SE31520 Assignment 2013-14

Car Insurance System

Chris Loftus
Hand-out date: Tuesday 29th October 2013
Hand-in date: Monday 9th December 2013
Feedback date: Tuesday 7th January 2014
50% of SE31520 assessment

Introduction

Where this assignment states any exact requirements, you MUST follow them precisely, in other areas you may take decisions as you see fit but you must be prepared to document and justify those decisions.
The whole of your solution for this assignment is worth 50% of the marks for this module. The remaining 50% of the marks come from the exam in January.
This assignment should take in the order of 50 hours, however the exact figure depends on whether you undertook sufficient preparatory work and study (worksheets, reading and attending lectures) and on your software development abilities.

Overall System Description

You are required to design and implement a prototype system that allows a customer to request a quotation for a car insurance policy. Given limited time for this assignment, many functions are omitted.
The system is intended to demonstrate how a new insurance broker application could be introduced, and link through to different insurance underwriters to obtain quotations and modifications to a policy. In this prototype your insurance broker application will only need to communicate with a single insurance underwriter application.
Figure 1 shows the high-level architecture of the required system. Two underwriter applications are shown, whereas you need only one. The online insurance broker is a web application that may be developed using a technology of your choice. For the prototype this web application has no persistent storage; it simply acts as a front to any number of online insurance underwriter applications. The underwriter application must present a RESTful interface that receives and delivers application data. This data can be transmitted in a representation format of your choice. In this simple prototype there is just one actor: the Customer. Clearly, in a real application other actors such as claims processors would be involved. The Customer only interacts via HTTP(S) with the broker. HTML and Javascript are the representation formats transmitted to the Customer. The underwriter application maintains data about customers or potential customers.
Where ambiguity in the specification exists then you are at liberty to act in the role of the client and provide your own interpretation. Make sure that you justify any decisions made. The key requirements of the system are as follows.

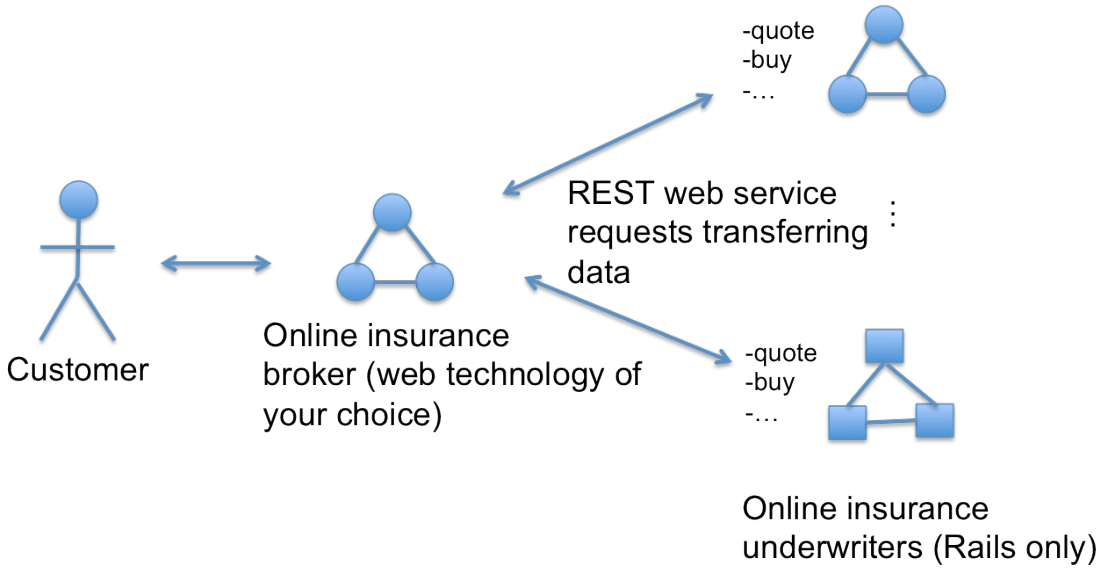

Figure 1: Requirements on architecture

Functional requirements

Given that this is a prototype and the limited time for the project, only three functional requirements are specified. There are no requirements for amending policy details, making a claim or calculating the insurance premium.

FR1. Request Quotation. A user acting as a *Customer* is able to request a quotation for a particular vehicle. The user will enter the following details:

- *Person Insured*. This will include the information that is listed in the Data Requirements section below.
- *Vehicle Details*. This will provide basic information about the vehicle. The information that must be entered is listed in the Data requirements section below.
- *Driver History*. This will record information about the driver's no-claims discount and any information about accidents in the past 3 years. A list of the required information is listed in the Data requirements section below.
- *Desired Policy Features*. The user will be able to indicate a preference for certain policy features, including breakdown cover and windscreen repair cover.

This quotation information should be stored in the Rails database of the underwriter to allow later access.
The function should calculate and return the quote premium. This quote premium and the date it was calculated should also be stored in the underwriter's database.

FR2. Save a Quotation. A Customer is able to save a quotation for later retrieval. The Customer is given a quotation identifier that will be used to retrieve the details at a future date.

FR3. Retrieve a Saved Quotation. A Customer is able to retrieve a saved quotation by entering their email and the quotation identifier. Once this has been entered, the quotation is redisplayed. Clearly, in a real application some way of identifying the underwriter provider would be required (perhaps as part of the quotation identifier). For this prototype you may assume a single provider.

Design constraints

DC1. Web access. System functions must be accessible via recent Information Services-based browsers.
DC2. Broker application technologies. You may decide which web application technology to use.
DC3. Physical separation of broker and insurance underwriter. To enable the loose coupling of different company applications, the broker and underwriter applications must run on separate computers with REST used to allow the broker and underwriter applications to communicate. Apply the ideas discussed during the lectures on REST and interoperability.
DC4. Security. For this prototype, HTTPS is required when the *Customer* interacts with the broker and sensitive data is being transferred. The use of self-signed certificates is acceptable. However, HTTPS is not required for the communication channel between the broker and the underwriter applications. Clearly, in a real system this would be essential.
DC5. Use of relational database and Rails ORM support. The underwriter application must use a relational database management system and the object-relational mapping support provided by Rails.

User interface requirements

Given that this is a prototype a slick GUI is not required. That said, don't use the default CSS provided by Rails.

Data requirements

The following sections list the information that the application should record.

Person

The following details should be stored for a person:

- Title (String or enumeration)
- Forename (String)
- Surname (String)
- Email (valid email address)
- Date of Birth (Date)
- Telephone Number (String)
- Address (Multiple parts, including street, city, county and postcode).
- Current License Type (Full or Provisional)
- License Period (Number of years that the person has held the license)
- Occupation (To be taken from a set list of job titles)

Vehicle

The following details should be stored for the vehicle:

- Vehicle Registration (String)
- Estimated Annual Mileage (Number)
- Estimated Vehicle Value (Number) Must be greater than 0.
- Typical Parking Location (String or Enumeration) Typical values will include 'In a Garage', 'On Driveway', 'On Street'.

- Policy Start Date (Date) The date and time that the policy starts.

Driver History

The following details should be recorded about driver history:

- Number of incidents that have resulted in claim in the last three years (Number)
- For each incident:
    - Date of incident (Date)
    - Value (Number) The total sum of the claim.
    - Type (String or Enumeration)
    - Description (String). A description of the claim.

Desired Policy Features

The following details should be recorded about the policy details:

- Policy Excess (Number) – This records the proportion of a breakdown claim that will be met by a customer. You are at liberty to decide the form this takes: e.g. discrete amounts such as £50, £100 or a continuous range, or percentages.
- Breakdown Cover (String or Enumeration) – Typical values: 'At Home', 'Roadside', 'European'. Also, *no breakdown cover* is a possibility, i.e. just support for windscreen repair cost.
- Windscreen Repair Cover: Yes or No, and the excess paid by the customer if Yes.

Documentation and program quality

Create a zip file and upload to Blackboard. The zip file must contain all source code, a README.txt file and a project report. The report must include appropriate screen shots/captures to show the extent to which your program runs. If there is no evidence of your program running then we will not award marks for that component of the assessment.

You must provide a system test table. See the following document for an example. http://users.aber.ac.uk/cwl/exampletesttable.pdf (also doc)

You are also required to submit a simple document, less than 10 sides of A4 paper, describing the design decisions you have made for each part of your system. In addition, the code must be internally documented with useful comments. Include a UML class diagram (or diagrams) of the program design. These can be drawn by hand if you wish.

In assessing your work, we will carefully consider issues such as layout, naming of variables, general readability, suitability of comments and so on as well as simple accuracy of the code. More information of detailed project requirements can be found in the Mark Breakdown section below.

Steps

Undertake the following steps:

- Study the above brief.
- Analyse the existing application. You will find existing design documentation in my Requirements slides (on Blackboard).
- Decide on the language you wish to use for the REST client. Ruby and Rails has support for web service clients (ActiveResource). A rather nice description

of this can be found here: http://ofps.oreilly.com/titles/9780596521424/activeresource_id59243.html

- Make sure you have implemented some examples of unit and functional (or integration, if you prefer) tests that test the application. Just provide a few examples for each model class and controller class. Note that you will find information online on how to pass session data (including authentication data) to the controller from a functional test[1].
- Write a report (five to eight pages of text with additional pages for screen shots, title page etc):
    - Write an introduction.
    - Write a section on the architecture of the underwriter application and rationale for decisions made. As part of this, produce a UML diagram(s) that shows the architecture of your application. The design diagram I used for the CSA application discussed in class might be a useful starting point. I drew mine using Powerpoint, but feel free to use another tool or even to draw neatly by hand!
    - Write a section on the architecture of the RESTful broker client, providing rationale for decisions made.
    - 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.
    - 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.

Burn a CD that contains the report, the Rails project and the screencast. Include a README.txt file that describes what I need to do to run the application. No printed documentation is required, just the CD and a signed Declaration of Originality form (obtain from the Department Office).

Learning outcomes

By undertaking this assignment you will:

- Demonstrate that you know how to use Ruby-on-Rails technologies: ActiveRecord, view templates, controllers and REST-based web services.
- Demonstrate the application of design patterns.
- Demonstrate that you are able to produce a UML diagram(s) during the design process and then translate this (these) into code.
- Demonstrate that you know how to undertake suitable testing.
- Demonstrate an ability to program secure communications.

Submission details

Your solution to this assignment must be deposited in the Department collection system in the foyer on Monday 9th December 2013 and between 10am-4pm. If you are late then please complete a Late Assignment Submission form and hand this in to the

[1] E.g. see http://garrickvanburen.com/archive/passing-sessions-and-referers-in-rails-functional-tests/

Department office. All required forms can be found at:
http://www.aber.ac.uk/~dcswww/intranet/staff-students-internal/teaching/resources.php

Note: this is an "individual" assignment and must be completed as a one-person effort by the student submitting the work.

This assignment is **not** marked anonymously.

I will attempt to provide marks and feedback by the Tuesday 7th January 2014.

Mark breakdown

The following table gives you some indication of the weights associated with individual parts of the assignment. This will help you judge how much time to spend on each part.

Design as reported in the documentation	Is there a design class diagram(s)? Is there an associated textual description? Is there rationale for design choices made? Quality of the design, and conformance to functional and non-functional requirements. Documentation quality.	25%
Implementation: Underwriter application	Does this application run? Appropriate use of Rails controllers, model classes and view templates. Code quality: comments, identifier names.	20%
Implementation: Broker application	Does this application run? Quality of the user interface, view templates. Code quality: comments, identifier names. Use of HTTPS.	15%
Interoperability: REST web services	Demonstrate the successful integration of the two applications using REST-based web services. Appropriate design of web service interface.	15%
Flair	You implemented and documented something in a way that really impresses me. I'll know it when I see it. One example is adding HTTPS between the broker and the underwriter and some form of authentication.	10%
Testing	System test tables and results and screen shots and screencast. A few examples of Rails unit and functional tests (I'm only looking to see that you know how to use them). Discussion of results and test strategy.	10%
Evaluation	Critique of your performance and suggested mark and why. Critical discussion of approach taken and technologies used. Problems encountered and if and how you solved them. What did you learn?	5%