University of Zurich[UZH]

ASSIGNMENT – Communication Systems Group, Prof. Dr. Burkhard Stiller

Design and Implementation of an Online Questionnaire Tool

Ronnie Schaniel
Zürich, Switzerland
Student ID: 11-707-254

Supervisor: Christos Tsiaras, Dr. Thomas Bocek
Date of Submission: Jan 09, 2014

University of Zürich
Department of Informatics (IFI)
Binzmühlestrasse 14, CH-8050 Zürich, Switzerland

ifi

Assignment
Communication Systems Group (CSG)
Department of Informatics (IFI)
University of Zürich
Binzmühlestrasse 14, CH-8050 Zürich, Switzerland
URL: http://www.csg.uzh.ch/

Abstract

Online Umfrage Tools werden heute genutzt, um auf einem einfachen Weg Feedback zu sammeln und die Resultate zu exportieren ohne den Overhead einer auf Papier basierenden Umfrage. Es existieren verschiedene Online Umfrage Tools. Aber diese Lösungen sind entweder nicht kostenlos oder bieten nur eine limitierte Anzahl von Umfragen ohne Kosten, und in den meisten Fällen ist es nicht möglich, zu steuern, wer Zugang zu den Daten hat. Datenschutz ist ein wichtiger Parameter, wenn Information wie E-Mail Adressen, Telefonnummern oder Namen gesammelt werden. Im Rahmen dieser Arbeit soll ein Open-Source Online Umfrage Tool, welches verschiedene Fragetypen unterstützt, designed und implementiert werden, so dass Administratoren, welche die Umfragen erstellen, als auch End-User das Tool in gleichem Masse und effizient sowohl auf Desktops, Laptops als auch Tablets über einen Web Browser nutzen können. Das System muss Multiple Choice Fragen, Single Choice Fragen, Freitext Fragen, Sprung Bedingungen und vordefinierte Dropdown Auswahlen unterstützen. Die Resultate einer Umfrage müssen in der Datenbank verfügbar und als Plain Text oder Excel Files exportierbar sein. Nicht zuletzt, muss jede Kommunikation zwischen Web Interface und dem Server das Secure Hypertext Transfer Protocol nutzen.
Gemäss den oben genannten Anforderungen wurde eine Webapplikation entwickelt, welche die einfache Erstellung von Umfragen erlaubt und sowohl auf Desktops als auch Tablets genutzt werden kann. In zukünftigen Arbeiten könnten neue Features hinzugefügt werden.

Online questionnaires are nowadays used as an easy way to collect people's feedback and export the results without having the overhead of a paper based questionnaire. There are several online questionnaires and online survey tools in place. However, the solutions are either not free of charge or only offer a limited number of surveys without a charge, and in most of these cases there is no control of who has access to the data. Privacy is a very important parameter, when it comes to collecting information such as e-mail address, phone numbers, or names. In this thesis an open-source online questionnaires tool that supports basic question types has to be designed and prototyped in a way that the administrator, who is the person that creates a survey, and the end-user interface can be equally and efficiently accessed in desktops, laptops, and tablets through a Web browser. The system must support multiple choice questions, multiple selection questions, free text answers, jump conditions, and pre-defined drop-down selections. The results of a questionnaire must be accessible in a database, exported in plain text and Excel files. Last but not least, any communication between the Web interface and the server must use the Secure Hypertext Transfer Protocol.
According to the above-mentioned requirements a webapplication was developed which

allows an easy creation of surveys and can be accessed in desktops as well as tablets. In future work new features could be added or the code could be refactored.

Acknowledgments

I would like to thank Prof. Burkhard Stiller for giving me the opportunity to complete this assignment. I would also like to thank Christos Tsiaras for the meetings and his quick responses in case of questions and Dr. Thomas Bocek for his support by email.

Contents

Chapter 1

Introduction

The Communication Systems Group (CSG) at the Computer Science Department of the University of Zurich needed an online questionnaire tool for internal as well as external use. The tool used in the past was not fully satisfying. Therefore a web application had to be developed which gives the users the needed functionality. In addition the interface of the tool had to be optimized for different devices and an interface to an existing authentication system had to be implemented.

1.1 Motivation

There existed several online questionnaires tool, however none of them covered all the needs while being free of charge at the same time. A few tools where free of charge but then only provided a limited number of surveys. Also in most cases there was no control of who has access to the data. But privacy was a very important parameter since it could come to collecting information such as e-mail addresses, phone numbers or names. Besides these limitations, an own implementation of such a questionnaires tool could also offer a working interface to the existing authentication tool of the CSG.

1.2 Description of Work

The goal of this assignment was to design and implement an online questionnaire tool. The following points had to be achieved at the end of the Online Questionnaires Tool implementation:

- Only HTTPS connections for every communication between the browser and the online questionnaire tool must be used
- Surveys must be created through a web interface (no client installation should be needed)

- A unique survey URL should be generated for each survey created
- A survey should be able to be marked as active/inactive and only active surveys should accept new answers
- The tool should support multiple choice questions (radio buttons)
- The tool should support multiple answer questions (check boxes)
- The tool should support pre-defined drop-down selections
- The tool should support free text answers
- The tool should support questions marking as optional or mandatory
- The tool should support jump conditions
- The tool should support welcome, survey finished and survey completed messages. One picture file must be able to be used if needed in each of those cases.
- The tool must be secure against SQL injection [6] attacks
- The tool must be secure against Cross-site Scripting (XSS) [8] attacks
- Only the CSG members must have access to the tool (an access similar to the CSG publication system [1] might be implemented)

1.3 Thesis Outline

In the next chapter the design choices are discussed as well as a set of arguments on the final design choice is provided. It follows the description of the implementation with a list of solved and open issues. In the last chapter the work is summarized.

Chapter 2

Implementation

The tool should be implemented from ground up and the only requirement regarding technologies was that it should be open source. Since I was most familiar with MySQL [2] I picked it for the database. Because programming with PHP [3] is less comfortable than with Python or Ruby, I had to chose either Python Django [4] or Ruby on Rails [5]. Due to more experience with it, Ruby on Rails was selected. Rails is a framework for the development of web applications and Ruby comes with a lot of useful plugins in this context, called gems. For the frontend of the application the Bootstrap framework [6] is used which helps to get a responsive design that works on multiple devices.

2.1 Design

When using the Rails Framework the design is more or less given. It follows the Model View Controller (MVC) pattern. The model part and some aspects of the controller and view are discussed. Finally the implementation of authentication as well as authorization mechanism is described.

2.2 Model

The relationship between the models looks like shown in Figure 2.1. Each class, except the subclasses of `Question`, correspond to a table in the database. On top there is basically the `Questionnaire` model to which users and questions are assigned. Each questionnaire is directly assigned to one users, i.e. the creator of it. All other users are connected to the questionnaire through a `Permission` model since it is a many-to-many relationship. This allows to add a user to multiple questionnaires and detailed permissions could simply get added in the intermediate table if needed.

A question has three basic types which are free text, multiple choice and single choice. To distinguish between the types the `Question` model has a subclass for each of them. For the database a single table inheritance (STI) approach is used. That means, although

there are several models, only one table is used to store them. The decision to use this structure is based on the fact that the question types only differ in one attribute. This single attribute is used to mark questions that needs to be rendered as drop down, what is, based on the requirements, relevant for single choice questions, but could also be for multiple choice.

Further, a question can have multiple jump conditions to control which questions should be provided next based on the answer of the current question. Therefore a `JumpCondition` object has two references to `Question`. First, for the question on which the condition is checked and second, for the question that should be displayed next if the condition holds.

The model `Option` has a many-to-one relationship to `Question` and it is only relevant for the multiple choice and single choice questions since the free text type has no options.

Finally the `Response` model belongs to `Question` as well as `User` and can have a reference to `Option`. Each entry of the `Response` model table has one null value, namely for the attribute option id or text answer. The other possibility would have been to create a table for text responses and one for the responses of multiple and single choice, but since there is only one null value, it is not that critical. One can argue about this decision, if you use multiple models the database has to do joins when retrieving all responses of a questionnaire.

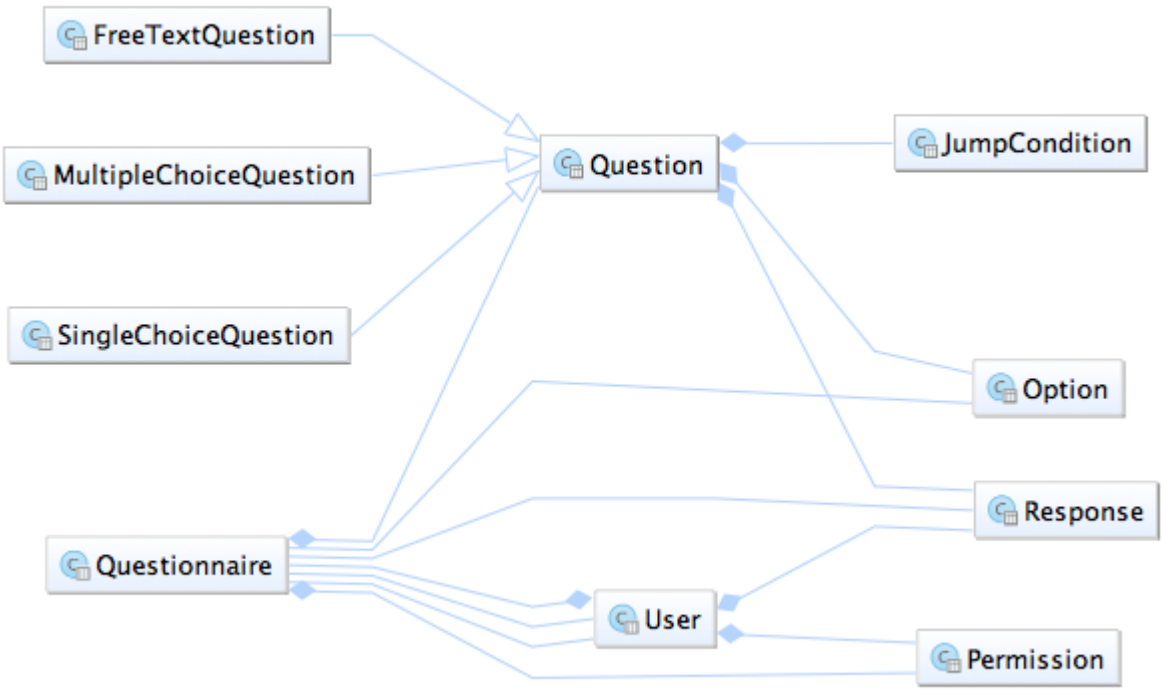

Figure 2.1: Overview of the models.

2.3 Controller

Typically in a Ruby on Rails application each model has a corresponding controller. These controllers inherit directly from the `ApplicationController` class. In the implementation a `SecurityController` is introduced which is added in the inheritance hierarchy between the `ApplicationController` and the other controllers. While the `ApplicationController` deals with things like error handling, the `SecurityController` handles the authentication which is a solution based on the devise gem [7]. In addition the `SecurityController` contains the interface to the basic auth mechanism and handles the creation of new users and signing in of already existing ones.

2.4 View

The view part is responsible for rendering the `HTML` in the end and works with several templates located at the app/views directory of the application. To prevent repetition of code a `application` template was introduced in which all other templates are integrated. The `application` template holds the basic structure of the page, including the head section, navigation bar and footer. In case of errors the `ApplicationController` renders one of the error templates. All other templates belong to a controller and are integrated into the `application` template. To facilitate the creation of all this templates I used two gems. For easier creation of `HTML` the plugin `haml-rails` [8] is used. For an ideal integretion of the Twitter Bootstrap Framework `twitter-boostrap-rails` [9] was installed. Rails provides serveral generators to build parts like models, controllers or views. With the Bootstrap gem it was possible to directly generate the appropriate html.

2.5 Authentication

When developing with rails the `devise` gem is the first choice when it comes to the need of authentication. It takes care of lots of things regarding user management, signing in, password recovery etc. and it is easy to extend the plugin or to add custom authentication mechanisms. Also in this project devise was choosen which facilitated the implementation of the authentication, specifically the interface to the existing authentication system. Basically, devise ensures that only admins can access the backend part of the application. For the taking of a questionnaire devise creates and signs in guest users. The interface to the user system of the Communication Systems Group is implemented in the `SecurityController`. In case of a successful http basic auth request on the subpath /auth of the application's url, a method is called that checks if a user with the current username is already in the database. If not, the method creates the new user and gives him admin permissions. Then the user is signed in using the devise authentication system.

2.6 Authorization

For the authorization the intention was to use the `cancan gem` [10] which is the most used in this context. Unfortunately this gem was not updated for the new version 4 of Rails. The attempt to fork the `cancan` repo and update the gem itself failed due to too many changes which would have been necessary. After studying and investigating several alternatives, with `Pundit` [11] an appropriate solution was found. The files located at app/policies allow detailed permission definition based on the controller actions and the given models or instances of models.

Chapter 3

Evaluation

In this chapter the solved and open issues are discussed which were introduced in the Description of Work section of chapter 1.

3.1 Solved Issues

All requirements, for which it was possible, have been implemented except the ones which were declared as invalid.

All relevant methods of the application as well as the authorization part are tested. These are more than 160 test cases which are in most cases very detailed, also to contribute to the documentation of the code itself. Where necessary code parts are documented. Also a successful deployment of the application to a private server has been achieved.
No client installation is needed because the whole functionality of the application can be accessed through a webbrowser. Due to the use of the bootstrap framework the webpages should be equally displayed no matter which browser is used (see Figure 3.1).
A unique url for each questionnaire is automatically provided when using the default routing configuration of Ruby on Rails. It has the form /questionnaires/:id/take/1, where :id is a unique identifier for a survey. It would also be possible to start the questionnaire on another question when using not 1 as the last parameter in the url, but a number that is higher.
Through the form of a questionnaire it is possible to toggle the state of it which could be either active or inactive. When the url of an inactive questionnaire is accessed the questions can not be answered and an appropriate message is shown.
All the 3 question types (free text, single choice, multiple choice) are implemented and one can use dropdown selections for the single choice type. Each question can also be marked as mandatory which forces the user to give an answer in order to proceed in the questionnaire. By default a question is optional. For single choice as well as multiple choice questions the specification of jump conditions is possible to allow the skipping of later questions based on the answer of previous questions.
The tool supports welcome and survey completed messages. The thesis goals included

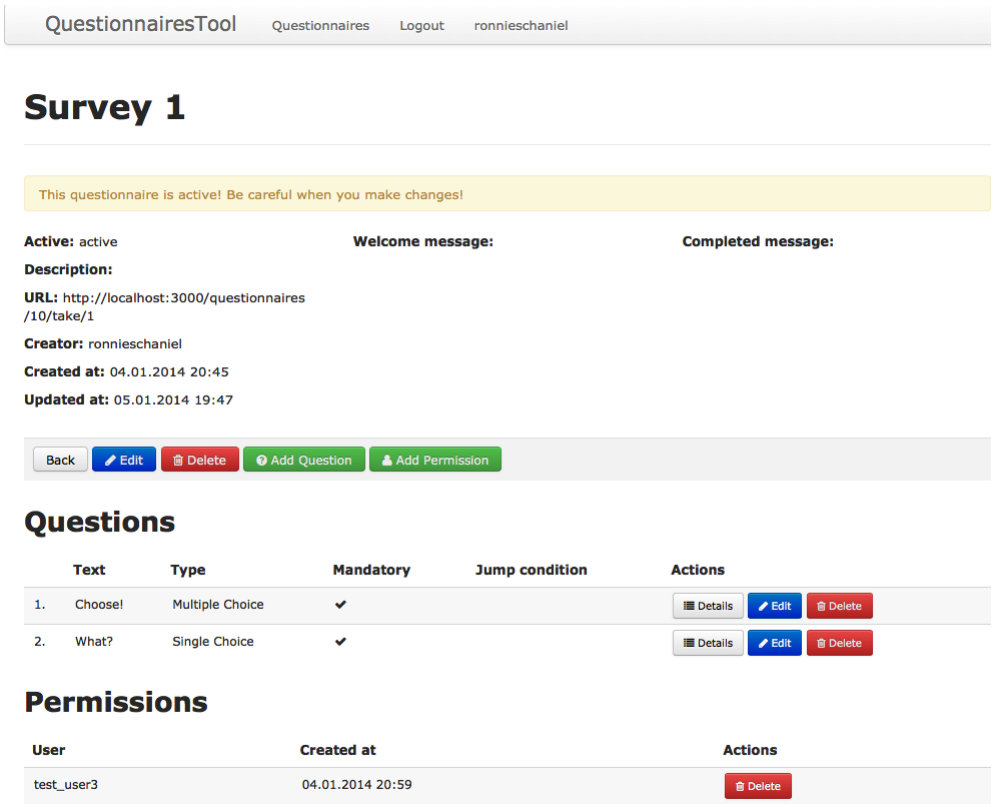

Figure 3.1: The detail page of one questionnaire with the associated questions and permissions in a typical bootstrap design.

also survey finished messages. But one of the supervisors and the student came to the conclusion that it makes no sense to introduce this third type of message. The other two messages, however, support a picture in each of the cases.
Due to the interface to the existing authentication system of the CSG only the specified members have access to the tool.
Regarding security, especially protection against SQL-Injection or XSS attacks, rails provides some helper methods [12]. In the built application user inputs in SQL statements are escaped. Attempts to introduce malicious javascript code fail due to escaping of output.

3.2 Open Issues

Open issues at this moment do not concern the implementation itself. The deployment to the CSG testbed is left which includes also the configuration of SSL. In the appendix of this report a description for the configuration of the relevant virtual hosts for the use of only https is included.

Chapter 4

Consideration of the task

The implementation of this online questionnaire tool was an interesting assignment for me and the amount of time I spent to solve all the issues was reasonable. Although I had some experience in the field of web applications, I could learn some lessons. Particularly the writing of tests was one thing that I never used to do with such intensity. I developed a usable application which fulfilles all of the requirements I received at the beginning. Nevertheless it is possible that some requests for changes will turn up or that some bugs get discovered while using the software. But that is the normal case when software gets developed.

Chapter 5

Summary and Conclusions

In the context of this assignment a new questionnaire tool for the Communication System Group was designed and developed. The implementation allows users in a existing authentication system to log into the application and enables them to create surveys. The interface of the webapplication is accessible equally and efficiently in desktops, laptops, tablets and even smartphones. It supports an export of the survey's data in excel format as well as in graphical format by showing bar charts. Administrators have only access to their own questionnaires or the one which they received a permission on. All basic question types and the jump conditions are supported. The use of the Ruby on Rails framework results in a well structured application which has also a reasonable unit test coverage. In addition the use of the application in a productive environment was successfully tested.

Further work could focus on the introduction of functional, user interface or integration tests. The simple architecture of the application enables developers to easily add new features and they are able to understand the code by having a look at the detailed unit tests. Another point, concerning the guest users, allows also some improvements. In the current implementation everytime one has filled out a questionnaire a new record in the user table is created which is not absolutely ideal.

Bibliography

[1] Communication System Group. (2014, Jan 5). Publication system, [Online]. Available: http://www.csg.uzh.ch/publications.html.

[2] Oracle Corporation. (2014, Jan 5). Mysql, [Online]. Available: http://www.mysql.com.

[3] PHP. (2014, Jan 5). Php, [Online]. Available: http://en.wikipedia.org/wiki/PHP.

[4] Django Software Foundation. (2014, Jan 5). Django, [Online]. Available: https://www.djangoproject.com.

[5] D. H. Hansson. (2014, Jan 5). Ruby on rails, [Online]. Available: http://rubyonrails.org/.

[6] Bootstrap. (2014, Jan 5). Bootstrap, [Online]. Available: http://getbootstrap.com.

[7] plataformatec. (2014, Jan 5). Devise, [Online]. Available: https://github.com/plataformatec/devise.

[8] indirect. (2014, Jan 5). Haml-rails, [Online]. Available: https://github.com/indirect/haml-rails.

[9] seyhunak. (2014, Jan 5). Twitter-bootstrap-rails, [Online]. Available: https://github.com/seyhunak/twitter-bootstrap-rails.

[10] ryanb. (2014, Jan 5). Cancan, [Online]. Available: https://github.com/ryanb/cancan.

[11] T. Klemm. (2014, Jan 5). Pundit, [Online]. Available: https://github.com/elabs/pundit.

[12] D. H. Hansson. (2014, Jan 6). Ruby on rails security guide, [Online]. Available: http://guides.rubyonrails.org/security.html.

List of Figures

Appendix A

Installation Guidelines

Ruby on Rails allows different variants for deploying an application. I recommend to use the phusion passenger application server with an apache as webserver and a MySQL database. Also a solution with JRuby and tomcat would be possible but deployment is easier with passenger and in addition the application runs faster. The variant which follows was tested on an Amazon AWS EC2 Instance on a CentOS machine.

1. Install rvm:

```
$ curl -L https://get.rvm.io | bash -s stable \\
$ source /etc/profile.d/rvm.sh
```

2. Install ruby:

```
$ rvm install 1.9.3-p125
```

3. Install passenger gem:

```
$ gem install passenger
```

4. Install mod rails:

```
$ passenger-install-apache2-module
```

   Follow the instructions which will guide you through the installation.

5. For the passenger config it is best to create the files passenger.load and passenger.conf under mods-available or insert the config directly into the apache conf file. In former case the paths come into the passenger.conf file and the "LoadModule" section finds place in the passenger.load file. After adding these configurations the apache server should get restarted.

6. Create a directory for the application (for example /home/ruby) and upload the app to this location.

7. Cd to the created directory and run bundle install.

8. Give the apache user read and write permission to the created directory:

```
$ chown -R www-data:www-data /home/ruby
```

9. Create the needed vhost entry which should look like the following.

   All http requests should be redirected to https:

```
<VirtualHost *:80>
  ServerAdmin email@example.com
  ServerName servername

  # disable TRACE on the virtual host
  RewriteEngine on
  RewriteCond %{REQUEST_METHOD} ^TRACE
  RewriteRule .* - [F]

  # redirect to HTTPS
  RewriteCond %{HTTPS} off
  RewriteRule (.*) https://%{HTTP_HOST}%{REQUEST_URI}
</VirtualHost>
```

   Now the vhost for the ssl requests follows. In the /auth Location section the path of the corresponding .htpasswd file should be provided. For the configuration of SSL insert the paths of the individual cert and key files. Maybe not all paths listed here have to be used. Restart apache after adding the virtual hosts.

```
<VirtualHost *:443>
  ServerAdmin email@example.com
  ServerName servername
  DocumentRoot /home/ruby/questionnaires_tool/public
  RailsEnv production
  <Directory /home/ruby/questionnaires_tool/public>
    AllowOverride None
    Options -MultiViews
  </Directory>
  ErrorLog /var/log/httpd/error.log
  CustomLog /var/log/httpd/custom.log common
  <Location /auth>
    AuthType Basic
    AuthName "Backend"
    AuthUserFile /home/ruby/.htpasswd
    require valid-user
  </Location>

  SSLEngine on
  SSLCertificateFile /etc/apache2/ssl/cert/file.crt
  SSLCertificateKeyFile /etc/apache2/ssl/key/file.key
```

```
    SSLCACertificateFile /etc/apache2/ssl/cert/file.crt
    SSLCertificateChainFile /etc/apache2/ssl/cert/file.crt

    <FilesMatch "\.(cgi|shtml|phtml|php)$">
      SSLOptions +StdEnvVars
    </FilesMatch>
    <Directory /usr/lib/cgi-bin>
      SSLOptions +StdEnvVars
    </Directory>
    BrowserMatch ".*MSIE.*" \
      nokeepalive ssl-unclean-shutdown \
      downgrade-1.0 force-response-1.0
  </VirtualHost>
```

10. Create a MySQL database and a user with the relevant privileges and insert the connection information into the config/database.yml file of the Rails application. Execute the following command to run the migrations:

```
$ rake db:migrate
```

11. Set up the assets.

```
$ rake assets:clean
$ rake assets:precompile
```

12. Start the application:

```
$ touch tmp/restart.txt
```

Appendix B

Contents of the CD

- Report in source files, including figures
- Report as a PDF
- Manual
- Source of the implementation
- Presentation Slides