

# **Car Insurance System**

## **SE31520 Assignment 2013-14**

### *Report*

*Author:* Stoyko Abrashev ([sta9@aber.ac.uk](mailto:sta9@aber.ac.uk))

Department of Computer Science, University of Aberystwyth  
Tuesday, January 7, 2014

## Declaration of originality

In signing below, I confirm that:

- This submission is my own work, except where clearly indicated.
- I understand that there are severe penalties for plagiarism and other unfair practice, which can lead to loss of marks or even the withholding of a degree.
- I have read the sections on unfair practice in the Students' Examinations Handbook and the relevant sections of the current Student Handbook of the Department of Computer Science.
- I understand and agree to abide by the University's regulations governing these issues.

Signature ..... (Stoyko Abrashev)

Date ...**07.01.2014**.....

## Consent to share this work

In signing below, I hereby agree to this dissertation being made available to other students and academic staff of the Aberystwyth Computer Science Department.

Signature ..... (Stoyko Abrashev)

Date...**07.01.2014**.....

# Contents

## Introduction .....

## Online Insurance Underwriter

Technologies .....

Architecture & Design Diagrams .....

REST API .....

Rationale .....

## RESTful Broker Client

Technologies .....

Architecture & Design Diagrams .....

RESTful Interoperability .....

Rationale .....

Resultant System .....

## Testing

Strategy .....

RESTful Client Testing .....

Results .....

## Evaluation

Approach and Technologies .....

Problems & Compromises .....

Areas of Expansion .....

Output & Assessment .....

## References

## Introduction

The purpose of the present paper is to describe the process of designing and implementing of a prototype system to be used by customers for requesting quotations for a car insurance policy.

As the system is a prototype, the architecture is quite simple: the main components are the customer, a web application that acts as an online insurance broker, and a single insurance underwriter application. The user interacts with the online broker through HTTPS in order to ensure the security of the data transfer. HTTPS is not used between the broker application and underwriter application.

The online broker has been written using PHP + HTML5 + JAVASCRIPT/AJAX, and it communicates with the insurance underwriter application via the REST protocol. The underwriter application has been developed upon Ruby on Rails. All the technologies used, high-level architecture, design solutions and diagrams as well as testing strategy and evaluation of results for both applications are presented in the respective sections.

## Online Insurance Underwriter

### Technologies

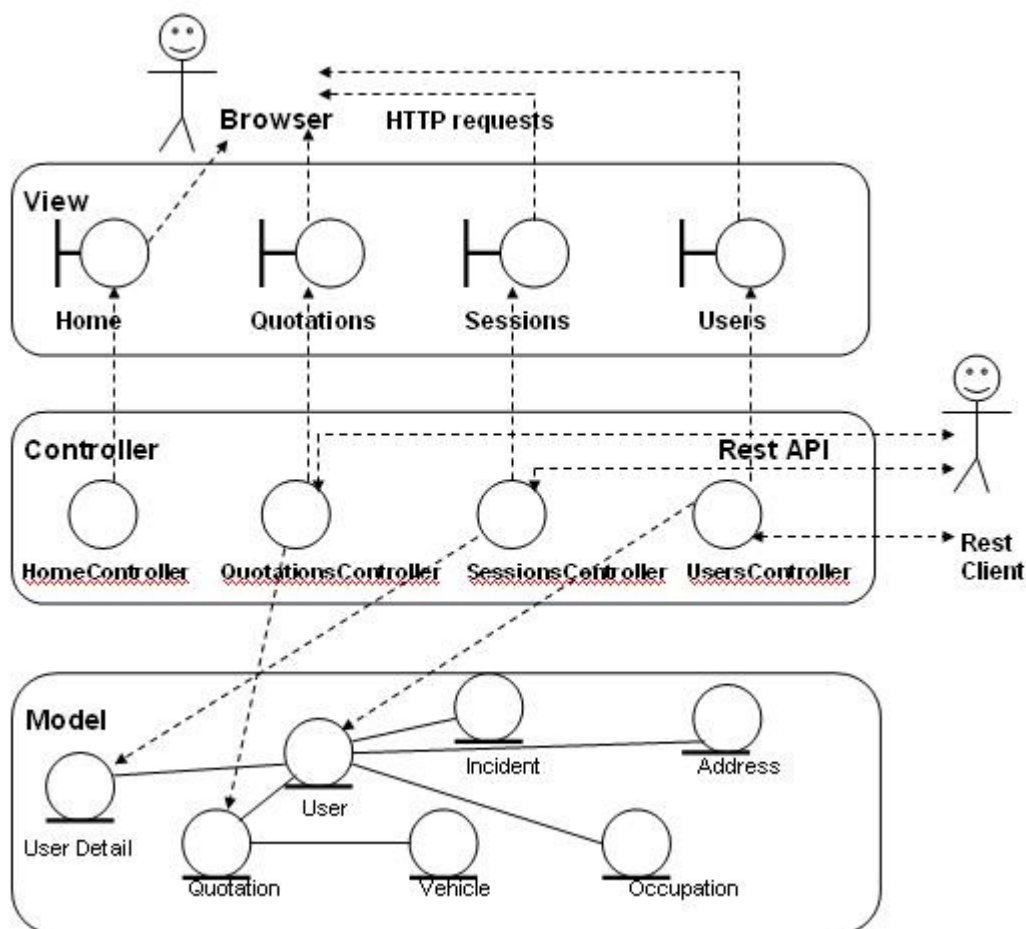
The technology used for the building of the insurance underwriter application is Ruby on Rails. The gems applied are listed below:

```
'rails', '4.0.0' – Rails version  
'thin' – Web Server  
'simple-navigation'  
'sqlite3' – Database  
'will_paginate', '>= 3.0.pre'  
'sass-rails', '~> 4.0.0'  
'uglifier', '>= 1.3.0'  
'coffee-rails', '~> 4.0.0'  
'therubyracer', platforms: :ruby  
'jquery-rails'  
'jquery-ui-rails'  
'turbolinks'  
'jbuilder', '~> 1.2'  
'bcrypt-ruby', '~> 3.0.0'  
'unicorn'  
'capistrano', group: :development  
'debugger', group: [:development, :test]
```

## Architecture & Design Diagrams

The architecture of the application has been constructed using a model-view-controller structure. The models, views, and controllers have been developed as independent functional blocks. This approach facilitates writing and maintaining the code, as every action and concept is located in one separate place only.

### Current overall class design



The given diagram can be described as follows:

The login process is almost the same as that for the CSA application already discussed, with a few changes.

The underwriter application's admin accesses the browser's home screen in order to login. He presses the login button, thus creating a GET request to the SessionsController's #new. The response is the login form with two fields and a submit button. He fills in his details – an email ([admin@abv.bg](mailto:admin@abv.bg)), and a password (**taliesin**); he then creates a POST request to the SessionsController's #create. SessionsController calls the authenticate method in the User

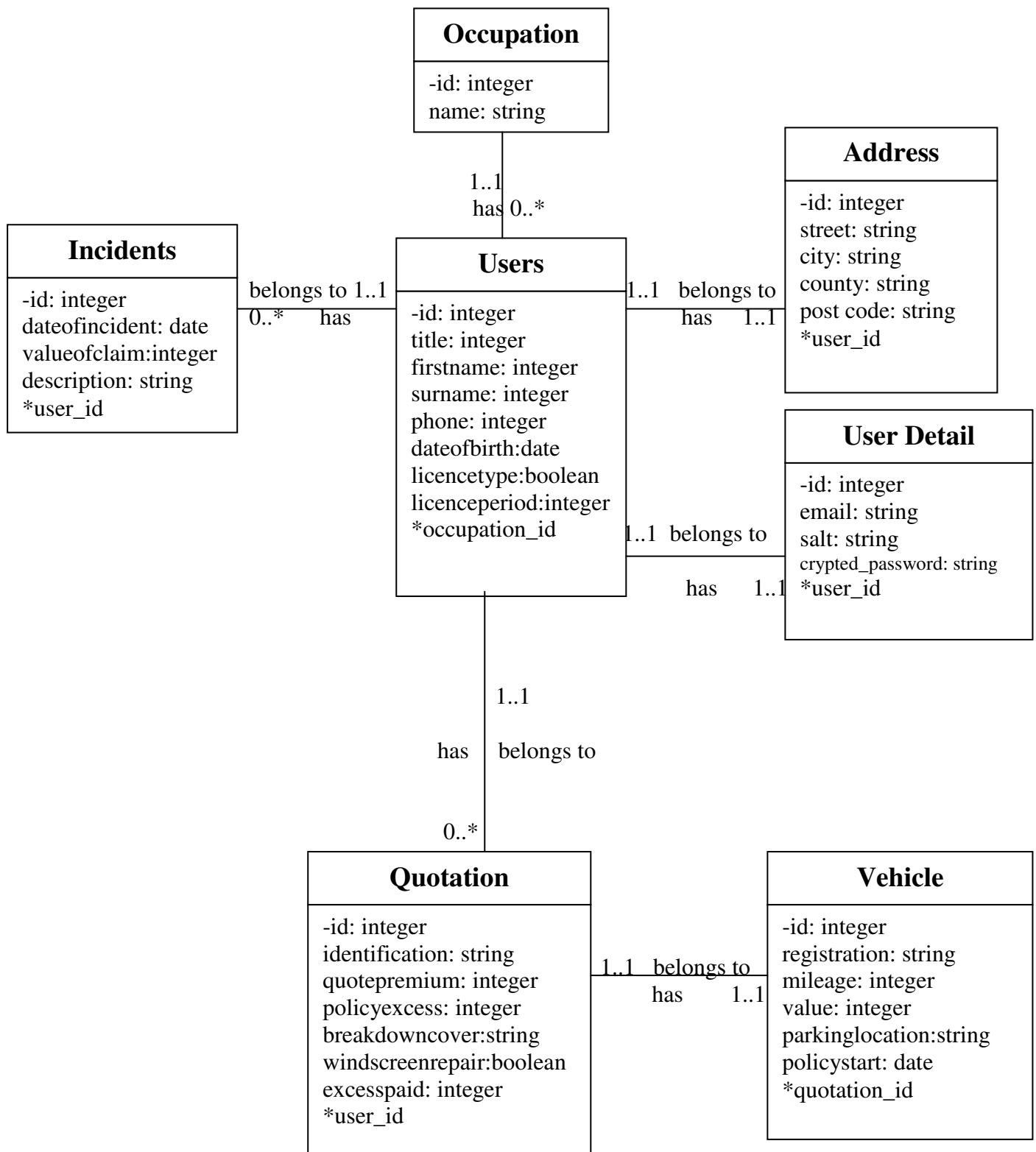
Detail model. The model searches for the given email address in the User Detail table of the database. If such an email exists it crypts the given password by means of *salt* from the database and checks if the expected password is the same as that from the database. After that, if both passwords are identical, a new session is created and the page redirects us to the home screen. The login process is finished.

If we select Users by pressing the navigation button, then a GET request is issued to the UsersController's method #index that returns pages with users (three users per page) showing all the information about the users. There are two links: the first one is Show. By a GET request, it calls the UsersController's method #show with the respective user id, which makes queries to the User Model to select all information from the users, incidents and address tables, and then the show.html.erb file is rendered in the app/views/users/ folder. The second link issues a confirm message asking OK or Cancel. In case of OK, it makes a GET request to the UsersController's method #destroy with the corresponding user's id. It destroys the user and renders the users' index page.

An analogous process runs when selecting Quotations from the navigation menu. There is also an Account tab where the admin can see and change his details. The admin is able to see the users and quotations ordered by the last id on the first page. For the pagination I use will\_paginate gem. If the admin wants to see the users or quotations in a JSON format, he must type in the location bar users.json or quotations.json.

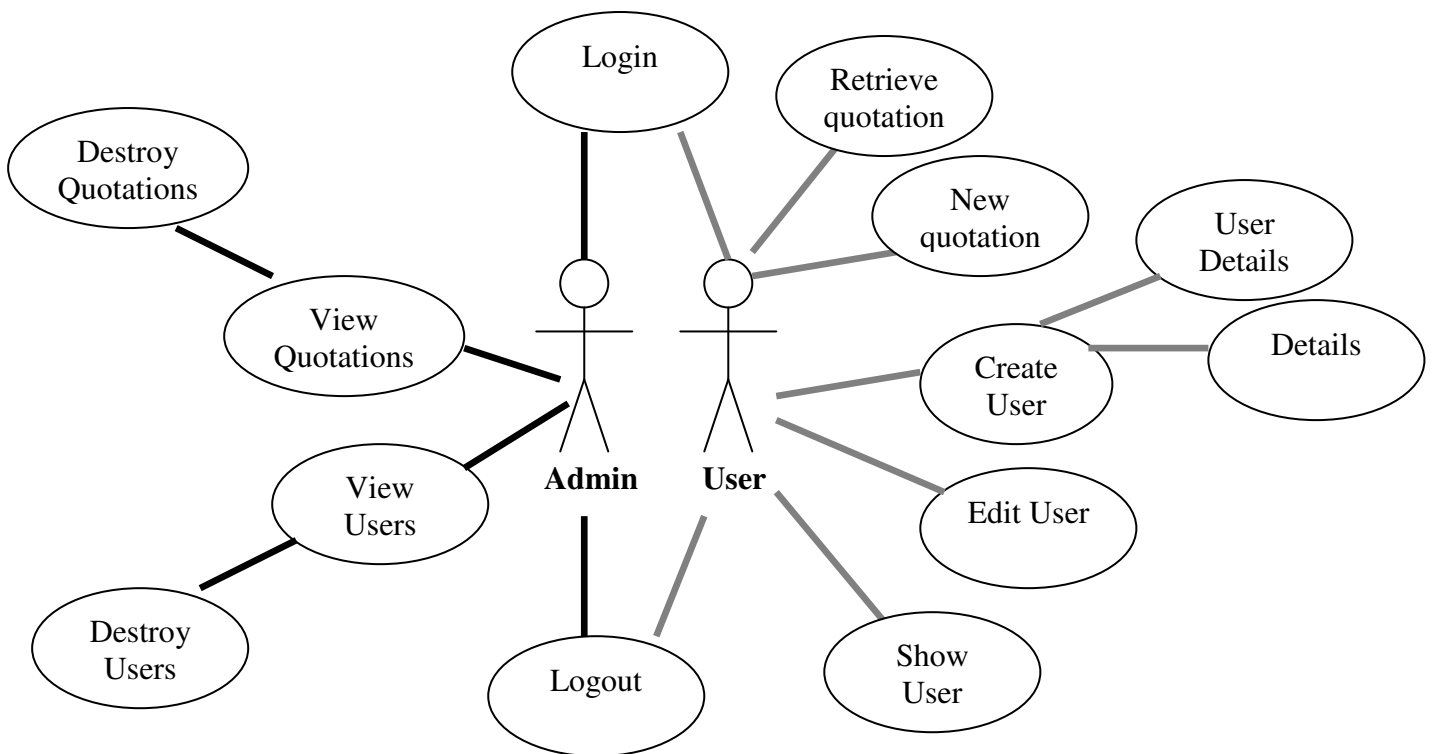
The application controller is almost the same as the controller in the CSA example.

## Current database design



Tasks of admin are shown in the screencast and screenshots as well.

The particular tasks of users and the admin that suit the use cases are shown below:



## REST API

The use of Rails facilitates integrating the RESTful interface into the code. The underlying idea is that a certain set of verbs (conforming to HTTP methods such as GET, POST, hidden fields in the forms with PUT, PATCH, DELETE) is used in order to operate on a large set of nouns (resources named using URLs). These are the routes in underwriter.

```

C:\Sites\cis>rake routes
Prefix Verb URI Pattern Controller#Action
search_quotations GET /quotations/search(:format) quotations#search
quotations GET /quotations(:format) quotations#index
POST /quotations(:format) quotations#create
new_quotation GET /quotations/new(:format) quotations#new
quotation GET /quotations/:id(:format) quotations#show
DELETE /quotations/:id(:format) quotations#destroy
users GET /users(:format) users#index
POST /users(:format) users#create
new_user GET /users/new(:format) users#new
edit_user GET /users/:id/edit(:format) users#edit
user GET /users/:id(:format) users#show
PATCH /users/:id(:format) users#update
PUT /users/:id(:format) users#update
DELETE /users/:id(:format) users#destroy
session POST /session(:format) sessions#create
new_session GET /session/new(:format) sessions#new
DELETE /session(:format) sessions#destroy
login POST /login(:format) sessions#userlogin
home GET /home(:format) home#index
root GET / home#index
GET /*a(:format) errors#routing
    
```



### Server details

```
C:\Sites\cis>thin start -p 3001 --ssl
>> Using rack adapter
>> Thin web server (v1.5.1 codename Straight Razor)
>> Maximum connections set to 1024
>> Listening on 0.0.0.0:3001, CTRL+C to stop
```

### Encryption-screenshot from Mozilla Firefox

#### Technical Details

**Connection Encrypted: High-grade Encryption (TLS\_RSA\_WITH\_CAMELLIA\_256\_CBC\_SHA, 256 bit keys)**

The page you are viewing was encrypted before being transmitted over the Internet.

Encryption makes it very difficult for unauthorized people to view information traveling between computers. It is therefore very unlikely that anyone read this page as it traveled across the network.

### Rationale

#### 1. REST Format

The admin can access the pages in HTML or JSON format. Requests from Broker to Underwriter return JSON which then is used for populating form and page elements.

#### 2. SessionsController

A separate function has been included especially for logging in. A request is sent to this function from the broker insurance application to the underwriter application with an e-mail and a password as parameters. After that the Sessions Controller calls the authenticate method in the User Detail model. If it does not return an object, the response to the request is "null", otherwise it returns a response {id of the user} in a JSON format and redirects to the main PHP page.

#### 3. QuotationController

Quotation premium is calculated by the formula:

**Breakdowncover** can be 0, 2, 3 or 4

**Windscreenrepair** is 30 if is clicked Yes radio button

**Discount**=0.30%(0.70) if there are no driver incidents

**Policy excess**=16% constant

**Excess paid**=5% constant

**Premium**=((Vehicle value \* Breakdowncover)+Windscreenrepair)\*Discount

## RESTful Broker Client

### Technologies

The online insurance broker application is constructed using PHP + HTML5 + JAVASCRIPT/AJAX. The jQuery libraries, Ajax and basic Javascript have been used very efficiently and are well implemented in the web page as well as the HTML5 language. I used JSON for storing and exchanging information.

All instructions and details on how to run the application are given in the accompanying README.txt.

## Architecture & Design Diagrams

The architecture model consists of a presentation layer and an application layer. As long as it is a prototype system, no persistent storage is required for this application, as it serves as a front-end only to the underwriter application.

The presentation is the way the broker application is shown on user's browsers. At start this layer makes a request to the server which returns the whole HTML page. It is the HTML 5 webpage which can be accessed from all known browsers both on desktop or mobile devices. The page is styled using the CSS 3 style sheets technology. In addition I have used JQuery for making requests to the application layer and JQuery User Interface libraries for a better look and a feel tabs capability.

Through the web front-end, the RESTful client provides users with access to the functionality permitted by the underwriter application. But it is not valid for the case of administrators. The administrator may login and use the application, but their experience will be limited, as they will be able to see and edit their profile only.

Users logging in: after the main page is loaded, a GET request is made with the getJSON function to the broker server. Then in the broker application, a GET request is made to the underwriter by using an e-mail, a password and an id, which returns the user data in a JSON format. The full user information is placed in the two tabs Profile and Edit profile.

## RESTful Interoperability

The two applications have been successfully integrated by means of REST-based web-services.

### 1. REST Format

The JSON format is used by the client for RESTful transactions with the underwriter application. JavaScript is used to take the JSON response of the GET requests, and JQuery is used for updating page elements with new data. I used also a JSON decode function in the PHP file when an user is logging in and the user id is returned and stored in the PHP session variable for later access.

### 2. REST Services & Entities

The client is able to communicate with the Underwriter application with making POST and GET requests via Ajax(\$.post or \$.getJSON JQuery functions) and PHP. PHP ones are made with **file\_get\_contents(\$url, false, \$context);** and **stream\_context\_create(\$options);** build in library functions. Along with user details we pass as header of the requests "Content-type: application/x-www-form-urlencoded\r\n"

Login is done with request to [https://\\$user:\\$pass@".\\$GLOBALS\['underwriter'\]. "/login.json](https://$user:$pass@)

which returns the used id saved in Session variable for later use.

### 3. CRUD

There are 5 tabs (Home, New quotation, Retrieve quotation, Profile and Edit profile). In each tab there are different HTML5 elements or forms and particular information depending on the tab.

### 4. Security & Authentication

All authentication is done through HTTPS basic authentication. The user has to authenticate every time so passing the email and password via secure connection is essential. Basically after login they are saved in session variables and used with every request made in the connect.php file to the target resource. For example a request for taking user's information is: `https://email:password@UnderwriterIP:3001/users/userid.json`

### 5. Resource Pagination

In the broker application there is no pagination because we do not need one. But in the underwriter application for the admin listing users or quotations I use will\_paginate gem demonstrated in the CSA application and other worksheets.

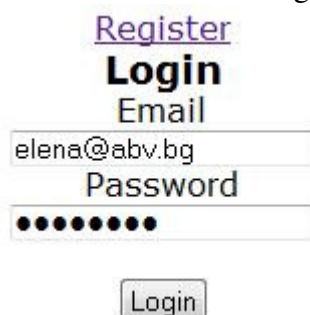
## Resultant system

Below are shown screenshots as evidence of the resultant system running. I am enclosing also screencasts to demonstrate how the underwriter application and RESTful broker client are working together via secure HTTPS connection on both sides of them. All screenshots are in a folder in cis-broker.

Server encryption:

Technical Details  
**Connection Encrypted: High-grade Encryption (TLS\_ECDHE\_RSA\_WITH\_AES\_256\_CBC\_SHA, 256 bit keys)**  
The page you are viewing was encrypted before being transmitted over the Internet.  
Encryption makes it very difficult for unauthorized people to view information traveling between computers. It is therefore very unlikely that anyone read this page as it traveled across the network.

Screenshots of user using the page:



The screenshot shows a web form with the following elements: a purple 'Register' link, a bold 'Login' heading, an 'Email' label above a text input field containing 'elena@abv.bg', a 'Password' label above a text input field filled with black dots, and a 'Login' button at the bottom.

After clicking Calculate premium and discount are showed

Signed in as  
**elena@abv.bg**  
[Logout](#)

[Request Quotation](#) [Retrieve Quotation](#) [Profile](#) [Edit Profile](#)

### Request a quotation

Your Discount is: 30%  
 Premium: £305  
 Policyexcess - 16%  
 Breakdowncover  
  
 Windscreenrepair  
☐ YES ☒ NO  
 Excesspaid - 5%  
 Vehicle:  
 Registration:   
 Annual mileage:   
 Estimated value:   
 Parking Location:   
 Start of policy:

[Calculate Premium](#) [Reset form](#) [Save Quotation](#)

After clicking Save request is made to connect.php where another request is made to quotations.json file in underwriter and response is identification number of just created quote. Identification is generated by secure random library in quotation.erb file.

[Request Quotation](#) [Retrieve Quotation](#) [Profile](#) [Edit Pr](#)

### Request a quotation

Your identification number is: UIOWY\_lIaQ  
 Please write it down for later retrieve  
 Your Discount is: 30%  
 Premium: £305  
 Policyexcess - 16%  
 Breakdowncover  
  
 Windscreenrepair  
☐ YES ☒ NO  
 Excesspaid - 5%  
 Vehicle:  
 Registration:   
 Annual mileage:   
 Estimated value:   
 Parking Location:   
 Start of policy:

[Calculate Premium](#) [Reset form](#) [Save Quotation](#)

Signed in as  
**elena@abv.bg**  
[Logout](#)

[Retrieve Quotation](#) | [Profile](#)

[Retrieve](#)

**Quotation Premium:**  
£305

**Date Calculated:**  
about 18 hours ago

**Policy excess:** 16%

**Breakdown cover:**  
At home

**Windscreen repair:**  
Not included

**Excess Paid** 5%

**Insurance for Vehicle:**

**Registration:**  
km52pgb

**Annual mileage:**  
2000 kilometers

**Estimated value:**  
£14500

**Parking location:**  
Garage

**Start of policy:**  
about a month from now

Signed in as  
**elena@abv.bg**  
[Logout](#)

[Quotation](#) | [Profile](#)

**Title:**  
Miss

**Firstname:**  
Elena

**Surname:**  
Petrova

**Phone:**  
7698356241

**Dataofbirth:**  
1976-11-12

**License type:**  
PROVISIONAL

**License period:**  
2 years

**Occupation:**  
Social worker

**Driver History:**  
0 incidents

**ADDRESS:**  
Bond Street  
Manchester  
UK  
sy23 3pg

## Testing

### Strategy

The strategy of testing involves completing of a few Rails unit and functional tests of the insurance underwriter application. The online insurance broker application has been tested in the process of functioning where a screencasts are attached as evidence.

### Results

#### Unit tests



```
# Running tests:
```

```
..
```

```
Finished tests in 0.286000s, 6.9930 tests/s, 17.4825 assertions/s.
```

```
2 tests, 5 assertions, 0 failures, 0 errors, 0 skips
```

## Functionals tests

```
14 tests, 8 assertions, 7 failures, 6 errors, 0 skips
```

# Evaluation

## Approach and Technologies

In my work I adhered strictly to the specified requirements, namely: building the underwriter application upon Ruby on Rails, using a RESTful interface for receiving and sending the application data, building the online insurance broker application using a technology at the student's choice.

I have selected PHP + HTML5 + JAVASCRIPT/AJAX not only because of its suitability, but considering also my knowledge of technologies, tools and programming languages. PHP is especially suited to server-side web development where PHP runs on a web server.

## Problems

Some problems occurred when I was trying to do the login functionality and then decided to do another function in sessions controller which will return the id of the user if email and password are correct. Another problem was the requests itself. Because when using basic authentication the email and password have to be passed as parameters every time so I make 2 requests usually. First one with Ajax to connect.php file where with session variables email and password another request (Post or Get) is made to underwriter and everything is working very good now.

## Output and Assessment

In my opinion, I managed to design and implement a functional prototype system that allows the customer to obtain car insurance quotations. It consists of three main components and features only a few key functions: requesting a quotation, saving a quotation and retrieving a saved quotation.

It is my belief that by creating the insurance underwriter application I applied successfully in practice my knowledge of Ruby-on-Rails technologies, demonstrating reasonable use of the object-relational mapping layer ActiveRecord, which functions as a part of Rails implementing the application's model, Rails controllers, view templates, and REST-based web services. Trying to catch hold of Ruby-on-Rails tricks was a real fun and exciting experience with the book *Agile Web Development with Rails* written by Ruby, Sam et. al. When building the underwriter application upon Rails, it turned out that it is fairly easy to

integrate the RESTful interface into the code, as support is built-in.

## References

- [1] Ruby, Sam et. al. Agile Web Development with Rails. *The Pragmatic Programmers, LLC*.
- [2] <http://www.rubyinside.com/nethttp-cheat-sheet-2940.html>
- [3] <http://fczaja.blogspot.co.uk/2011/07/php-how-to-send-post-request-with.html>
- [4] <http://www.jonasjohn.de/snippets/php/post-request.htm>
- [5] <http://php.net/manual/en/function.stream-context-create.php>
- [6] <http://timeago.yarp.com/>
- [7] [http://guides.rubyonrails.org/active\\_record\\_querying.html](http://guides.rubyonrails.org/active_record_querying.html)
- [8] [http://guides.rubyonrails.org/form\\_helpers.html](http://guides.rubyonrails.org/form_helpers.html)