Abstract

Web programming is getting more and more important every day and as a consequence, many new tools are created in order to help developers design and construct applications quicker, easier and better structured. Apart from different IDEs and Technologies, nowadays Web Frameworks are gaining popularity amongst users since they offer a large range of methods, classes, etc. that allow programmers to create and maintain solid Web systems.

This research focuses on two different Web Frameworks: Ruby on Rails and Google Web Toolkit and within this document we will examine some of the most important differences between them during a Web development.

Keywords

web frameworks, Ruby, Rails, Model-View-Controller, web programming, Java, Google Web Toolkit, web development, code lines
List of Figures

Figure 2.1. mraible - History of Web Frameworks ................................................. 4
Figure 2.2. Java BluePrints - MVC Pattern ......................................................... 6
Figure 2.3. Libros Web - MVC Architecture ...................................................... 7
Figure 2.4. Ruby on Rails - Logo ........................................................................ 8
Figure 2.5. Windaroo Consulting Inc - Ruby on Rails Structure ......................... 10
Figure 2.6. Ruby on Rails Architecture ............................................................... 12
Figure 2.7. GWT - Logo ..................................................................................... 13
Figure 2.8. Product Overview - GWT Architecture ............................................ 15
Figure 3.1. Class Diagram ................................................................................... 19
List of tables

Table 2.1. Features of a Framework.................................................................7
Table 4.1. Ruby on Rails Configuration Files.................................................48
Table 4.2. GWT Configuration Files...............................................................48
Table 5.1. Advantages and Disadvantages....................................................52
Table 5.2. Time and Code lines........................................................................53
Abbreviations

AJAX: Asynchronous JavaScript And XML
API: Application Programming Interface
AWT: Abstract Window Toolkit
CRUD: Create, Read, Update and Delete
CSS: Cascading Style Sheets
GWT: Google Web Toolkit
HTML: Hypertext Markup Language
IDE: Integrated Development Environment
JSP: JavaServer Pages
J2EE: Java Platform, Enterprise Edition or Java EE
ORM: Object-relational mapping
REST: Representational State Transfer
RoR: Ruby on Rails
RPC: Remote Procedure Call
SDK: Software Development Kit
SOAP: Simple Object Access Protocol
XML: Extensible Markup Language
# Table of Contents

1. Introduction .................................................................................................................. 1  
   1.1. Problem and Motivation .......................................................................................... 1  
   1.2. Scope ......................................................................................................................... 2  
   1.3. Goals and Criteria ................................................................................................. 2  
   1.4. Report Outline ......................................................................................................... 3  

2. Background .................................................................................................................... 4  
   2.1. Web Development Frameworks ............................................................................... 4  
   2.1.1. History ...................................................................................................................... 4  
   2.1.2. Overview ................................................................................................................. 5  
   2.2. Ruby on Rails ........................................................................................................... 8  
   2.2.1. History ...................................................................................................................... 8  
   2.2.2. Overview ................................................................................................................. 8  
   2.3. Google Web Toolkit ............................................................................................... 11  
   2.3.1. History ...................................................................................................................... 12  
   2.3.2. Overview ................................................................................................................. 12  

3. Application development ............................................................................................. 15  
   3.1. Project description .................................................................................................. 15  
   3.2. System design ......................................................................................................... 16  
   3.3. Class Diagram ........................................................................................................ 17  

4. Implementation ............................................................................................................. 18  
   4.1. Common Tools ....................................................................................................... 18  
   4.2. Set up ....................................................................................................................... 19  
   4.2.1. Ruby on Rails ......................................................................................................... 19  
   4.2.2. GWT ......................................................................................................................... 19  
   4.2.3. Comparison .......................................................................................................... 20  
   4.3. Learning ................................................................................................................... 21  
   4.3.1. Ruby on Rails ......................................................................................................... 21  
   4.3.2. GWT ......................................................................................................................... 21  
   4.3.3. Comparison .......................................................................................................... 22  
   4.4. Basic application development, Hello World! ....................................................... 22  
   4.4.1. Ruby on Rails ......................................................................................................... 22  
   4.4.2. GWT ......................................................................................................................... 23  
   4.4.3. Comparison .......................................................................................................... 23
1. Introduction

In the introduction of the thesis, the purpose of the project and the most important goals will be pointed out.

1.1. Problem and Motivation

The origin of the WWW started twenty years ago. Since then, WWW has grown significantly becoming one of the main business markets, especially since the web 2.0 era. The constant evolution suffered by the WWW has become Web Development an indispensable tool in our world due to strong demand by companies. To facilitate the Web Development have been created tools called frameworks. Frameworks have popularized the concept of rapid development that in our society is essential.

The framework can be defined as a set of classes, usually some of them being abstract. Collaborations between them provide an abstract design of the solutions for a set of problems (Ralph E. Johnson and Vincent F. Russo, 1991)

The framework captures the most common design decisions regarding a specific type of application, creating/providing a common model amongst all of them, assigning responsibilities and establishing collaborations between classes that make up the model. Besides, this model contains common points of variability. (W.Pree, 1995)

Reuse has been one of the goals of software engineering and Web Development since its inception, and the improvement of the re-usability was one of the targets for the creation of Object-oriented techniques. Several development techniques have been proposed, which are based on coarser assets as frameworks that have noted remarkable successes in the Web Development using reuse techniques such as in the field of graphical user interfaces, where these techniques were original made in. (R.J. Wirfs-Brock and R.E. Johnson, 1990).

Reuse is then produced in the process of instantiation of the framework, in which the developer by using the reuse provides specific functionality to each application.

Even if different classifications have been proposed to characterize the types of frameworks (M. Fayad and D.C.Schmidt, 1997), all of them seem to agree that there are two different types called application frameworks and frameworks domain. An application framework encapsulates a horizontal layer of functionality that can be applied in the construction of a large variety of programs. In addition to the frameworks that implement the graphical interfaces user, we can classify in this category to the dedicated to establish communications or processing of XML documents...

On the other hand, a domain framework implements a layer of vertical functionality corresponding to an application domain or a product line (R.E. Johnson and B. Foote, 1988).
We will focus our study in the applications frameworks where there are even different classifications depending on the structure PHP, JAVA … in which they are based. As we can see there is a rich variety of frameworks, which is very beneficial for developers because they could choose from a great selection.

Even if with the different benefits provide by the frameworks, some obstacles have prevented to implement on a successfully way some reuse models based frameworks. These events have occurred due to main difficulties given because they are tough to build and are not easy to learn and use. This problem is due to inexperience in the use of a specific framework and if you do not have enough knowledge can be difficult to modify default behaviors in the framework as classes, libraries established. A good proof of this is the number of papers dedicated to report the experience obtained in the construction and use of frameworks, as it was published in the “Electronic Symposium on Object-Oriented Application Frameworks” (Mohamed E. Fayad, 2000).

To understand the origin of these difficulties we have to establish in our mind that the construction of frameworks is a process purely artisan. This process needs a costly development that requires to be highly experienced and prone to errors.

1.2. Scope

The purpose of this thesis is to establish bases in Web Development using development tools provided by the Frameworks. Web Frameworks Development Tools have evolved rapidly in the world of WWW (World Wide Web) becoming essential in web development. We have chosen and focused our study on two of the most important web frameworks Google Web Toolkit and Ruby on Rails. They both have experienced major changes and developments since their inception due to strong growth in web market.

For studying these Web Frameworks we will develop a medium size application and we will compare the effort and time spent doing the work.

1.3. Goals and Criteria

The thesis is composed of the following main objectives:

- Discover the world of Frameworks. Learn, understand and use a branch of the framework: the Web Development Frameworks.

- Specifically analyze two Web Frameworks: Google Web Toolkit and Ruby on Rails.

- Google Web Toolkit VS Ruby on Rails. Compare the development using these Frameworks following different criteria such as learning time, development time, simplicity, time spent, code lines written, etc. For this
reason, we will compare the following 10 tasks of the application’s development:

1. Set up the Framework and tools.
2. Study and learn about the Frameworks.
3. Create a simple “Hello World” application.
4. Implement the main classes and interfaces.
5. Use of the Model-View-Controller Pattern.
6. Connection to the Database.
7. Create sessions and the Log in system.
8. Create the GUI.
9. Configuration Files.
10. Test the application.

- Conclusions about comparative and the study made of the Frameworks.
  Point of view and opinion about which Framework is more satisfactory to work with.

1.4. Report Outline

The thesis includes seven chapters. Chapter 2 highlights the history of both frameworks and their features. Chapter 3 describes the project we developed to compare the web frameworks. The main chapter of the thesis, chapter 4, point out the comparison between both web frameworks during the development. In chapter 5 we show the results we obtained in the chapter 4 and we discuss these results. Finally, chapter 6 presents the conclusion based in our research and work and the future work.
2. Background

Along this chapter of the thesis we are going to describe what a Web Framework is and show the main features of some of them. We are also going to introduce the features of our two Web Frameworks selected, Ruby on Rails and GWT.

2.1. Web Development Frameworks

In this point we point out the history of the Web Frameworks and a short summary of some of the most famous ones.

2.1.1. History

In figure 2.1. we can see the history of some of the most famous Web Frameworks. We miss in the figure 2.1. Rails 3.0, released in 2010 that is the one we used in this project.

Looking at the figure 2.1. we can see how Web Frameworks history started about 10 years ago, so it is something almost new for us and it has increased its development very fast in the past years. We can also see that the most developed frameworks during this time have been Java Frameworks as Struts, Spring and GWT and even if we cannot see them in the picture PHP Frameworks are also very popular and very commonly used by most of PHP programmers, for example, Zend Framework, CakePHP, Symfony, etc.

It is noteworthy that the last years JavaScript Frameworks are being used to a great extent by programmers due to the increase of the usage of AJAX Technology in Web 2.0. These frameworks help the developers to create dynamic connections with the database, visual effects, etc.
Nowadays, most frameworks are Java or PHP based but, in the future this can change and for example, Ruby or Python frameworks could be on top.

### 2.1.2. Overview

A framework is basically a development environment in which programmers can develop applications of all types much easier and faster, including Web applications. A framework may consist of source code libraries, utilities, plug-ins, development models, and all kinds of tools, the only purpose of which is to accelerate the development pace of an application. A framework can help us implement functions such as security, view templates, authentication and validation of data, methods for handling data in a DB, etc. in much less time. Probably these processes would take a long time to be done from the beginning, but with fragments this problem is eliminated.

In other words, a framework can be considered as a common incomplete and configurable application to which the latest parts are being added in order to build a particular application.

The main objectives of a framework are:

- Accelerate the process development.
- Reuse existing code.
- Promote good development practices with the use of patterns.

Therefore we can define a Web framework as a set of components that compose a reusable design which facilitates and speeds up the development of Web systems.

To understand how Web Frameworks works are available it is essential to know the Model-View-Controller (MVC) pattern.
The Model-View-Controller (see figure 2.2) is a guide to architecture design applications and offers a strong interactivity with users. This pattern organizes the application into three individual models. The first one is a model that represents the application data and business rules, the second one is a set of views representing the entry forms and out-give information and the third one is a set of drivers that processes user requests and controls the execution flow of the system.

**The model:** defines the business rules (the database belongs to the model). It also responds to the request that the controller makes about the state of the data.

**The view:** allows the user to interact with the system through the user interface. Multiple views can exist for the same model.

**The controller:** makes calls to the model in order to respond to requests made by the view. It also usually sends the responds to the view. (Chuidiang – Ejemplos Java y C/Linux, 2007)
In figure 2.3, we can see the Model-View-Controller architecture and how it works. We have a request in the Controller that demand any data from the model. Once the Controller gets the data, it sends the information to the View to show the response in any format.

All different types of framework have a number of common features:

<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abstraction of URLs and sessions</td>
<td>Manipulation directly to the URLs is not needed otherwise the framework sessions do it for you.</td>
</tr>
<tr>
<td>Data access</td>
<td>Include tools and interfaces necessary to integrate itself with tools data access in Databases, XML, etc.</td>
</tr>
<tr>
<td>Drivers</td>
<td>Most frameworks implement a series of drivers to manage events as input using a form or access to a page. These drivers usually adapt easily to the needs of a particular project.</td>
</tr>
<tr>
<td>Authentication and access control</td>
<td>Include mechanisms for identifying users by user name and password allowing to restrict access to certain pages to specific users.</td>
</tr>
<tr>
<td>Internationalisation</td>
<td>Include tools to adapt the content to different languages, translating the text and adding some components depends on the localization.</td>
</tr>
<tr>
<td>Separation between design and content</td>
<td>Most frameworks separate the data from the way in how the data is showed. This separation in different layers allows programmers to work in different levels during the development.</td>
</tr>
</tbody>
</table>

Table 2.1. Features of a Framework
2.2. Ruby on Rails

Next, we are going to describe one of the Web Frameworks we have chosen for our thesis. Ruby on Rails is one of the most popular Frameworks currently and we are going to present the benefits of its use.

![Figure 2.4. Ruby on Rails – Logo, 2011](image)

2.2.1. History

Ruby on Rails is not a programming language itself. In fact it is the union between the programming language Ruby and the framework Rails.

The Ruby programming language is the work of Yukihiro Matsumoto. Matsumoto began working at Ruby in 1993 and showed the first version in 1995. Ruby has syntax similar to Perl, another programming language. Matsumoto's main goal was to create a programming language to facilitate the programmer's life. It is a great advantage because the programmer can work easier than other languages and can finish the work before, being more productive. (Tripod – Ruby on Rails, 2006)

The history behind Rails completes the process of Ruby on Rails. David Heinemeier Hansson, a Danish programmer, created Ruby on Rails after he was working on Basecamp, a project control tool famous. The first version of Ruby on Rails appeared in December 2005. It uses the MVC architecture pattern and it focuses on the principle of DRY (Don't repeat yourself).

The result was a language very efficient, powerful and with a very clear syntax. The current version of Ruby on Rails is 3.0.9. and it is licensed by MIT license. (Ruby on Rails, 2011)

2.2.2. Overview

Rails is an open source framework written in Ruby, created for developing Web applications with communication to databases. This architecture-based Model View Controller (MVC) is one the best feature of Ruby. What makes it different is its incredible speed which is much faster than others. There are many things in Ruby that can be done in a very simple way whereas in other languages it is not even possible to accomplish. This is because of the language structure; it is a truly scripting language, object-oriented that allows faster development with fewer code
lines and fewer errors. In general it is easy to use, quick to implement and programming becomes a pleasant experience.

Ruby is portable to multiple platforms: UNIX, Linux, DOS, Windows, OS / 2, and runs on all of them without the code having to be modified.

Rails works with several web servers and databases. For the web server, we recommend Apache or lighttpd running FastCGI or SCGI. For the database, one can use MySQL, PostgreSQL, SQLite, Oracle, SQL Server, DB2, and Firebird. It will work on any operating system, but we recommend one based on Unix for development. If you need hosting, TextDrive is the official hosting of Ruby on Rails. (Maestros del Web, 2006])

At the beginning Ruby on Rails used SOAP for Web Services but in the current versions, it uses REST. REST describes an architecture style of networked systems.

This is the Ruby on Rails structure:

As previously explained, Ruby on Rails, uses MVC pattern as structural pattern. Next the different parts of MVC on Ruby on Rails will be described. In the figure 2.5. we can see the different Ruby on rails and how they work, the events that make them contact to each other. Looking at these components, we can see how Ruby on Rails uses a MVC separating the model (and its connection with the database), the controller getting the responses from the model, and the view with some components like the Action View.
Model

In object-oriented web applications on databases, the model consists in classes that represent the tables in the database. In Ruby on Rails, Model classes are handled by Active Record.

Frequently, the only thing that the programmer has to do is to inherit from the class ActiveRecord::Base, and the program will automatically find what table has to be used and how many columns it consists of.

The definitions of the classes also detail the relationships between classes with object-relational mapping sentences. The data validation routines (eg validates_uniqueness_of: checksum) and routines related to the update (eg after_destroy: delete_archive, before_update: update_details) are also specified and implemented in the model class. (Software Developer, 2007)

The model represents:

- Tables in the database.
- Migration (Express Changes in the DB)
- Observers

View

In MVC, View is the display logic, or how to display data from the control class. Often in web application the view is a minimal amount of code in HTML.

There are now many ways to manage the views. The method used in Rails by default is to use Embedded Ruby (archives.rhtml from version 2.x onwards archives.html.erb RoR), which are basically pieces of HTML with some Ruby code, following syntax similar to JSP. Views can also be constructed in HTML and XML with Builder or by using the Liquid templating system.

It is necessary to write a small piece of HTML code for each method of the controller needs to show the user. The "model" or distribution of the elements on the page is described separately from the controller action and the fragments can be invoked at each other. (Software Developer, 2007)

Controller

In MVC, the Controller classes respond to user interaction and invoke the logic application, which manipulates the data model classes and displays the results using the Views. In MVC-based web applications, methods are applied by the controller using the browser.

The implementation of the controller is handled by the Actionpack of Rails, which contains the class ApplicationController. A Rails application simply derives from this class and defines the necessary actions as methods that can have derived from the web, usually in the form http://application/example/method.
Rails also provides scaffolding which can quickly build most of the logic and views needed to perform the operations frequently. (EcuRed – Ruby on Rails, 2010)

Figure 2.6. shows the simply the MVC and explains in the legend how it works. It is a simpler example than figure 2.5. showing how the MVC works.

Main features from Rails:

a. MVC better defined.
b. Scaffolding to create models, views and controllers at once.
c. Naming convention very comfortable and easy.
d. Use of YAML instead of XML.
e. DRY, REST and CRUD entering the vocabulary of the framework as a philosophy.
f. All the power of Ruby (simplicity, open classes, meta-programming, etc).
g. A community creating amazing gems and plugins for almost everything.
h. Testing integrated with the project from the start.
i. Flexibility to configure and extend the vast majority of objects simply.

2.3. Google Web Toolkit

In this section we are going to describe the other Web Framework we have chosen for our thesis; Google Web Toolkit. We are going to use this framework to compare it against Ruby on Rails.
2.3.1. History

GWT (Google Web Toolkit) is a framework created by Google that allows hiding the complexity of various aspects of technology AJAX. GWT is compatible with all browsers which is really important because different browsers require different coding. The concept of Google Web Toolkit is based on a premise and it is quite simple. Basically what you should do is create the Java code using any development environment (IDE) for Java and the compiler will translate it to HTML and JavaScript.

Google announced the GWT library in the JavaOne conference in 2006 and launched Version 1.0 on May 16, 2006. Since GWT has been released, the GWT open source project has helped Java developers take the richness of Ajax in a multi-platform environment and friendly Web.

The principal mission of GWT must improve radically the web experience for users allowing the developers who use Java tools to build Ajax without any modern browser barriers. GWT has been established as a Java software development platform very popular that can create sophisticated Web applications of high-performance. GWT provides us a much richer experience than other usual Java technologies. (EcuRed - GWT, 2011)

2.3.2. Overview

GWT uses the Apache License 2.0, as a free software license created by Apache Software Foundation. This license applies to all components, including:

- GWT compiler of Java to JavaScript.
- Accommodation in the Browser.
- Class Libraries.
- GWT Web user interfaces.

GWT is a development framework in Java open source. It allows users to escape from the technologies currently used to write AJAX applications, which are difficult to manage and could possibly make a number of mistakes. With GWT one can develop and debug AJAX applications using the Java programming language in the development environment of one’s choice (operating system and IDEs).
When the user has finished his application written in Java and has compiled the program, GWT automatically translates into JavaScript and HTML compatible with any Web browser. This is the GWT life cycle:

1. Use your favorite integrated development environment (IDE) for writing and debugging a Java application using GWT libraries as you need.

2. Use the Java to JavaScript compiler of GWT to transform the application into a set of files JavaScript and HTML that you can hang on any server and run from a Web browser.

3. You have to verify your work applications on every one of the browsers thinking in which your customers will use. (Google Web Toolkit - Google Code, 2010)

Google Web Toolkit Architecture:

In figure 2.8 we can see the architecture of GWT separated in the Class libraries and the Development Tools.

- **GWT Java-to-JavaScript Compiler**: The role of this component is to translate code in Java to JavaScript. We use it when GWT is on Web mode.

- **Hosted Web Browser**: This component executes Java application without translating JavaScript, host mode using the Java virtual machine.

- **JRE Emulation Library**: It contains the most important libraries of Java classes: `java.lang` where are fundamental classes to program in Java language and a subset of classes from Java package `useful`. `Java.lang` includes the class `Java.lang.Object`, this is a fundamental class in Java
because this inherits or extends all classes in Java. The rest of the packages are not supported by GWT.

- GWT Web UI Class Library: contains a set of user interface elements that enables the creation of objects such as text, text boxes, images, button, etc.

Main features from GWT (Blogspot – GWT, 2008):

a. Dynamic graphics and reusable components, developers can use art classes to implement behaviors that otherwise would consume a lot of time.
b. Simple RPC mechanism.
c. Management of Web browser history.
d. Support of debugging of Java.
e. Control of different features of the browser.
f. Integration with JUnit.
g. Internalization.
h. Developers can mix code written in JavaScript within Java code using the JavaScript Native Interface (JSNI).
i. Support for the Google API’s (initially, support for Google Gears).
j. It is open source.
k. Developers can design and develop object-oriented applications. Common errors in JavaScript as the data types discrepancy, are checked at compile time.
l. The generated JavaScript code can be obfuscated to optimize the performance.
m. There are a large set of libraries developed by Google and others that extend the functionality of GWT.
3. Application development

Within this chapter we will describe the application we are going to develop with the purpose of comparison between both Web Frameworks. We will develop it with Ruby on Rails and we will make the comparison with GWT.

3.1. Project description

The application that we are going to develop will consist of a web-based Project Management System and the following main features should be included in the system:

- **Log in system:** The system has to allow the user to log in/out the system using a user name and password in the Log in screen. Using sessions the application must keep the user logged for some time and in case any user wants to visit any page of the application without logging in, the system has to reject the request and show the Log in screen.

- **Users Management:** The system allows administrator users to manage all the users of the system granting them some specific roles and permissions. One user (even not administrators) can manage his own account. Every user can have one picture as avatar on the system.

- **Dashboard:** Once the user is logged in, the first screen the system shows is the dashboard. In this screen the user can see and access to the list of all the projects and the tasks where he is taking part.

- **Customers Management:** The system allows all users to manage different customers allowing them to manage the data of the customer and the projects that the customer has. One customer can have many projects.

- **Projects Management:** The system also allows all users to manage all the projects in the system. Users can create, modify and delete different projects. They can also upload a file related with the project as, for example, the budget in PDF format. One project belongs to only one customer. One project can have many tasks to do.

- **Tasks Management:** Each project consists in several tasks that users can manage. They can create, modify and remove different tasks related with one
At the same time, each task can be assigned to one or many users to make the work. One task belongs to only one project.

3.2. Design of the system

Looking at the requirements for the project management System, we have to design the system considering that the system has to meet all the requirements and be as efficient as possible. This design must be independent of the technology we are going to use to develop the application.

To address the whole functionality of the required system, we are going to use the Client / Server model, so we have a server side with the web server and the database, and the client side with the web browsers and the operating systems.

Regarding to the structure of the system, we divide it in three tiers: presentation, application and storage. Next we will describe each tier of this structure.

Presentation

It includes the web browser and for that reason it also includes the client languages as HTML, CSS and JavaScript. This tier is responsible of interact with the user and send the appropriated requests to the application. It uses HTML and CSS to display the user interface in the web browser and JavaScript for the dynamic behavior in the GUI.

Application

This tier includes the dynamic programming language we chose for developing our web application. It is responsible for the logic of the system and it works as the engine of the whole application.

We include here an ORM for the database interaction and we use the Model-View-Controller pattern as our architectural pattern. This pattern will give a lot of benefits during the development and it also makes the application more robust and scalable as we said before. In our case, we are going to use on one hand Ruby on Rails, and on the other hand Google Web Toolkit + Java for our development in this level of the structure of the system.

Storage

At this level of the structure of the application, we are going to use MySQL as our database. We will use MySQL server and the different connectors we need for both programming languages. As we said in the previous point, the application tier will interact with the storage tier through one ORM for make this interaction easier to implement.
3.3. Class Diagram

Figure 3.1. Class Diagram
4. Implementation

This chapter describes the steps we have followed in order to develop the application which compares both technologies and shows partial results of each task.

4.1. Common Tools

For the development of the project and the study we require a series of essential tools to run and get started with Google Web Toolkit and Ruby on Rails. Such tools are very popular and well known and are usually used in many applications and developments in the computer world, these tools are:

MySQL

MySQL is one of the most important system operational databases presently. The software provides a MySQL database server SQL (Structured Query Language) fast, multi-threaded, multi-user and robust.

The program uses MySQL as a server through which multiple users can connect and use simultaneously. One of the most interesting features is that MySQL allows the use of multi-user database through websites and in different programming languages adapting different needs and requirements.

On the other hand, MySQL is known for developing high-speed data search and information, unlike previous systems. (MySQL, 2011)

MySQL is a very important tool in our project because it is an essential requirement to have a database where we save/load users, projects, customers and tasks. MySQL can be downloaded in MySQL official website (MySQL, 2011) and in our case we are using MySQL 5.0.67.

ArgoUML

ArgoUML is a UML diagramming application written in Java and published under the BSD License. As it is a Java application, is available on any platform supported by Java.

We have used ArgoUML to design the classes of the system and for exporting it as an image file to include it in this document. ArgoUML can be downloaded in its official website (ArgoUML, 2011). We are using 0.32.2 version in this project.
4.2. Set up

The first step to develop the application is to set up all the tools and requirements for the tool in the computer. We have to set up both frameworks, Ruby on Rails and GWT + Java.

4.2.1. Ruby on Rails

The first thing we have to do is install Ruby in our computer. To accomplish this we can go to the official website and download the version we prefer. We have to be careful with the version we chose since not all versions of Ruby work with Rails. In our case, after some failed attempts, we have downloaded Ruby v1.9.2-p290 (Windows installer) from the official website (Ruby Installer for Windows, 2011).

Using this installer we have the Ruby core, the console "Interactive Ruby" and some other applications and documents. One important tool that is included in the Windows installer is RubyGems, one tool we will use to download and install other plug-ins and modules for ruby.

Once we already have Ruby in our computer, we download Rails using the console. For that, we write: $ gem install rails --include-dependencies in the Ruby console and that will install Rails in our Ruby installation. In our project we are using Rails 3.0.5 version which is compatible with our version of Ruby.

Moreover, we can install Ruby Development Kit. The tool that will help us in the Ruby development because we will use it for debug the application. We can download it from the project website (Development Kit – GitHub, 2011).

Finally, we have to install an IDE that provides syntax checking and will help us to develop the application. We have chosen two IDEs, Aptana Studio 3 and Netbeans 6.9.1. Lastly we have used Aptana Studio 3 since Netbeans 6.9.1 does not work with Rails 3. For installing Aptana Studio 3, we have to download the tool in the official website (Aptana Studio, 2011), and we will install it as we use to do with any other Windows application.

4.2.2. GWT

Since GWT is a Java Framework, the first thing we have to do is download and install the SDK for Java EE. We can download it from Oracle website (Oracle - Java Platform, Enterprise Edition, 2011).

Once we have Java installed we need to configure the tools we are going to use in our development. The first one is Git. This tool is a revision control system tool that will help us during the development, specially since we are working two
people at the same time. We can download Git from its project site on Google Code (msysgit – Git for Windows, 2011).

In this development we are going to use Eclipse IDE as our code editor and we will use its tools providing us a good environment for the development. Eclipse can be downloaded from its website (Eclipse Downloads, 2011). In our case we are going to use Eclipse Galileo.

When we already have Eclipse in our computer, we only need to install GWT plug-in on Eclipse for allow us to develop GWT applications. This plug-in can be downloaded from GWT through Eclipse. We also, have installed GWT Designer for the User Interface. This tool helps us to create users interfaces faster. We can also download it through Eclipse.

For testing we are going to use JUnit 4.8.1, so we have to download it from its website (Downloads for KentBeck’s junit – GitHub, 2011) and follow the instructions to install it and configure it on Eclipse.

Finally, as GWT does not have any ORM for persistence we have tried some frameworks for that and the one we have chosen is Hibernate. For that reason we have had to download it from its official website (Hibernate – Jboss Comumnity, 2011).

### 4.2.3. Comparison

If we look at the time we spent setting up the Frameworks, we can see that while for GWT we have needed 2 hours more or less, for Ruby on Rails we have spent 4 hours. The main problem with the Ruby on Rails setting up has been the problems between Ruby version and Rails versions.

Another problem that has made us lose time is the IDE. At the beginning we tried to use Netbeans but we found problems with Rails 3 because Netbeans do not support Rails 3. Finally we tried Aptana Studio 3 and the results are satisfactory.

We also want want to point out that the setting up of GWT has been quite easy because there is plenty of information on the Internet.

Speaking about the features that both IDEs have, both of them are quite good because they provide so many tools to debug, edit code, search, etc. Apart from that, Aptana Studio 3 includes WEBrick Server for testing Ruby on Rails applications and it can be used during the development.

If we mention the server we use for test the GWT application, found that GWT plug-in includes also a free web server, AppEngine Server. We can easily use this server for testing and seeing our web application.

Briefly, setting up both Frameworks entails some work. Ruby on Rails requires more work than GWT if we previously do not know about the compatibilities of the
different version of Rails and Ruby. So with Ruby on Rails we have spent 6 hours in total for setting up the framework, the server, the IDE and the database. On the other hand, in the GWT project, we needed 4 hours in total for the same work.

4.3. Learning

In this section we are going to discuss the effort we have spent learning Ruby on Rails and GWT. We will compare the time we have spent and the problems we have found.

4.3.1. Ruby on Rails

The first step we have to follow before starting programming with Ruby on Rails is to learn the Ruby programming language. After reading some documents, books and tutorials (Ruby User’s Guide, 2008) we can see the difference between Ruby and some other languages, in this specific case, between Ruby and Java.

The study of Ruby language looks hard at the beginning but once you understand a bit the language, the rest of the knowledge is easier. For studying the Ruby language, we can find plenty of books and websites and a lot of information on the Internet. One interesting book is the official Ruby book (Huw Collingbourne, 2009) and by reading this book we can easily start programming in Ruby.

Once we know Ruby better, we are ready to learn more about Rails and the best way to do it is by reading the official documentation we can find online (Ruby on Rails Guides, 2011). In Rails API (Ruby on Rails Documentation, 2011), we can also find examples and descriptions of every class, method, etc. we need to use in our application. This API is quite complete and there we can find almost everything we need easily.

4.3.2. GWT

The learning of GWT has been quite easy because to use this framework, programmers only have to learn the different GWT components and their behaviors. Once the developer knows the performance of these components he only has to use them correctly and GWT will translate the Java code to HTML and JavaScript code. One good way to start learning GWT is by the tutorial which can be found in the GWT website (Getting Started – GWT – Google Code, 2011)

Most of the problems with the learning of GWT have been related to the use of any ORM for persistence. We have tried to use ActiveObjects (Java.net – ActiveObjects, 2008) and ActiveJDBC (activejdbc - Google Code, 2011) but since we have had too many problems setting them up, finally we have decided to use Hibernate (Using GWT with Hibernate – GWT – Google Code, 2009).
For testing the classes and the whole application we are going to use JUnit so we can find documentation about it in its website (JUnit, 2011)

4.3.3. Comparison

The learning of Ruby on Rails is harder at the beginning since we are used to develop in programming languages similar to Java. Ruby is also a very complex language and difficult to understand for us since we are used to program in C-based languages. For this reason, the learning curve for Ruby on Rails is, at the beginning, long and hard but once you have understood the structure and the performance, it is quick, easy and enjoyable to program with.

On the other hand, learning GWT is not that difficult once you know Java or another C-based language.

If we speak about the features or both languages, Java and Ruby, we find the following differences and similarities:

- Ruby is an interpreted programming language and Java is not.
- Java is a compiled language. Ruby is not compiled.
- Both languages allow serialization; Java implementing the `Serializable` interface and Ruby using Marshall class.
- Polymorphism is much more achieved in Ruby than in Java.
- Both languages have Garbage Collector.
- Ruby is dynamically typed and Java is not.
- Java has `JavaDoc` to generate documentation and Ruby has `RDOC`.
- They both have tools to use AJAX easily.

Finally, speaking about the time we needed to learn each framework, we have spent 14 hours more or less studying Ruby on Rails. On the other hand, we have spent 16.5 hours studying GWT but most of them have been because of the problems choosing the ORM for the application (6.5 hours). In summary, with Ruby on Rails we have spent more time learning the new language and with GWT we have used this time trying to learn how to assemble all the tools we need to use and they are not included in the framework.

4.4. Basic application development, Hello World!

Along this point we are going to explain how we have created a basic application. The typical "Hello World!".

4.4.1. Ruby on Rails

In order to create this application in Ruby on Rails we have proceeded the following steps:
1. In Aptana Studio we create a New Rails Project called “HelloWorld”.
2. For installing the required Gems we execute in the console ’bundle install’
3. We need to create at minimum a Controller and a View. Thus, we use the command: ’rails generate controller home index’. This will create the HomeController class and the index.html.erb file inside the home folder
4. We edit the file index.html.erb including the Hello World sentence between the <h1> tag.
5. Inside the folder public we remove the file index.html because Ruby on Rails always looks for any static html file to load at the beginning.
6. We edit the file config/routes.rb adding this line to load our index file: root :to => "home#index"
7. We run the server writing the command ’rails server’ and we can see the Hello World in our web browser by writing http://localhost/3000

4.4.2. GWT

In order to create this application with GWT we have proceeded the following steps:

1. In Eclipse we create a New Web Application Project called “HelloWorld”.
2. We mark the Use Google Web Toolkit check box.
3. We mark also Generate GWT project sample code for making it easy to create the application.
4. Once the application is created, we only have to edit the file HelloWorld.html to show the Hello World!! message inside <h1>
5. If we want to delete all the components GWT has inserted in the document we just have to delete the lines we do not need and delete the lines associated in HelloWorld.java also.
6. We run the server clicking the button Run in Eclipse IDE and we can see the Hello World in our web browser by writing http://127.0.0.1:8888/HelloWorld.html?gwt.codesvr=127.0.0.1:9997

4.4.3. Comparison

For both basic applications the time we spent was less than 30 minutes. It is a very simple example but even here we can see how with Ruby on Rails we have to work with HTML tags directly and with GWT we can work both in Java or using HTML. In this case if we would have used Java, the work would have been harder and we would have needed more time to develop the small application.

Other things we would like to point out, is that in Ruby on Rails we had to modify configuration files and write some commands in the console while in GWT we can do everything using only the IDE and writing code.
Finally, speaking about the code lines we had to write, using Ruby on Rails we only needed to add one code line in the HTML code and modify one code line in the `routes.rb` file. On the other hand, using GWT we also had to add one code line in the HTML code and specify the entry point class in the `gwt.xml` file (one code line).

### 4.5. Implement the classes and interfaces

In this case we are going to compare the work we had to use to implement and design the main classes and interfaces of the system. We have done it in both programming languages; Ruby and Java.

#### 4.5.1. Ruby on Rails

Developing the classes in Ruby has been quite easy. Once we have a clear view how Rails works with the classes, how to connect them to the database and also how to relate the classes between them, it is quite easy to create the main classes. In Rails we do not need to create the getters and setters because it automatically creates these methods. It is also easy to relate the classes just adding some tags we have to know by reading Rails documentation. Here we can see the Customer class in Ruby:

```ruby
class Customer < ActiveRecord::Base
  validates :name, :presence => true,
            :length => { :maximum => 100 }
  validates :address, :length => { :maximum => 200 }
  validates :city, :length => { :maximum => 40 }
  validates :phone, :presence => true,
            :length => { :maximum => 20 },
            :numericality => { :only_integer => true }
  validates :email, :presence => true,
            :length => { :maximum => 60 }
  validates :web, :length => { :maximum => 40 }
  has_many :projects, :dependent => :destroy
end
```

As we can see, inheriting from `ActiveRecord::Base`, the class uses the Customers table in the database and in the class definition we only have to validate the data that the user has to introduce before saving the information in the database. In this case, we are checking the presence of some fields and the length allowed for them.
In the last part of the class we can see how easy it is to relate this class (Customer) with the class Project, saying also that in case one customer is deleted, the application has to destroy the projects of the customer also.

All the models have to be included in the folder ‘models’ and for creating these models we can use “scaffolding”. This is a command that developers can use for creating all the structure of any application in Rails. For example we can use it like this:

```
$ rails generate scaffold Card name:string title:string content:text
```

This command will generate a model called Card and its fields will be name, title and content, all of them string. This command also will create the views, controllers and helpers for the Card model including the basic operations and edit, show, etc. Hence, we have quite powerful tools for developing faster and safer.

### 4.5.2. GWT

In order to develop the main classes for GWT, we have to implement them in Java and thereupon, at the beginning it can seem that is easier but, when we start to work on it, we see we have to create by hand, all the getters and setters and also the constructors. To make this work, we first have to create the attributes and after write all the methods we need.

We can see below the implementation of the Customer class written in Java and having the same behavior than the same class in Ruby:

```java
public class Customer implements Serializable {
    private static final long serialVersionUID = 3L;

    Integer id;
    String name;
    String address;
    String city;
    String phone;
    String email;
    String web;
    String notes;
    Set<Project> projects;

    public Customer() {
    }
    public Customer(Integer id) {
        this.id = id;
    }
}
```
public String getAddress() {
    return address;
}
public void setAddress(String address) {
    this.address = address;
}
public String getCity() {
    return city;
}
public void setCity(String city) {
    this.city = city;
}
public String getWeb() {
    return web;
}
public void setWeb(String web) {
    this.web = web;
}
public String getNotes() {
    return notes;
}
public void setNotes(String notes) {
    this.notes = notes;
}
public Integer getId() {
    return id;
}
public void setId(Integer id) {
    this.id = id;
}
public String getName() {
    return name;
}
public void setName(String name) {
    this.name = name;
}
public String getEmail() {
    return email;
}
public String getPhone() {
    return phone;
}
public void setEmail(String email) {
    this.email = email;
}
public void setPhone(String phone) {
    this.phone = phone;
}
As we can see we have to write so much source code to have the same features we had in Ruby. It is not a hard job but the effort is quite big.

4.5.3. Comparison

Looking at the class Customer we have implemented before, we can easily see that more of the effort to develop in Java is by writing code and the different methods. So if we look at the time we have spent (2 hours for Ruby and 4 hours for GWT) we can say that implementing the classes is faster and maybe more enjoyable in Rails. This means that once we know perfectly how we have to write the code in Ruby, developing new classes is something easy and something that does not last a long time.

We also want to point out that in Ruby on Rails we learn while we write and we focus on developing the model whereas in Java + GWT we spend so much time by writing the repetitive code. We can also say and see that the number of source code lines is completely different so in less than the half of lines in Rails, we can have the same performance than in Java. In detail, we have written 139 code lines for implementing the main classes of the model in Ruby on Rails, and 967 code lines in GWT for implementing the same functionality more or less.

In short, based on this comparison, we can say that in Ruby on Rails, we need to write less code, spend less time and make less effort. We only have to study a bit before start working on it but once we have the knowledge, it is something quite easy.
4.6. Use of MVC Pattern

Model-View-Controller is an architectural pattern used in most of the frameworks we can find nowadays. Below we are going to see the use that these two frameworks make of it.

4.6.1. Ruby on Rails

In the next lines we are going to describe how Ruby on Rails uses the MVC as its architecture. As every system with MVC, Ruby on Rails has the next layers:

**Models:** they are used in our case to interact with the tables in the database, so, in our system we have four models corresponding four tables in the database. The models are: user, customer, task and project.

Rails provides some components we can use for implement the model of our system. First of all we have Active Model that provides an interface between the Controller of the system (Action Controller) and, in case we use one, any ORM we use for persistence.

By default Rails provides the component Active Record. This component is an ORM that provides CRUD functionality, database independence and many other features. We can use another ORM in case we do not want to use Active Record.

Rails framework also provides Active Resource that implements a way to map CRUD semantics with local objects.

**Views:** in our application they are the user interface so HTML documents with Ruby code embedded. View show through the web browser the data to the user.

Rails provides Active View for managing the views in our system. This component can create HTML and XML outputs, render templates and it also includes built-in AJAX support.

**Controllers:** they are the link between the models and the views and they are responsible for requesting the models the data that is required from the view.

For the controller, Ruby on Rails includes the Action Controller component for managing the incoming requests, dispatching them to right resource. It also includes tools for managing sessions, template rendering and redirections.

Finally, Rails also includes Action Dispatch to handle routing for web requests.
4.6.2. GWT

Google Web Toolkit by default does not use the MVC so we tried to implement it adding new plug-ins as gwt-mvc but finally we could not use it so far. We realised that since GWT is a framework mostly used for the client side of a web development, it does not really need to use the MVC. In any case it is strongly recommended to use this pattern so we think it would be a good improvement for the future of this framework.

As a Java framework one thing we can do is to develop the Java application using MVC pattern and use only GWT for the View. That means that GWT can only cover the View part of an application.

4.6.3. Comparison

As we cannot compare the use of MVC pattern because GWT does not use it, we can only say that the MVC components Ruby on Rails provides are more than useful and very powerful tools helping the web development, following the MVC pattern, to be easier, very productive and intuitive. We can say that Rails is 100% MVC and this makes this tool a great framework for web developing.

On the other hand, GWT does not use the MVC pattern and although there are some projects trying to change this, nowadays most of them are still beta projects. For that reason we cannot use the MVC pattern in our GWT development to compare it with the Ruby on Rails one.

In this comparison we cannot speak about code lines or time spent because as we said before GWT does not make use of this pattern and we have not used it in the GWT project.

4.7. Connection to the Database

In this section we are going to compare the difficulties and facilities that both frameworks give us during the development for connecting to the database. We have used MySQL as database for both applications.

4.7.1. Ruby on Rails

One of the best experiences during the development with Ruby on Rails has been the use of the tools that it provides for the database connection and the persistence. As we said before, Rails includes the component Active Record, an
ORM that makes quite easily all the tasks that need to perform with the database and our models.

Active Record is very powerful but it is also very easy to use and actually some of the new ORM of the market try to copy its features and abilities because the development is very comfortable with it. For using it, we only have to inherit our model classes from ActiveRecord::Base and write the restrictions and methods we want to use with this model.

To configure the database access in Rails, first of all we have to edit the file config/database.yml and set our data connection to the database we want to use. In this file we can also chose the adapter we are going to use. In this case, mysql. Here we have our database.yml document:

```yaml
# MySQL
#   gem install mysql
development:
  adapter: mysql
  encoding: utf8
  database: project_dev
  pool: 5
  username: root
  password: 1331
  host: localhost
```

As we also said before, we can create models using the `generate` command in the console to generate the models we want to create. This command also creates files called Migrations. They are files with sentences that ruby can understand and translate to SQL instructions to create/modify and delete tables in the database. Below we can see an example of a Migration file:

```ruby
class CreateCustomers < ActiveRecord::Migration
def self.up
  create_table :customers do |t|
    t.string :name
    t.string :address
    t.string :city
    t.string :phone
    t.string :email
    t.string :web
    t.string :notes
  end
end
def self.down
  drop_table :customers
end
end
```
These files are created automatically by using the command `generate` and for using it with the database we only have to execute the command `rake db:migrate` in the console. This also writes the structure of the table in the file `db/schema.rb` where we can see the whole database structure in Rails format.

In our project we needed to look up all the data in the database from the different entities of the system, users, projects, etc. To make these queries we found that with Active Record, this task is very easy, fast and intuitive. For example, if we look at the Log in system we can see that we can create a function without writing a code like this:

```
find_by_username_and_password(@user.username, @user.password)
```

We have not defined this function, but Rails just uses the Find function and checks the user name and password just by taking this idea from the name of the function. It is really incredible how easy this kind of tasks can be with Rails. We can also have a look in the defined functions Search and Filter from the Project model:

```
def self.search(search)
    if search
        find(:all, :conditions => ['name LIKE ?', "%#{search}"])
    else
        find(:all)
    end
end

def self.filter(filter)
    if filter != "all"
        (filter == "open") ? filter = "0" : filter = "1"
        find(:all, :conditions => ['status = ' + filter])
    else
        find(:all)
    end
end
```

We can also see how powerful is the `Find` function. We can pass it through many conditions and parameters to perform the right query and it will translate it to an SQL query.

Finally if we speak about the many to many relationships, we also find how easy is to make this work with Rails since it automatically saves the data in the database just by saying to the model implementation that it has this kind of relationship with another model. If we see in Task model we have this sentence that indicates this:
has_and_belongs_to_many :users

In short, the work with the database with Rails is completely achieved with the least effort possible and with a great group of tools that help the developer.

4.7.2. GWT

The first problem we have found trying to connect to the database in the GWT project is that we cannot find any tool to connect to any database. GWT does not include any component for that so we decided to integrate any ORM in the system to perform this.

As we did not know too much about the different options we could use, we started to try some free ORMs looking for a tool that can help us as Active Record does with Rails.

The first tools we tried were ActiveJDBC and ActiveObjects. Both are almost new projects and both have tried to work as Active Record work in Rails. We finally could not use any of them because of different problems about versions, incompatibilities, etc. So we used Hibernate (Using GWT with Hibernate – GWT – Google Code, 2009) for persistence in our GWT project.

Configuring Hibernate was relevantly easy at the beginning but when we wanted to start to use it we had many problems. Knowing that Hibernate is a great ORM for Java we had many problems with it. First of all we had to create many XML documents. These documents are very strict with names and tags so in case you make a mistake it is not easy to find the right solution. So for that reason we have lost so much time configuring Hibernate with GWT.

Once Hibernate is already configured and running, we can see the examples of files we have to create. First of all, this is the Hibernate configuration file:

```xml
<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE hibernate-configuration PUBLIC
 "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
 "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">
<hibernate-configuration>

<session-factory>

<!-- Database connection settings -->
<property name="connection.driver_class">com.mysql.jdbc.Driver</property>
<property name="connection.url">jdbc:mysql://localhost:3306/projects</property>
<property name="connection.username">root</property>
<property name="connection.password">1331</property>
```
As we can see we have to map all the resources to another XML file and we also have to configure the connection data to the database.

Apart from that file, we have to create one XML file for each model we have in our application and Hibernate will use this XML file to map the fields of the database with the attribute of the class. One example of these XML files is the next:

```xml
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC
 "-/Hibernate/Hibernate Mapping DTD 3.0//EN"
 "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<hibernate-mapping>
  <class name="lnu.project.domain.User" table="USER">
    <id name="id" column="ID">
      <generator class="native"/>
    </id>
    <property name="name"/>
  </class>
</hibernate-mapping>
```
In summary, we have to be very careful with these XML files and we have to be sure that we have created them for implementing the application.

4.7.3. Comparison

Comparing both frameworks in access to the database refers, the first thing we can say is that in Ruby on Rails we have an ORM system and in GWT we do not have it. This is a great point for Rails and it is even more important if the ORM is Active Record. This component is completely amazing and the work using it is very comfortable and safe. Of course we can use any ORM with GWT but for that we have to find which one is compatible with GWT, configure it, test it, etc. So if we speak about effort and the tools, Ruby on Rails and Active Record are a much better option than GWT and any other ORM used.

We can also compare the time we have used in both projects, and in this case, Ruby on Rails wins also in this aspect of the development. While we have spent 2 hours to make all the database tasks with Ruby on Rails, with GWT and Hibernate we spent more than 8 hours. It is true that we had to find the best solution for that, in this case Hibernate, but it is also true that we had to manage and create too many XML files with Hibernate and this can get a bit complex sometimes.

Regarding to the code lines we had to write, in Ruby on Rails all the code we needed to write is included in the 139 code lines we wrote for the model classes. This means we need less than 139 code lines for implementing all the functionality we need. On the other hand, with GWT we needed more than 500 code lines in Java and around 100 code lines in XML.

4.8. Sessions and Log in System

Next we are going to point out the work we have made for creating sessions, keeping the sessions active and developing the Log in system. We will describe the work we have made and the tools we have used with each framework.
4.8.1. Ruby on Rails

This part of the application has been one of the first we have programmed with Ruby on Rails and we could say that it has been quite easy and we did not need that much time and effort to make it.

The first step we made was the Log in system. For this reason we created the log in screen in the view and added the log out button on each page of the application. Once we were done with this, we used the powerful of Active Record to check if the user name and password that the user has entered are correct. If this is the case, we allow the user to log in into the system.

Next step, once the user is logged in, we had to create the session on the system that allows the user to be online for a certain time allowing him to visit all the pages the system has. So if we see the code:

```ruby
user = User.find_by_username_and_password(@user.username, @user.password)
if user
  session[:user_id] = user.id
  user.login!(session)
else
  @user.clear_password!
  flash[:notice] = "The user name or password is invalid"
end
```

We are using the session variable to keep the user online in case he has access to the system. Now we look at the User class to see the method "login!"

```ruby
def login!(session)
  session[:user_id] = self.id
end
```

Moreover, we can see also the log out functions both in the controller and in the model:

- User controller:

```ruby
def logout
  User.logout!(session)
  flash[:notice] = "See you soon"
  redirect_to :action => "login"
end
```

- User model:

```ruby
def self.logout!(session)
  session[:user_id] = nil
end
```

Another feature we have implemented is to protect the web pages against users who are not logged in. This protects the pages against unauthorized visitors. For that we have added this sentence in each controller of the system:
before_filter :login_required

And we also have to add this method to the application_helper.rb file to check if the user is logged in or not:

```
module ApplicationHelper
  def logged_in?
    not session[:user_id].nil?
  end
end
```

And this one to the user_controller.rb file to protect the pages:

```
def protect
  unless logged_in?
    session[:protected_page] = request.request_uri
    flash[:notice] = 'You need to login before you can view that page.'
    redirect_to :action => "login"
    return false
  end
end
```

Finally, we have to mention that we used the database for sessions instead of cookies because we thought it would be safer for important information as user names and passwords. In order to use the database for this task, first of all we have to edit the file session_store.rb changing the property and we also have to create the session table through the command:

```
rails generate session_migration
```

### 4.8.2. GWT

For performing the Log in system in GWT, we had to use Java and Java sessions but first of all we created the method that checks the user name and password with Java and Hibernate and once the user is authorized to enter in the system we had to create the session variable. For that purpose, we use Java code as this:

```
HttpSession session = request.getSession();
```

And we can set and load attributes from the session using the following lines:

```
session.setAttribute(name, data_to_save);
Object obj = session.getAttribute(name);
```

So by using this class and its methods we can easily check the session and protect the pages we do not want to show to users who are not logged in.
4.8.3. Comparison

In this aspect of the development we can see that both technologies give us utilities to create sessions, keep them and use them to protect pages and resources.

Using Ruby on Rails it is quite easy to manage sessions and create the login system. It was only a bit complicated to get to know how to use the session variable.

Finally, speaking about GWT, this framework does not manage session by itself but we can use Java code for doing it since GWT works always under a Java development.

So in summary, we can say that in each case the work with sessions has been satisfactory and we have spent 4 hours on each project for this task. Otherwise, regarding to the number of code lines we had to write, with Ruby on Rails we needed 75 code lines in total (including ruby and HTML code) and with GWT we needed 198 code lines including java and HTML code.

4.9. GUI

Perhaps this is the part of the thesis where we can compare better the two frameworks because, as we said before, GWT is mostly oriented to the user interface so in this section we will be able to speak more about the possibilities that GWT offers us. We will compare them with Ruby on Rails that also provides very good tools for developing the user interface.

4.9.1. Ruby on Rails

Next we are going to describe the tools that Rails provide us during the web development. We have seen how this framework has given us so many tools for the Model and for the Controller, tools for the database etc. In this case, we are going to focus on the view layer, the user interface or GUI.

Looking at the Rails guides (Ruby on Rails Guides, 2011) and the API (Ruby on Rails Documentation, 2011), we can see that the framework has many tags, helpers and other tools for creating HTML code without writing code but, we see that we also can write HTML code and CSS and use these technologies to create the user interface we want.

If we speak about styles, we have to speak about CSS. In Ruby on Rails one CSS file is created by default and it is placed in the folder public/stylesheets. We can edit this file and we can also create other files that we can include in our project. We only have to add this line:

```ruby
<%= stylesheet_link_tag :all %>
```
In our `views/layouts/application.html.erb` file to include all the CSS files we have placed in the stylesheets folder.

Concerning HTML files generated with Ruby on Rails, we see they are HTML files with ruby code embedded. For that reason we can structure our files as needed and after that we can apply to the styles. This file extension is html.erb. We have a view folder for each model and we can place on it as many files as we need to create the view of this model. In our case we mainly have this files: `_form.html.erb, edit.html.erb, new.html.erb, show.html.erb, index.html.erb.`

Once we have the HTML files created, we can use inside them any helpers we have created for the model, and for generating HTML code with Ruby, we have many modules and methods that generate HTML tags automatically linking them to records from the database. They are very useful and easy to use and, maybe the best advantage, they generate HTML 5 code following the standard of this new version of HTML with the advantages that means. For example, for creating HTML forms we have the modules: `ActionView::Helpers::FormHelper` and `ActionView::Helpers::FormTagHelper`.

FormHelpers has been designed to create forms in an easier way. FormHelpers are:

- check_box
- email_field
- fields_for
- file_field
- form_for
- hidden_field
- label
- number_field
- password_field
- radio_button
- range_field
- search_field
- telephone_field
- text_area
- text_field
- url_field

We can visit Ruby on Rails API to see how they work (Ruby on Rails – ActionView Helpers FormHelper, 2011). As we said before, we also have FormTagHelpers that also help in the implementation of forms in our HTML file. This tag helpers are:

- check_box_tag
- email_field_tag
- field_set_tag
- file_field_tag
- form_tag
- hidden_field_tag
- image_submit_tag
We can also see in the API all the information about this module (Ruby on Rails – ActionView Helpers FormTagHelper, 2011).

One of the best advantages of these Form Helpers is that they includes some validation for the fields and they check the values showing by JavaScript the errors they find. For example, if we use phone_field, it will check in the value entered is a number and it looks like a phone number. For email_field, it will check if the value entered is an email, this means, it it has an @ symbol, dots, etc. This feature helps the developers a lot and makes the development going faster.

Furthermore, we also have “helpers” for the navigation inside the HTML files. For that purpose we have the module ActionView::Helpers::UrlHelper. These are the methods we can find in this module:

- button_to
- current_page?
- link_to
- link_to_if
- link_to_unless
- link_to_unless_current
- mail_to
- url_for
- url_options

We can visit its API page for learn more about these helpers (Ruby on Rails – ActionView Helpers UrlHelper, 2011).

Finally, we are going to explain how Ruby on Rails works with AJAX and the utilities that it gives us. By default Rails includes Prototype JavaScript Framework, but we can change it and use another one. It uses this framework to improve the user experience. So, let’s see what do we have to do to use AJAX in our pages. For example, if we want to insert a form in a database without submitting it and redirecting, we have to change the code from here:

<%=< form_for(@user) do |f| %>
To here:

```ruby
<%= form_for(@post, :remote => true) do |f| %>

As easy as that, we only have to add the option `:remote => true` and Rails will transform it to the corresponding HTML 5 tag for using Prototype. Here another example:

```ruby
<%= link_to 'Destroy', user, :confirm => 'Are you sure?', :method => :delete, :remote=>true, :class=>'delete_user' %>
```

4.9.2. GWT

Google Web Toolkit provides Java developers an easy way to create HTML pages without writing HTML code and for this reason this framework is strongly recommended to this particular group of developers.

This framework translates the Java code into HTML and JavaScript code so if we need to change anything in the final view of the page, we have to change it in the Java file and then compile it to generate again the HTML and JavaScript code.

In the following lines, we are going to describe the use we have done of the utilities GWT brings. All the components included in GWT work as AWT and SWING do in Java, we have to import the components we need and after that we have to add them to another component that allows us to include components inside, e.g.: panels, windows, etc. In GWT they call them panels and Widgets for the rest of the components. Next we can see a list of some of the Widgets available in GWT:

- Button
- CheckBox
- DatePicker
- DialogBox
- Hyperlink
- ListBox
- MenuBar
- PasswordTextBox
- PushButton
- RadioButton
- SuggestBox
- RichTextArea
- TabBar
- Table
- TextArea
- TextBox
- ToggleButton
- Tree
The best advantage of these components is that they use AJAX so we can see effects and other dynamic behaviors using these components. Now we can see a list with some of the panels available in GWT:

- DisclosurePane
- DockLayoutPanel
- FlowPanel
- HorizontalPanel
- HorizontalSplitPanel
- PopupPanel
- SplitLayoutPanel
- StackPanel
- StackLayoutPanel
- TabLayoutPanel
- TabPanel
- VerticalPanel
- VerticalSplitPanel

These panels work like other Java panels, so we have to add component inside them to configure the final view of the page.

4.9.3. Comparison

In this level of the comparison we can see that both frameworks have more or less the same level.

On the one hand, Rails provides so many modules and methods for developing in HTML 5, the last version of HTML and it also includes Prototype for AJAX and its use is very simple so this can give the developer a comfortable way to create great web applications.

On the other hand, Java developers will find GWT as an awesome tool for creating web pages since they are used to work with panels and components. They will not need also to learn HTML so it could be a good choice for them.

In summary, on the basis the two frameworks work perfectly with AJAX, if you are used to work with HTML and CSS, Ruby on Rails is the best choice, the helpers it provides are just prefect and although you have to learn how to use them, once you get it, developing is easier and faster. However, if you are used to programming in Java language, GWT is a fantastic tool that will allow you to develop web pages without spending too much time.

Finally if we speak about time, we spent more time with the GWT development because we are used to work with HTML and CSS but we found that the development with GWT has been very interesting and satisfactory also. So in detail, we spent around 10 hours with RoR and more than 30 hours with GWT; and we wrote around 1200 code lines for the Ruby on Rails project (including ruby, HTML and CSS code) and more than 3500 code lines in the GWT project, including java, HTML and CSS code.
4.10. Configuration Files

After the web development we want to point out the differences between the configuration files in Ruby on Rails and these files in GWT. We are going to show and describe the different files we have on each framework.

4.10.1. Ruby on Rails

Below we can see a table with the most important configuration files we can find a Ruby on Rails project and their description:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>config/application.rb</td>
<td>In this file we can configure some properties for the whole application like the encoding type, time zone, plug-ins, etc.</td>
</tr>
<tr>
<td>config/database.yml</td>
<td>This file contains all the information about how to connect with the database.</td>
</tr>
<tr>
<td>config/routes.rb</td>
<td>In this file we can load all the resources we need in the application. We can also define the root file to load and route resources matching them with words.</td>
</tr>
<tr>
<td>config/environments/*.rb</td>
<td>In this folder we can find 3 different files, one for development, one for production and one for test. They are files we can configure depending in which fase of the development we are.</td>
</tr>
<tr>
<td>config/initializers/session_store.rb</td>
<td>In this file we can configure how we want to save the sessions in the application. We can also configure some parameters like the key to use.</td>
</tr>
</tbody>
</table>

Table 4.1. Ruby on Rails Configuration Files

4.10.2. GWT

Below we can see a table with some of the configuration files we can find in GWT and their description. We do not include the configuration files of each library added to the project neither the XML files we create for each class for Hibernate.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Project.gwt.xml</td>
<td>In this file we configure GWT. We can inherit classes, styles, etc. We also define the entry point of the application and the CSS.</td>
</tr>
<tr>
<td>hibernate.cfg.xml</td>
<td>Configuration files for Hibernate. We specify the data for the database connection and some other mappings.</td>
</tr>
</tbody>
</table>
This file is used when we want to create an App Engine application.

This file defines the mapping between the servlets and the URLs of the system. It also defines the welcome file for the application.

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>appengine-web.xml</td>
<td>This file is used when we want to create an App Engine application.</td>
</tr>
<tr>
<td>web.xml</td>
<td>This file defines the mapping between the servlets and the URLs of the system. It also defines the welcome file for the application.</td>
</tr>
</tbody>
</table>

### 4.10.3. Comparison

We see that in both frameworks we have to use some configuration files. In Ruby on Rails we have ruby files and in GWT we have XML files. We cannot say which one is better or not as we believe XML are perfect files for this purpose but ruby files are also good.

Before we started with Ruby on Rails we have never used Ruby files but now we can see how powerful they can be since we can write Ruby code following some rules, specially in `routes.rb` file.

Comparing both we can say that with XML we have very structured files but sometimes they are a bit complicated to read and to edit. Still, we definitely believe they are one of the best options for that purpose. On the other hand, with ruby files we have so many possibilities to use Ruby language to create the file and configure the system. These files are not very structured and they are also more difficult to read but they seem that they work perfectly.

Regarding to the time we spent configuring these files, in Ruby on Rails we needed 1 hour more or less ad in GWT we needed around 2 hours because we needed to create one XML file for every class in the model.

Finally, if we look at the number of code lines we see that for the Ruby on Rails configuration files we have 277 code lines whereas in GWT we have 204. We have more or less the same number of code lines but while in GWT we have to write almost all the code by hand, in RoR we only need to change some code lines since when we create a new Ruby on Rails project, all the files and code lines are created automatically.

### 4.11. Testing the application

In this section, the last one, we are going to compare the utilities that both frameworks provide us to test our applications.

#### 4.11.1. Ruby on Rails

Every time we create a model and its controller, Rails also create some test files we can modify to add our own tests. These files are created in the `test` folder and we can easily create our test cases both for models and controllers.

If we want to test the controllers, we have to check the files inside the folder `test/functional` and as we can see every test case has to inherit from
ActionController::TestCase. In these files we can use assert instructions like for example: assert_response, assert_not_nil, assert_difference, assert_redirect_to, etc. Each one is used with one purpose and we can find one for every requirement we have.

Speaking about testing models, we have to check in the folder test/unit where we can find the test cases for the main classes of the application. We have to inherit from ActiveSupport::TestCase for creating the unit tests for each class and we can, of course, use assert instructions to test the classes.

Once we have all the tests designed and implemented we can, for example, run the tests by using the command:

```
rails db:test:load
-Itest test/unit/user_test.rb
```

There are also other test files we can configure and adapt to our requirements.

### 4.11.2. GWT

As it happened before with persistence, for testing GWT application we have to use an external program we can integrate in our project. In this case we used the popular JUnit since it can be included in GWT by importing the right library.

To configure JUnit we have to follow some instructions we can find in the official website and once we have it installed and working, we are ready to start developing tests. For that, the first step is to import JUnit by writing:

```
import com.google.gwt.junit.client.GWTTestCase;
```

After that, our test class has to inherit from GWTTestcase writing it like this:

```
public class UserTest extends GWTTestCase {}
```

And finally we can use assert instructions as we did with Ruby before. Some of these instructions are: assertNot Null, assertEquals, etc. We can run the tests using the Run or Debug button in Eclipse IDE selecting the correct option with JUnit.

### 4.11.3. Comparison

In this point we can see how the two frameworks have integrated tools to make tests to each class and to test the whole application. Ruby on Rails has its own tools and GWT uses the open source and free very popular tools JUnit. In this case JUnit seems more powerful than the Ruby on Rails tools but it does not mean that Rails tools are not good.

If we would have to choose between both speaking about testing, perhaps we will choose GWT and JUnit but only for the power that this tool has. Rails has the
same features so develop and test applications is still quite good using this
framework.

Looking at the time we spent doing the different tests we have done, we see
that we used same time to implement the tests, 4 hours each but we spent more
time running the test in Ruby on Rails since we had some problems to run them.
5. Results and discussion

If we take a look at the different comparisons we have made in the thesis, we can find the differences between Ruby on Rails and GWT concerning web development.

We have to keep in mind that both frameworks have not been developed for the same purpose. Both frameworks are Web Frameworks but on one hand we have Rails that we can use for the whole web development, including database access, MVC, etc. and on the other hand, we have GWT that is mostly used for the user interface part of a web development. For the development of a whole web system we need to use other tools integrated to GWT and based on Java.

Below, table 5.1 shows the comparison of the advantages and disadvantages for each one of the tasks we have developed during the development of our application.

<table>
<thead>
<tr>
<th>Framework</th>
<th>Ruby on Rails</th>
<th>Google Web Toolkit</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Set up</strong></td>
<td><strong>Aptana Studio</strong></td>
<td><strong>Configure Java</strong></td>
</tr>
<tr>
<td><strong>Learning</strong></td>
<td>Plenty of information on Internet</td>
<td>Hard and long learning curve</td>
</tr>
<tr>
<td><strong>Basic application, Hello World!</strong></td>
<td>Only writing HTML code</td>
<td>Modify configuration files</td>
</tr>
<tr>
<td><strong>Implement the classes and interfaces</strong></td>
<td>Less code, less time. ActiveRecord</td>
<td>Learn Ruby language before</td>
</tr>
<tr>
<td><strong>Use of MVC Pattern</strong></td>
<td>Implemented</td>
<td><strong>Not implemented</strong></td>
</tr>
<tr>
<td><strong>Connection with the Database</strong></td>
<td><strong>ORM included ActiveRecord Migrations</strong></td>
<td><strong>External ORMs can be used</strong></td>
</tr>
<tr>
<td><strong>Sessions and Log in system</strong></td>
<td>Easy to use and implement</td>
<td>Easy to use and implement using Java</td>
</tr>
<tr>
<td><strong>GUI</strong></td>
<td>Prototype HTML 5 Helpers/Tags</td>
<td>AJAX by writing Java code</td>
</tr>
<tr>
<td><strong>Configuration Files</strong></td>
<td>Ruby files</td>
<td>Learn Ruby language before</td>
</tr>
<tr>
<td><strong>Testing the application</strong></td>
<td>Test Cases ActiveSupport</td>
<td>Problems running tests</td>
</tr>
</tbody>
</table>

Table 5.1. Advantages and Disadvantages
<table>
<thead>
<tr>
<th>Framework</th>
<th>Ruby on Rails</th>
<th>Google Web Toolkit</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Task</strong></td>
<td><strong>Time</strong></td>
<td><strong>Code lines</strong></td>
</tr>
<tr>
<td>Set up</td>
<td>6 hours</td>
<td></td>
</tr>
<tr>
<td>Learning</td>
<td>14 hours</td>
<td>16.5</td>
</tr>
<tr>
<td>Basic application,</td>
<td>0.5</td>
<td>&lt; 10 lines</td>
</tr>
<tr>
<td>Hello World!</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Implement the</td>
<td>2 hours</td>
<td>≈140 lines</td>
</tr>
<tr>
<td>classes and</td>
<td></td>
<td></td>
</tr>
<tr>
<td>interfaces</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Use of MVC Pattern</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Connection with the</td>
<td>2 hours</td>
<td>&lt; 139 lines</td>
</tr>
<tr>
<td>Database</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sessions and Log in</td>
<td>4 hours</td>
<td>75 lines</td>
</tr>
<tr>
<td>system</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GUI</td>
<td>≈ 10 hours</td>
<td>≈ 1200</td>
</tr>
<tr>
<td>Configuration Files</td>
<td>1 hour</td>
<td>≈ 277 lines</td>
</tr>
<tr>
<td>Testing the</td>
<td>4 hours</td>
<td></td>
</tr>
<tr>
<td>application</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 5.2. Time and Code lines

Regarding the time we need to develop in both languages, we can say that we needed less time with Ruby on Rails than with GWT in the developing process. Furthermore, we can see also that using Ruby on Rails we have to write much less code than in GWT.

With Ruby on Rails, we have a harder curve of learning because of the differences existing between Ruby language and the languages we are used to use, but after learning, the development is faster and more effective giving us a great experience during the development. On the other hand, the learning curve for GWT is softer because we are used to use C-based programming languages but, we need much more time to develop the application which also means more source code lines.

In short, a Java programmer who does not want to change the programming language and develop rich web applications can easily use GWT since it saves time and energy. If one needs to use the MVC pattern, persistence, etc. he has to use other tools integrated with GWT like Hibernate, Spring, Struts, etc. However, in case a programmer is keen to learn a new technology, that may be the technology of the future, and develop rich web applications faster, very efficient and in some way more comfortable, he has to choose Ruby in Rails and its impressive tools that it provides.
6. Conclusion

Based on the results we have obtained, and we can see in Table 5.1 and Table 5.2, and our own opinion about the work made during the development, we want to point out that learning Ruby on Rails has been a great challenge for us. We would say that it was hard at the beginning but in the end we found an incredible framework that helps the developers a lot and gave us a great experience to work with. Specifically, as far as Active Record is concerned we discovered an ORM very easy to use, flexible and solid, facts that prove that it is an amazing tool. Thus, we will keep learning and hopefully keep developing in Ruby on Rails.

GWT framework has given us also a really good experience but we have noticed that we need other tools to have a great development and sometimes it has been very hard to integrate these tools and has caused many problems. Nevertheless, it is a really good tool for Java developments and has let us with the best impressions.

To finish this thesis, we want to mention that learning modern Web Frameworks has been a great experience for us and it has inspired us to study some other Web Frameworks of the market. Therefore we will continue learning these very useful tools and we will use them in our future developments.

In the future research, we should prove that other users agree or disagree with us about our conclusion and the comparison we have made, paying attention to their suggestions and points of view.
References

Figure References

Figure 2.1. mraible - History of Web Frameworks [image online]
Available at: <https://github.com/mraible/history-of-web-frameworks-timeline>
[Accessed August 2011]

Figure 2.2. Java BluePrints - MVC Pattern [image online]
Available at: <http://java.sun.com/blueprints/patterns/MVC-detailed.html>
[Accessed August 2011]

Figure 2.3. Libros Web - MVC Architecture [image online]
Available at: <http://www.librosweb.es/jobet/capitulo4/la_arquitectura_mvc.html>
[Accessed August 2011]

Figure 2.4. Ruby on Rails - Logo [image online]
Available at: <http://rubyonrails.org/>
[Accessed August 2011]

Figure 2.5. Windaroo Consulting Inc - Ruby on Rails Structure [image online]
Available at: <http://windarooci.com/service_list/view_service/1>
[Accessed August 2011]

Figure 2.6. Ruby on Rails - Architecture [image online]
Available at: <http://www.ils.unc.edu/~uskala/MVC.htm>
[Accessed August 2011]

Figure 2.7. GWT - Logo [image online]
Available at: <http://code.google.com/intl/es/webtoolkit/>
[Accessed August 2011]

Figure 2.8. Product Overview - GWT Architecture [image online]
Available at: <http://code.google.com/p/google-web-toolkit-doc-1-4/wiki/ProductOverview>
[Accessed August 2011]

Figure 3.1. Class Diagram
Designed with ArgoUML

Text References


(Huw Collingbourne, 2009) - The Book of Ruby (Huw Collingbourne - 2009)


Ruby on Rails [online] Available at: <http://www.rubyonrails.org> [Accessed August 2011]


ArgoUML [online]
Available at: <http://argouml.tigris.org/>
[Accessed July 2011]

Ruby Installer for Windows [online]
Available at: <http://rubyinstaller.org/downloads/>
[Accessed August 2011]

Development Kit - GitHub [online]
Available at: <https://github.com/oneclick/rubyinstaller/wiki/Development-Kit>
[Accessed August 2011]

Aptana Studio [online]
Available at: <http://www.aptana.com/products/studio3/download>
[Accessed August 2011]

Oracle – Java Platform, Enterprise Edition [online]
Available at: <http://www.oracle.com/technetwork/java/javaee/downloads/index.html>
[Accessed August 2011]

msysgit – Git for Windows [online]
Available at: <http://code.google.com/p/msysgit/>
[Accessed August 2011]

Eclipse Downloads [online]
Available at: <http://www.eclipse.org/>
[Accessed August 2011]

Downloads for KentBeck’s junit [online]
Available at: <https://github.com/KentBeck/junit/downloads>
[Accessed August 2011]

Hibernate – JBoss Community [online]
Available at: <http://www.hibernate.org/downloads>
[Accessed August 2011]

Ruby User’s Guide [online] (Updated in 2008)
Available at: <http://www.rubyist.net/~slagell/ruby/>
[Accessed August 2011]

Ruby on Rails Guides [online]
Available at: <http://guides.rubyonrails.org/>
[Accessed August 2011]

Ruby on Rails Documentation [online]
Available at: <http://api.rubyonrails.org/>[
[Accessed August 2011]

Getting Started – GWt – Google Code [online]
Available at: <http://code.google.com/intl/es/webtoolkit/doc/1.6/tutorial/gettingstarted.html>
[Accessed August 2011]

Java.net - ActiveObjects [online]
Available at: <http://java.net/projects/activeobjects/pages/Home> [Accessed August 2011]

activejdbc - Google Code [online]
Available at: <http://code.google.com/p/activejdbc/> [Accessed August 2011]

Using GWT with Hibernate – GWT – Google Code [online] (Updated on July 2009)

JUnit [online]
Available at: <http://junit.sourceforge.net/> [Accessed August 2011]

Ruby on Rails – ActionView Helpers FormHelper [online]
Available at: <http://api.rubyonrails.org/classes/ActionView/Helpers/FormHelper.html> [Accessed August 2011]

Ruby on Rails – ActionView Helpers FormTagHelper [online]
Available at: <http://api.rubyonrails.org/classes/ActionView/Helpers/FormTagHelper.html> [Accessed August 2011]

Ruby on Rails – ActionView Helpers UrlHelper [online]
Available at: <http://api.rubyonrails.org/classes/ActionView/Helpers/UrlHelper.html> [Accessed August 2011]