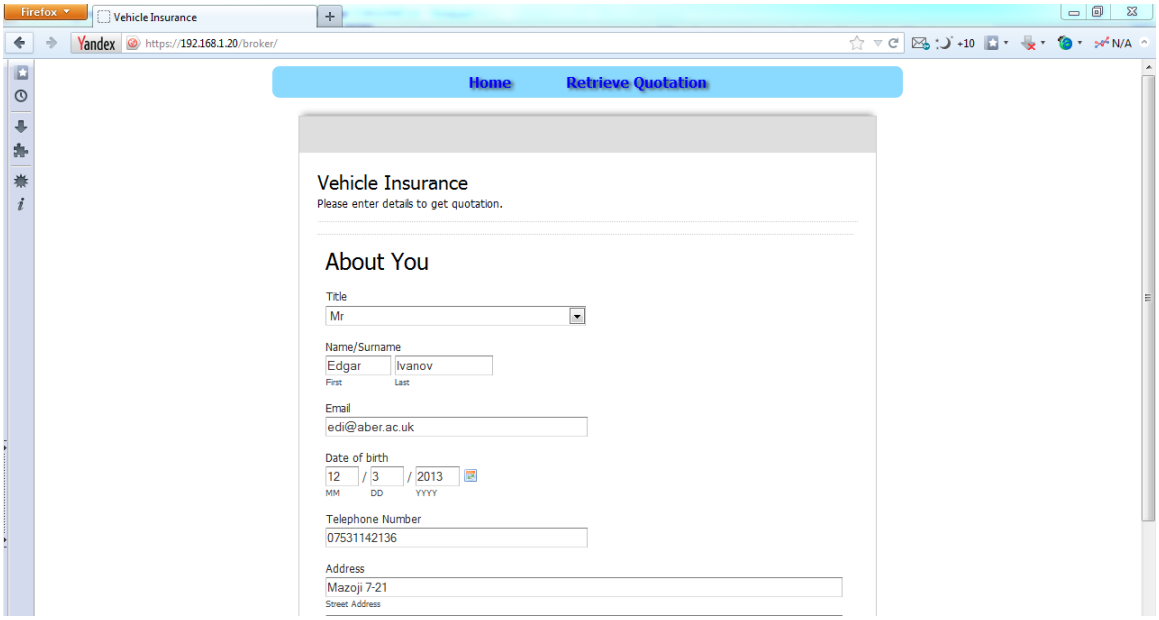

Figure 4: Broker Home Page

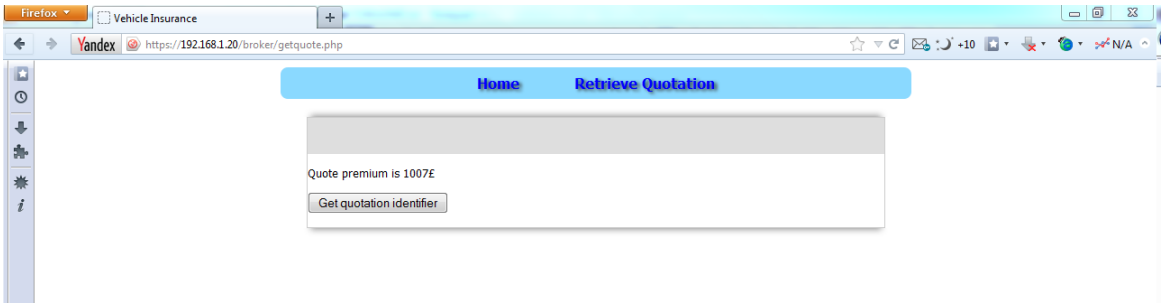

Figure 5: Quote premium displayed

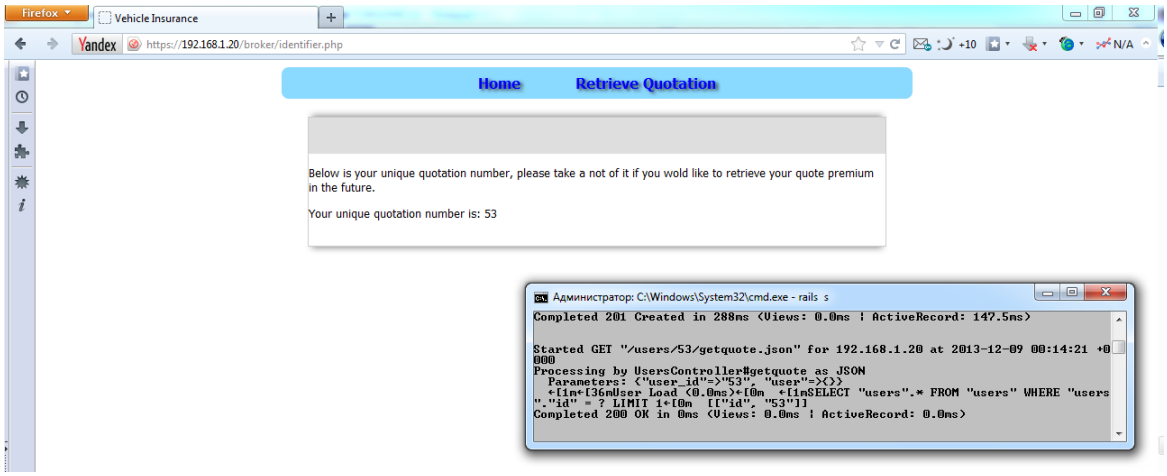

Figure 6: Quote Identifier

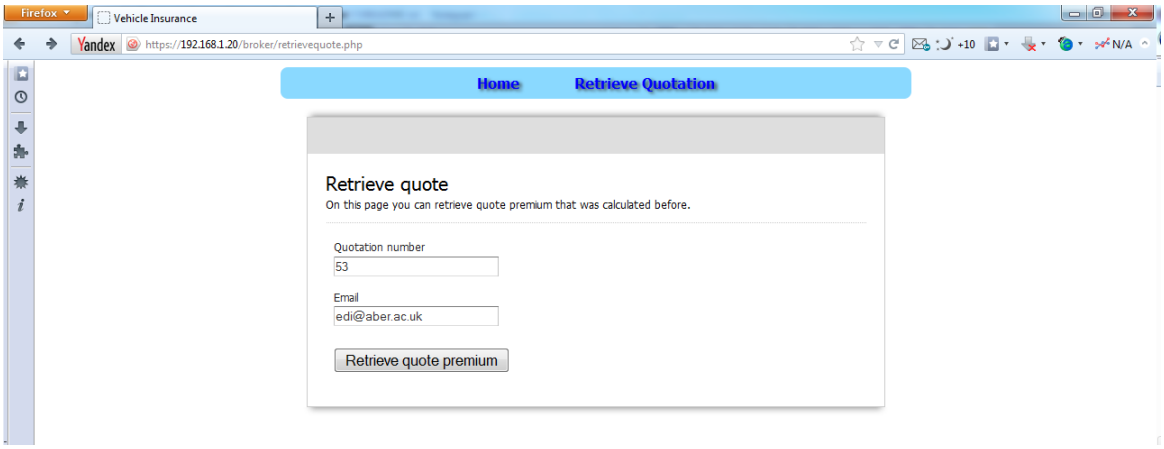

Figure 7: Page to retrieve quote premium

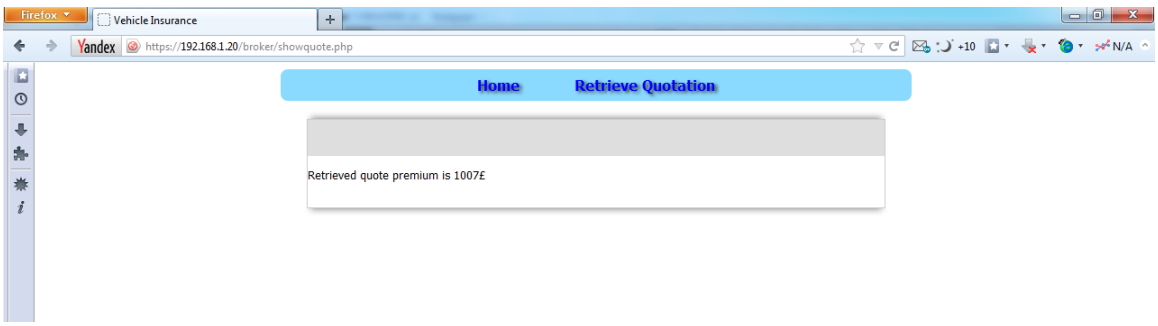

Figure 8: Retrieved quote premium

Test strategy

Write a section on your test strategy. IMPORTANT: Provide a screencast of your underwriter application and RESTful broker client working (some free screen-casting tools can be found online). This

must focus on the broker to underwriter interworking and be no longer than five minutes long.

ID	Requirement	Description	Inputs	Expected outputs	Pass/Fail	Comments
1	FR1	Request a quotation	Standard user details are entered	Display page with quote premium	P	
2	FR2	Save a Quotation	"Get quote identifier" button is clicked	Display page with unique identifier	P	
3	FR3	Retrieve a Saved Quotation	Identifier and user email are entered	Redisplay page with quote premium	P	

Self-evaluation

Write a self-evaluation section. Say what mark you should be awarded and why. Say what you found hard or easy, and what was omitted and why. Provide an analysis of your design and the appropriateness, or otherwise, of the technologies used.

Bibliography

[1] cURL, 2013. cURL. [Online] Available at:`http://curl.haxx.se/`, [Accessed 7 December 2013].

[2] W3Techs - World Wide Web Technology Surveys, 2013. Most popular server-side programming languages. [Online] Available at:`http://w3techs.com/`, [Accessed 7 December 2013].