

A Practical Comparison of Agile Web Frameworks

www.blog.websitesframeworks.com



David Díaz Clavijo

Tutors:

Cayetano Guerra Artal

Lydia Esther Bolaños Medina

Alexis Quesada Arencibia

February 16, 2014

Contents

1	Introduction	1
1.1	Solution approach	1
2	Document structure	3
3	State of the art	5
3.1	A Comparison Model for Agile Web Frameworks	5
3.2	Matt Raible Comparison	7
3.3	Wahner web frameworks comparison based on types of websites	8
3.4	Conclusions	10
4	Objectives	11
5	Methodology	13
5.1	The comparison of web frameworks	13
5.1.1	Selection of the web frameworks	13
5.1.2	Test design	13
5.1.3	Test execution	13
5.1.4	The conclusions	14
5.2	Project diffusion	14
6	Resources	15
6.1	Hardware resources	15
6.2	Software resources	15
6.3	Service resources	15
7	Planning	17
8	Project dissemination - the Blog	21
8.1	Setting up the blog	21
8.2	Logo	22
8.3	Theme	22
8.4	Social networks	23
8.5	WordPress SEO	23
8.6	Avoiding spamming	23
8.7	Analyzing incoming traffic	23

9	Analysis - Which technologies to choose?	25
9.1	Types of web frameworks	25
9.2	Statistics usage of server-side programming language	26
9.3	Statistics usage of Web frameworks	28
9.3.1	Web frameworks usage growth	29
9.4	Web frameworks benchmarks	30
9.4.1	World Wide Wait	30
9.4.2	The Great Web Framework Shootout	31
9.4.3	Benchmarking Web Applications and Frameworks	33
9.4.4	Rails, Wicket, Grails, Play, Tapestry, Lift, JSP and Context	33
9.4.5	Conclusions	35
9.5	Decided technologies	35
10	Website and language exercises design	39
10.1	Student background	39
10.2	Chronological development	39
10.3	Steps	40
10.4	Timing steps	40
10.5	Documentation	41
10.6	Programming exercises	41
10.6.1	The Strings, Files and Regular Expressions exercise	41
10.6.2	The Numbers Set exercise	42
10.6.3	The Composite Pattern exercise	43
10.7	Web Application to develop	44
10.7.1	Requisites	44
10.7.2	Actors	44
10.7.3	Use cases	44
10.7.4	Data Model	46
10.7.5	Mockup	47
10.8	The order of development	58
11	Test execution	61
11.1	Ruby and Ruby on Rails	61
11.1.1	Ruby	61
11.1.2	Programming exercises with Ruby	64
11.1.3	Ruby on Rails	65
11.1.4	Website development with Rails	66
11.2	Groovy and Grails	70
11.2.1	Groovy	70
11.2.2	Programming exercises with Groovy	72
11.2.3	Grails	72
11.2.4	Website development with Grails	74
11.3	Python and Django	78
11.3.1	Python	78
11.3.2	Programming exercises with Python	80
11.3.3	Django	80
11.3.4	Website development with Django	85
11.4	PHP and CodeIgniter	89
11.4.1	PHP	89
11.4.2	Programming exercises with PHP	92

11.4.3	CodeIgniter	92
11.4.4	Website development with CodeIgniter	97
12	Test results	101
12.1	Programmimg languages	101
12.1.1	Comparison of developing time	101
12.1.2	Readability	102
12.1.3	Performance	103
12.1.4	Conclusions	106
12.2	Web framework comparison	107
12.2.1	Documentation	107
12.2.2	Implementing authentication	108
12.2.3	Implementing Blog, Post and Comment	109
12.2.4	Comparing general features	111
12.2.5	Comparing lines of code	113
12.3	Conclusions	114
12.3.1	Concerning the documentation	114
12.3.2	Concerning the authentication system	114
12.3.3	Concerning developing common operations: CRUD	114
12.3.4	Concerning the developing times	115
12.3.5	General conclusions	115
13	Conclusions and future works	117
13.1	Conclusions	117
13.2	Future works	118
14	Results of Project Diffusion	119
14.1	Key experts in dissemination	119
14.2	Social networks	119
14.3	Referrals	119
14.4	Visits	120
14.5	Search Engine Optimization	120
	Appendices	121
A	Source code of programming exercises	123
A.1	Source code of programming exercises in Ruby	123
A.1.1	Source code of The Strings, Files and Regular Expressions exercise in Ruby	123
A.1.2	Source code of The Numbers Set exercise in Ruby	124
A.1.3	Source code of The Composite Pattern exercise in Ruby	126
A.2	Source code of programming exercises in Groovy	128
A.3	Source code of programming exercises in Python	132
A.3.1	Source code of The Strings, Files and Regular Expressions exercise in Python	132
A.3.2	Source code of The Numbers Set exercise in Python	133
A.3.3	Source code of The Composite Pattern exercise in Python	135
A.4	Source code of programming exercises in PHP	136
A.4.1	Source code of The Strings, Files and Regular Expressions exercise in PHP	136
A.4.2	Source code of The Numbers Set exercise in PHP	137
A.4.3	Source code of The Composite Pattern exercise in PHP	140

B	Examples of implementation source code with different web frameworks	143
B.1	Code Generated vs Modified code in Rails and Grails	143
B.1.1	Rails generated code and adaptation	143
B.1.2	Grails generated code and adaptation	147
B.2	Source Code of Post in Django: controllers, views and templates	156
B.3	Source code of Post in CodeIgniter: models, controllers and views	160
B.4	Search posts implementation in different web frameworks	170
Bibliography		173
List of figures		174
List of tables		176
List of lists		177

Chapter 1

Introduction

Web application frameworks are defined as a set of classes that make up a reusable design for an application or, more commonly, one tier of one application. This specialization in tiers (persistence, web flow, ...) has led to their classification due to their proliferation. Agile web frameworks are defined as full stack web frameworks for developing an application. In contrast with web application frameworks, they are not specialized in one layer, but offer the full stack.

Agile web frameworks started with Ruby on Rails, which defined a new approach to web development, based on a single web framework. Ruby on Rails follows the principles of convention over configuration and *don't repeat yourself*, providing an agile web development framework that simplifies the development process and increases productivity for prototyping web applications.

When a subject, a programmer or a business, is beginning to develop a web project, it is a hard decision which agile web framework to choose. Learning how to use a full stack web framework is a high time consuming task, it can take two months to get started and one year to have some level of expertise. Therefore, experienced programmers can wonder if it is worthy to migrate from a known framework to a new one and inexperienced programmers or startups do not know which one to choose, neither they have tools to choose the right one.

This project tries to analyze the existing web agile frameworks, to choose, compare and review a set of them to facilitate stakeholders the selection of the most suitable for each case.

1.1 Solution approach

This project will make a test with a student. The student which will execute the test is the same person that designs it. The test will be the development of a website with different technologies. The student has a fixed time for learning each technology and developing the website. After the test is finished, it is analyzed the levels reached in development, lines of code and opinion of the subject.

Firstly, it is necessary to decide which technologies will be tested. It is considered interesting to compare a cross-language web frameworks set. In this manner, the ease of learning of the languages and the ease of learning of the web frameworks are tested.

Secondly, a website and a set of programming language exercises have to be designed. The exercises will be used to test language knowledge and the website development will be used to test the ease of

learning of the frameworks. These have to be minimalist because time for developing is fixed and short.

Thirdly, the experiment, learning technologies and development of the website, is done for each technology in fixed time. If the student is not able to finish within the deadline, development is left at the state it is at that moment.

Afterwards, based on this experience a review of each framework is done.

Chapter 2

Document structure

This document is divided into 14 chapters and appendices. The previous and first chapter is the introduction, where a description of the project is presented. The next list explains the rest of the chapters:

- **State of the art:** A set of previous comparisons is listed and analyzed
- **Objectives:** The objectives of the project are listed
- **Methodology:** The methodology to reach the objectives described in the previous chapter is explained
- **Resources:** Necessary resources to carry out the project are described
- **Planning:** The project steps and their schedule are shown
- **Project diffusion:** It is explained how the project will be spread
- **Analysis - Which technologies to choose?:** An analyses of the current web frameworks technologies and the reasons to choose four of them are exposed
- **Website and language exercise design:** It defines the student knowledge previous to this comparison, the website and exercises to be developed, the steps in development, developing times, documentation used and the order of development
- **Test execution:** It is explained the experience developing with each of the programming languages and frameworks. A brief introduction to each programming language and framework is done
- **Test results:** Results of the comparison are analyzed: times in development, readability of the solutions, ease of learning and development and lines of code, among others
- **Conclusions and future works:** The methodology followed in the comparison, pros, cons and results are considered. Furthermore, several proposals of future works in this area of knowledge are presented
- **Results of project diffusion:** The audience that this project has reached
- **Appendices:** Listing code, tables, figures and bibliography

Chapter 3

State of the art

Comparing web frameworks is not a mature discipline. There are a few conferences about this topic and only one university article. This task is hard due to the high complexity of full stack web frameworks. Learning how to use a full stack web framework is a high time consuming task, it can take two months to get started and one year to have some level of expertise. On the other hand, there are more than one hundred frameworks and their number is growing. Furthermore, there are some variables in comparison that are difficult to measure, for example the human perception about the framework. How can be measured the productivity? How much does the level of expertise with a framework influence in the productivity?

In this chapter a set of previous comparisons has been selected. There are a few more but they have been dismissed because of lacking accuracy. The first article analyzed is the university article. It is the oldest article but it is the most precise. The second article is the last comparison made by Matt Raible, a well known expert who compares web frameworks. He says his comparisons are not neutral but the matrix he uses is useful. Last comparison is a website type based comparison: it defines which kind of websites can be built and which frameworks are right to choose, the right framework for the right website.

3.1 A Comparison Model for Agile Web Frameworks

This article is part of a project by The Polytechnic University of Madrid which is now abandoned. It was written by José Ignacio Fernández-Villamor, Laura Díaz-Casillas and Carlos Á. Iglesias. It is a comparative study focused on agile web frameworks, high productivity frameworks. It compares Ruby on Rails and its Java partners: Roma, Grails and Trails.

The authors define a model composed by a set of parameters described in figure 3.1. To evaluate these parameters, they ask some questions and give them a grade. For example, for *domain & persistence*, one of the questions is: *Is domain definition automatically inferred from schema data?* and it has a 20% grade. Answers to the questions can be given a percentage too.

In figure 3.2, results are shown in a graphic way which leads us to understand that the best framework does not exists, but it is a question of parameters.

Nevertheless, It is interesting to know which would be the result if an addition of all the grades is calculated. Table 3.1 shows that Roma would be the first framework, followed by Rails, Trails and Grails.

The article tries to expose a model to compare agile web frameworks and to help in choosing among them. This objective is achieved and the model has been exposed.

Table 3.1: Addition of all the grades in the results of A Comparison Model for Agile Web Frameworks

Roma	Rails	Trails	Grails
750	650	600	475

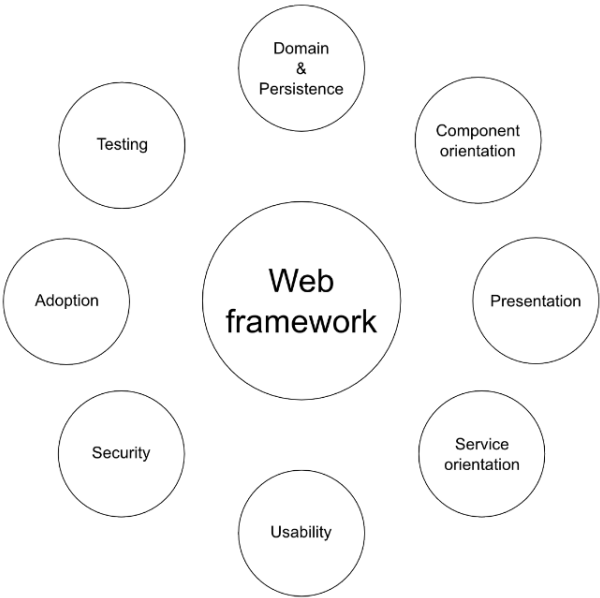


Figure 3.1: Comparison model. Source [1]

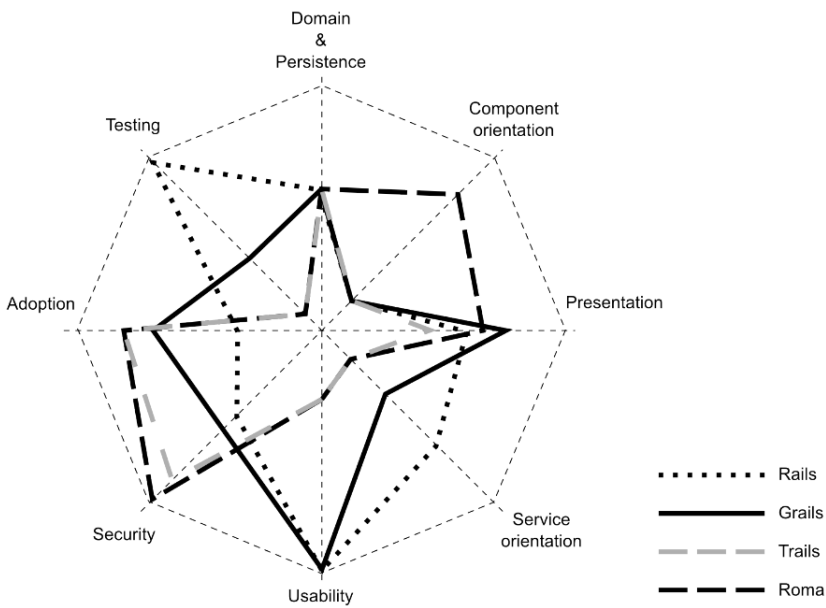


Figure 3.2: Framework evaluation results. Source [1]

3.2 Matt Raible Comparison

Matt Raible has been presenting Java web frameworks comparisons since 2007. Last presentation was at JFokus 2012. His presentations have been highly criticized due to the lack of objectivity. He says that it is his opinion. It is important to remember that comparing web frameworks is still not an established discipline and neutrality has not been reached yet.

The frameworks which have been compared in his last presentation are:

- Struts 2
- Spring MVC
- Wicket
- JSF 2
- Tapestry
- Stripes
- GWT
- Grails
- Flex
- Vaadin
- Lift
- Play
- Rails

In his presentation at JFokus 2012 he explains that there are too many frameworks, therefore, it is important to eliminate some of them from the list of choices. He argued why Struts is a framework which should be deleted from the list of choices. He shows how to choose a framework based on the type of application that will be built. For example, for a Rich Interface Application, closer to a desktop application, he recommends to use a component-based web framework.

Next, he explains that he built a matrix based on a list of features he considered to be important. He posted it to the community to receive feedback and add new features which should be part of the matrix. The result was a matrix for comparing different web frameworks. It is shown in figure 3.3. The matrix has a row per each defined parameter. Every web framework has a grade for each parameter and every parameter is weighed. The numbers in the matrix are just his opinion and the criteria for choosing them is far away from being scientific or precise. Mainly, there are 0 if it does not have this characteristic at all, 0.5 if it has it at some level and 1 if it is fully satisfied. Based on Matt Raible opinion, the winners are Grails, Spring, Rails and GWT [2].

Criteria	Struts 2	Spring MVC	Wicket	JSF 2	Tapestry	Stripes	GWT	Grails	Rails	Flex	Vaadin	Lift	Play
Developer Productivity	0.50	0.50	0.50	0.50	1.00	0.50	1.00	1.00	1.00	0.00	1.00	0.50	1.00
Developer Perception	0.50	1.00	1.00	0.00	0.50	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Learning Curve	1.00	1.00	0.50	0.50	0.50	1.00	1.00	1.00	1.00	1.00	1.00	0.50	1.00
Project Health	0.50	1.00	1.00	1.00	0.50	0.50	1.00	1.00	1.00	0.50	1.00	1.00	1.00
Developer Availability	0.50	1.00	0.50	1.00	1.00	0.50	1.00	0.50	1.00	1.00	0.50	0.00	0.50
Job Trends	1.00	1.00	0.50	1.00	0.50	0.00	1.00	0.50	1.00	1.00	0.00	0.00	0.50
Templating	1.00	1.00	1.00	0.50	1.00	1.00	0.50	1.00	1.00	0.50	0.50	0.50	0.50
Components	0.00	0.00	1.00	1.00	1.00	0.00	0.50	0.50	0.50	1.00	1.00	0.00	0.00
Ajax	0.50	1.00	0.50	0.50	0.50	0.50	1.00	0.50	0.50	0.50	1.00	1.00	0.50
Plugins or Add-Ons	0.50	0.00	1.00	1.00	0.50	0.00	1.00	1.00	1.00	1.00	1.00	0.50	1.00
Scalability	1.00	1.00	0.50	0.50	0.50	1.00	1.00	0.50	0.50	0.50	0.50	1.00	1.00
Testing	1.00	1.00	0.50	0.50	1.00	1.00	0.50	1.00	1.00	0.00	0.50	0.50	1.00
i18n and l10n	1.00	1.00	1.00	0.50	1.00	1.00	1.00	1.00	0.50	0.50	1.00	1.00	1.00
Validation	1.00	1.00	1.00	0.50	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.50	0.50
Multi-language Support (Groovy / Scala)	0.50	0.50	1.00	1.00	1.00	1.00	0.00	1.00	0.00	0.00	1.00	0.00	0.50
Quality of Documentation/Tutorials	0.50	1.00	0.50	0.50	0.50	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Books Published	1.00	1.00	0.50	1.00	0.50	0.50	1.00	1.00	1.00	1.00	0.50	0.50	0.00
REST Support (client and server)	0.50	1.00	0.50	0.00	0.50	0.50	0.50	1.00	1.00	0.50	0.50	0.50	0.50
Mobile / iPhone Support	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.50	1.00	1.00	1.00
Degree of Risk	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.50	0.50	0.50
Totals	14.5	17	15	13.5	15	14	17	17.5	17	13.5	15.5	11.5	14

Figure 3.3: Comparison frameworks matrix. Source [2]

In the rest of the speech, he made references to two benchmarks. The Peter Thomas *Perfbench* [3] and the World Wide Wait [4]. He showed pros and cons of Grails, GWT, Ruby on Rails, Spring MVC, Vaadin, Wicket and Tapestry. He showed some graphs with variables such as searching volume of the key words in Google, jobs on dice, mailing list traffic or skills in LinkedIn, among others. He referred to these graphs as *funny graphs* due to the different ways they could be interpreted. He showed a list of the frameworks that he considered the main innovators: Struts, Rails, GWT and Play!.

To finish the presentation he said "There is not best framework, just a lot of awesome choices" and recommended the audience to pick a set of frameworks they like and test them for a week with a small project.

3.3 Wahner web frameworks comparison based on types of websites

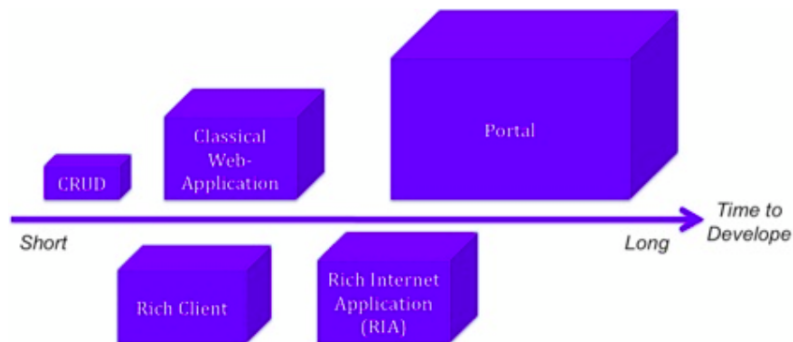


Figure 3.4: Website types. Source [5]

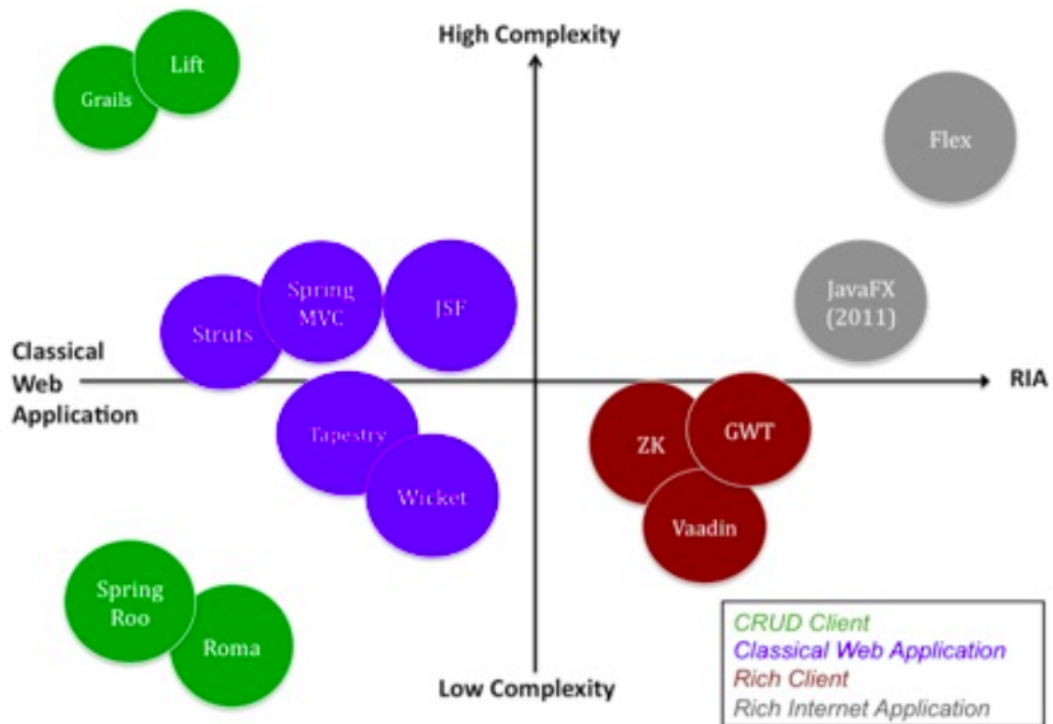


Figure 3.5: Frameworks complexity and website types. Source [5]

Wahner [5] published a more neutral web frameworks comparison through defining types of web applications and assigning a web framework to each of them. In figure 3.4, types of web applications are exposed.

The web frameworks in the comparison are:

- Grails
- Lift
- Struts
- Spring
- JSF
- Tapestry
- Wicket
- Roma
- ZK
- Vaadin
- GWT

- JavaFX
- Flex

CRUD client makes reference to a simple classical web application where the main actions are Create, Read, Update and Delete (CRUD). A portal makes reference to websites which contain more than one web application. RIA (Rich Internet Applications) refers to applications which need a plugin to satisfy their requirements, for example Adobe Flash. A Rich Client tries to create a better experience using HTML with AJAX.

Taking into account this typology, frameworks can be distributed according to figure 3.5 where complexity means difficulty to learn the framework for a Java developer.

For a CRUD client Spring Roo, Roma, Grails or Lift should be used. For a Classical Web Application Struts, Spring MVC, JSF, Tapestry and Wicket suit. While for a Rich Client GWT, Vaadin and ZK are recommended. JavaFX and Flex are used for RIA projects.

Wahner tries to expose a more neutral comparison which leads to an agreement among developers. Positions of the circles of the frameworks in the graph are approximated, they could be a little lower or higher.

3.4 Conclusions

More articles than those cited above have been found but they were a long talk about each framework. Those are the only ones which presented a final result in a graphic way and tried to approach a more neutral answer. The fact that no article is neutral, from my point of view, the best comparison has been proposed by Fernández-Villamor, Díaz-Casillas and Iglesias. Unfortunately, the article is a bit old and frameworks have changed enough to doubt about its results. A neutral comparison model has not been reached yet and discussions about the best framework, even based on types of websites, lead to opinions without proofs.

Chapter 4

Objectives

The objectives of this project are:

- To analyze the state of web frameworks technologies, focusing on agile web frameworks
- To compare a set of agile web frameworks through a practical test
- To learn to program with the web frameworks which present higher productivity
- Diffusion of the project

Chapter 5

Methodology

This project has two different objectives. The comparison of agile web frameworks and the project diffusion. First, it is described the methodology for the comparison of agile web frameworks, second, the methodology used for the project diffusion.

5.1 The comparison of web frameworks

To carry out the comparison, a set of web frameworks will be chosen, then it will be described a website to be built with each framework in order to test them. Afterwards, the website will be developed with each framework. Finally, conclusions will be made based on the results from the development.

5.1.1 Selection of the web frameworks

To select the web frameworks the following criteria will be used:

- Results in previous comparisons
- Results in benchmarks
- Statistics of usage
- Popularity in the community

5.1.2 Test design

The test will be developing a small website within deadlines which satisfy the project requirements. In order to design the test, to prove each framework, experts of agile web development and language learning will be interviewed. These experts will guide the test definition and will review the final website design.

5.1.3 Test execution

The student will execute the test with each of the technologies in a fixed time. He will develop the website with the rules specified by the test design.

5.1.4 The conclusions

When development with each of the web frameworks is finished, a view in perspective will be made. The following details will be compared:

- Line of codes
- Times in development and level of development reached
- Main problems encountered along the development process
- Readability of the final code
- Understandability of the solution
- Quality of the documentation
- The steps to implement some of the modules will be listed with each framework

5.2 Project diffusion

In order to diffuse the project, articles adapted from this report will be published in a blog. This blog will be SEO optimized and connected to social networks in order to get readers from search engines and social networks.

Chapter 6

Resources

6.1 Hardware resources

- Personal Computer : Toshiba U200-192
- Screen: SAMSUNG SyncMaster 940n

6.2 Software resources

- TexMaker
- Evince
- Libre Office
- Geany
- Enterprise Architect
- Balsamiq Mock up
- Ubuntu 12.04 64 bits
- Windows 7 64 bits
- Firefox

6.3 Service resources

- Hosting capable of executing Wordpress: This service is paid by the student. The business for the chosen service was **1and1** with a cost of 26 euros a year

Chapter 7

Planning

This project will have an investment of 800 hours of the student time. These hours will be distributed across five different main tasks. This distribution is showed in figure 7.1.

- State of the art (100 hours): Analyses of previous researches, statistics of web frameworks usage, interviews with experts in the area and decision of the set of frameworks to be compared
- Test description (50 hours): Definition of the test to be passed with each framework. Opinion of experts will be asked
- Application requirements and design (50 hours): Datamodel and mockup design of the application to be built. Interview with experts to review the decisions
- Implementation (500 hours): Passing the test with each of the web frameworks. It will take 125 hours with each framework
- Summary (100 hours): Analyzing results: Comparison of lines of code, developing times, found problems, readability of the solutions, quality of documentation, timing of steps, etc

The report will be written along the development of the project, therefore each quantum of time will be used for the task and reporting it. The blog for the project will be set up in the state of the art task.

Table 7.1: The project plan.

The project Plan																								
Activities	Hours																							
	100	200	300	400	500	600	700	800																
1. State-of-the-art																								
1.1. Analysis of previous researches in frameworks comparisons																								
1.2. Analysis of statistics of technology usage																								
1.3. Interview experts in the area in order to receive feedback about the project																								
1.4. Writing the report about the state-of-art																								

Continued on next page

Table 7.1 – Continued from previous page

Activities	Hours															
	100		200		300		400		500		600		700		800	
1.5. Deciding the selection of the web frameworks and giving reasons why they are chosen																
2. Test description																
2.1. Interview with experts in education about programming languages to define an exercise to test language knowledge																
2.2. Defining an exercise to be developed in the different programming languages to test programming language knowledge																
2.3. Interview with experts in web development to describe a website adequate in order to test the web frameworks																
2.4 Defining the website to develop in order to test the web frameworks																
3. Application requirements and design																
3.1. Defining website requirements																
3.2. Website design: Mockups and database																
3.3. Interview with experts in order to receive feedback																
4. Implementation																
4.1. First implementation																
4.1.1. Reading the programming language documentation																
4.1.2. Making the exercises to demonstrate language knowledge																
4.1.3. Reporting the language features and development experience																
4.1.4. Reading the web framework documentation																
4.1.5. Developing the website																
4.1.6. Reporting the web framework features and development experience																
4.2: Second implementation																
4.2.1. Reading the programming language documentation																
4.2.2. Making the exercises to demonstrate language knowledge																
4.2.3. Reporting the language features and development experience																

Continued on next page

Table 7.1 – Continued from previous page

Activities	Hours															
	100				200				300				400			
4.2.4. Reading the web framework documentation																
4.2.5. Developing the website																
4.2.6. Reporting the web framework features and development experience																
4.3: Third implementation																
4.3.1. Reading the programming language documentation																
4.3.2. Making the exercises to demonstrate language knowledge																
4.3.3. Reporting the language features and development experience																
4.3.4. Reading the web framework documentation																
4.3.5. Developing the website																
4.3.6. Reporting the web framework features and development experience																
4.4: Forth implementation																
4.4.1. Reading the programming language documentation																
4.4.2. Making the exercises to demonstrate language knowledge																
4.4.3. Reporting the language features and development experience																
4.4.4. Reading the web framework documentation																
4.4.5. Developing the website																
4.4.6. Reporting the web framework features and development experience																
5: Summary																
5.1. Analysis of development result with each technology																
5.2. Comparison of code lines and time to develop																
5.3. Evaluation of the frameworks																
	TOTAL HOURS															800

Chapter 8

Project dissemination - the Blog

A blog and social networks will be used to diffuse this project. During the reporting of the project, it will be tried to find articles which could be interesting for the Internet community to publish them. Afterwards, social networks will be used to diffuse articles. Figure 8.1 illustrates the methodology.

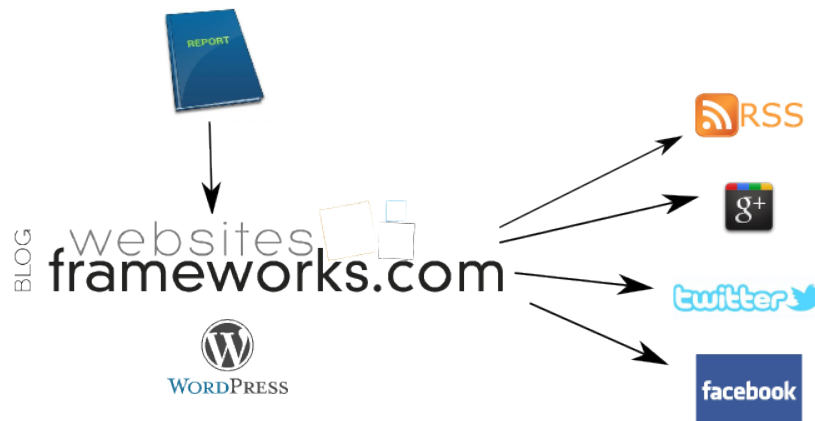


Figure 8.1: Diffusion diagram.

8.1 Setting up the blog

Setting up the blog is not the main objective of the project, therefore, it will be tried not to invest too many time. A thorough research about the different blog system is not done. Any tool which satisfies the project diffusion requisites is enough.

Requisites of the Blog :

- Easy configuration and deployment. It is not necessary to write code
- To have an editor which facilitates the task of publishing an article
- Integration with social networks
- Easy tools to improve SEO
- To avoid spamming

- To analyze incoming traffic

The software which could satisfy these requirements are:

- Wordpress
- Joomla
- Drupal

Joomla and Drupal are oriented to more complex applications than Wordpress. On the other hand, Wordpress is highly used, therefore it is more likely to be well supported. Furthermore, the student has previous knowledge of Wordpress because he has built up a blog before with this technology. Wordpress is easily customizable, it is not necessary to program code to have a functional blog: there are plugins for SEO optimization, integration with social networks, avoid spamming and analyses incoming traffic. These are the reasons why Wordpress is chosen.

8.2 Logo

The logo for the project that has been created is shown in figure 8.2. It was made by Jose Luis Spicoli Pereida, a student at the Art School in Gran Canaria. The square version of the logo in figure 8.3 will be used for profile pictures in social accounts.



Figure 8.2: Header logo of the blog for the project diffusion.



Figure 8.3: Square logo of the blog for the project diffusion.

8.3 Theme

It is needed a theme with a header and a two column layout. The header will be used to display the logo and the menu. The left column would be bigger and will be used to display the articles. The right column will be used to display the Twitter account information and recent comments. RtPanel, by RtCamp,

satisfies this common requirements.

8.4 Social networks

The next list of plugins satisfies the social requirements of the blog:

- Social by Mailchimp: It broadcasts newly published posts and pulls in discussions using integrations with Twitter and Facebook
- rtSocial: It displays buttons near the articles to share them in different social networks

8.5 WordPress SEO

The plugin Wordpress SEO offers tools to optimize search engine results.

- It counts density of the key word in the document
- It allows to define meta description, the focus key word and SEO Title
- It gives an automatic analysis and advises for improving SEO

8.6 Avoiding spamming

Akismet is a plugin included by default in a new Wordpress installation which protects the blog from spam comments.

8.7 Analyzing incoming traffic

To analyze incoming traffic, Google Analytics is used. Estefanía Díaz Clavijo, expertise in marketing online, was asked and she considered that Google Analytics will satisfy the requirements of this project. There is a plugin called Google Analytics for Wordpress which activates Google Analytics in Wordpress.

Chapter 9

Analysis - Which technologies to choose?

In this section, it will be analyzed useful information to decide which technologies to choose. First, a definition of web framework is given. Second, types of web frameworks are explained. Third, statistics usage of web technology are exposed. Fourth, benchmarks made to different technologies are shown. Finally, based on this analysis, a set of technologies is chosen.

9.1 Types of web frameworks

According to [6], there are five types of Java web frameworks: Request-based, Component-based, Hybrid, Meta and RIA-based framework.

- **Request-based:** The frameworks directly handle incoming requests. "Each request is essentially stateless but with server-side sessions, a certain degree of statefulness has been achieved" [6]
- **Component-based:** The framework abstracts the internals of the request handling and encapsulates the logic into reusable components. The state is automatically handle by the framework. "Together with some form of event handling, this development model is very similar to the features by desktop GUI toolkits"[6]
- **Hybrid Framework:** "The framework combines both request-based and component-based frameworks by taking control of the entire data and logic flow in a request-based model" [6]. Developers have full control over URLs, forms, parameters, cookies and pathinfos. "However, instead of mapping actions and controllers directly to the request, a hybrid framework provides a component object model that behaves identically in many different situations such as individual pages, intercepted request, portal-like pages fragments and integratable widgets. Components can be wired together and be packaged as groups that are components in their own right. They can be distributed separately and be seamlessly integrated into other projects. This combines the form of reusability in component-based frameworks with the raw controls of a request-based approach" [6]
- **Meta:** "The framework has a set of core interfaces for common services and a highly extensible backbone for integrating components and services"[6]." A meta framework is close to a framework of frameworks
- **RIA-based framework:** The framework for developing Rich Internet Applications (RIA). RIA "refers to a web page-based application running in a browser" with rich user interface features which are common in desktop application such as drag and drop. Generally, a plugin is needed to run the application or AJAX

There are more than 100 web frameworks. Therefore, a sift has to be applied to fit the project deadlines. In this context, this project will focus on request-based frameworks.

9.2 Statistics usage of server-side programming language

World Wide Web Technology Surveys (W3Techs) publishes daily statistics of usage of web technology. In this section, a data analysis is presented. It is interesting to know which technologies are used to build web applications using frameworks. Unfortunately, W3Techs does not offer data about frameworks usage. CMS and server-side usage are surveyed. Crossing this data, it will be tried to analyze which technologies are used in the non-CMS sites.

Table 9.1 shows that PHP is the most server-side language used. But the purpose at this section is finding the most used server-side language to build web applications using frameworks. Therefore, CMS might be taken out of the results.

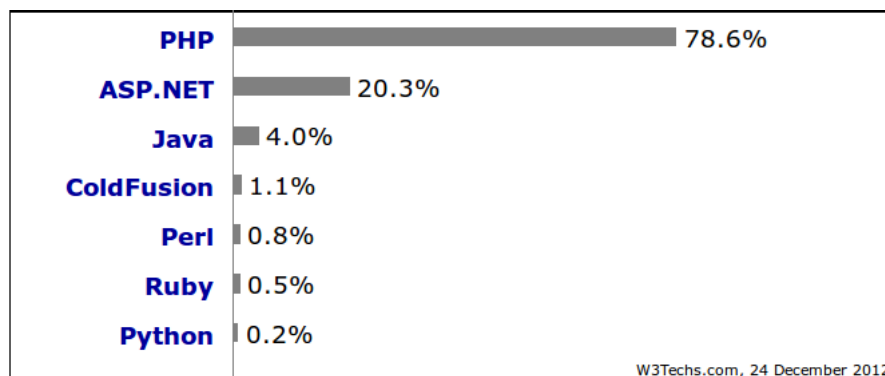


Figure 9.1: Usage of server-side programming languages for websites. Source [7]

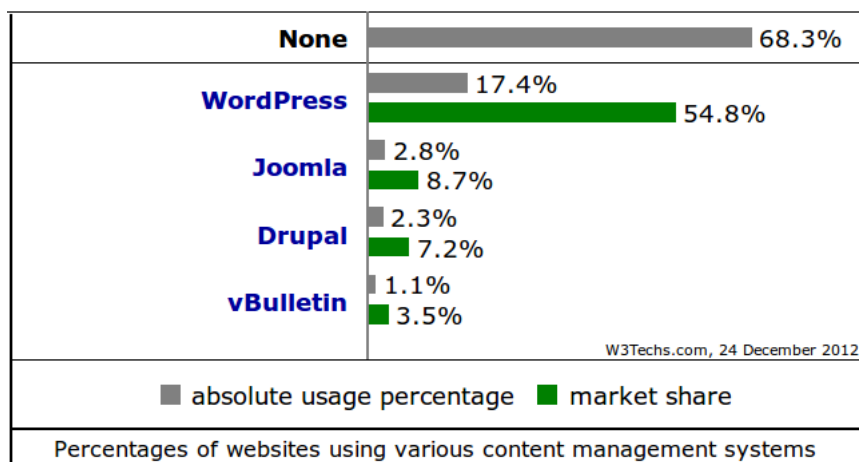


Figure 9.2: Usage of CMS for websites. Source [7]

Table 9.2 shows the high usage of CMS: Wordpress, Joomla, Drupal, VBulletin, Typo3 are built with PHP. Figure 9.3 is the result of subtracting the usage of CMS from the PHP statistics. PHP is still the

most used server-side language but with a smaller advantage. This table gives a more accurate view of the client-side language usage in applications built with frameworks.

Figure 9.4 shows that PHP is the most used language in websites with low level traffic. While ASP.NET is in the middle between those used by many sites and high traffic sites. Java, Javascript, ColdFusion, Perl, Ruby, Python and Lasso are used in few sites but with high traffic.

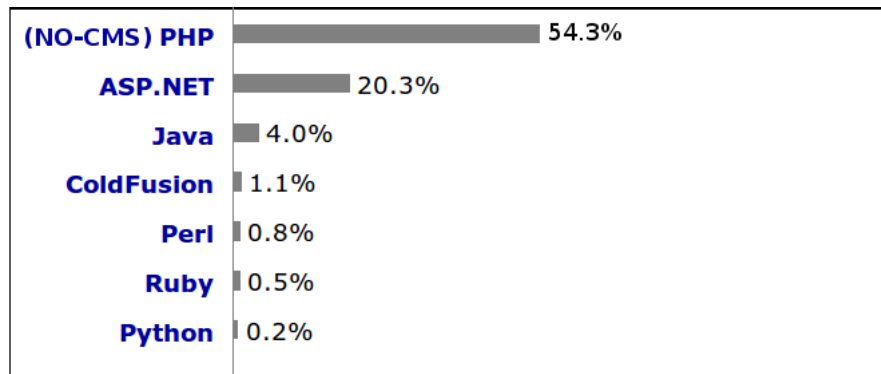


Figure 9.3: Usage of server-side programming languages for websites not using CMS.

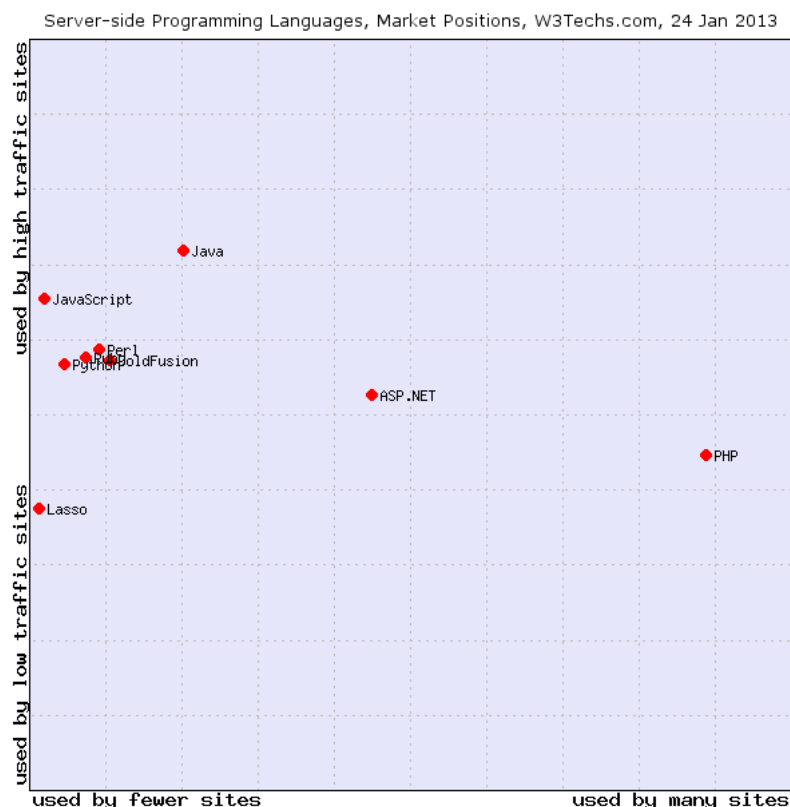


Figure 9.4: Server-side programming languages market position report. Source [7]

Figure 9.5 represents the historical trends. Java keeps stable along last four years. PHP has been in-

creasing, CMS has increased almost the same amount, as it can be seen in figure 9.6. Subtracting CMS growth, PHP usage still increased 1.10%. ASP.NET usage has a great decrease while Perl, Coldfusion, Ruby and Python usage has a slight decrease, less than 0.5%.

	2010 1 Jan	2011 1 Jan	2012 1 Jan	2013 1 Jan	2013 25 Jan	2010-2013 Growth	PHP growth without CMS
PHP	72.5%	75.3%	77.3%	78.7%	78.7%	6,20%	1,10%
ASP.NET	24.4%	23.4%	21.7%	20.2%	20.1%	-4,30%	
Java	4.0%	3.8%	4.0%	4.1%	4.1%	0,10%	
ColdFusion		1.3%	1.2%	1.1%	1.1%	-0,20%	
Perl		1.1%	1.0%	0.8%	0.8%	-0,30%	
Ruby	0.5%	0.5%	0.6%	0.5%	0.5%	0,00%	
Python	0.3%	0.3%	0.2%	0.2%	0.2%	-0,10%	

Figure 9.5: Historical trends in the usage of server-side programming languages for websites. Source [7]

	2011 1 Jan	2012 1 Jan	2013 1 Jan	2013 25 Jan	Growth
WordPress	13.1%	15.8%	17.4%	17.5%	4,4%
Joomla	2.6%	2.8%	2.8%	2.7%	0,1%
Drupal	1.4%	1.9%	2.3%	2.3%	0,9%
vBulletin	1.4%	1.3%	1.1%	1.1%	-0,3%
				TOTAL	5,10%

Figure 9.6: Historical trends in the usage of CMS for websites. Source [7] on 25 Jan 2013

In conclusion, PHP is the leader in server-side language usage with a wide advantage. Main used technologies are PHP, ASP.NET and Java. Perl, Ruby, ColdFusion and Python are represented by a few sites with high traffic. The highest traffic sites use Java and the lowest traffic sites use PHP. The usage of CMS sites is growing and they are built with PHP.

9.3 Statistics usage of Web frameworks

BuiltWith provides web frameworks usage statistics. Its data does not seem totally accurate because several libraries appear as frameworks, for example, Telerik Controls. However, it is the only source of web frameworks statistics usage. Therefore, it will be analyzed.

In figure 9.7, the websites column represents websites using this technology and the last column represents use percentage in the top 10.000 sites. Web frameworks which are not full stack have been removed from the table. According to BuiltWith, they use two detection techniques to recognize Ruby on Rails use. To avoid confusion, results from both techniques have been added in this table.

ASP.NET is the most used web framework with a great advantage, followed by Ruby on Rails. Private web frameworks are ASP.NET, Silverlight and Adobe ColdFusion. Open source web frameworks are Ruby on Rails, CodeIgniter, CakePHP, Django and GWT. Silverlight, CodeIgniter, Cake PHP, Django and GWT present a use percentage of less than 0.5% in the top 10.000.

Name	Websites	Usage 10k %
ASP.NET	10.236.465	19,48
Ruby on Rails	139.374	3,65
Adobe ColdFusion	204.389	1,12
Silverlight	99.799	0,09
CodeIgniter	96.572	0,27
CakePHP	60.141	0,13
Django CSRF	12.114	0,20
Google Web Toolkit	9.056	0,10

Figure 9.7: Web frameworks usage. Adapted from [8] consulted on 1 March 2013

9.3.1 Web frameworks usage growth

BuiltWith offered a five day trial to access web frameworks usage growth. In figure 9.8 is shown the growth since 12, 6, 3 and 1 months ago and in the top 10.000, 100.000 and 1 million sites.

The best growth in 12 months is presented by GWT in all the tops. Spring presents the second best growth in the top million and in the top 100.000 sites but it presents a really low growth in the top 10.000, only 1%. Django, Tapestry and CodeIgniter are the only ones which show better growth in the top 10.000 than in other tops. Silverlight and Adobe Coldfusion are declining.

Web frameworks growth/decline												
Technology Name	Top 10k Growth/Decline				Top 100k Growth/Decline				Top Million Growth/Decline			
	12m	6m	3m	1m	12m	6m	3m	1m	12m	6m	3m	1m
Google Web Toolkit	82%	-9%	9%	0,00%	90%	-36%	-21%	0,00%	98%	-11%	-3%	0%
Spring Source	1%	-1%	1%	0,00%	78%	57%	-35%	-9%	79%	61%	26%	3%
Ruby on Rails	43%	25%	10%	2%	36%	16%	4%	0%	58%	23%	4%	0%
Django CSRF	59%	32%	9%	0,00%	39%	6%	-11%	-1%	49%	12%	0%	0%
ASP.NET MVC	32%	19%	15%	7%	29%	17%	6%	2%	37%	-10%	-7%	1%
Apache Tapestry	67%	0,00%	0,00%	0,00%	17%	0,00%	0,00%	0,00%	30%	30%	13%	0%
CodeIgniter	50%	4%	-4%	0,00%	18%	11%	-1%	1%	29%	10%	3%	0%
Lift	---	---	---	---	33%	0,00%	-33%	0,00%	23%	31%	-15%	0%
CakePHP	-57%	-57%	-29%	14%	-2%	4%	2%	2%	11%	0%	0%	0%
Seagull	---	---	---	---	0,00%	67%	33%	0,00%	6%	-28%	-11%	-6%
Adobe ColdFusion	-12%	0%	-18%	-3%	-11%	-1%	-3%	0%	-4%	-5%	-2%	0%
Silverlight	-83%	-33%	0%	17%	-3,00%	-6%	-10%	-1,00%	-9%	-7%	-5%	-1%

Figure 9.8: Web frameworks growth. Adapted from [8] consulted on 1 March 2013

To summarize:

- ASP.NET is leader in web frameworks usage with a great advantage
- GWT has the greatest growth

- Every technology which is highly used is growing, excluding Silverlight and Adobe ColdFusion which are declining
- Spring, Ruby on Rails, Django, ASP.NET and Tapestry usage are growing

9.4 Web frameworks benchmarks

In this section, an overview of the main web frameworks benchmarks made is carried out. To begin, it is analysed a presentation by Van den Enden which is probably the best web framework benchmark done. Only Java-based frameworks are compared but the methodology and the results are worthy. Next, there is a set of small web frameworks benchmarks posted on blogs, although they could be more precise, the similarity between the results lead to conclusions which could help in taking the decision on which frameworks to choose.

9.4.1 World Wide Wait

Stijn Van den Enden, Ward Vijfeijken, Guy Veraghtert made a presentation at Devovx 2011 where they compared different Java frameworks request per second and scalability. Compared web frameworks were:

- GWT
- Spring
- JSF
- Wicket
- MyFaces

The study is scientific and precise. A web application based on common patterns was built and a set of request tests was sent to the application. It consists of more than 700 hours of test runs and more than 16Gb to analyse. They made different validations to prove results were accurate. The database was simulated to avoid bottlenecks.

In figure 9.9, it is shown the mean page load time versus concurrent users, ThreadCount. In the case of GWT, DOM updates executed after onLoad event are not counted. Therefore, it should be a little higher.

Figure 9.10 is a brief diagram, it shows framework cost of scale for 10000 users and a limit of 5 seconds to answer. GWT is the cheapest to scale despite being the second faster in average mean page load time due to the fact GWT sends JSON to the client and it renders it. Because of this, server has lower load and it is cheaper to scale, while client has higher load rendering. Furthermore, authors of the research said GWT was the most difficult to develop with.

In conclusion, Spring is the fastest framework but it costs a 50% more to scale with it than with GWT. GWT is the cheapest framework to scale but it gives a slower response time. Wicket and JSF are slower and more expensive.

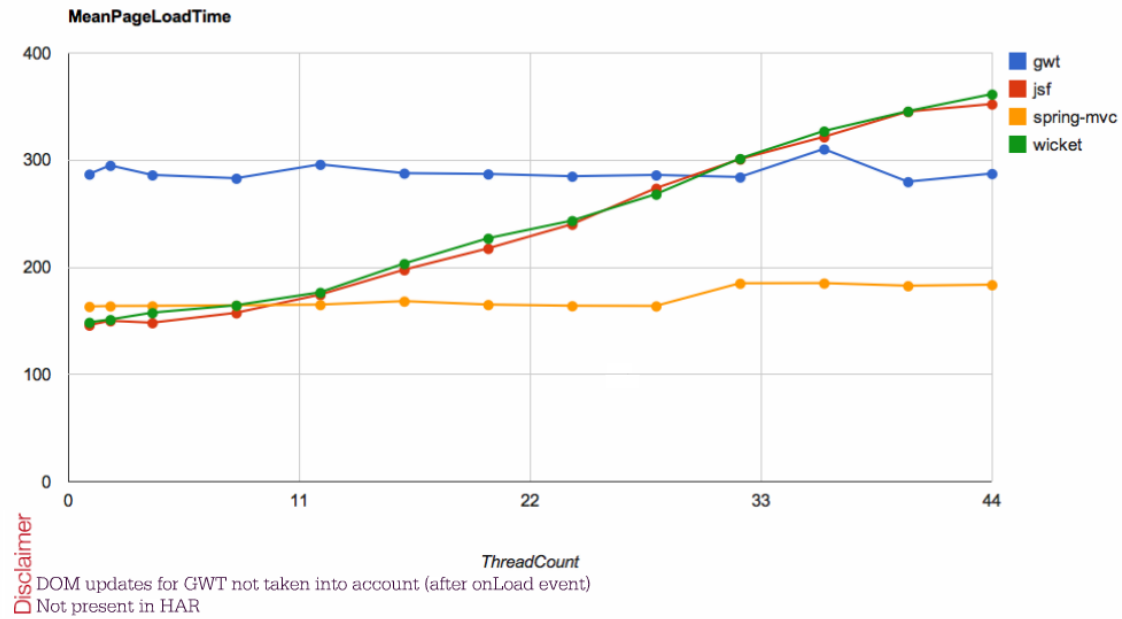


Figure 9.9: Mean page load time with java web frameworks . Source [4]

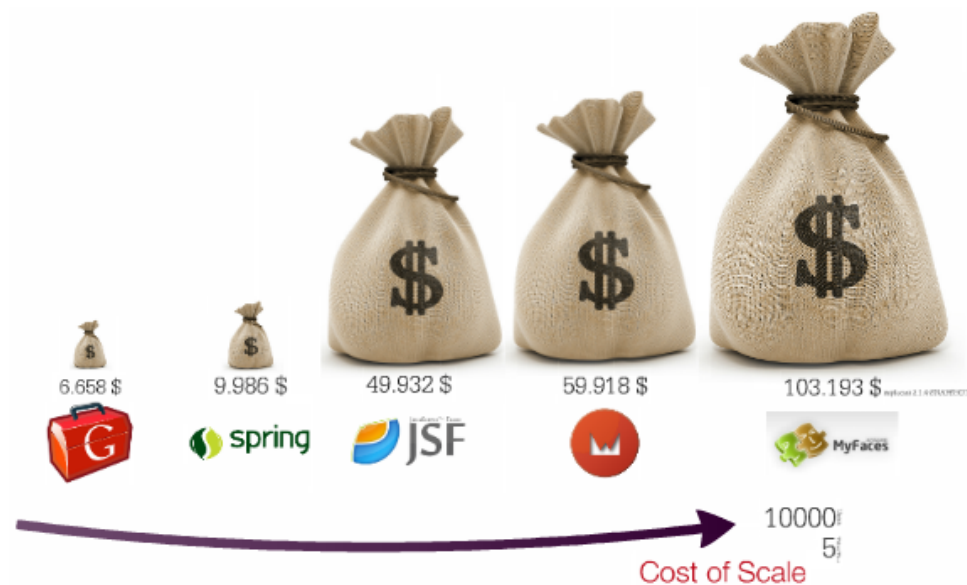


Figure 9.10: Cost of scale with different java web frameworks. Source [4]

9.4.2 The Great Web Framework Shootout

Davis published a more imprecise benchmark [9]. According to its words "it would be foolish to consider these results as scientific in any way". He tried to test frameworks performance just out-of-box. Data collected are not as precise as World Wide Wait but it compares different frameworks based on PHP, Python and Ruby. The frameworks which have been compared are:

- Bottle
- Flask
- Sinatra
- Web.go
- Pyramid
- Django
- Rails
- Kohana
- TG
- Yii
- CakePHP
- Symfony

Figure 9.11 shows requests per second for a template with a DB query. Bottle, Flash, Sinatra and Web.go are microframeworks which do not have so many capabilities as full stack frameworks have. The full stack frameworks which have higher requests per second are Pyramid, CodeIgniter, Django and Rails.

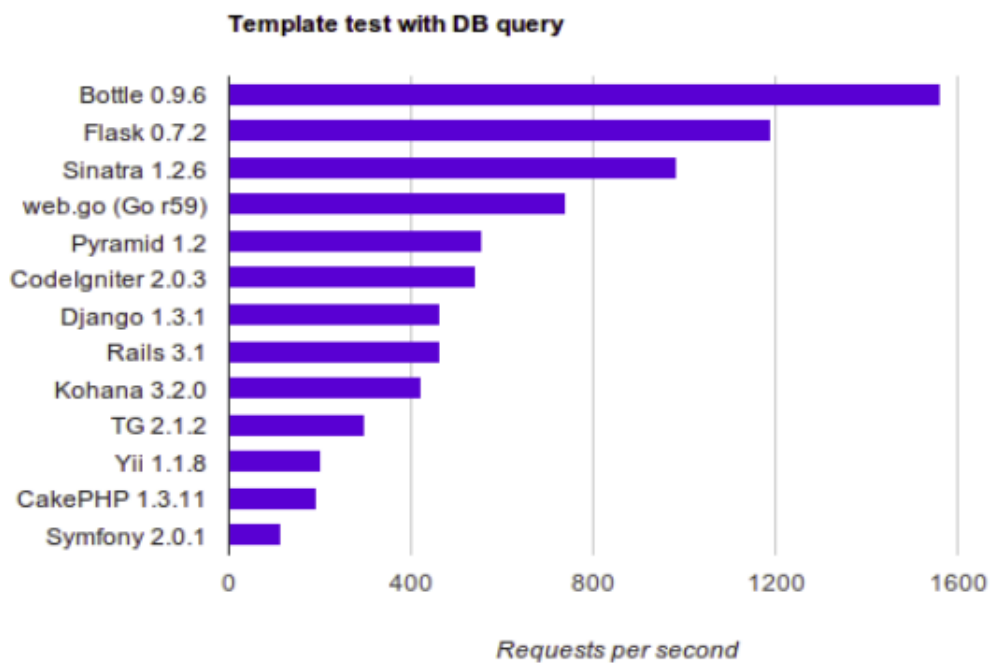


Figure 9.11: Request per second with different frameworks. Source [4]

9.4.3 Benchmarking Web Applications and Frameworks

Jones published a benchmark for PHP frameworks [10]. The PHP frameworks which have been compared are:

- Aura
- CakePHP
- CodeIgniter
- Lithium
- Solar
- Symfony
- Zend

First, he tested the maximum responsiveness of the benchmark environment without any framework. The tests were a base line HTML page with "hello world" and a PHP page with "Echo hello world". Afterwards, he tested each framework rendering the HTML page. Results in figure 9.12 are the requests per second in relative percentage to the PHP page rendering time.

CodeIgniter presents the best performance.

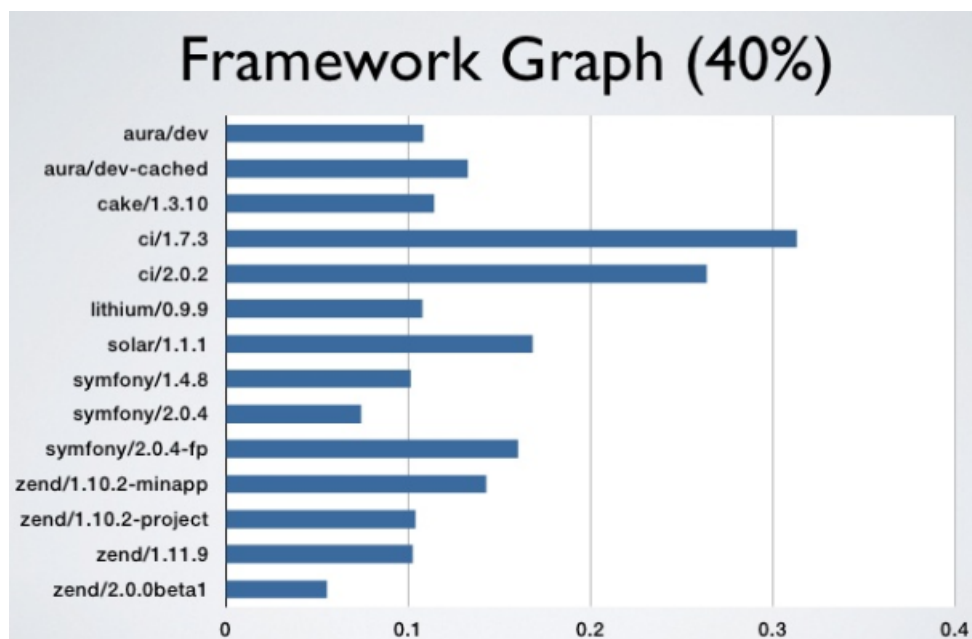


Figure 9.12: Relative request per second with different frameworks. Source [10]

9.4.4 Rails, Wicket, Grails, Play, Tapestry, Lift, JSP and Context

In the website *jtict.com* it was published a benchmark for Java web frameworks and Ruby on Rails [11]. The frameworks which have been compared are:

- Grails
- JRails
- JSP
- Lift
- Play
- Rails
- Wicket
- Context
- Tapestry

The test is a single html page where a list of products objects with associated category are rendered. All data are in memory to avoid the use of a database.

Figure 9.13 shows response time versus number of users. The best performance is presented by *Play rendering with Japid and Netty*, followed by *Play with Japid*, *Play with Scala and netty*, *Tapestry*, *JSP*, *Grails* and more. Rails has a poor performance compared to the rest. JRails presents the worst performance.

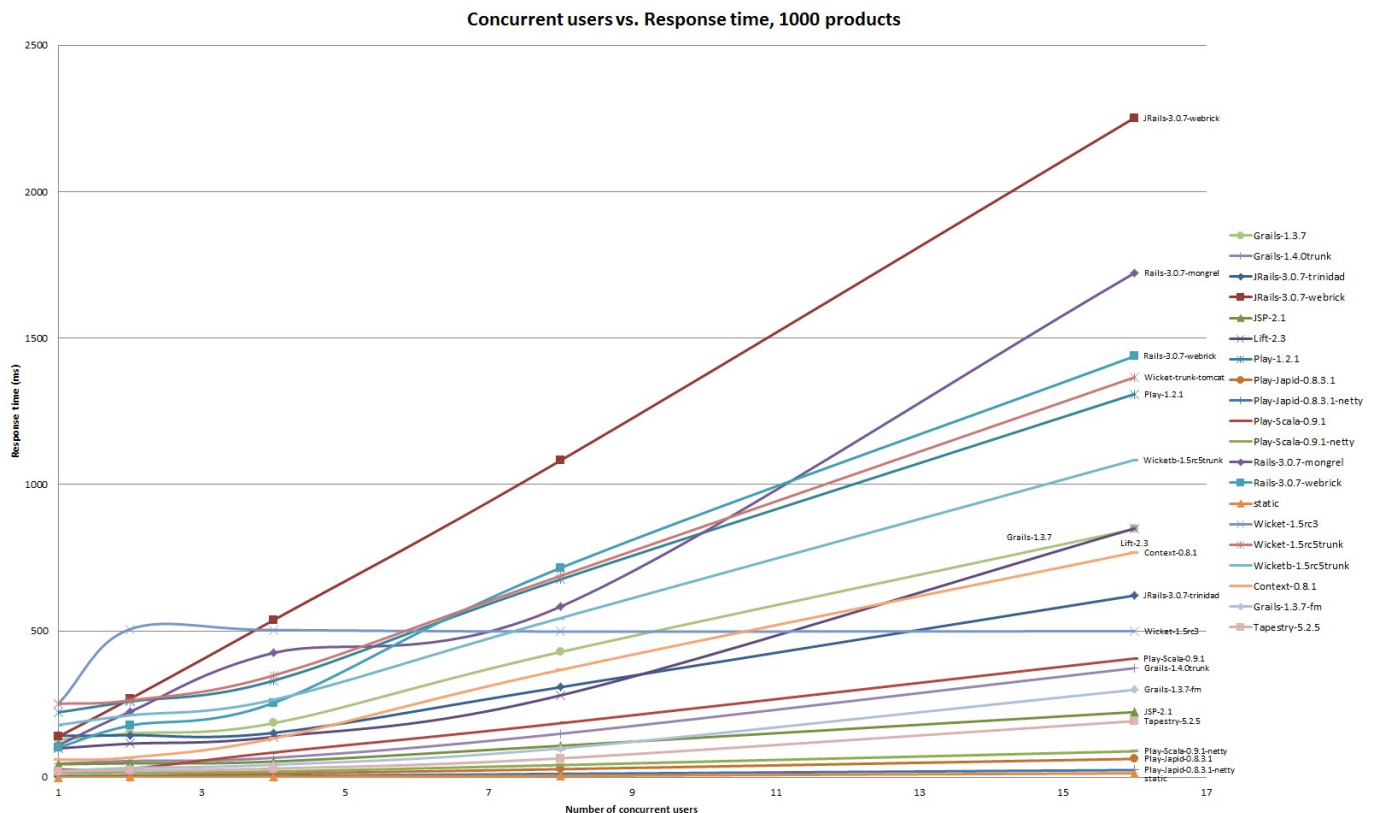


Figure 9.13: Relative request per second with different frameworks. Source [11]

9.4.5 Conclusions

It is important to notice that each benchmark measures different features. In the first one, it has been measured mean page load time, in the second and the third ones request per second, while in the last, one response time.

The best benchmark approach is the World Wide Wait. It shows that Spring is the fastest framework but it costs a 50% more to scale with it than with GWT. GWT is the cheapest framework to scale but it gives a slower response time. Wicket and JSF are slower and more expensive.

The second and the third benchmarks show CodeIgniter as the leader in PHP. Pyramid and Rails are the leaders in Python and Ruby languages. Django is following Pyramid in Python language and close to Ruby on Rails.

Wicket is presented in the first and the last benchmark. It could be extrapolated that all the frameworks which last longer than Wicket on response time in the last benchmark have worse performance than Spring or GWT.

9.5 Decided technologies

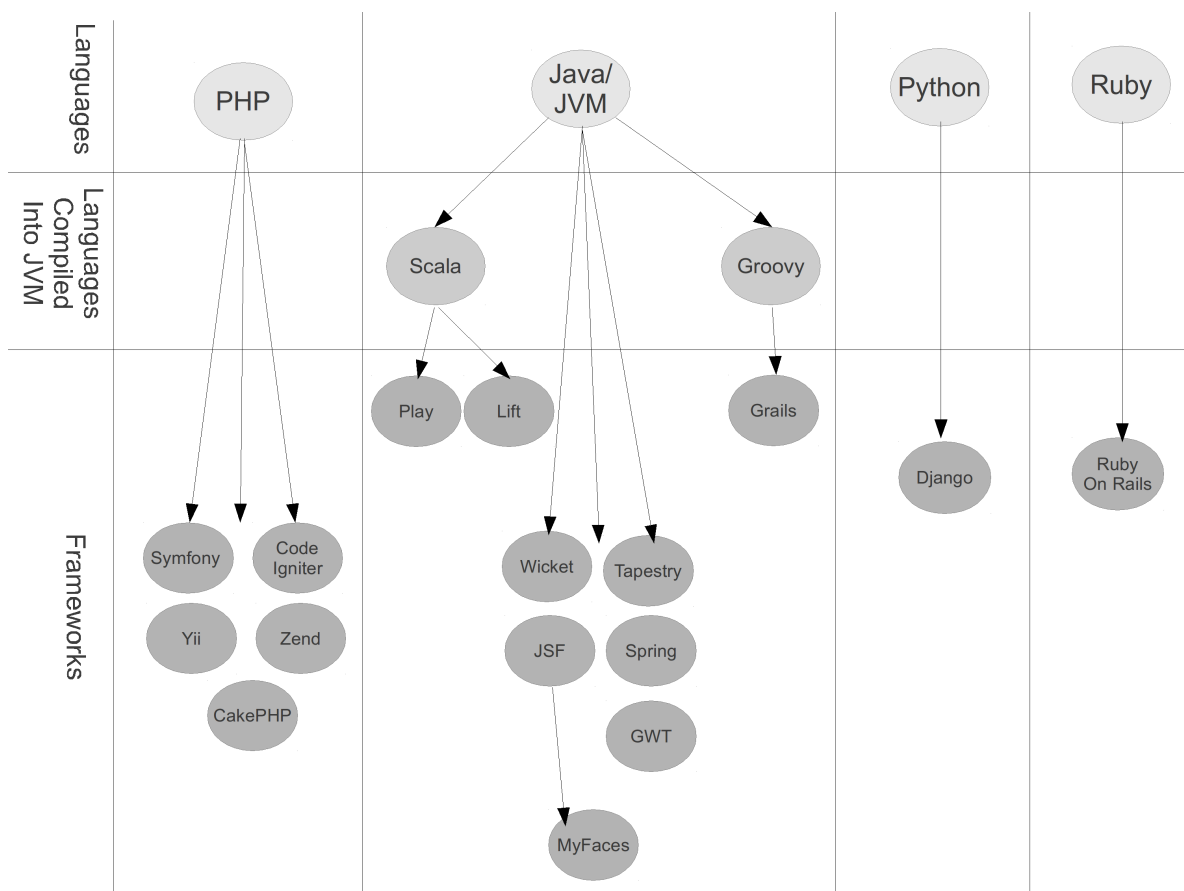


Figure 9.14: Main frameworks map.

Figure 9.14 shows popular frameworks which have better characteristics of usage and ease of development. Perl has been excluded despite of being a highly used technology, its frameworks are not considered in comparisons. Python has been included although not being named in comparison because it presents a notable use in high traffic sites and its usage is increasing.

Although there are more frameworks to choose in Ruby and Python frameworks universe, Ruby on Rails and Django are the most widely used. It is said to be a good characteristic community focused on one framework. On the other hand, there are plenty of Java and PHP based frameworks.

Component-based frameworks are an approach to desktop application development [6]. Their http request abstraction is not accurate for CRUD, classical websites development. A request-based approach is closer to web development and this project will focus on developing a CRUD website. Therefore, component-based frameworks are excluded from options to choose. Figure 9.15 lists request-based frameworks and compares their characteristics. All of them support every attribute in some manner.

Project	Language	Ajax	MVC framework	MVC push-pull	118n & L10n?	ORM	Testing	DB migration	Security	Template	Caching	Form validation	Request / component based
Ruby on Rails		Prototype, script.aculo.us, jQuery	ActiveRecord, Action Pack	Push	Yes	Active Record	Unit Tests, Functional Tests and Integration Tests	Yes	Plug-in	Yes	Yes	Yes	Request
Django	Python2.*	Yes	Full stack	Push	Yes	Django ORM	Yes	Provided by South	ACL-based	Django Template Language	Cache Framework	Django Forms API	Request
CodeIgniter	PHP >= 5.1	Any	Yes	Push	Mostly [15]	Third party only	Ready for next release	Yes	Yes	Yes	Yes	Yes	Request
Grails	Groovy	Yes	Active record pattern	Push	Yes	GORM, Hibernate	Unit tests, integration test, functional test	multiple plugins: autobase, dbmigrate, more	Spring Security, [13] Apache Shiro [14]	Yes	Yes	Yes	Request
Spring	Java	Yes	Yes	Push	Yes	Hibernate, iBatis, more	Mock objects, unit tests		Spring Security (formerly Acegi)	JSP, Commons Tiles, Velocity, Thymeleaf, more	ehcache, more	Commons validator, Bean Validation	Request
Play	Scala	Yes	Yes	Push-pull	Yes	JPA, Hibernate	JUnit, Selenium	Yes	via Core Security module	Yes	Yes	Server-side validation	Request

Figure 9.15: Main frameworks table. Modified from Wikipedia

The objective of this project is to evaluate the ease of development in different web frameworks based on different programming languages.

Matt Raible matrix has been weighed according to priorities in this project. Weights are ranged between 0, less important, to 3, most important.

- Developer productivity(2): It is related to ease of development
- Developer perception(2) : A better perception probably means an easier learning curve
- Learning curve(3) : It is the most important characteristic
- Project Health (1)
- Developer availability (2): A project should start with technologies with available developers
- Job Trends (0) : It is barely important for a project
- Templating (1)
- Components (2) : Reusable components accelerate development
- Ajax (2)
- Plugins or Addons (2)
- Scalability (0) : It really does not matter for agile development
- Testing (2) : Tools for testing applications allow to find errors faster
- i18n and l10n (0) : It is not important for agile development
- Validation (1)
- Multi-language Support (0)
- Quality of Documentation/Tutorials (2) : Better documentation, faster learning
- Books published (1): With Documentation is enough
- REST Support (client and server) (1)
- Mobile / iPhone Support (1)
- Degree of Risk (1)

Results in figure 9.16 show as request-based winners Grails and Rails.

Weight	Criteria	Struts 2	Spring MVC	Wicket	JSF 2	Tapestry	Stripes	GWT	Grails	Rails	Flex	Vaadin	Lift	Play
2	Developer Productivity	1,00	1,00	1,00	1,00	2,00	1,00	2,00	2,00	2,00	0,00	2,00	1,00	2,00
2	Developer Perception	1,00	2,00	2,00	0,00	1,00	2,00	2,00	2,00	2,00	2,00	2,00	2,00	2,00
3	Learning Curve	3,00	3,00	1,50	1,50	1,50	3,00	3,00	3,00	3,00	3,00	3,00	1,50	3,00
1	Project Health	0,50	1,00	1,00	1,00	0,50	0,50	1,00	1,00	1,00	0,50	1,00	1,00	1,00
2	Developer Availability	1,00	2,00	1,00	2,00	2,00	1,00	2,00	1,00	2,00	2,00	1,00	0,00	1,00
0	Job Trends	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
1	Templating	1,00	1,00	1,00	0,50	1,00	1,00	0,50	1,00	1,00	0,50	0,50	0,50	0,50
2	Components	0,00	0,00	2,00	2,00	2,00	0,00	1,00	1,00	1,00	2,00	2,00	0,00	0,00
2	Ajax	1,00	2,00	1,00	1,00	1,00	1,00	2,00	1,00	1,00	1,00	2,00	2,00	1,00
2	Plugins or Add-Ons	1,00	0,00	2,00	2,00	1,00	0,00	2,00	2,00	2,00	2,00	2,00	1,00	2,00
0	Scalability	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
2	Testing	2,00	2,00	1,00	1,00	2,00	2,00	1,00	2,00	2,00	0,00	1,00	1,00	2,00
0	i18n and l10n	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
1	Validation	1,00	1,00	1,00	0,50	1,00	1,00	1,00	1,00	1,00	1,00	1,00	0,50	0,50
0	Multi-language Support (Groovy / Scala)	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
2	Quality of Documentation/Tutorials	1,00	2,00	1,00	1,00	1,00	2,00	2,00	2,00	2,00	2,00	2,00	2,00	2,00
1	Books Published	1,00	1,00	0,50	1,00	0,50	0,50	1,00	1,00	1,00	1,00	0,50	0,50	0,00
1	REST Support (client and server)	0,50	1,00	0,50	0,00	0,50	0,50	0,50	1,00	1,00	0,50	0,50	0,50	0,50
1	Mobile / iPhone Support	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	0,50	1,00	1,00	1,00
1	Degree of Risk	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	0,50	0,50	0,50
	Totals	17,0	21,0	18,5	16,5	19,0	17,5	23,0	23,0	24,0	19,0	22,0	15,0	19,0

Figure 9.16: Matt Raible's Matrix weighed according to priorities

Consequently, Ruby on Rails and Django are chosen due to the fact of being the only famous frameworks in these technologies. In PHP, CodeIgniter presents the best performance and it is said to be easier than Symfony, CakePHP and Zend. Yii could be an alternative but, due to its less adoption, CodeIgniter seems the best option. There are so many JVM frameworks that more than one should be proved. Nevertheless, because of time limitation, only one can be chosen. Candidates are Play and Grails. Grails seems to be a good option due to the fact that it is an alternative to Ruby on Rails in Java environment [1]. But Play! has gained a lot of popularity in the last years, it has a mixed MVC Push-pull which could be worthy to test and presents an incredible performance. Nevertheless, because Grails is more adopted than Play! and close in performance to it, this is the reason why it is finally chosen.

To summarize, web frameworks to be compared are:

- Ruby on Rails
- Grails
- Django
- CodeIgniter

Chapter 10

Website and language exercises design

In this chapter, it is described the subject background, the chronological development of all the tests, the test steps to do with each web framework, their times and the documentation that will be used along the process.

Due to the time limitations of the project, a deep research could not have been done in this section. Therefore, the experts, Juan Carlos Rodríguez del Pino, a language programming professor at ULPGC, the author of Virtual Programming Lab, and Carlos Hernández, founder of Endeve, expertise in agile development, have supervised the language programming exercises definition and the website definition.

10.1 Student background

The student's relevant background for this project is:

- The student has a five year degree in computer science, final year project remains but every course has been passed
- Several courses on C and C++
- He has programmed in Java for some university courses but he did not receive any Java programming course
- He programmed a course exercise in PHP
- He read Yii documentation for several hours
- He used Django for several hours in a Startup Weekend

10.2 Chronological development

The chronological development of the frameworks will be:

- Ruby on Rails
- Grails
- Django

- CodeIgniter

It is a hypothesis that it will be more difficult to develop with the first frameworks, whatever they are. Therefore, those which are supposed to be the easiest ones to develop with are the first ones.

10.3 Steps

These are the six practical test steps which will be accomplished with each web framework:

- **Reading the programming language documentation:** The student can read the official documentation of the programming language, use search engines for answering questions and do tutorials, but he is not allowed to do the programming exercises
- **Making the exercises to demonstrate language knowledge:** The student has to focus on solving the exercises with the language. He can read the documentation and search in Google, but the priority is solving the exercises. The exercises will be described in a few days
- **Reporting the language features and development experience:** A report has to be done along this project. In this step, the student can write part of the report. Furthermore, the report improves his learning
- **Reading the web framework documentation:** It is close to the *Reading the Programming Language Documentation* step. The student has to focus on reading the documentation, he cannot work in the website
- **Developing the website:** The student has to focus on developing the website. He can read documentation and search in Google to help him in his tasks but the main priority is developing the website
- **Writing notes about the development and experience:** In this task, notes about web framework functionality and problem solution are written in the report

10.4 Timing steps

In this section, the fixed times for each step are defined. Each time is multiple of five. If the student makes five hours per working day, most tasks are separated for at least one day from the previous one. Furthermore, main tasks are separated by weeks. Language related tasks are done in one week. Learning web framework documentation is done in another week. Developing the website is done in two weeks and notes are written in one last week. It means five weeks for each framework test with five hours per working day.

Next, tasks divided by hours:

- **Reading the programming language documentation:** 8 hours
- **Making the exercises to demonstrate language knowledge:** 12
- **Reporting the language features and development experience:** 5
- **Reading the web framework documentation:** 30
- **Developing the website:** 45

- **Reporting the web framework features and development experience:** 25

Reading language documentation is so short due to the fact that the subject already knows how to program and simply has to learn the new syntax and structures of the language. The main learning is done through the exercises.

10.5 Documentation

The official documentation of the web frameworks and the languages will be used along the process. In the development steps, search engines can be used to solve problems.

10.6 Programming exercises

A subset of the exercises used at the Computer Science School (Escuela de Ingeniería Informática) in ULPGC has been chosen. Figure 10.1 shows which features of the programming language are going to be practiced.

The first exercise demonstrates knowledge of files operations, strings and regular expressions. The second exercise demonstrates basic knowledge about classes, types and structures in the language. The third exercise is the composite pattern, it demonstrates inheritance and polymorphism knowledge. The third and last exercise is the composite pattern because, despite being important in programming language knowledge, it is not considered to be important for developing the website.

Basic components such as variables, constants, data types, control structures, data structure and functions are the most used because they seem to be more important for programming with a web framework.

	Variables, constants, expressions, operators	Data types	Control structure	Static methods	Exception	Files	Classes and methods	Inheritance and polymorphism	Regular expressions
File, strings and regular expressions	X	X	X	X		X			X
Numbers set	X	X	X		X		X		
Composite	X	X	X				X	X	

Figure 10.1: Exercises practicing component

10.6.1 The Strings, Files and Regular Expressions exercise

Given a text in a file and a dictionary in another file, the output has to be a file with every word included in the text that is not inside the dictionary surrounded by square brackets and emails have to be surrounded by braces. Words which are parts of emails are not tested if they are in the dictionary.

Example:

Text:

```
Welcome to a great comparison framework. Contact me
on contact@websitesframeworks.com
```

Dictionary:

```
to
a
great
comparison.
```

Output:

```
[Welcome] to a great comparison [framework]. Contact me
on {contact@websitesframeworks.com}
```

10.6.2 The Numbers Set exercise

A class NumbersSet which contains a set of numbers and has the following operations:

- **Empty constructor:** A default constructor which creates empty sets
- **Vector constructor :** An optional constructor which allows to initialize the set with numbers without repeating them through a vector. If repeated, it throws an exception
- **size:** Number of numbers in the set
- **isEmpty:** It gives an answer if the set is empty or has at least one number
- **add:** It adds a number if it does not exist, if it exists, it throws an exception
- **belong:** It tests if a number is inside the set
- **equal:** Given a set, it tests if it is equal. Two sets are equal if they have the same numbers
- **subset:** Given a set, it tests if the set is a subset of the instance
- **vector:** Given a vector, it returns all the elements of the set in the vector
- **union:** Given a set, it creates the union between the set and the instance

Example of use:

```
array v1={1, 3, 4, 7, -10};
array v2={3, 5, -2, 1, 4, 8};
array v3;
NumbersSet A(v1);
NumbersSet B(v2);
NumbersSet D;
D=A+B;
D.elements(v3);
for(i=0; i<D.size(); i++){
```



```

        print v3[i];
        print " ";
    }
    print "\n"

```

10.6.3 The Composite Pattern exercise

Some documents have to be represented. A document is composed by components. A component can be some text, a number, a date or a composite component. A composite component is a collection of components. Every component offers the operation:

- **toString** : String representing it

The composite component has the following operations too:

- **add**: it adds a component to the collection
- **size**: it returns the numbers of elements in the collection
- **modify**: Given a component and an index, it substitutes the element indicated with the index for the component in the input

Define the classes Component, TextComponent, NumberComponent, DateComponent and CompositeComponent.

Use example:

```

p= new CompositeComponent;
p.add(TextComponent("String. Number:"));
p.add(NumberComponent(1));
p.add(TextComponent("\n"));
p.add(TextComponent("Date: "));
p.add(DateComponent(today));
p.add(TextComponent("\n"));
p.add(TextComponent("Yesterday:"));
p.add(DateComponent(yesterday));
d= new CompositeComponent;
d.add(TextComponent("Testing composite"));
d.add(TextComponent("\n"));
d.add(p);
delete p;
d.add(TextComponent("End testing composite"));
print d.toString();
print "\n";
d.modify(0,TextComponent("Beginning modified"));
print d.toString();
print "\n";

```

10.7 Web Application to develop

A multiblog-only-one-author application has been decided. It requests the developer to know implementing CRUD operations with related data, validations, user session management and searching in a database.

10.7.1 Requisites

- A visitor can read blog posts
- A visitor can read comments in blogs
- A visitor can search for posts
- A visitor can comment posts
- A visitor can register
- A user can login
- A user must have a blog and no more blogs
- A user can create posts in his/her blog
- A user can comment in his/her posts and other posts
- A user can search for posts
- A user can read blog posts and comments

10.7.2 Actors

There are two actors in the website: visitor and registered user.

- Visitor: It is a visitor of the page who has not registered or has not logged in
- User: It is a registered person, a user. He has a blog

10.7.3 Use cases

Next, the use cases:

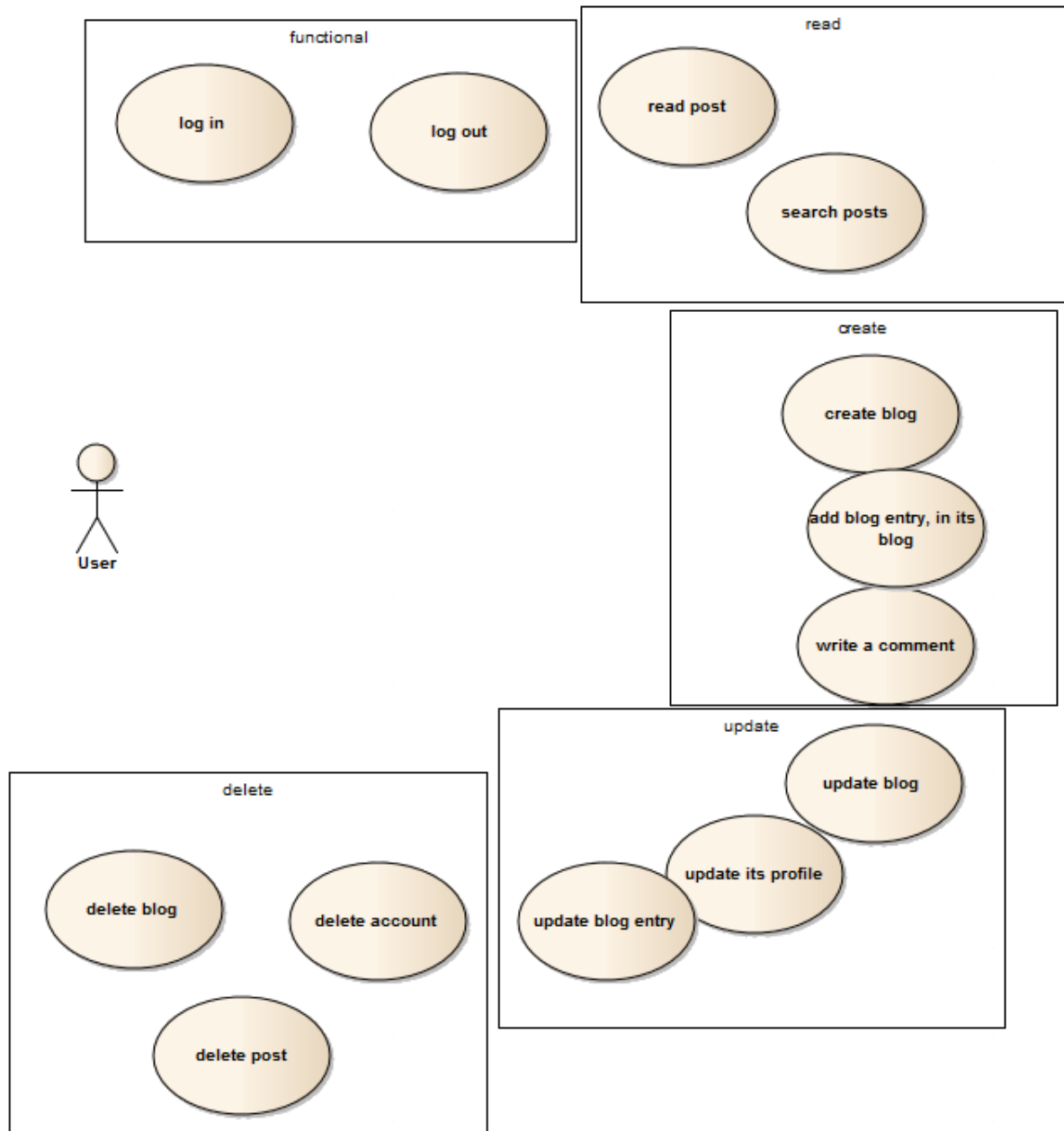


Figure 10.2: Use cases - User

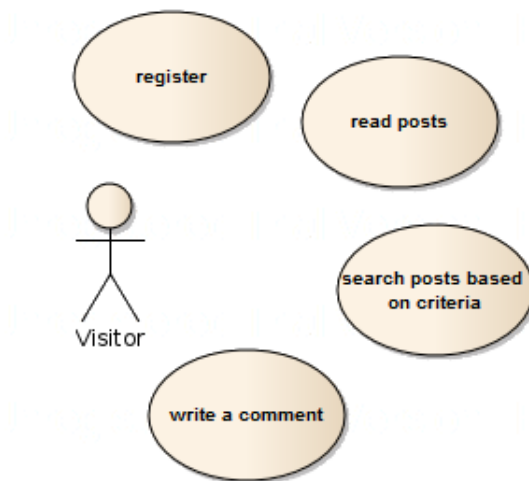


Figure 10.3: Use cases - Visitor

10.7.4 Data Model

Data stored for each entity:

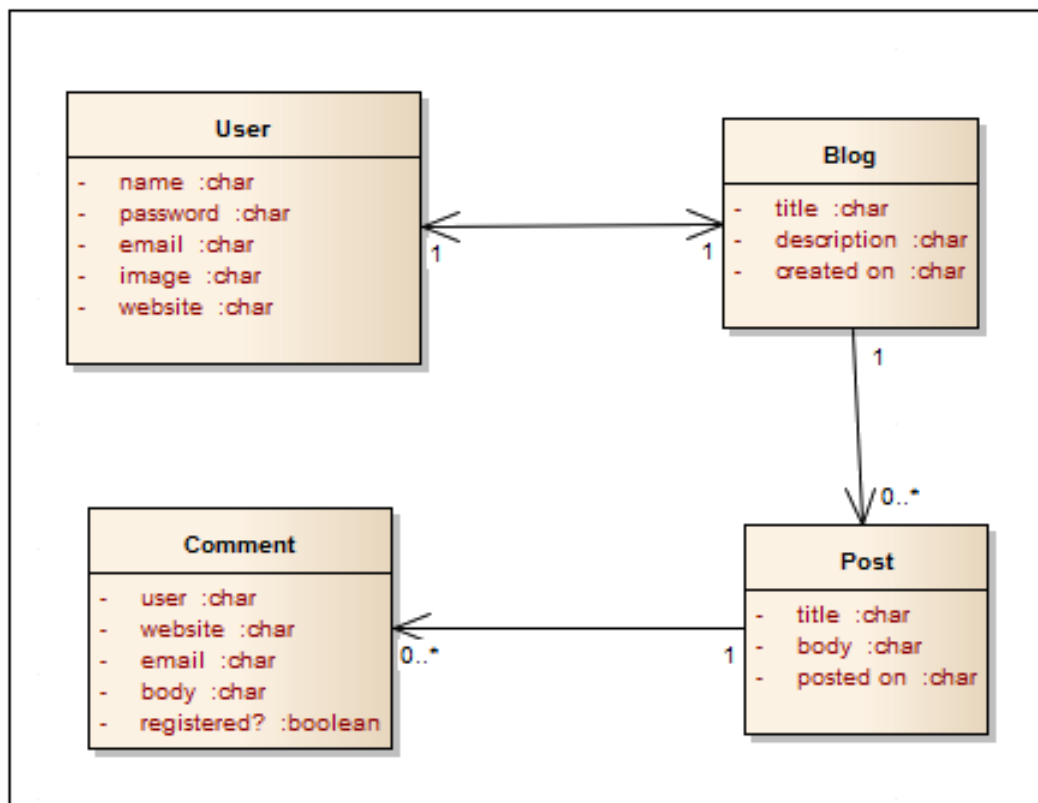


Figure 10.4: Data Model

10.7.5 Mockup

The final website will not be well designed, aesthetic and complex interaction using AJAX are not considered in this comparison. The views will be just HTML divs with titles to show interaction and options. Nevertheless, a mockup can be useful to understand the website.

Visitor's view of the front page

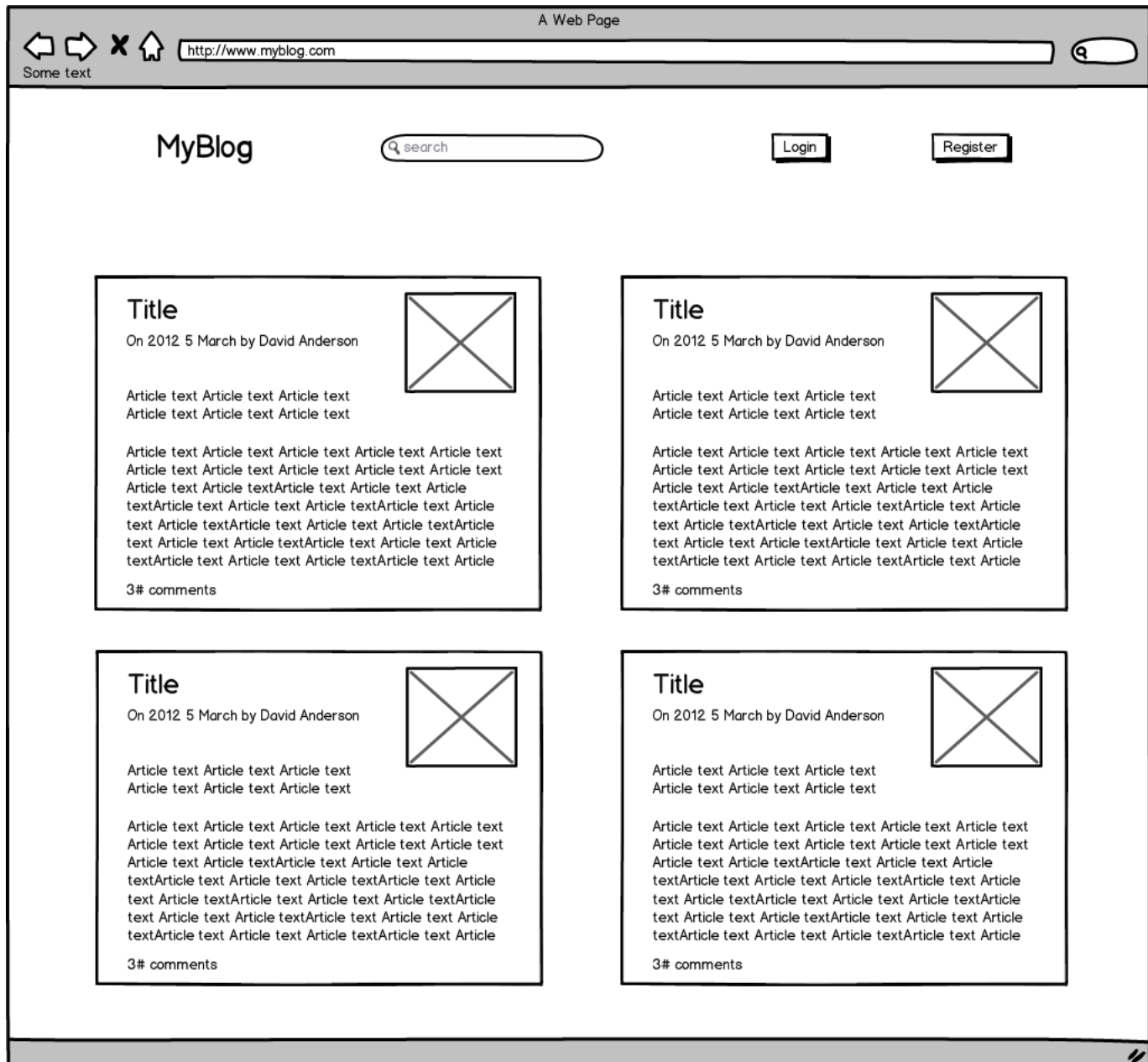


Figure 10.5: Visitor's view of the front page

Visitor's view of a blog

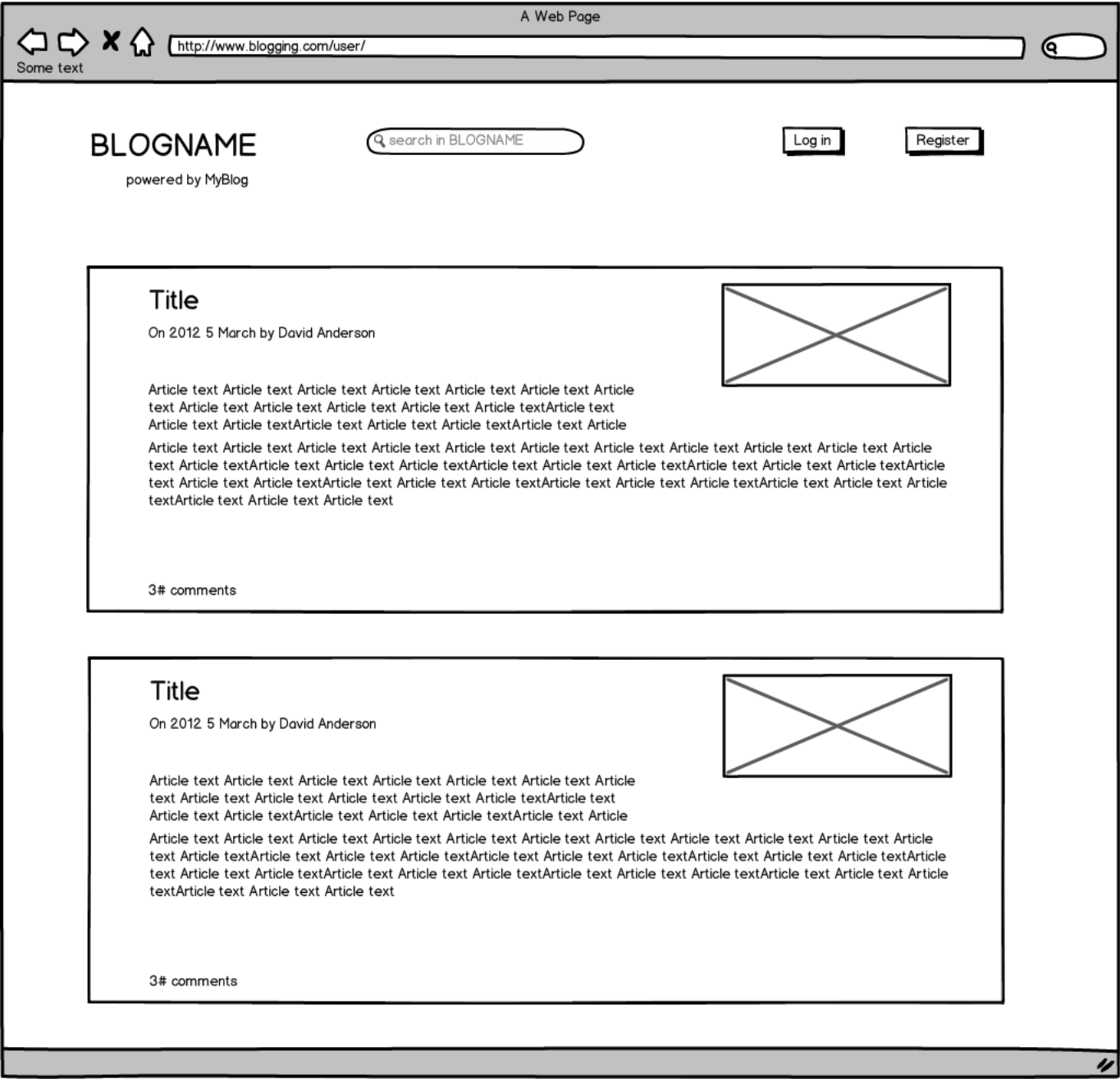


Figure 10.6: Visitor's view of a blog

[illegible]

49

Visitor's view of a register screen

A Web Page

Some text

http://www.myblog.com/sign up

MyBlog - Sign up

user

email

password

register

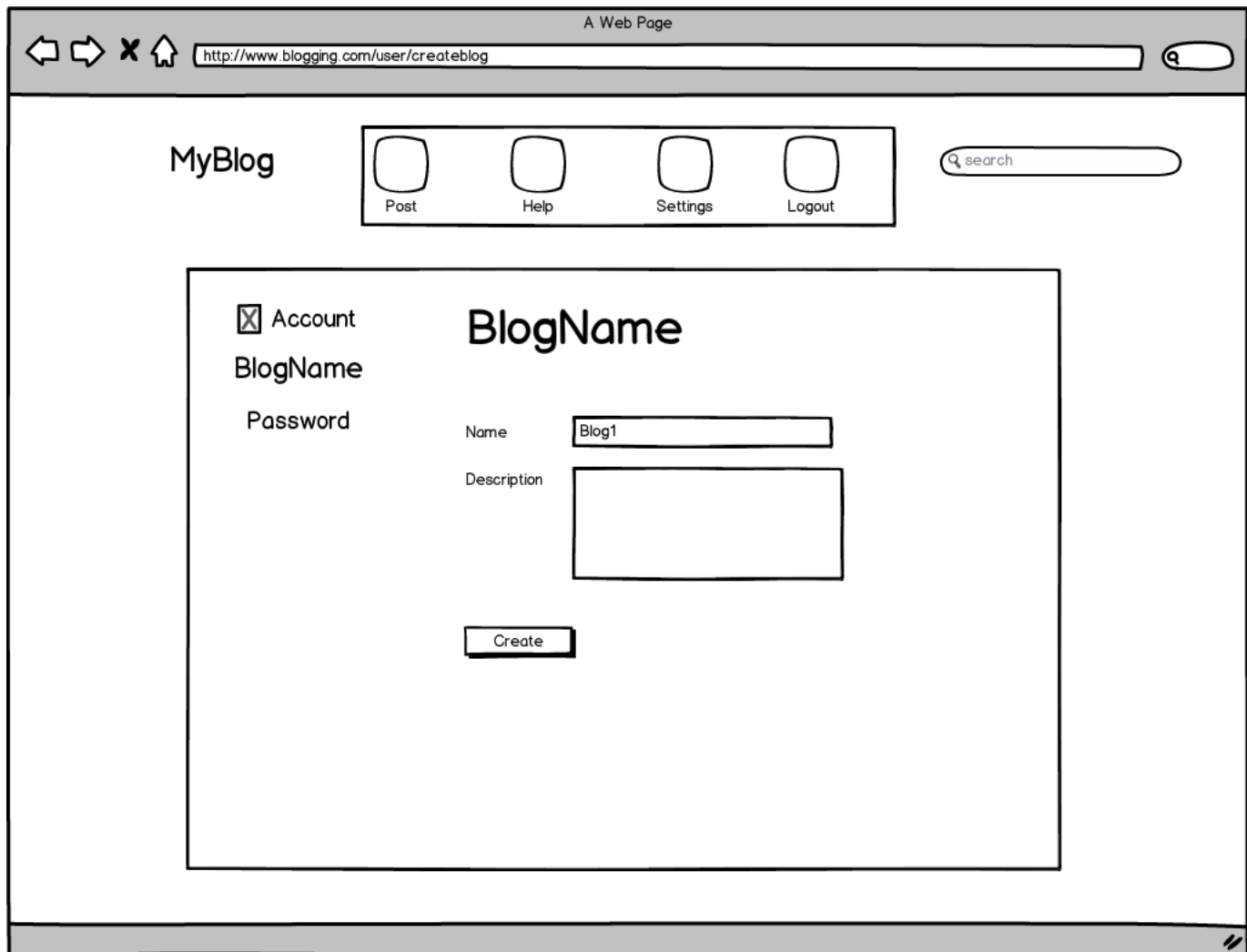
Figure 10.8: Visitor's view of a register screen

Visitor's view of login screen

A hand-drawn diagram of a web browser window. The browser's title bar says "A Web Page". The address bar contains the URL "http://www.myblog.com/login". Below the address bar, the page content is centered and consists of the text "MyBlog - Log in", followed by an input field labeled "email", another input field labeled "password", and a button labeled "Login".

Figure 10.9: Visitor's view of login screen

User's view of creating a blog



A Web Page

http://www.blogging.com/user/createblog

MyBlog

Post Help Settings Logout

search

☒ Account

BlogName

Password

Name Blog1

Description

Create

The image shows a web browser window titled 'A Web Page' with the address bar displaying 'http://www.blogging.com/user/createblog'. The page content includes a 'MyBlog' header, a navigation bar with 'Post', 'Help', 'Settings', and 'Logout' links, and a search bar. The main form area contains a 'BlogName' section with a 'Create' button. The form fields are: 'Account' (checked), 'BlogName', 'Password', 'Name' (filled with 'Blog1'), 'Description', and 'Create'.

Figure 10.10: User's view of creating a blog

User's view of the front page

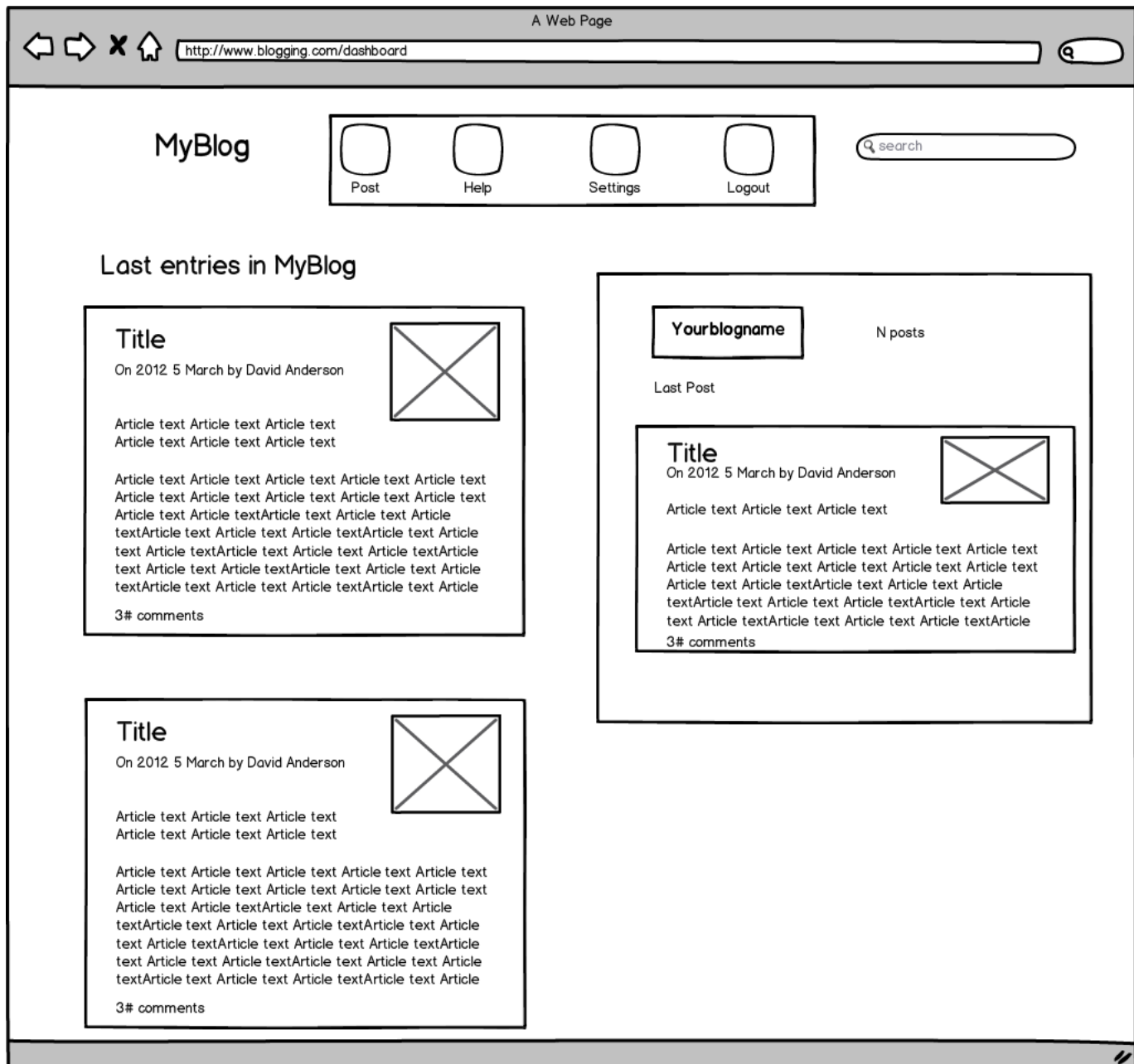


Figure 10.11: User's view of the front page

User's view of a blog

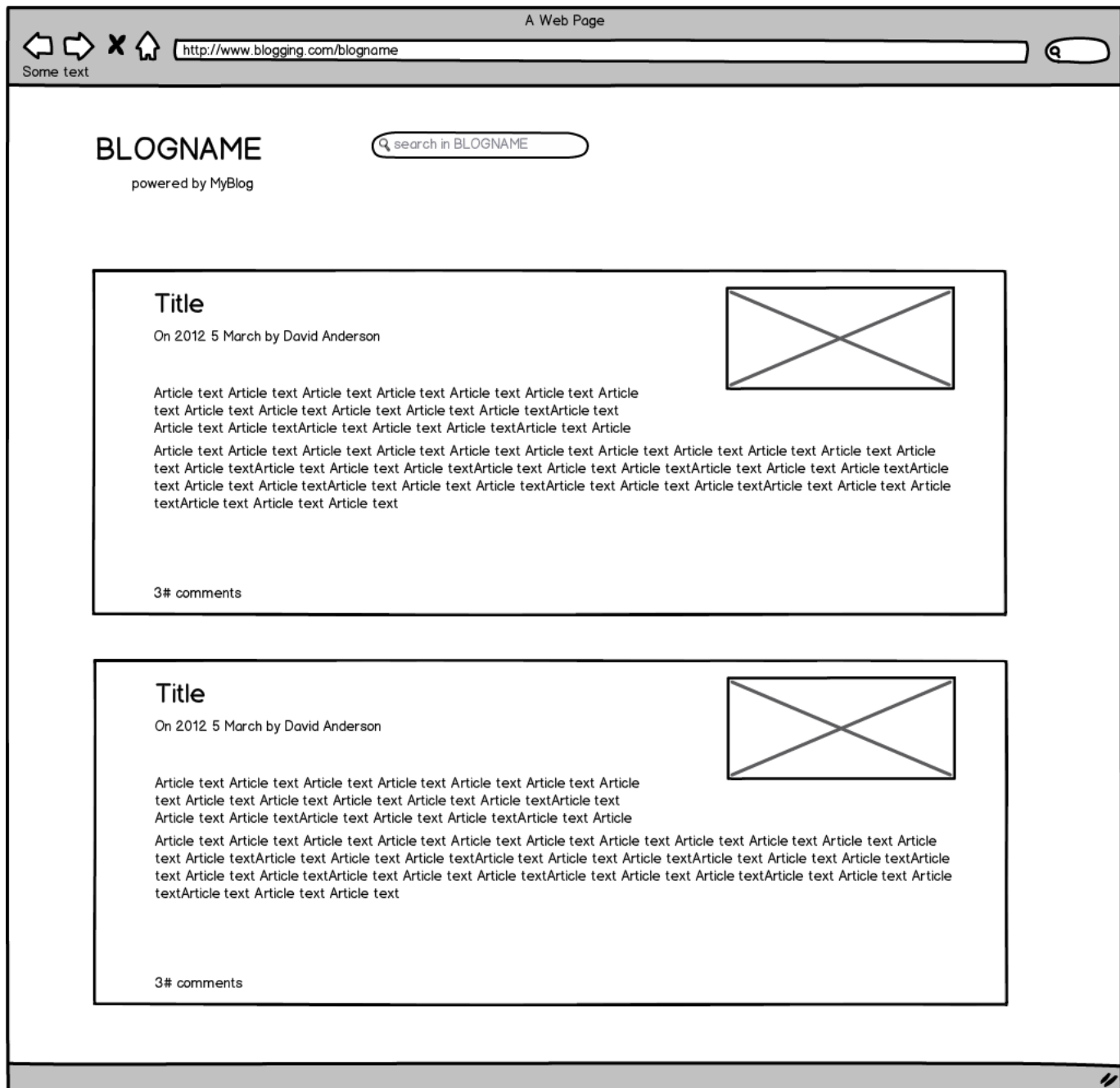
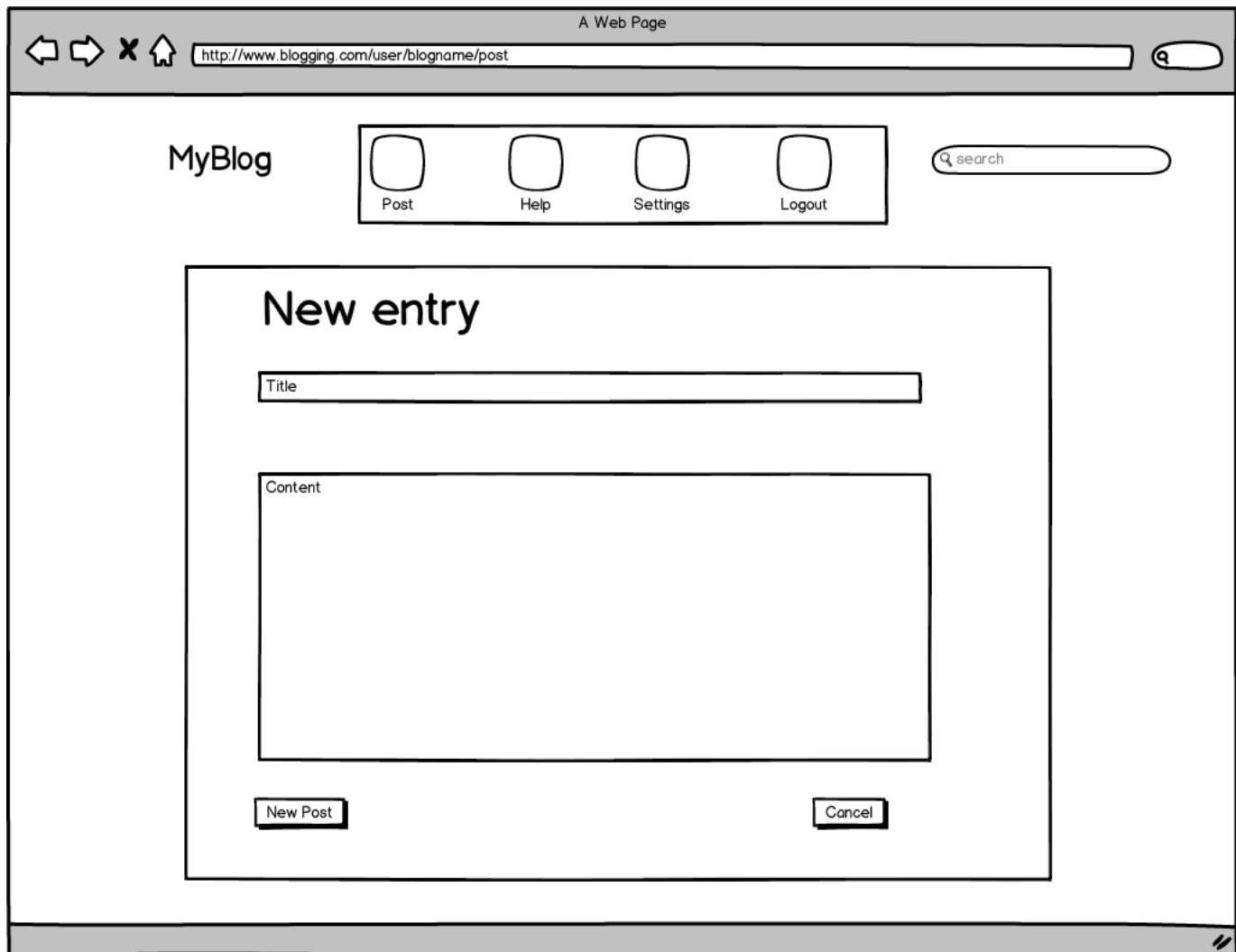


Figure 10.12: User's view of a blog

A hand-drawn sketch of a web browser window titled "A Web Page". The browser's address bar contains the URL "http://www.myblog/user/ID". Below the browser window, the page layout is sketched. At the top left is the blog name "BLOGNAME" with the text "powered by MyBlog" underneath. To the right of the blog name is a horizontal navigation bar containing four rounded square buttons labeled "Post", "Help", "Settings", and "Logout". Further right is a search bar with the placeholder text "search in BLOGNAME". The main content area features a large article titled "Title" by "David Anderson", dated "On 2012 5 March". The article text is represented by several lines of placeholder text. To the right of the article text is a large square placeholder with an 'X' inside, likely for an image. Below the article is a comment section titled "Andrew commented:" containing two paragraphs of placeholder text and a "Greetings." line. At the bottom of the page is a "Comment..." input field and a "Reply" button.

55

User's view for creating a new post



A Web Page

http://www.blogging.com/user/blogname/post

MyBlog

Post Help Settings Logout

search

New entry

Title

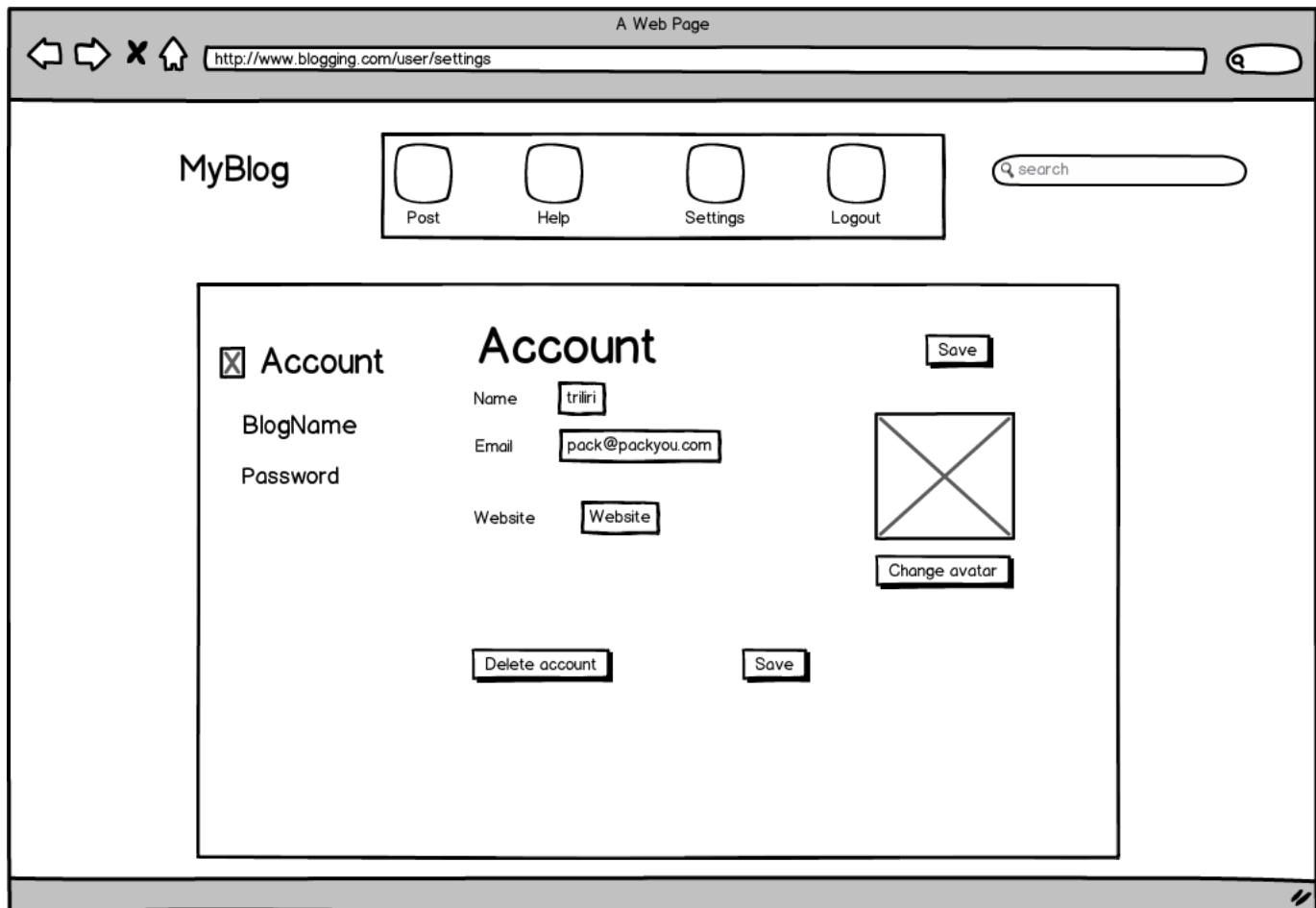
Content

New Post Cancel

The image is a wireframe of a web browser window. The browser's address bar shows 'http://www.blogging.com/user/blogname/post'. The page has a header with the site name 'MyBlog' and a navigation menu with four items: 'Post', 'Help', 'Settings', and 'Logout', each represented by a square icon. To the right of the navigation menu is a search bar with the placeholder text 'search'. The main content area is titled 'New entry' and contains a form for creating a new post. The form has two input fields: a single-line text field labeled 'Title' and a larger multi-line text area labeled 'Content'. At the bottom of the form are two buttons: 'New Post' and 'Cancel'.

Figure 10.14: User's view for creating a new post

User's view for the setting screen



A Web Page

http://www.blogging.com/user/settings

MyBlog

Post Help Settings Logout

search

☒ Account

BlogName

Password

Account

Name triliri

Email pack@packyou.com

Website Website

Save

Change avatar

Delete account

Save

Figure 10.15: User's view for the setting screen

10.8 The order of development

These are the tasks in the order in which they will be carried out in the development process:

1. User authentication system

- Create a user
- Show a user
- Update a user
- Delete a user
- Secure authentication: Login, logout, remember session. Allow to update a user only to signed users and to their accounts
- Image attached to the user: Add the image for the users. Upload system and display the image in the profile
- Change password

2. Blog system

- Create a blog
- Show a blog
- Update a blog
- Delete a blog
- Update delete user: Remove blog if user is removed
- Authentication functionalities: Only a signed user can create a blog. Only the owner of a blog can update it and delete it

3. Post system

- Create a post
- Show a post
- Update a post
- Delete a post
- Show the list of posts in the blog view
- Update delete user: Remove posts if user is removed
- Authentication functionalities: Only allow owner of the blog to create, update and delete posts
- Search posts with any words in the title or in the body

4. Comment system

- Create a comment
- Show a comment
- Delete a comment
- Authentication functionalities: Only allow owner of the post to delete comments
- Update comments upon deletion of user: Leave comments if the user is removed with its last data

5. Fixing everything together

- Friendly urls: `myblog.com/timy` must show Timy's blog and the url of the posts must be friendly too
- View as described in design: The interactive design must be as described. For example, the front page must be an index of all the posts ordered by date
- Pagination: When there are too many posts to be listed in a page, it must be divided by pages with links to choose between them

Chapter 11

Test execution

This chapter will explain the technologies to be compared and the experience developing with them. It is divided into four sections, one for each technology. Each section has four topics:

- The programming language description: A small report about the programming language
- The experience developing the programming exercises: developing times and main problems found along the process
- The web framework: A small report about the web framework
- The experience developing with the web framework: A report about developing times, main problems found, quality of documentation, etc

11.1 Ruby and Ruby on Rails

11.1.1 Ruby

Ruby is a dynamic, general-purpose programming language influenced by Perl, Smalltalk, Eiffel, Lisp and Python. It was created by Yukihiro Matsumoto in Japan in the mid-1990s. It supports multiple programming paradigms: functional, imperative, object-oriented and reflective.

Features

It has the following features:

- Thoroughly object-oriented with inheritance, mixins and metaclasses
- Dynamic type system
- It supports Duck typing: It is a style of dynamic typing in which an object's methods and properties determine the valid semantics, rather than its inheritance from a particular class
- Everything is an expression (even statements) and everything is executed imperatively (even declarations)
- Succinct and flexible syntax that minimizes syntactic noise and serves as a foundation for domain-specific languages
- Lexical closures, iterators and generators, with a unique block syntax

- Literal notation for arrays, hashes, regular expressions and symbols
- Four levels of variable scope (global, class, instance, and local) denoted by sigils or the lack thereof
- Default arguments
- Garbage collection
- Operator overloading
- Exception handling
- Built-in support for rational numbers, complex numbers and arbitrary-precision arithmetic
- Interactive Ruby Shell (a REPL)
- Centralized package management through RubyGems

Syntax

The syntax of Ruby is broadly similar to Perl and Python.

- Indentation is not mandatory
- A symbol attached to the variable names, a sigil, is used to show variable scope. *@variablename* references to a variable within the object. *@@variablename* references to class variables, shared among all objects of a class. *\$Global* variables are prefixed with a dollar sign
- Local variables, method parameters and method names should all start with a lowercase letter or with an underscore. Class names, modules names and constants should start with an uppercase letter
- Semicolons are not mandatory. Instead, line breaks are significant as the end of the statement

A 'hello world' program in Ruby will be:

Listing 11.1: hello world example

```
puts 'hello world'    >>(output) "hello world"
```

A definition of a class syntax is:

Listing 11.2: Class example

```
class Name
  def method1
  end

  def method2( argument1 )
  end

  def method3(argument2 = defaultargument2value )
  end
end
```

A class definition is composed by the key word *class* followed by the class name and *end*. Class names should start with an uppercase letter. The method definition begins with the key word *def* followed by the method instructions and it is finished by the *end* key word. A method can have arguments which must be between parenthesis, separated by commas and after the method name. An argument can have a default value which is expressed with an equal sign, =, after the argument and the default value.

Semantic

Ruby is fully object-oriented: every value is an object. Variables always hold references to objects. Every function is a method and methods are always called on an object. Methods defined at the top level scope become member of the Object class, which is an ancestor of every other class.

Ruby supports inheritance with dynamic dispatch and mixins. Though Ruby does not support multiple inheritance, classes can import modules as mixins.

Standard types

Ruby standard types are:

- Ranges : A range of numbers or letters, such as 1..10 or a..z , is an instance of the *Range* class in Ruby
- Regular expressions: Regular expressions are a class too. They can be defined using a literal which is a character sequence between slashes. */%d/* It is a digit pattern

Containers

You have two kind of containers. Hashes and arrays. Arrays can be created using literals. A literal array is a list of any object between square brackets. For example:

```
aArray=[0, 'hello', anyObject, 2.243]
```

Hashes are similar to arrays but they are indexed with an object of any type. They are a pair of keys and values, where a key and the value can be an object of any type. Hashes can be created using literals. A literal hash is a list of pair of *key=>value* between braces, where key and value can be any object of any type.

```
aHash={key1=>value1 , key2 => value2 , key3 => value3 , ...}
```

Blocks and iterators

A Ruby iterator is simply a method that can invoke a block of code. To invoke the block of code inside a method, the *yield* keyword is used. Afterwards, the method can be called with a block of code as input. For example, the iterator *threetimes* is defined in the example and it is called afterwards with block of code "puts hello". It results in the block of code being executed three times.

Listing 11.3: An iterator example

```
def threeTimes
  yield
  yield
  yield
end
threeTimes { puts "Hello" }
```

Furthermore, Ruby remembers the context in which the block appeared and the context from where it was called.

To pass parameters from the block code to the yield statement, the yield statement is called with the parameters and the block has to indicate them between vertical bars before the code block. This is shown in the next example:

Listing 11.4: Iterator with arguments example

```
def fibUpTo(max)
  i1, i2 = 1, 1          # parallel assignment
  while i1 <= max
    yield i1
    i1, i2 = i2, i1+i2
  end
end
fibUpTo(1000) { |f| print f, " " }
```

11.1.2 Programming exercises with Ruby

Table 11.1: Hours needed for developing Ruby programming exercises

Exercise	Time (hours)
Strings, files and regular expressions	4,5
Numbers set	4
Composite Pattern	1,5
TOTAL	10

Times for each exercise are shown in table 11.1.

4.30 hours were required to finish the first exercise. It was almost clear from the beginning that a hash would be used for the dictionary representation. The student was mainly troubled finding the regular expression for the email address and understanding regular expressions. He was trying to avoid a solution where an exception code had to be included, as it was when the word was at the beginning of the line. Finally, he did not find any solution which avoided to use an exception code, and had to use the one exposed in listing A.1.

4.00 hours were required to finish the second exercise. The *Vector constructor* method and the *subset* method were the hardest tasks. He spent a long time trying to solve the *subset* and discovering that he was using a variable which was the key-value pair of the dictionary, while he was thinking it was the

key. Source code is shown in listing [A.2](#).

1.30 hours were required to finish the third exercise. As Ruby is a duck typing system, it was even not necessary in the beginning to declare all the super classes. But, in case it was wanted to test the type, it could be done through testing it and throwing an exception. Source code is shown in listing [A.3](#).

A table describing the times is shown in table [11.1](#).

11.1.3 Ruby on Rails

Ruby on Rails is an open source full-stack web application framework which emphasizes the use of the following engineering patterns and principles: active record pattern, convention over configuration, *don't repeat yourself* and model-view-controller.

In a default configuration, the data in the web application is managed by *models*. The models are Ruby files, with the *rb* extension, which define a table in a database. These files are server database-independent. Database servers can be *PostgreSQL*, *MySQL* or *SQLite*, among others.

The code [11.5](#) shows the model file of a post in a blog application example. It defines:

- The columns: content, name and title
- The data rules: Name and title must be presented and title must have a length longer than 5 characters
- Relations: A post is related to many comments presented in the table comments

Following the principles of convention over configuration, it must exist a table in the database named posts with the columns defined in the model file.

Listing 11.5: Model example. Source [\[12\]](#)

```
class Post < ActiveRecord::Base
  attr_accessible :content, :name, :title

  validates :name, :presence => true
  validates :title, :presence => true,
                  :length => { :minimum => 5 }

  has_many :comments
end
```

Ruby on Rails use a file named **routes.rb** to connect every HTTP request to a controller action. The actions are controller methods. Every HTTP request runs a controller action, the controller action renders a view and may ask the model for data from the database.

In figure [11.6](#) the show action in the Post controller is listed. It is executed through a HTTP request. It asks the model for the post indicated in the url through the *id* identifier and it uses this value to initialize the *@post* variable. Afterwards it renders the *show.html.erb* view.

Listing 11.6: Controller example. Source [12]

```
def show
  @post = Post.find(params[:id])

  respond_to do |format|
    format.html # show.html.erb
    format.json { render :json => @post }
  end
end
```

If an action is not mapped to the Rails router, the action will not be accessible by HTTP requests.

A view in the default configuration of Rails is an *erb* file. Following the convention, the action *show* will look for the view named *show.html.erb*. *Partials* and *layouts* are used to avoid repetition. A *partial* is a view code which can be called from several view files. A layout is the common HTML code in all the views.

Listing 11.7: View example. Source [12]

```
<p>
  <b>Name:</b>
  <%= @post.name %>
</p>

<p>
  <b>Title:</b>
  <%= @post.title %>
</p>

<p>
  <b>Content:</b>
  <%= @post.content %>
</p>
```

Furthermore, Ruby on Rails offers the following tools:

- Scaffolding: Code generator which can create basic code, CRUD operations, for a model
- WEBrick: A simple Ruby server
- Rake: A built system which allows you to create migrations for databases among other functions
- Rubygems: It is a package manager which is often used to install Ruby on Rails itself. It substitutes plugin installation for packaged gems installation, facilitating the task

11.1.4 Website development with Rails

Development was finished in 47 hours. Time was distributed according to figure 11.2.

In this first approach to development, it was not clear how to organize the modules and which step to do first, that is the reason why it was made some mistakes. For example, *Post* module began to be developed but *Blog* module had some details left to be implemented yet. Nevertheless, timing for each module had

Table 11.2: Developing hours for the website with Rails

Documentation	30
User + Auth	13,5
Blog	8
Post	8
Comment	8
Friendly urls	6
Views as described in design	2
Pagination	2
Total Development	47.5
Total	77.5

tried to be extracted accurately from the diary.

Table 11.3 shows the structure of files for MyBlog on Ruby on Rails. Location is relative to the root folder of the application. Lines have been counted removing white lines and comments. The structure is mainly divided into the folders controllers, models and views folder. There are two exceptions. The first one is where helpers are located. Helpers are functions which can be called in controllers, views or models. The second one is the *routes.rb* file. The views folder has a folder per each controller where its views are located. 725 lines is the total of the application source code lines.

Next, it will be explained the documentation that has been read and the main problems found in each of the modules:

- **Documentation(30):** Read documentation was v3.2.13. It was clear, full of examples and extensive. The division of "Digging Deeper" and previous chapters led the student to understand that he should only read the *Start Here*, *Models*, *Views* and *Controllers*. The documentation articles that were read before beginning the development were:
 - Getting started
 - Rails database migrations
 - Active Record validations and Callbacks
 - Active record associations
 - Active record query interface
 - Layouts and rendering in Rails
 - Action view form helpers
 - Action controller overview
 - Rails routing from the outside in
 - Ruby on Rails tutorial: Chapters 6,7,8 and 9. Avoiding learning how to do tests.

Along development, search engine *Google* was used to solve problems. It usually drove to *StackOverflow*. Most of the questions were solved in *StackOverflow* and they had great popularity.

Table 11.3: Source code files of MyBlog on Ruby on Rails with counted lines

Rails			
location	name	type	lines
config/	routes.rb	url	20
App/views/layouts	application.html	view	16
App/views/layouts	_header	view	27
App/views/users	edit	view	30
App/views/users	new	view	30
App/views/users	new_password	view	25
App/views/users	show	view	12
App/views/blogs	edit	view	2
App/views/blogs	_form	view	23
App/views/blogs	new	view	2
App/views/blogs	show	view	9
App/views/comments	_comment	view	39
App/views/comments	_form	view	29
App/views/posts	signed_user_frontpage.html.erb	view	6
App/views/posts	edit	view	2
App/views/posts	_form	view	21
App/views/posts	index	view	11
App/views/posts	new	view	22
App/views/posts	_post	view	20
App/views/posts	show	view	6
App/views/sessions	new	view	14
App/models/	blog	model	7
App/models/	comment	model	10
App/models/	post	model	20
App/models/	user	model	22
App/helpers/	session_helper	helper	29
App/controllers/	application_controller	controller	23
App/controllers/	blogs_controller	controller	40
App/controllers/	comments_controller	controller	41
App/controllers/	posts_controller	controller	69
App/controllers/	session_controller	controller	19
App/controllers/	users_controller	controller	67
App/controllers/	welcome_controller	controller	12
TOTAL			725

- **User authentication(13,5):** User CRUD operations and authentication was made following the Ruby on Rails tutorial: Chapters 6 ,7 ,8 , 9. Tests were not implemented. Adding the functionality of inserting an avatar was developed using *Paperclip*, a Ruby Gem which made almost everything. Changing the password was the most difficult task because it was necessary to deal with error validations which differed from the model. There are two password fields in the form but, in the model and database, only one field is stored: the password encrypted. Adapting the structure of Ruby on Rails to satisfy this requirement was complex
- **Blog(8):** At first, it was tried to develop the application with just 3 models. The objective was to join the Blog model and User model because it seemed to be easier. Along the process, the student realized that it was acquiring complexity. Therefore, it was adapted to a system of 4 models. It was an adaptation from previous work which required a notable effort
- **Post(8):** It was straight forward following the knowledge acquired in previous modules. It was needed to implement more views and listing than in the rest of modules: List posts in front page, of the user, of the blog. To implement the *search posts* functionality, it was needed to review the query strings and form tags in Ruby on Rails
- **Comment(8):** It was necessary to design all the comments and that was a task similar to previous modules. Nevertheless, implementing a form in the view of a post and showing conditionally some buttons depending on if the user was signed in or not was complex. Most of the extra time was invested in learning how to use the framework to implement those requirements
- **Friendly urls(6):** It was necessary to understand well the routing system but the main problem was to adapt modification in the urls to the rest of the application
- **Views as described in design(2):** It did not present difficulties
- **Pagination (2):** It was not known any library to do it neither it was expected it exists. Therefore it was developed in Ruby

To finish, the main problems along development with Ruby on Rails were:

- The programmer error of trying to develop with just 3 models and having to redesign part of the code
- Designing friendly-urls: At beginning, it was made following the convention and at the end, it had to be modified. Modifications affected most of the code and understanding how it had to be treated was not easy
- Controlling the errors in the password form. Rules are usually binded to a model, but it was difficult to define rules which only were applied to the form and, afterwards, to create the encrypted password

And the main pros were:

- Easy documentation with a huge amount of examples
- Great support from community. Almost every question was already asked and answered in Stack-Overflow

11.2 Groovy and Grails

11.2.1 Groovy

Groovy is a dynamic, object-oriented programming language for the Java Platform. It is compiled into Java Bytecode. Its design has been influenced by Python, Ruby, Perl and Smalltalk. It was designed by James Strachan and Guillaume LaForge. It supports multiple programming paradigms: functional, imperative and object-oriented.

Features

- Dynamic language
- Duck typing
- It is compiled into the Java Virtual Machine
- Support closures
- Support operators-overload

Syntax

- Indentation is not mandatory
- Semicolons are not mandatory. They can be used for writing more than one statement in a line
- It uses a Java-like bracket syntax

An example of Hello World is shown in listing [11.8](#).

Listing 11.8: Hello world example in Groovy

```
println 'hello world'
```

As shown in listing [11.9](#), a class definition is composed by the key word *class* followed by the class name and the class content between brackets. Methods are defined with the *def* key word followed by the name of the method, the arguments between parenthesis and the statements block between brackets. Groovy accepts the Java syntax and strong typing can be used in the code. Therefore, methods can be written as in Java.

Listing 11.9: Class example in Groovy

```
class Name{
    def count = 0
    def incr ( n ) { count += n }
    def decr ( n ) {
        count -= n
    }
    String toString(){
        count
    }
}
```

Standard types

Groovy supports all Java types. Furthermore, it adds native support for maps (equal to dictionaries) and lists. Listing 11.10 shows a list example. A list is defined by a set of objects separated by commas between square brackets.

Listing 11.10: List example in Groovy

```
a = [1, 'uno', 3, 4.0]
assert ( a[0] == 1)
assert ( a[3] == 4)
```

A map, shown in listing 11.11, is defined by pairs of key and values joined by colons and separated by commas between square brackets.

Listing 11.11: Map example in Groovy

```
a = [1:'one', 'two':2]
assert ( a[1] == 'one')
assert ( a['two'] == 2)
```

Closures and iterators

A closure is a sequence of statements which can be assigned to a variable and executed later. It can be passed to a function as an argument and the function can execute the closure when it is adequate.

Listing 11.12: Closure example in Groovy

```
square = { it * it }
```

In listing 11.12, *Square* is a closure which squares the argument passed to it. In listing 11.13, the *Collect* method of list has a closure as a possible argument which is executed for each item in the list and the results are collected in another list.

Listing 11.13: Closure example in Groovy

```
result = [ 1, 2, 3, 4 ].collect(){
it * it
}
assert ( result == [ 1, 4, 9, 16 ])
```

Closures are able to remember the context in which they were called and can receive more than one argument. For example in the listing 11.14, to iterate through a map, it would be used the each iterator and the variable for storing keys and values are indicated at the beginning of the code block.

Listing 11.14: Iteration over a Map in Groovy

```
[ 1: 2, 3: 4 ].each(){ key, value ->
```

```
    print key
}
```

11.2.2 Programming exercises with Groovy

Table 11.4: Hours needed for developing Groovy programming exercises

Exercise	Time
Strings, files and regular expressions	3
Numbers set	2.5
Composite Pattern	1
TOTAL	6.5

Groovy is similar to Ruby, therefore, solutions for the different exercises were known in the beginning. Times for each exercise are shown in table [11.4](#).

3 hours were required to finish the first exercise. At the beginning, it was clear that a dictionary would suit for storing the words and which regular expressions would solve the problem. In fact, the solution reached in the Groovy exercise is shorter and clearer mainly due to the use of a better regular expression. Source code is shown in listing [A.4](#).

2.5 hours were required to finish the second exercise. The most difficult problem to solve was the subset method. It did not imply complexity due to the fact that it was already understood the use of the closures, as it was really similar to Ruby. Source code is shown in listing [A.5](#).

1 hour was required to finish the third exercise. It was easy to restrict the type to Component type, as the syntax for arguments can be equal to Java. It was invested thirty minutes in Ruby, probably looking for the duck typing system and how to check the types. Source code is shown in listing [A.6](#).

11.2.3 Grails

Grails is an open source web application MVC framework which uses the Groovy programming language (which is in turn based on the Java platform). It is intended to be a high-productivity framework by following the "coding by convention" paradigm and "don't repeat your self", providing a stand-alone development environment and hiding much of the configuration detail from the developer. Grails reuse Java technologies as Hibernate and Spring under a simple and consistent interface.

In the default configuration, GORM is used to persist the data. Hibernate underlies the GORM and connects to the database selected. By default, an Hyper Structured Query Language Database (HSQLDB) is used.

To create an app, the following command is used.

```
$ grails create-app
```

It creates the skeleton of the application.

Persistence objects are called domains and use the GORM. They are presented in the *grails-app/domain* directory as classes. They can be created using the command:

```
$ grails create-domain-class <domainname>
```

Grails recommends to use always a package for every class. In this description the package *shelf* will be used. For example, it is going to be created a Book domain within the package *shelf*:

```
$ grails create-domain-class shelf.Book
```

The file is called by convention with the name of the domain without any addons. The code in 11.15 shows the book class with a title and its date properties. Property types are expressed before the name of the property as if it would be Java types. Static variable constraints define the rules of the model. For example, in this case the title must have a minimum size of 5 and a maximum size of 45 characters.

Listing 11.15: Grails model example: A book

```
class Book {
    String title
    Date published

    static constraints = {
        title size: 5..45
    }
}
```

By default, controllers are placed in the *grails-app/controllers* directory. They must be the name of the controller suffixed by the word *controller*. Furthermore, they can be created with the command:

```
$ grails create-controller shelve.Book
```

It generates the controller for the Book model described before, if the model is presented. Every controller has a group of actions which can be called and optionally, can create a new instance of some model and render a view. By default, the url */Book/list* is mapped to the *Book* controller and the list action. 11.16 shows a possible list action for the book controller. This action will render the view *list.gsp* in the folder *grails-app/views/book/*, at least other view is specified.

Listing 11.16: Grails controller example: A book

```
class BookController {
    def list() {
        [ books: Book.findAll() ]
    }
}
```

Grails supports JSP and GSP views. In the 11.17, it is shown an example of the *list.gsp*. It is rendered a simple web page with the title "Our books" and it renders an unordered list with the book title of each

of the books and the author. Any view is rendered with a model. The model is the data available to the view. In the listing 11.16, the model is all the books.

Listing 11.17: Grails view example: A list of books

```
<html>
  <head>
    <title>Our books</title>
  </head>
  <body>
    <ul>
      <g:each in="${books}">
        <li>${it.title} (${it.author.name})</li>
      </g:each>
    </ul>
  </body>
</html>
```

Furthermore, Grails offer the following tools:

- A standalone server which automatically reloads resources
- Scaffolding: Automatic code generation for domains, controllers and views

11.2.4 Website development with Grails

Documentation was finished earlier than its time assigned because it was shorter. 22 hours were invested in reading the documentation. Therefore, it allows to spend 55.5 hours developing the application. Time was distributed according to figure 11.5.

In 55.5 hours, MyBlog was almost finished but with some bugs. Updating a user threw an error that it was not possible to solve in the fixed time. Deleting a user requires an HTML redirect because redirecting directly in the controller threw an error. Nevertheless, the application was finished except for those issues.

Table 11.5: Developing hours for the website with Grails

Documentation	22
User + Auth	31.5
Blog	9
Post	3.75
Comment	6.5
Friendly urls	1.5
Views as described in design	3
Pagination	0.25
Total Development	55.5
Total	77.5

Table 11.6 shows the structure of files for MyBlog on Grails. Location is relative to the root folder of the application. Lines have been counted removing white lines and comments. Structure is mainly divided

into the folders *controllers*, *domain* and *views* folder. There is an exception which is where the library for authentication is located, */src/groovy* and *UrlMapping.groovy* file. The views folder has a folder per each controller where its views are located. 1336 lines is the total amount of application source code lines.

Table 11.6: Files of source code of MyBlog on Grails with counted lines

location	name	type	lines
Grails-app	UrlMapping.groovy	url	44
Grails-app/views/layouts	main.gsp	view	66
Grails-app/views/user	edit.gsp	view	47
Grails-app/views/user	_editform.gsp	view	24
Grails-app/views/user	create.gsp	view	46
Grails-app/views/user	_form.gsp	view	27
Grails-app/views/user	newPassword.gsp		58
Grails-app/views/user	show.gsp	view	55
Grails-app/views/blog	edit.gsp	view	43
Grails-app/views/blog	_form.gsp	view	12
Grails-app/views/blog	create.gsp	view	39
Grails-app/views/blog	show.gsp	view	37
Grails-app/views/blog	delete.gsp	view	9
Grails-app/views/post	_comment.gsp	view	21
Grails-app/views/comment	_form.gsp	view	44
Grails-app/views/post	edit.gsp	view	43
Grails-app/views/post	_form.gsp	view	15
Grails-app/views/post	list.gsp	view	24
Grails-app/views/post	create.gsp	view	39
Grails-app/views/blog	_post.gsp	view	18
Grails-app/views/post	show.gsp	view	36
Grails-app/views/auth	login.gsp	view	35
Grails-app/domain/com/myblog	blog	model	15
Grails-app/domain/com/myblog	comment	model	17
Grails-app/domain/com/myblog	post	model	20
Grails-app/domain/com/myblog	user	model	24
src/groovy/com/myblog	authutils.groovy		39
Grails-app/controllers/com/myblog	blogcontroller	controller	83
Grails-app/controllers/com/myblog	commentcontroller	controller	44
Grails-app/controllers/com/myblog	postcontroller	controller	101
Grails-app/controllers/com/myblog	authcontroller	controller	57
Grails-app/controllers/com/myblog	usercontrolller	controller	151
TOTAL			1333

Next, it will be explained the documentation which has been read and the main problems found in each of the modules:

- **Documentation (22):** Documentation used was *The Official User Guide* of Grails, v2.2.0, located at <http://grails.org/doc/2.2.2/>. It was large and confusing. There was not a differentiation between

advanced chapter and beginning chapters. Though, documentation was large, it was spent only 22 hours in reading because the student tried to avoid reading advanced topics which were not necessary for the application development. There were not examples, this fact increased the difficulty to understand how to use the technology. The chapters read were:

- 2. Getting Started: It was read the whole chapter excluding: Deploying an application, Supported Java EE Containers, Generating an Application and Creating Artefacts subchapters
 - 4. Configuration : It was skimmed and it was only read Basic Configuration subchapter
 - 5. The Command Line: It was read only the beginning and the first subchapter, 5.1 Interactive Mode
 - 6. Object Relational Mapping (GORM) : It was read the whole chapter excluding 6.5 *Advanced GORM features* and the following subchapters. The 6.5.1 Events and Autotimestamping section was read
 - 7. The Web Layer:
 - * 7.1 Controllers: It was read until 7.1.6 data binding. The rest was not considered worth to read
 - * 7.2. Groovy Serve Pages: It was read until 7.2.4 Layouts with Sitemesh
 - * 7.3 Tag Libraries: It was read
 - * 7.4 URL Mappings: It was read excluding 7.4.12 Customizing URL formats
 - * 7.6 Filters: It was read
 - 13. Internationalization
 - 14.5 Security Plugins
 - 18. Scaffolding
- **User + Authentication (31.5 hours):** It was divided in this in three tasks:
 - **Basic CRUD User + Authentication (27 hours):** Many hours were spent in this task due to the following reasons: Documentation was not clearly enough, therefore, it had to be read again several times and deeply along the coding. It was not clear which plugin to use in authentication. It was tried to use Shiro, but the documentation seemed too complex and time consuming, so it was switched back to try to develop an own authentication system. While the student was trying to do so, it was learned the mechanism of messages, mixins, services and command objects. It was read before about them in the documentation, but it did not clear how to use them. Finally, looking for the own authentication system, it was found a clearer tutorial to use Shiro. This tutorial plus knowledge acquired along the process, allowed to develop the authentication system with the user. Even though, after the development it was not clear how Shiro worked. There were several complaints in forums about lacking of documentation
 - **User avatar (1 hour):** A plugin which stores images in the database was used to do the task. It seemed to work properly. It is said that is not good practice to store images in databases, nevertheless, the main aim was agility, therefore it was used. But at the end of the development, updating a user thrown an error that it was not possible to solve due to the time limit
 - **Changing password (3.5 hours):** It was not a complex task. Command objects and Shiro functions were useful in this task
 - **Blog (8.5 hours):** Documentation of one-to-one entity relationships was confusing. Hibernate was another feature difficult to understand, mainly due to the lack of documentation

- **Post (3.25 hours):** It was divided into two tasks:
 - **CRUD Post (3 hours):** It has already been acquired some level of knowledge in model relations, hence it was easier
 - **Search posts (15 minutes):** Grails has an understandable function to search based on pre-defined parameters which allowed to build this up quickly
- **Comment (6 hours):** The reason why it was more time consuming than CRUD Post is that the form had to be code in another view, it had to control if the user was signed in or not and other special features had to be implemented
- **Authentication related filters (1.5 hours):** It was written for every model in 1.5 hours due to the centralized filter system. The centralized filter system was not flexible enough to write the permissions as needed. Therefore, a custom code had to be written
- **Friendly URLs (1.5 hours):** The URL mapping was similar to Rails, hence it was easy. Furthermore, all the redirects had to be changed
- **View as described in design (3 hours):** It had to be created some different views and listed some information from different models
- **Pagination (15 minutes):** Grails functions to build pagination are integrated inside the framework and are easy to understand and set up

To finish, the main problems along development with Grails were:

- Lack of clear documentation and examples
- Lack of organized documentation, a short guide for beginners
- It was not clear which plugin to use for authentication system
- The Groovy/Grails Tool Suite (GGTS) was buggy and used a lot of memory

And the main pros were:

- Plugins for pagination and searching were easy to set up

11.3 Python and Django

In this section the Django experience is explained. First, it is shown a short introduction to Python and its features, followed by the solution to the exercises and timings. Finally, it is made an overview of the web framework Django together with the development experience with this technology.

11.3.1 Python

Python is a general-purpose, high-level programming language. It was created in 1993 by Guido van Rossum. It was influenced by Java, C, C++, Perl, Lisp among others and it has influenced Ruby, Groovy, Javascript and others. It supports multiple programming paradigms: functional, imperative, object-oriented, procedural and reflective. It tries to improve productivity and readability of the language.

A hello world example

Listing 11.18: Hello world example in Python

```
print 'hello world'
```

Syntax

The syntax is similar to Ruby and was intended to be a highly readable language.

- Indentation is mandatory and it is used to delimit blocks of code
- Semicolons are not mandatory
- *If* conditions do not need parenthesis surrounding them

Standard types

There are two kinds of types: mutable and immutable. Mutable types cannot be keys in dictionaries, for example.

- Str: A string. They are strings delimited by single or double quotation marks. It is possible to define them using triple-quoted delimiters and write the string in multiple lines
- ByteArray: It is a mutable sequence of bytes
- Bytes: It is the same than a ByteArray but it is immutable
- List: It is an array of any type. It is mutable and is represented between square brackets: [4.0, 'string', True]
- Tuple: It is close to an array, it can also contain mixed types but it is immutable. It is represented between parentheses: (4.0, 'string', True)
- Set, frozenset: A set is an array with no duplicates. A frozenset is immutable
- Dict: An associative array and mutable
- Int: Immutable integer

- Float: Immutable float
- Complex: Immutable complex number
- Bool: An immutable value which can be True or False

Semantic

Python was designed to be a readable language. Python uses indentation to delimit block of code. In Python, everything is an object. It supports multiple inheritance and mixins. All variables in Python hold references to objects and these references are passed to functions.

Classes

A class is composed by the keyword *class* followed by the name of the class and colons. The body of the class is expressed with indentation. There is no end key word. The class ends when the indentation returns back to its initial level.

Every method inside the class follows the same rules. The body is indented and ends when indented increase is finished. The init method represents the constructor of the class. Every method has as an argument the self variable which refers to the object instantiated.

There are no private methods, neither attributes.

Listing 11.19: Class example in Python

```
class NumberSet:
    def __init__(self, vector=None):
        if vector==None:
            self.the_set = set()
        else:
            self.the_set = set (vector)

    def size(self):
        return len(self.the_set)
```

List comprehensions

List comprehensions are a useful tool to create *lists* and *dictionaries* in a concise way. A list comprehension is composed by a group of variables in the beginning and a *for* statement. It can be surrounded by brackets

For example:

Listing 11.20: List comprehension example in Python

```
>>> squares = [x**2 for x in range(10)]
>>> squares
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

11.3.2 Programming exercises with Python

Table 11.7: Hours needed for developing Python programming exercises

Exercise	Time
Strings, files and regular expressions	2
Numbers set	1
Composite Pattern	1
TOTAL	4

Times for each exercise are shown in table [11.7](#).

Two hours were required to finish the first exercise. Opening files was untroubled, in contrast to Groovy which had some issues related to problems to open a file for just writing it. In the documentation there was not the *split* method, but it was found it on *StackOverFlow* used in conjunction with *.strip(string.punctuation)*. This methods led to the solution of splitting each line in words, removing punctuation and testing if this word was an email or in the dictionary. Regex expressions were difficult to find and there were some issues related to create a filter with variables and expressions but, after looking for it in Google, a solution was found. Source code is shown in listing [A.7](#).

One hour was required to implement the second exercise. Python has a type which is a set and satisfied all the conditions required by the exercise. Therefore, it was just a translation of the methods from the set to the ones asked for the exercise. Source code can be seen in listing [A.8](#).

One hour was required to finish the third exercise. Control argument type is not offered and, as in Ruby, it was needed to check for the right type and throw an exception. Source code is shown in listing [A.9](#).

11.3.3 Django

In this section, an overview of the Django framework and its main features will be shown.

Django is a free and open source web application framework, written in Python, which follows the model–view–controller architectural pattern. It is maintained by the Django Software Foundation (DSF), an independent non-profit organization.

Django's primary goal is to ease the creation of complex, database-driven websites. Django emphasizes reusability and "pluggability" of components, rapid development, and the principle of *don't repeat yourself*. Python is used throughout, even for settings, files, and data models. Django also provides an optional administrative create, read, update and delete interface that is generated dynamically through introspection and configured via admin models.

Django was created by Adrian Holovaty and Simon Willison in fall 2003 [[13](#)] while working as developers in the World Online.

Features

The core framework and its bundled applications offer the following features:

- A lightweight, standalone web server for development and testing
- A form serialization and validation system which can translate between HTML forms and values suitable for storage in the database
- A caching framework which can use any of several cache methods
- Support for middleware classes which can intervene at various stages of request processing and carry out custom functions
- An internal dispatcher system which allows components of an application to communicate events to each other via pre-defined signals
- An internationalization system, including translations of Django's own components into a variety of languages
- A serialization system which can produce and read XML and/or JSON representations of Django model instances
- A system for extending the capabilities of the template engine
- An interface to Python's built-in unit test framework
- An extensible authentication system
- The dynamic administrative interface
- Tools for generating RSS and Atom syndication feeds
- A flexible commenting system
- A sites framework that allows one Django installation to run multiple websites, each with their own content and applications
- Tools for generating Google Sitemaps
- Built-in mitigation for cross-site request forgery, cross-site scripting, SQL injection, password cracking and other typical web attacks, most of them turned on by default
- A framework for creating GIS applications

Overview

The core Django MVC framework consists of an object-relational mapper which mediates between data models (defined as Python classes) and a relational database ("Model"); a system for processing requests with a web templating system ("View") and a regular-expression-based URL dispatcher ("Controller"). In the next paragraphs it will be overseen these pieces working together. To begin, it will be created a project.

Django differences between projects and applications. A project references to a website while an application is a web application that has some functionality. A project is composed by applications.

To create a Django project, the following command has to be used:

Listing 11.21: Command for creating a project in Django

```
django-admin.py startproject projectname
```

The `projectname` can be exchanged for the name of the project.

It creates a skeleton of a web project in Django with following structure:

Listing 11.22: Basic project skeleton in Django

```
projectname/  
  manage.py  
  projectname/  
    __init__.py  
    settings.py  
    urls.py  
    wsgi.py
```

The most important files in the folder are *settings.py* and *urls.py*. *Settings.py* is the configuration for the project. While *urls.py* describes the url mapping between urls and views.

A project usually has a set of applications. To create an application, it has to be used the following command:

Listing 11.23: Command for creating an app in Django

```
python manage.py startapp appname
```

This command creates a basic skeleton of an application which is laid out as described below:

Listing 11.24: Basic app skeleton in Django

```
polls/  
  __init__.py  
  models.py  
  tests.py  
  views.py
```

Data are represented by Python classes inside the *models.py* file. The object-relational mapper mediates between these classes and the relational database. Generally, each model class maps to a single database table.

Listing 11.25 shows two python classes representing a Poll and a set of choices. Each model has a number of class variables, each of which represents a database field in the model. Each field is represented

by an instance of a Field class, for example *CharField* for character fields. Note that the relationship is defined using a *ForeignKey* field. Django supports the common database relationships: many-to-ones, many-to-manys and one-to-ones.

Listing 11.25: A model.py file in Django

```
from django.db import models

class Poll(models.Model):
    question = models.CharField(max_length=200)
    pub_date = models.DateTimeField('date published')

class Choice(models.Model):
    poll = models.ForeignKey(Poll)
    choice_text = models.CharField(max_length=200)
    votes = models.IntegerField(default=0)
```

Models are used by views. Views are very similar to controller classes in Rails or Grails frameworks. The *url.py* file maps each url to a view. They may be defined as functions in *views.py*, although they can be classes too.

In listing 11.26, the function *index* takes a request as input as every views must do. It uses *Poll* class to retrieve a list of polls from the database, loads a template and returns an *HttpResponse* where the template is rendered with a context.

Listing 11.26: A views.py file in Django

```
from django.http import HttpResponse
from django.template import RequestContext, loader

from polls.models import Poll

def index(request):
    latest_poll_list = Poll.objects.order_by('-pub_date')[:5]
    template = loader.get_template('polls/index.html')
    context = RequestContext(request, {
        'latest_poll_list': latest_poll_list,
    })
    return HttpResponse(template.render(context))
```

Templates indicate how the data is presented. Listing 11.26 uses a template which is defined in listing 11.27. Templates are HTML with code snippets. To show a variable value, it is used double brackets. Tags can be called inside templates. *If* is a tag inside a template and it behaves as it could be expected.

Listing 11.27: A template file in Django

```
{% if latest_poll_list %}
<ul>
{% for poll in latest_poll_list %}
    <li><a href="/polls/{% poll.id %}"/>{% poll.question %}</a></li>
{% endfor %}
```

```

    </ul>
{% else %}
    <p>No polls are available.</p>
{% endif %}

```

To define the url to connect with a view, the *urls.py* file is used. Listing 11.28 is an example. It is a file with a variable *urlpatterns* where an assignment is ordered. It is assigned the value of a *patterns* class which is instantiated with the set of urls to map. The first url, the root */polls/*, maps to the views described in 11.26.

Below, there are other examples of more complex urls which use regular expresions and capture groups of characters to pass them to the views.

Listing 11.28: A url file in Django

```

from django.conf.urls import patterns, url

from polls import views

urlpatterns = patterns('',
    # ex: /polls/
    url(r'^$', views.index, name='index'),
    # ex: /polls/5/
    url(r'^(?P<poll_id>\d+)/$', views.detail, name='detail'),
    # ex: /polls/5/results/
    url(r'^(?P<poll_id>\d+)/results/$', views.results, name='results'),
    # ex: /polls/5/vote/
    url(r'^(?P<poll_id>\d+)/vote/$', views.vote, name='vote'),
)

```

Class-based views

A view is a callable object which takes a request and returns a response. This can be more than just a function, and Django provides an example of some classes which can be used as views. These allow you to structure your views and reuse code by harnessing inheritance and mixins.

There is a set of generic classes of common tasks, for example, listing objects. For this common task, there is a class-based view called *ListView*. In listing 11.29 a *PollList* class is defined. It inherits from *ListView* and defines the model to work with, *Poll*, and the name of the object to be used in the templates, *polls*.

It is as flexible as the programmer needs because the source code is very readable and is prepared to override any method to achieve the desired functionality.

Listing 11.29: Class-based view example in Django

```

# views.py
from django.views.generic import ListView
from polls.models import Poll

```

```
class PollList(ListView):
    model = Poll
    context_object_name = 'polls'
```

11.3.4 Website development with Django

Documentation was read in 30 hours and development was finished in 40.5 hours. Time was distributed according to figure 11.8.

Table 11.8: Developing hours for the website with Django

Documentation	30
User auth	20
Blog	6
Post	7
Comment	4,5
Friendly urls	0
Views as described in design	0
Pagination	1
Refactoring	2
Total Development	40,5
Total	70,5

Table 11.9 shows the structure of files for MyBlog on Django. Location is relative to the root folder of the application. Lines have been counted removing white lines and comments. The structure is mainly divided into application folders. In this case, there are three applications: *userprofile*, *blog* and *myblog* application. *Myblog* application is the root application which owns the routing file and the index page. Inside each application, there is a file for the views, called *views.py*, a file for the models, called *models.py*, a file for the forms, called *form.py* and a folder for the templates, called *templates*. Inside the template folder there is a file for each template. Templates are inside another folder with the same name of the application to distinguish between them when they are called. 692 lines is the total of the application source code lines.

Next, it will be explained the documentation read and the main problems found in each of the modules:

- **Documentation:** It was read Django 1.3 documentation (<https://docs.djangoproject.com/en/1.3/>). It is quite complete and clear. It is divided into a very short tutorial, *First Steps*; *The Model Layer*; *The Template Layer*; *The View Layer*, *Forms* and other topics, specially more advance, which were not considered valuable to read for this project
 - *First Steps*: It was read the whole chapter
 - *The Model Layer*:
 - * *Models: Model Syntax* was read. *Field Types* and *Meta Options* were skimmed
 - * *QuerySets: Executing Queries* was read and *Query Set Method Reference* was skimmed
 - * *Model Instances*: It was completely read

Table 11.9: Source code files of MyBlog on Django with counted lines

Django			
location	name	type	lines
myblog/urls.py	url.py	url	26
myblog/templates/base.html	base.html	view	49
myblog/templates/	base_settings.html	view	13
userprofile/templates/userprofile/	update.html	view	45
userprofile/templates/userprofile/	user_form.html	view	15
userprofile/templates/registration/	password_change_form.html	view	15
blog/templates/blog	blog_form.html	view	33
blog/templates/blog	blog.html	view	14
blog/templates/blog	post_list.html	view	20
blog/templates/blog	post_form.html	view	34
blog/templates/blog	post.html	view	19
blog/templates/blog	post_detail.html	view	30
blog/templatetags/	blog_extras.py	tags	6
userauth/templates/registration/	login.html	view	21
blog/	models.py	model	49
userprofile/	models.py	model	7
blog/	views.py	controller	151
blog/	forms.py	form	6
myblog/	views.py	controller	14
Userprofile/	views.py	controller	94
userprofile/	forms.py	form	31
TOTAL			692

- *The Template Layer*: Syntax overview was read but the rest of topics were skimmed
- *The View Layer*:
 - * *The basics*: All sub chapters were read
 - * *File Uploads: Overview* and *Managing files* were read, while the rest was skimmed
 - * *Generic Views: Overview* was read and *Built-in generic views* was skimmed
- *Forms*:
 - * *The Basics: Overview* and *Form API* were read and *Built-in fields* and *Built-in widgets* were skimmed
 - * *Advanced: Forms for models* and *Customizing validation* were read, the rest was skimmed
- **User + Auth (20 hours)**:
 - **Basic CRUD User + Authentication (15.5 hours)**: Django comes with a bundle authentication system. It was recommended to use this system. Most of the time was invested in understanding how to extend the system to satisfy the requirements. It was needed to use class-based view and understand them. This feature is slightly complex and it requires source code to be read in some cases
 - **Image attached to user (3)**: It was used a form with files. Django offered a field to upload a file to a define folder easily
 - **Change password (1.5)**: There was already the common forms in the authentication system, therefore, it just had to be adapted
- **Blog(6 hours)**: Initially, there was an error by the developer try to test if a user had a blog in the template. This error consumed several hours. Afterwards, main parts of this module were built using the class-based views with the acquired knowledge
- **Post and Search Posts (6.5 hours)**:
 - **CRUD Post (6 hours)**: It was made using class-based views: There was a problem when it was needed to alter the tables, Django does not offer any tool to alter tables. It had to be done by hand, usually dropping and creating the table, this fact consumed more time. In this part, some adaptation to the "view as described in design" was made
 - **Search posts (0.5)**: Using a class based views in list-view was easy to define the required request in the form and return the set of objects. Displaying them in a template was similar to previous templates with this class
- **Comment (4.5)**: It has to be looked for a way to show a form inside another view. Problem was common and a solution was explained in documentation. It consists in a class which inherits from `DetailView` and `FormMixin`. To solve the problem of a user account which has been deleted and its comments remain with its last data associated, it was used signals and it was needed to review documentation
- **Friendly urls (0)**: Django requires you to define the urls from the beginning. Therefore, it was made along the process
- **Views as described in design (0)**: Django does not have scaffolding. Therefore, templates had to be created for each page and designed view was made along development
- **Pagination (1)**: List-based views offered tools which implemented it. It was needed to understand and customize it

- **Refactoring (2):** Templates for small objects were more difficult to implement in Django than in other frameworks. Therefore, it was decided to do this task at the end. After reading documentation, it was possible to implement them using tags

To finish, main pros and cons that were founded in this experience are

Pros:

- A bundle authentication system
- Class-based views
- Admin panel allows to create fast administration panels
- Useful documentation

Cons:

- To update a table, it is needed to drop and re create the table. Furthermore, it does not have any migration tool for databases
- Loose coupling makes it difficult to implement some options. For example, templates for small objects have to be defined in different layers to create them
- No integration with Javascript libraries. There was not any utility for making an "Are you sure?" Javascript dialog window

11.4 PHP and CodeIgniter

In this section an overview of the experience with PHP and CodeIgniter will be exposed. First, it will be described PHP and its features. Second, it will be analyzed the experience solving the programming exercises with PHP, timings and lines. Third, the CodeIgniter framework will be described and, finally, the experience developing with CodeIgniter will be exposed.

11.4.1 PHP

PHP is a server-side, scripting, object-oriented language designed for web development but also used as a general-purpose programming language. PHP code is interpreted by a web server with a PHP processor module which generates the resulting web page: PHP commands can be embedded directly into an HTML source document rather than calling an external file to process data.

PHP was created by Rasmus Lerdorf in 1995. It was influenced by Perl, C, C++, Java and TCL. It has a weak and dynamic typing.

Features

- Object-oriented
- Web server-side oriented
- It supports public, private, protected methods
- It support interfaces
- It support abstract classes
- It support final classes
- All types are passed by value except objects
- It support pass by reference
- It support traits
- It support static methods and properties
- It does not support overloading operators
- It does not support overloading methods

Syntax and Semantic

PHP code is delimited by a couple of symbols. The most common used are `<?php` and `?>`. Every statement must be terminated by a semicolon, as in C++. Variables are prefixed by a dollar sign, `$`. It is possible to define classes and functions. Classes and function names are case insensitive. PHP has three types of comment syntax: `/* */` which serve as block comments, and `//` as well as `#` which are used for inline comments.

A function is defined by the key word *function* followed by the name of the function, the list of arguments separated by commas inside parenthesis and the function code delimited by brackets. It is possible

to define the class of an argument, but it is not mandatory.

Classes are defined by the key word *class* followed by the name of the class. It is possible to write the key word *extends* after the name to indicate inheritance from another class. The body of the class is defined after the name delimited by brackets. It can define methods and properties. Within an object, *\$this* pseudo-variable refers to the actual object.

Standard types are passed by value unless it is specified to be passed by reference. Classes are passed by reference by default.

The listing 11.30 shows a class and function example.

Listing 11.30: Syntax and semantic example in PHP

```
<?php

class SimpleClass
{
    // property declaration
    public $var = 'a default value';

    // method declaration
    public function displayVar() {
        echo $this->var;
    }
}

function foo($arg_1, $arg_2, ClassType $arg_n)
{
    echo "Example function.\n";
    return $retval;
}

$asimpleclass = new SimpleClass();
$asimpleclass->displayVar();

?>
```

Standard Types

PHP supports eight primitive types.

- **Boolean:** It can be True or False. Similar to C++: 0 integers and float are considered false
- **Integer:** It is an integer number
- **Float:** Real numbers
- **String:** A string is series of characters where a character is the same as a byte
- **Array:** It is an associative array
- **Object:** It is a class object

- **Resource:** A resource is a special variable, holding a reference to an external resource
- **Null:** The special NULL value represents a variable with no value. NULL is the only possible value of type null

Control structures

PHP supports the following control structures:

- If/else/elseif
- While
- Do-while
- for
- foreach
- break
- Continue
- Switch
- Declare
- Goto

All of them are similar to those control structures in C++ except *foreach* which is a PHP control structure.

The foreach construct provides an easy way to iterate over arrays. There are two syntaxes shown in listing 11.31 .

Listing 11.31: Foreach syntaxes in PHP

```
foreach (array-expression as $value)
    statement
foreach (array-expression as $key => $value)
    statement
```

The first form loops over the array given by array-expression. On each iteration, the value of the current element is assigned to \$value and the internal array pointer is advanced by one (so, on the next iteration, it will be looking at the next element). The second form will additionally assign the current element key to the \$key variable on each iteration.

Classes

Classes in PHP support visibility for methods and properties. It has the same behavior than in C++. There are three possible prefixes: public, protected and private. Public methods and properties can be accessed everywhere. Members declared protected can be accessed only within the class itself and by inherited and parent classes. Members declared as private may only be accessed by the class that defines

the member.

PHP supports abstract classes. Abstract classes allow to define abstract methods. Methods defined as abstract simply declare the method's signature, they cannot define the implementation. When inheriting from an abstract class, all methods marked as abstract in the parent's class declaration must be defined by the child; additionally, these methods must be defined with the same (or a less restricted) visibility.

It supports the final methods defined by the final key word. It prevents child classes from overriding a method by prefixing the definition with the *final* keyword.

To solve multiple inheritance, PHP uses Traits. A Trait is similar to a class, but only intended to group functionality in a fine-grained and consistent way. It is not possible to instantiate a Trait on its own. A class inserts a trait to acquire the desired methods.

11.4.2 Programming exercises with PHP

Table 11.10: Hours needed for developing PHP programming exercises

Exercise	Time
Strings, files and regular expressions	2
Numbers set	2
Composite Pattern	0,5
TOTAL	4,5

Times for each exercise are shown in table [11.10](#).

Two hours were required to finish the first exercise. The exercise did not present any difficulty. A dictionary was created reading each of the lines of the file. The white spaces and line breaks were removed using the *trim* method. An associative array was used to keep words in the dictionary. Afterwards, each line of input file is separated by words and each word is tested to be an email or be contained in the dictionary. Source code is shown in listing [A.10](#).

Two hour were required to implement the second exercise. It had to be spent more time than in Python because PHP did not have native libraries for number set support. Operators could not be implemented and the key word union had to be used to define the *union* method. Nevertheless, implementing the exercise was not difficult. Source code can be seen in listing [A.11](#).

Half hour was required to finish the third exercise. PHP support of abstract classes permits to implement the abstract public function `__toString()` to force every child class to implement it. Furthermore, it allowed to control the argument type, hence exceptions to control correct type did not have to be used. Source code is shown in listing [A.12](#).

11.4.3 CodeIgniter

CodeIgniter is an open source rapid development web application framework, for use in building dynamic web sites with PHP. "Its goal is to enable developers to develop projects much faster than writing

code from scratch, by providing a rich set of libraries for commonly needed tasks, as well as a simple interface and logical structure to access these libraries."

CodeIgniter is loosely based on the popular Model-View-Controller development pattern. While view and controller classes are a necessary part of development under CodeIgniter, models are optional. CodeIgniter is most often noted for its speed when compared to other PHP frameworks.

Features

- Extremely Light Weight
- Full Featured database classes with support for several platforms
- Active Record Database Support
- Form and Data Validation
- Security and XSS Filtering
- Session Management
- Email Sending Class. It supports Attachments, HTML/Text email, multiple protocols (sendmail, SMTP, and Mail) and more.
- Image Manipulation Library (cropping, resizing, rotating, etc.). Supports GD, ImageMagick, and NetPBM
- File Uploading Class
- FTP Class
- Localization
- Pagination
- Data Encryption
- Benchmarking
- Full Page Caching
- Error Logging
- Application Profiling
- Calendaring Class
- User Agent Class
- Zip Encoding Class
- Template Engine Class
- Trackback Class
- XML-RPC Library

- Unit Testing Class
- Search-engine Friendly URLs
- Flexible URI Routing
- Support for Hooks and Class Extensions
- Large library of "helper" functions

Overview

CodeIgniter uses the Model-View-Controller approach, which allows great separation between logic and presentation. Maps between URIs and controllers are defined in the *routes.php* file.

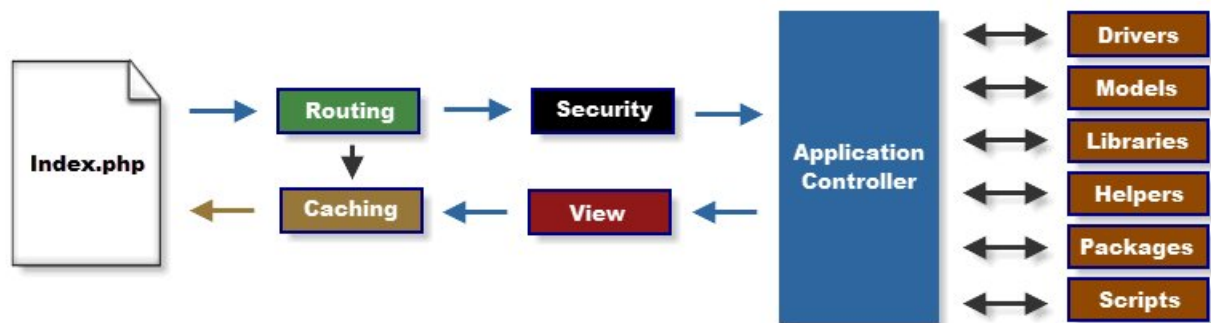


Figure 11.1: Data flow in CodeIgniter. Source [14]

The figure 11.1 shows how a request is received by the index.php:

1. The index.php serves as the front controller, initializing the base resources needed to run CodeIgniter.
2. The Router examines the HTTP request to determine what should be done with it.
3. If a cache file exists, it is sent directly to the browser, bypassing the normal system execution.
4. Security. Before the application controller is loaded, the HTTP request and any user submitted data is filtered for security.
5. The Controller loads the model, core libraries, helpers, and any other resources needed to process the specific request.
6. The finalized View is rendered and sent to the web browser to be seen. If caching is enabled, the view is cached first so that, on subsequent requests, it can be served.

To install CodeIgniter, the software is uncompressed in server directory. A minimal database configuration has to be done to have access to a database. This configuration is specified in the *application/config/database.php* file.

To load a controller you specify the map between the URI and the controller in the *routes.php* file. For example, the code in 11.32 shows a map to connect the URI *blog/show/3* to the show method in the blog controller. The end of the URI has to be a digit and it is passed to the method as a parameter.

Listing 11.32: Example of routing rule in CodeIgniter

```
$route['blog/show/(:num)'] = 'blog/show/$1';
```

Code in 11.33 shows a possible controller for the show action. First, it loads the blog model using the load method of CodeIgniter. To call methods of the framework, it has to be used the `$this` variable. Afterwards, the model method `get_blog_by_id($id)` is called. The `get_blog_by_id($id)` method is defined in the model file. It returns the blog data in an array. Next, if the blog entry exists, the elements *title* and *description* are extracted from the blog array to be sent to the view file. Finally, it is called the load method to render the different views.

Listing 11.33: Example of controller in CodeIgniter

```
class Blog extends CI_Controller{
    function show($id)
    {
        $this->load->model('mblog');
        if ($blog = $this->mblog->get_blog_by_id($id))
        {
            $data = elements(array('title','description'), $blog);

            $this->load->view('templates/header', $data);
            $this->load->view('blog/show', $data);

        }else{//does not exists
            show_404();
        }
    }
}
```

CodeIgniter uses an ActiveRecord class. It allows to execute common database operations in an independent manner. Code in 11.34 shows the method `get_blog_by_id($id)`, which was called in 11.33 to obtain the data of the blog. It executes the method `get_where` of the ActiveRecord class. The first parameter is the table name, and the second is an array with the conditions that filter the results. To obtain the results, it is necessary to execute the method `row_array()` to receive the result inside an array. It is possible to receive results in an object type. Finally, the result is returned.

Listing 11.34: Example of model class in CodeIgniter

```
class MBlog extends CI_Model
{
    private $table_name = 'blog'; // blog table name

    function get_blog_by_id($id){
        $query = $this->db->get_where(
            $this->table_name,
            array('id' => $id)
        );

        if ($query->num_rows() == 1) return $query->row_array();
        return NULL;
    }
}
```

In CodeIgniter, a view is simply a web page or a page fragment, like a header, footer, sidebar, etc, which is finally rendered by the web browser. It is loaded in the controller and information related to the database is passed to the view. For example, in 11.33, the view *blog/show* is loaded after the view *header*. *Blog/show* view is shown in code 11.35. It is an HTML page with PHP code inserted when it is necessary to show the value of variables.

Listing 11.35: Example of view template in CodeIgniter

```
<h1> <?= $title ?> </h1>

<h3> <?= $description ?> </h3>

<? if (isset($posts)) foreach ($posts as $post){ ?>
    <? echo show_post($post) ?>
<? }
else
    echo anchor('/post/create','You haven\'t post anything yet, why don\'t you post some
```

Furthermore, CodeIgniter has a set of helpers to approach different problems. For example, *anchor* is a URL helper. Group of helpers of CodeIgniter are:

- Array
- CAPTCHA
- Cookie
- Date
- Directory
- Download
- Email
- File
- Form
- HTML
- Inflector
- Language
- Number
- Path
- Security
- Smiley
- String

- Text
- Typography
- URL
- XML

11.4.4 Website development with CodeIgniter

23.5 hours were invested in reading CodeIgniter Documentation. Documentation was short and the framework does not have as many features as others in this comparison. There were less functionality to learn to use it. Development required 37 hours to be finished. Time was distributed according to figure 11.11.

Table 11.11: Developing hours for the website with CodeIgniter

Documentation	23,5
User auth	15
Blog	5,5
Post	8
Comment	6,5
Friendly urls	0
Views as described in design	0
Pagination	2
Total Development	37
Total	60,5

Table 11.12 shows the structure of files for MyBlog on CodeIgniter. Location is relative to the root folder of the application. Lines have been counted removing white lines and comments. Structure is mainly divided into the folders controllers, models and views folder. There is an exception, it is where *routes.php* is located. The views folder has a folder per each controller where its views are located. 1452 lines is the total of the application source code lines.

Table 11.12: Source code files of MyBlog on CodeIgniter with counted lines

location	name	type	lines
application/config	routes.php	url	18
application/views/templates	header.php	view	43
application/views/templates	submenu.php	view	10
application/views/auth	update.php	view	78
application/views/auth	register_form.php	view	55
application/views/auth	change_password_form.php	view	40
application/views/blog	update.php	view	34
application/views/blog	create.php	view	32
application/views/blog	show.php	view	7
application/views/post	update.php	view	34
application/views/post	index.php	view	12
application/views/post	create.php	view	31
application/views/post	show.php	view	81
application/views/auth	login_form.php	view	50
application/models	mblog.php	model	76
application/models	mcomment.php	model	84
application/models	mpost.php	model	115
application/models/tank_auth	users.php	model	154
application/controllers	blog.php	controller	91
application/controllers	comment.php	controller	34
application/controllers	post.php	controller	178
application/controllers	auth.php	controller	213
Total			1452

Next, it will be explained the documentation read and the main problems found in each of the modules:

- **Documentation:** The official documentation that was used is located at <http://ellislab.com/codeigniter/user-guide/toc.html> version 2.1.4. It was completely read and it was short and concise
- **User + Auth (15 hours):**
 - **Basic CRUD User + Authentication (11.5 hours):** *Tank Auth* is a CodeIgniter library which is well accepted as the best authentication library by the *Stackoverflow* community [**sof-ci-auth**]. It was used to develop the authentication system and the user-related operations. It took 11 hours and a half to finish the system. *Tank Auth* library supports captcha, email validation among other features that were not needed. The development was made using the library source code and adapting it to the system requirements, removing and adding features, for example, the *current user* method to receive the current user data was added. The library code was clean and readable as described. The common methods to be executed, for example, *current user* or the condition *logged-in-or-redirect*, were tried to be implemented in a super class but it seemed to be more time consuming than implementing it in the library *Tank Auth*
 - **Image attached to user (3):** CodeIgniter's File Uploading Class permits files to be uploaded. This class was used to upload user's avatar. If a redirect was executed, errors about uploading the file were not shown. This fact drove to expend more time trying to solve the problem
 - **Change password (0.5):** There was already the common forms in the authentication system, therefore, it just had to be adapted
- **Blog (5.5):** The common set of tasks to develop a CRUD in CodeIgniter was applied. These are: create the database tables, at database level; create the model class, their methods; create the controller; create the views and support methods to answer questions like *does has a user a blog?*
- **Post (8):** The common set of tasks to develop a CRUD in CodeIgniter was applied. This task was more time consuming because it was needed to implement list and search operations. Furthermore, it had to be shown buttons for each post depending on the viewer and rules for deletion were more complex than before, because the connection between Post module and User module
- **Comment (6.5):** The common set of tasks to develop a CRUD in CodeIgniter was applied. The main problem was to implement the special rules for comment: different views based on the viewer and changes in database after user deletion
- **Friendly urls (0):** CodeIgniter requires you from the beginning to define the URIs. Therefore, it was made along the process
- **Views as described in design (0):** In this case, it was made along the development
- **Pagination (2):** CodeIgniter has a class for pagination. It was easy to customize it to list resources, but it was more difficult to understand how to customize it for the view of the search results. Solutions found searching for it in Google were difficult to implement and they did not allow to bookmark the rendered results because they used the session to keep the search variable. These solutions did not satisfy requirements. Finally, it was developed a solution which was easy to implement and it permitted to bookmark the rendered page

To finish, main pros and cons that were founded in this experience are:

Pros:

- Documentation was short and concise, it was easy to begin developing because there were no many features to understand

Cons:

- Trying to use SQLite3 as the database threw some errors. It was tried to solve them, but solutions found in Internet were not efficient. Due to the fact that trying to use SQLite3 was a very time consuming task and all the documentation was written for MYSQL, it was decided to use MYSQL
- Knowing the errors was difficult in many situations, due to redirections. If there is any redirect, you do not receive the error messages

Chapter 12

Test results

The development has been carried out with each of the frameworks. In this section, a comparison from different perspectives is made. First, it will be made a programming language comparison and after that the web framework comparison.

12.1 Programmimg languages

This project is focused on comparing web frameworks. Nevertheless, a set of exercises had to be done to learn the programming language and a comparison of this experience can be done. This will be exposed in this section. First, time developing each exercise will be analyzed. Second, readability of the language. Third, a performance comparison and, finally, conclusions will be extracted.

The source code of the exercises implemented can be read in [appendix A](#).

12.1.1 Comparison of developing time

Table 12.1: Hours required to develop programming exercises with programming languages

Exercise	Ruby	Groovy	Python	PHP
Strings, files and regular expressions	4,5	3	2	2
Numbers set	4	2,5	1	2
Composite Pattern	1,5	1	1	0,5
TOTAL	10	6,5	4	4,5

Table [12.1](#) shows a comparison of timings used to implement each exercise with each technology. According to the results, Python was the programming language which required less hours to finish the exercises and Ruby was the one which needed the most. It is important to remember that the order of columns is the order in development. The first programming languages used required more time than the last ones. Python and PHP were the exception.

In the first exercise, Ruby was the first language used to implement. Therefore, at the beginning, it had to be understood the regular expressions and how to use them. For example, The Ruby solution to the first exercise used a regular expression which did not satisfy all the cases, therefore, an exception code had to be written. Nevertheless, in Groovy, the regular expression which satisfied all the cases was used. Python already had a method to separate between words and another method for removing punctuation,

which increased the speed in resolving it. PHP solution was close to Python solution.

In the second exercise, in the case of Ruby, it was necessary to understand well the code blocks. For the Groovy case it was easier, since Groovy code blocks were similar to Ruby. For the Python case it was even easier, since Python has a number set class. When using PHP, it had to be programmed in a manner closer to C++.

In the third exercise, Ruby and Python did not support control type argument, hence exceptions had to be written to control type arguments. On the other hand, Groovy and PHP allowed to control type argument. Furthermore, PHP supported abstract classes, therefore, the Component class could be implemented as abstract.

Table 12.2: Source code lines in programming exercises

Exercise	Ruby	Groovy	Python	PHP
Strings, files and regular expressions	30	27	28	35
Numbers set	56	51	28	56
Composite Pattern	50	48	38	50
TOTAL	136	126	94	141

In table 12.2, the number of source code lines for each exercise was counted. Python presented the shortest code. The reasons were: Python did not need end statements in block of code, neither it used curly braces syntax. Furthermore, in the Number Set exercise, it already had a library for implementing it, which resulted in a shorter code. The rest of programming languages were close to each other. PHP presented the longest code. It used curly braces and did not have any special method for iteration, it used the *for* structure which resulted in more lines. Furthermore, Ruby and Groovy were really close, but Groovy had control type argument, for example, which allowed to write less lines in the third exercise.

Based on this analysis and experience, Python presented the best productivity with the least amount of lines of code. Mainly due to the following facts: it provided a complete set of libraries which allowed to solve each exercise quickly and it had a mandatory indentation syntax. Differences in time between Groovy, Ruby and PHP could be a bit tricky, because it was learnt how to solve the exercises by using Ruby. Furthermore, the programmer had knowledge about how to use PHP because the manner to solve the problems is really similar to C++. Nevertheless, according to the facts, PHP presented a high productivity, close to Python.

12.1.2 Readability

In this section, an analysis of readability of each programming language will be made. A comparison between programming languages and the natural languages will be carried out.

In natural language, a predicate is the most complete part of a sentence. The subject names the "do-er" or "be-er" of the sentence; the predicate does the rest of the work. There are several types of predicate but it is always composed by a verb which may connect the subject with other modifiers or information. Object-oriented syntax is the closest one to natural language. In the common form *Object.method argu-*

ments, the object is the subject, the method is the verb and the arguments are the modifiers.

In table 12.3, different orders are expressed in natural language and their implementations in the different programming languages are shown. Examples have been extracted from the exercise whenever it has been possible.

In the first example, it is shown that Ruby allows to use a structure closer to a subject, the number set; the action, the method; and the question symbol which shows it expects a boolean. Groovy is close, but it does not allow the ? symbol. In Python and PHP implementation the verb, the method, acts over the subject, in a less object-oriented manner.

In the second example, a structure for collecting inside a list of elements is shown. In this case, Groovy and Ruby are really similar, they both use the iterators and blocks of code, providing a solution really close to natural language. Groovy presents a shorter solution with using the *it* keyword. Python and PHP present a closer syntax to C++. PHP uses the dot to sum strings, which is not as intuitive as the plus.

The third example is similar to the first one. Ruby uses the ! symbol to make reference to a method which will change the state of the object. Groovy uses an object-oriented manner and closer to natural language but it does not use the exclamation symbol. In this case, Python is closer to natural language than before. PHP still uses the verb, function over the object.

Finally, the fourth example shows the ability of Ruby to use the *unless* statement.

In table 12.4 a list of capabilities which make a language readable are shown. They are extracted from the experience developing these exercises and the website. Python is supposed to be highly object oriented, but some examples have been seen where Ruby and Groovy are more object-oriented than Python. Furthermore, Python is the only programming language which considers indentation mandatory, which is a good feature for improving readability. The fifth feature shows languages which support keyword arguments. This feature was added to Ruby in version 2.0. It increases readability since it is not necessary to remember the order of the arguments for a method, instead, it can be used named arguments.

Based on these examples and the experience developing, Ruby is considered to be the closest one to the natural language because it has special syntax features. Groovy is the second, Python the third, and PHP the last one.

12.1.3 Performance

In this subsection a brief analyses of performance for each programming language will be made. Results in figure 12.1, shows that Ruby, PHP and Python are similar in performance. Groovy is not shown. But Groovy is considered to be 8 times slower than Java. Hence, it would be around 10 times slower than C.

Table 12.3: Comparing readability of languages

Natural Language	Code examples	
Is the list of numbers empty?	Ruby @theNumberSet.empty? Python len(self.the_set) == 0	Groovy numbersMap.size() == 0 PHP count(\$this->arrayset);
Collect names of each element and write them in a paper called result	Ruby @composite.each{ item result+=item.toString } Python for element in self.listComponents: string += element.toString()	Groovy components.each{ string += it } PHP foreach (\$this->list as \$element){ \$string .= \$element->__toString(); }
Remove white spaces from the line	Ruby line.chomp! Python line = line.rstrip()	Groovy Line.trim() PHP \$word = trim(\$word);
If variable is not a component type, throw an exception	Ruby raise 'No component type' unless component.kind_of?(Component) Python if not isinstance(component,Component):raise NameError("No component type")	

Table 12.4: Comparing features for readability in programming languages

Feature	Example	Ruby	Groovy	Python	PHP
Highly object oriented	3.times{ puts 'hello' }	yes	yes	almost	no
It allows to use ? And ! in methods	line.chomp!	yes	no	no	no
It is possible to write arguments for a method avoiding parenthesis and separated by commas	line.gsub 'f','a'	yes	yes	no	no
Unless and until statements	println 'hello' unless dont-knowhim	yes	no	no	no
It uses block codes and iterators	listnumbers.each{ number print number }	yes	yes	no	no
Indentation is mandatory and is used to delimit blocks of code	-	no	no	yes	no
Keyword arguments. Arguments for a method can be passed by a name	travel(from='pointa', to='pointb')	yes	no	yes	no

examples. In some examples Python is shorter and in others larger.

12.1.4 Conclusions

Doing these exercises allowed me to have some level of experience with the different programming languages. It was difficult to measure productivity, due to the fact that the first implementation was when the first solution for the problem was described, hence it required more time. Nevertheless, the next conclusions have been extracted:

- Python was the most productive language. Mainly due to the existence of libraries. It was the most concise too. It is important to remember that in The Computer Language Benchmark Game [15], Ruby and Python were similar in lines of code
- Ruby was the most readable language. Groovy was really similar to Ruby. Python presented new syntax features which were not common in C++, while PHP was really close to C++, hence less readable
- Groovy presents the best performance, it is approximately five times faster than others. PHP, Python and Ruby have a similar performance

12.2 Web framework comparison

In this section, a comparison of the experience with the web frameworks will be shown. Points of comparison will be timing, documentation, community support, steps to execute common functions, some features, lines of code and readability of solutions.

Complete source code of the implementations can be found in Github at the address:

github.com/diaclavijo/practical-comparison-of-agile-web-frameworks

In table 12.5, it is shown a comparison of developing time for each step with each framework. The same data is presented in a graph manner in 12.3. The order of columns represents the order in development. Developing with CodeIgniter required the least amount of time compared with other frameworks, 17 hours less to finish the exercise compared with Rails. Myblog application was finished with every framework in fixed time excluding Grails, which was almost finished with some errors. Developing with Django required less time than with Grails and Rails: 7 hours less.

Before comparing different steps, it will be exposed the commonalities between these frameworks:

- All of them are MVC frameworks
- All of them have a route file to map the urls to the controller

In the next sections, a comparison of different steps throughout development will be exposed.

Table 12.5: Developing times with Rails, Grails, Django and CodeIgniter

Step	Rails	Grails	Django	Codeigniter
Documentation	30	22	30	23,5
User + Auth	13,5	31,5	20	15
Blog	8	9	6	5,5
Post	8	3,75	7	8
Comment	8	6,5	4,5	6,5
Friendly urls	6	1,5	0	0
Views as described in design	2	3	0	0
Pagination	2	0,25	1	2
Refactoring	0	0	2	0
Total	77,5	77,5	70,5	60,5

12.2.1 Documentation

Rails had an excellent documentation. It had several examples which facilitated to understand the framework, it also had a getting started guide which gave an overview of the framework and had a division between beginner and advanced topics. Furthermore, it presented the highest popularity in StackOverflow, most questions were already asked and solved.

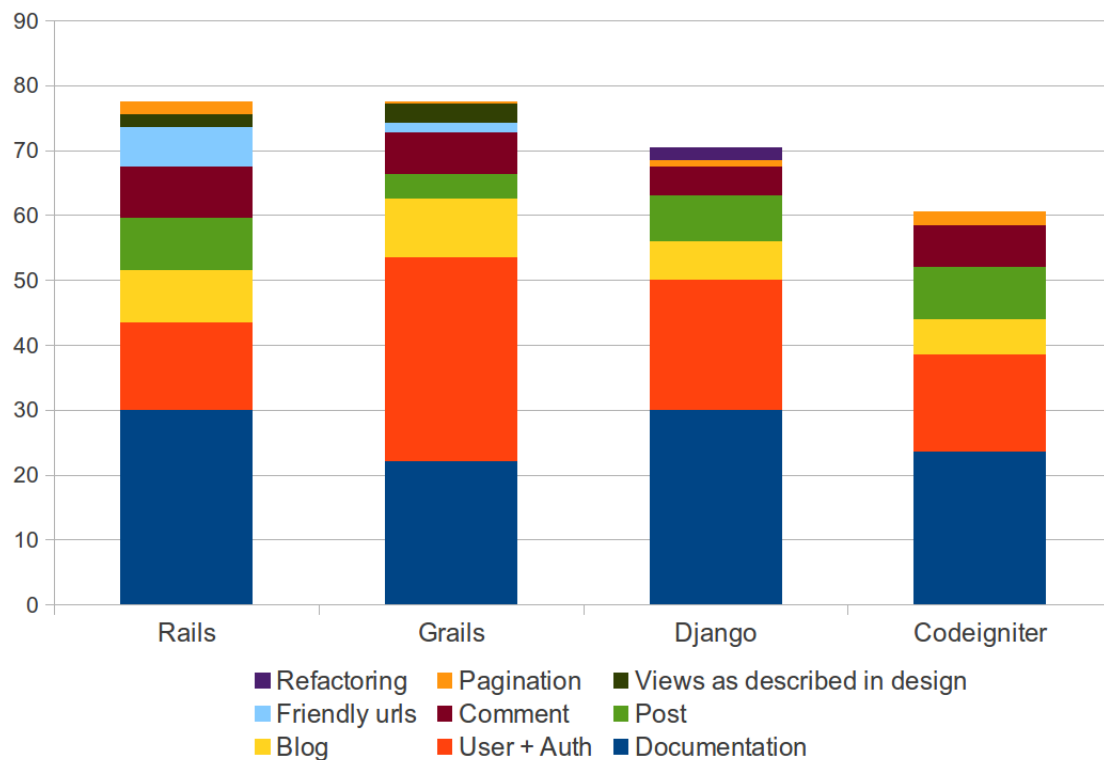


Figure 12.3: Developing times with Rails, Grails, Django and CodeIgniter

Grails required less than 30 hours to read the documentation. It was difficult to understand and there was not a general division for beginners and advanced readers. It was considered, probably, a wrong consideration, that it had been read enough documentation to start implementation and it would be well understood throughout development. Therefore, Grails required more time than others in the second step, when development began, because the documentation was not clear. Other sources of documentation were used to understand it and people were complaining about lacking of documentation in blogs and forums. Main problems were Shiro and Hibernate.

Django had an excellent documentation, it was easy to follow, understand and it had several examples. Most questions were solved in StackOverflow clearly too, but it presented a bit less popularity than Rails.

CodeIgniter presented a concise and short documentation. It lacked examples but it could be completely read and understood in 23.5 hour.

12.2.2 Implementing authentication

In **Rails**, implementing the User module and authentication required the least amount of time. Probably due to the fact that it was implemented following the Ruby on Rails Tutorial [16]. Only with this framework, an own authentication system was built. The reason was that plugins(gems in Rails vocabulary), which offered this functionality, advised to build an own authentication system if the programmer had

never done that.

In **Grails**, implementing authentication was the most difficult task. It was not known if it should be used Shiro or Spring Security plugin. At the beginning, it was tried with Shiro, but the official documentation was complex to be understood. This was the reason why it was tried to build an own authentication system as in Rails. All over this process, some better documentation for Shiro was found and finally, Shiro authentication library was used.

Django came with a bundled authentication system which was well documented, it simply had to be understood and adapted.

CodeIgniter had a Tank Auth library which was well accepted [17]. This library was copied into the application. The source code was easy to understand and it was modified according to requirements.

12.2.3 Implementing Blog, Post and Comment

The modules Blog, Post and Comment are really similar: CRUD (Create, Read, Update and Delete) operations on database tables which are related between them. Therefore, it has been chosen a small example to compare them. This example implements the following use cases and features:

- A user can create a post
- A user can read a post and the comments associated to this post. He/she has the possibility to publish a new comment
- The url to show a post must be friendly
- The owner of a blog is allowed to delete comments of his/her blog

In table 12.6, the steps that were followed to implement this example in each framework are shown. This allows us to compare the different methodologies used to develop with each framework. From a wide perspective, code generators were used with Rails and Grails, while class-based views were used with Django and most of the code had to be written with CodeIgniter.

Rails and Grails steps are really similar. It may be used code generators and adapt them. These generators implement the CRUD operations in a resource with basic views. You can see code generated and adapted for the post controller, the post model and the post show view in Grails and Rails in appendix B.1. These generators can be used to speed up development and make the required changes to the code.

Django offers a different approach with the class-based views, although it is possible to write similar code in Django than the one generated by Grails or Rails. It would be more time consuming than using class-based views. The Django approach requires the programmer to understand well the class-based views, their workflow and methods. This approach follows more strictly the DRY principle than Rails and Grails. In appendix B.2, views, models and a template are listed for the post module. Complete source code can be found in the source code of the full implementation. The action of creating a post is implemented by defining a class *PostCreate*, which is a subclass of *CreateView*, to create a record based on a model. The model to be used is defined and two methods are overwritten: the dispatch method and the *form_valid*. The first one is modified to require a user to be signed in, while the last one is modified to connect the post to a blog, afterwards it is called the super method in both cases.

Table 12.6: Comparing steps in development

Rails	Grails
<ul style="list-style-type: none"> • In console, use scaffold generator to create Post and Comment CRUD operations: it would generate basic views, controllers and models. Migrations to create database tables are created too. They simply have to be run. It is necessary to understand scaffold and the generated code • Edit Models: Write validations and relations between models. Modify the method <i>to_params</i> to use the title as the param to pass in parameters. In the post model: Create a filter to autogenerate the permalink based on the title. It is necessary to understand Rails validations, relations and the <i>to_param</i> method • Edit Post Controller: modify <i>create</i> method to create the post and link it to the user and <i>show</i> method to receive the permalink • Edit routes file: adapt it to use the permalink of the post. It is necessary to understand routing • Edit Comment controller: allow only <i>create</i> and <i>delete</i> methods. Adapt <i>create</i> comment to relate the comment to a user if he/she is signed in. Adapt <i>delete</i> method to control if the user is the owner of the post • Write template for showing a post 	<ul style="list-style-type: none"> • Create the domains Post and Comment, write validations, relations and create a filter to auto generate the permalink based on the title in post domain. It is necessary to understand Grails domains • Scaffold Post and Comment based on the domains defined. Similar to Rails but there are no migrations • Edit Post Controller: Adapt <i>save</i> method to create the post and link it to the user. Adapt <i>show</i> method to find the post using the permalink • Edit <i>UrlMappings</i> file. Adapt it to receive the permalink of the posts to look for it. It is necessary to understand routing • Edit Comment controller: Only allow to <i>create</i> and <i>delete</i> methods. Adapt <i>create</i> comment method to relate the comment to a user if he/she is signed in. Adapt <i>delete</i> method to allow execution only if the user is the owner of the post • Write templates for showing a post
Django	CodeIgniter
<ul style="list-style-type: none"> • Define new models for Post and Comment in models file: Their fields, relations and methods for setting friendly urls. It is necessary to understand well Django models and their methods • In console use <i>syncdb</i> to make Django create the database tables • Create the views inheriting from the Class Based Views and adapt them to satisfy the requirements. It is necessary to understand the class-based views well: their workflow and their methods to know which ones are necessary to be modified • Configure routes to map the urls to the methods • Write templates for showing a post 	<ul style="list-style-type: none"> • Create the databases you need with SQL statements or any database management software • Create the models: Implement the method to publish a post. Implement the method to read a post. Implement the method to read a comment and to delete a comment. Implement the method for retrieving all the comments from a post • Create the controllers: Implement the create method for creating a post. Implement the show method for showing a post. Implement the methods for creating and deleting a comment. Load classes needed, retrieve data to pass it to the view, test if input data is valid • Create the views

CodeIgniter does not have generators neither class-based views. It requires the programmer to write most of the needed methods. It has libraries to help in the task but the active record implementation does not offer a complete set of functions to treat database records as objects. The source code for the model, controller and view are listed here. It can be seen that several methods used in the rest of the web frameworks in this comparison had to be implemented, for example: *create*, *get_post*, *update*, among others.

Code generated by Grails vs Rails

Rails and Grails code generated and adapted for the post controller, model and show post are listed in appendix [B.1](#). As it can be seen, Grails did not generate a model. First, the model had to be written and afterwards the code could be generated. The code generated by Grails was larger than the code generated by Rails. But Grails offered more features with the code generated: it was already prepared for internationalization and, when updating a resource, it controlled if the resource had been updated since it was retrieved. On the other hand, Rails was prepared to render JSON with the code generated. Views generated by Rails were cleaner and more concise than those generated by Grails.

12.2.4 Comparing general features

Some features which have been discovered throughout development are compared here: database tools, active record implementation, Javascript support, support for forms not binded to models, support of DRY principle and convention over configuration.

- Database tools: Tools for creating databases, tables, fields, get back to previous state or move forward
 - Rails had a complete functionality for managing databases. It could be created databases, tables, updated fields, among other features. Furthermore, it supported migrations: any change made to the database was stored in a file called migration which could be set up or down, to commit the changes described in the file or recover to previous state
 - In Grails, it was defined the model in a file and the database was created in HSQLDB. It is not known how difficult is to migrate from this DB to MYSQL, SQLite or others
 - Django supported creating database, although it did not have migrations. Therefore, updates in table structure had to be made with other software
 - CodeIgniter did not offer tools for creating the database
- Active Record Implementation, searching in models and pagination: Tools for operating with records in database in an object-oriented manner. For example, to get a post is possible with the statement *post = Post.find(id)*; to access its fields: *post.title*; or its relations: *post.blog*
 - Rails had a full active record implementation. It did not support search queries in database, while Grails and Django did
 - Grails had a full active record implementation. It supported searches and pagination. It can be seen an example of use in Myblog in [B.18](#)
 - Django had a full active record implementation. It supported searches and pagination It can be seen an example of use in Myblog in [B.19](#)

- CodeIgniter active record implementation lacked several features. There were no standard methods to retrieve an object from a database, or to find a record by an attribute, among others
- Javascript feature: send a dialog to confirm to delete a resource. This project built the application with different frameworks without taking care about the front end. Therefore, it was not necessary to use Javascript. There was only one case where it was useful: to open a dialog requesting if a resource should be deleted or not. Based on this problem:
 - Rails: it was really easy to implement and was full supported
 - Grails: Similar to Rails
 - Django: It was not supported
 - CodeIgniter: It was not supported
- Follows DRY Principle:
 - Rails: Its structure followed greatly the DRY principle but the code generated was repeated in several controllers
 - Grails: Same as Rails
 - Django: It was the best framework in the comparison following this principle due to the fact that it used class-based views
 - CodeIgniter: It did not follow the principle as strongly as other web frameworks did. Most of the methods had to be written and they were usually very similar
- Forms not binded to models: There are special cases where a form is presented in a website but it is not binded to a model. A contact form is an example. What is the approach of each framework to this problem?
 - Rails used virtual attributes to define models which were not connected to a model. It was not well understood after development. It did not seem as clear as other solutions
 - Grails had a class that was called command objects and they served for this purpose
 - Django had a class called forms
 - CodeIgniter had a form library to help in this task
- Convention over configuration
 - Rails followed strictly the convention over configuration principle. It was positive to obtain a more concise code, as most of its part was assumed by the convention. But, if it was necessary to implement something different from the convention, for example not using REST convention, it increased complexity by implementing it
 - Grails followed less strictly convention, more configuration had to be done but it also helped when defining the personalized behaviour
 - Django followed this principle less strictly than Grails
 - CodeIgniter lacked following a convention, for example, there was no standard to call the models. It was decided to use the letter m in front of the name of the resource for naming it

Table 12.7: Comparison of lines of code of implementations with Rails, Grails, Django and CodeIgniter

	Rails	Grails	Django	CodeIgniter
views/templates	375	777	314	507
models/domains	59	76	56	429
controller/views	271	436	296	516
routing files	20	44	26	18
total	725	1333	692	1470

12.2.5 Comparing lines of code

Table 12.7 shows the total amount of lines of source code for every implementation with each web framework. Grails called the models domains and Django, the controller views, but from now on it will be used the words model, controllers and views following the names used in the MVC pattern. It has been divided into four groups: views(it includes small functions to help called helpers and layouts), models, controllers(forms included) and routing files.

According to the results, Django presented the least amount of lines of code, followed by Rails and Grails and CodeIgniter. Django used the class-based views which implied less repetition and shorter code. Rails used code generation, but it generated a really short code which was possibly due to all the conventions assumed. Grails made generation, but it added more functionality in the code and was more verbose. Furthermore, it did not follow so many conventions as Rails. In CodeIgniter, it was necessary to write most of the functions, therefore, it had the biggest amount of lines of code.

12.3 Conclusions

Conclusions of the comparison are divided into five concerns: documentation, authentication system, developing common operations: CRUD, developing times and general conclusions.

12.3.1 Concerning the documentation

Rails was a really well documented framework and had the greatest support from the community but learning how to use it required a huge amount of time. Documentation could not be finished and many features could not be discovered throughout this implementation.

Grails documentation was not well organized, it was difficult to understand, it lacked examples and it had a poor support from the community. It seemed to point to developers coming from Java frameworks, like Spring or others. Nevertheless, when it was understood, the development began to be fast, as it had been proved in the developing times after the first module was implemented: second, third, and fourth modules presented a similar implementation time in every web framework. It had more features than those which were discovered in this implementation.

Django documentation was as good as Rails, at least, and it had great support from the community. It had more features than those which were discovered in this implementation too.

CodeIgniter documentation could be completely read in less than the fixed time. Documentation was clear and concise. It was not necessary to look for questions because most of the times it was already known how to solve the problems. Mainly, because it was the programmer who had to create the methods.

12.3.2 Concerning the authentication system

Rails was the framework which required the least amount of hours to implement this requirement despite the fact that the authentication system was built, it was not used any plugin(gem). It showed that, for an inexperienced programmer, it was faster having a tutorial than learning how to use the plugins in the other different frameworks. Being built by the programmer had the cons of being probably more insecure.

In Grails and Django the authentication systems had to be adapted, but the code was not easy to understand, neither how to customize it, it required more time. It is important to keep in mind that probably Grails and Django approaches were more secure as they underlay in the plugin implementation.

On the other hand, CodeIgniter offered a library whose code was copied into the application and adapted to it. It was easier to customize because it could be read the whole code, remove unnecessary parts and adapt others. Nevertheless, as it was copied into the project and fully adapted, it could have security problems as in Rails.

12.3.3 Concerning developing common operations: CRUD

Rails and Grails generated code were good approaches, specially for an inexperienced programmer. It was more common that they would not know how to approach these problems, scaffold gave a fast solution and it could be worked from there. Nevertheless, code was repeated several times in different

controllers, it did not follow so strongly the DRY principle. As it was commented previously, Rails generated code was clearer than Grails, although Grails offered internationalization and version control.

Django approach with the class based views was a bit harder to understand, but it was the best approach to follow the DRY principle and it gave the most concise code.

CodeIgniter did not offer generators, class-based views and even it did not have a complete ActiveRecord implementation, a library for accessing database records in an object oriented way. The framework helped with libraries and facilitates to follow the MVC pattern, but it had to be written several methods which were already supported in other frameworks. Due to this fact, it weakly followed the DRY principle and the Convention over Configuration.

12.3.4 Concerning the developing times

Rails was the first framework to develop with. There were a lot to learn about the problem and web development in general, for example, differences between POST and GET verbs. Furthermore, when it was tried to modify the conventions, it increased complexity solving the problems. In Rails, in general, everything was going to be implemented, there was a Rails manner to implement it and it required the programmer to learn it.

Grails is the only framework whose implementation was not possible to be finished. It was almost finished but there were some functionalities left and some errors. The main problem with Grails was the documentation, the IDE, Groovy and Grails ToolSuite based on Eclipse, and the fact that the JVM had to be loaded for developing slowed down development speed.

Django was the first framework which presented less time to implement the full application. Reasons could have been: experience from previous implementations, readable and understandable documentation and class-based views.

CodeIgniter was the only framework which required less time than fixed to read documentation and to implement the application. Nevertheless, it was the framework which less adequately followed the DRY and the Convention over Configuration principles. CodeIgniter did not have as many features as other frameworks in this comparison, therefore, it was not necessary to read the documentation all the time. Throughout the development, most of the times, it was not necessary to read how to use libraries and how they were connected to each other, it simply could be programmed as it was needed. Therefore, it required the least amount of time, compared to others in this comparison.

12.3.5 General conclusions

If a newbie programmer just wants a small web application to get started fast, CodeIgniter is an adequate choice.

Rails, Django and Grails are frameworks which require a great effort to learn how to use them. Nevertheless, they are better prepared to support bigger applications, using them will result in a clearer, better organized, less repeated, easier to support code and to follow better a convention. Learning those frameworks is a time investment to obtain benefits in several months.

It is more common for bigger corporations to use Java, therefore Grails, based on Groovy, which is compiled into JVM, is more probable useful to work with big corporations. Furthermore, it presents the best performance. Although, as it has been said, it is more difficult to learn it than others in this comparison.

Chapter 13

Conclusions and future works

In this chapter, conclusions about the methodology followed to make the comparison are firstly exposed. Afterwards, possible future works about web frameworks comparison and web development are presented, extracted from the experience in this project.

13.1 Conclusions

At the beginning of this document, it was said that comparing web frameworks was not an established discipline. This project has taken an approach different to the ones that have been seen before. In order to achieve objectivity, the definition of the test has been strict and precise and every framework has been tested under the same rules. Nevertheless, some factors have influenced in the results of the project and they could not be avoided:

- The order of the test execution with each of the web frameworks influenced the developing times. The first technology to develop with was more probable to need more time to finish
- The background of the student. Previous knowledge influenced the learning process with each technology

Furthermore, it is said that it is necessary nearly a year of experience developing with a web framework to acquire a certain level of expertise. In this test, 125 hours were spent with each of the web frameworks. Therefore, this approach is not feasible for measuring the productivity of an expert. Nevertheless, spending 1 year with each framework would have implied a 4 year project. During this imaginary project, it is highly probable that each web framework would have delivered a new version, making the results of the comparison useless. While this project was being executed, Rails developed a new version and other web frameworks have delivered new subversions.

Nevertheless, due to the fact that the student carrying out the project was inexperienced in web development, it has been a good test for measuring the ease of learning of web frameworks. Furthermore, it has been shown that the web frameworks which have more features are harder to learn, but they offer a more concise code and an overview of the main methodologies to develop with each framework has been exposed, with practical examples and built with the same test.

The kind of approach to the problem where the web framework features are analyzed and grades are given to them, similar to Raible and Villamor works, would have kept this comparison very theoretical.

These kind of approach requires more time to be spent in the project, as Villamor did, or more experience in web development, as Raible has. With the knowledge acquired along this project, it will be interesting to make this kind of comparison. But, at the beginning, being an inexperienced web developer who makes the comparison, it would have produced a really theoretical work by someone who has never built a real application with the tools he/she is reviewing.

This comparison still lacks objectivity and the web framework comparison is still not a established discipline. It has not been found any method to carry out comparisons. Nevertheless, a valuable analysis with practical examples has been exposed. The fact that the student was inexperienced in web development has allowed to compare the ease of learning of each web framework and an overview of the main features and methodologies.

13.2 Future works

Based on this experience, a list of possible future works is proposed:

- To complete the Villamor's model for a set of web frameworks
- To compare less popular frameworks and more innovative
- To compare more distinct web frameworks. Request-based vs component-based
- To compare front-end web frameworks
- Far away from comparisons, but interesting in the web development knowledge. To choose a web framework and improve it

Chapter 14

Results of Project Diffusion

In this chapter, an analysis of the results obtained in the project dissemination is presented.

14.1 Key experts in dissemination

An email asking for opinion was sent to different experts interested in the subject. The following list presents the experts who answered to the emails:

- Matt Raible: He has made web frameworks comparison around the world
- David Heinemeier Hansson
- Russel Keith-Magee: Developer of Django

Matt Raible published a post in his blog with the question and answer to my email with a link to the blog of the project. This produced a lot of income traffic and even an article was published in *Java.Dzone.com*, which has an Alexa rank of 2676.

14.2 Social networks

- The twitter account reached 49 followers
- Facebook page guided some attention but mainly by author's friends
- Google plus 3 followers

14.3 Referrals

T.co makes references to short urls, mainly used when sharing articles through social networks. It can be inferred that the Twitter account and the articles from *Java.dzone.com* and *Raibledesigns.com* were the top 3 referrals. *Stackoverflow* is presented as a referral because, at least, two questions about web framework comparison were answered in it along the development with a link to the project blog.

Table 14.1: Top 10 referrals in the blog

Source	Visits
1. t.co	578
2. raibledesigns.com	472
3. java.dzone.com	234
4. plus.url.google.com	171
5. websitesframeworks.com	131
6. facebook.com	115
7. stackoverflow.com	94
8. forum.root.cz	68
9. google.com	64
10. m.facebook.com	52

14.4 Visits

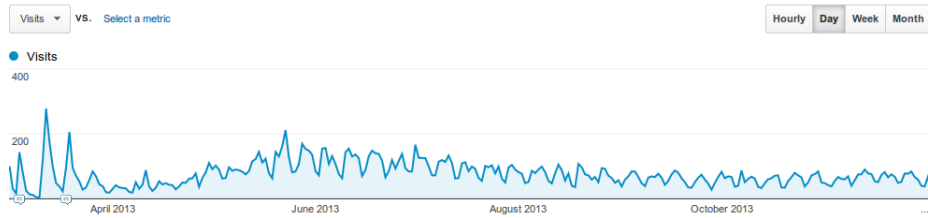


Figure 14.1: Visits to the blog.

Figure 14.1 shows the visits to the blog during the project, from 1 March to December 1. At least 1 article was published every two weeks until 1 May, after this date it was not published any new article until November because the programming of the comparison was being carried out and it required all the time. These articles produces traffic in the following months, around 100 visits per day. This visit rate was decreasing until the end of December reaching around 50 visits per day.

14.5 Search Engine Optimization

On December 3, one of the blog's article appears in the 17th result, second page, by searching for the words "comparison of web frameworks".

Appendices

Appendix A

Source code of programming exercises

A.1 Source code of programming exercises in Ruby

A.1.1 Source code of The Strings, Files and Regular Expressions exercise in Ruby

Listing A.1: The Strings, Files and Regular Expressions exercise in Ruby

```
#First input file names are written in these variables**
ninput = 'input'
ndictionary = 'dictionary'
noutput = 'output'

#Second, the dictionary is created. It is used a hash for create the
  dictionary
#because it is easy to test if a word is in the dictionary by evaluating
#the dictionary[word].
dictionary = Hash.new()
File.foreach(ndictionary){ |line|
  line.chomp! #removes \n marks from the line.
  dictionary[line]=true
}

#Create the output file. Creates the file if it does not exists and for
  write only.
theOutput = File.new(noutput,File::CREAT|File::WRONLY)

#Gets every word of every line and do the block code. Surround emails and
  word with brackets and braces.
File.foreach(ninput){ |line|
  #Surrounded every email defined by the regular expression with braces.
  line.gsub!(/([\w-]+\([\.\[\w-]+\)*@[A-Za-z0-9-]+\([\.\[A-Za-z0-9-]+\)*\([\.\[A-Za-z
    ]{2,4}\)\)/,
    '{\1}')

  #Gets every word, test if it is in the dictionary, if it is it is
    surrounded by brackets, if it is not, it is leave in its state.
  line.gsub!(/(^([A-Za-z'-]+))/) {
    if dictionary[$1]
      '['+$1+']'
    else
      $1
    end
  }
}
```

```

    end
  }
  #The previous example does not take into account words which are in the
  #beginning of the line. With this substitution this exception is solved.
  line.gsub!(/(\s+)([A-Za-z'-]+)/) {
    if dictionary[$2]
      $1+'['+$2+']'
    else
      $1+$2
    end
  }

  #write the modified line in the output file
  theOutput.puts line
}

#close file
theOutput.close()

puts '*** END OF THE PROGRAM ***'
#end

```

A.1.2 Source code of The Numbers Set exercise in Ruby

Listing A.2: The Numbers Set exercise in Ruby

```

#A hash has been chosen to represent the NumbersSet due to the fact that
#numbers cannot be repeated inside the set and the hash allow to test
#easily this feature using the numbers as keys and the values as true or
#false.
class NumbersSet
  attr_reader :theNumberSet #define the attributes which are accesible
  #from outside the class.

  #Constructor function. It supports 0 arguments and creates an empty
  #NumbersSet or a vector with numbers and creates the numbers set with
  #the numbers inside the vector. It can be created with a Hash too.
  def initialize(vector = nil)
    @theNumberSet= Hash.new()
    if vector.kind_of?(Array)
      puts 'Initialize: arg Vector'
      vector.each() { |number|
        if @theNumberSet[number]
          raise '**MY MESSAGE**Vector with repeated component'
        else
          @theNumberSet[number]=true
        end
      }
    elsif vector.kind_of?(Hash)
      puts 'initialize with a hash'
      @theNumberSet=vector
    else
      puts 'initialize:EmptySet'
    end
  end
end

```

```

end

def size
  return @theNumberSet.size
end

def isEmpty
  return @theNumberSet.empty?
end

def add(number)
  if @theNumberSet.include?(number) then
    raise "**MSG** NUMBER REPEATED"
  end
  @theNumberSet[number]=true
end

def belong(number)
  return @theNumberSet[number]
end

def ==(theOtherSet)
  return @theNumberSet==theOtherSet.theNumberSet
end

def <(theSuperSet)
  return !(
    @theNumberSet.find{|number|
      !(theSuperSet.belong(number[0]))
    }
  )
end

def to_v()
  @theNumberSet.keys
end

def +(theOtherSet)
  theNewNumberSet = @theNumberSet.dup
  theOtherSet.theNumberSet.each_key{|key|
    theNewNumberSet[key]=true
  }
  return NumbersSet.new(theNewNumberSet)
end
end

aNumberSetEmpty = NumbersSet.new();
#raise "**MSG**NO EMPTY" unless aNumberSetEmpty.isEmpty #EXCEPTION RAISED
aVector = [1,2,3,4]
aRepVector = [1,2,3,4,2]
aNumberSet = NumbersSet.new(aVector)
#aNumberSet = NumbersSet.new(aRepVector) #EXCEPTION RAISED
if aNumberSet.size != 4 then raise "**MSG**SIZE ERROR " end
unless aNumberSetEmpty.isEmpty then raise "**MSG**EMPTY ERROR " end
#aNumberSet.add(2) #EXCEPTION RAISED

```

```

aNumberSet.add(9)

unless aNumberSet.belong(9) then raise "**MSG**BELONG ERROR 9 SHOULD BELONG
" end
if aNumberSet.belong(11) then raise "**MSG** BELONG ERROR" end

unless aNumberSet==NumbersSet.new([1,2,3,4,9]) then raise "**MSG** WRONG
EQUAL, MUST BE TRUE" end

if aNumberSet==NumbersSet.new([1,2,3,4,8]) then raise "**MSG** FAIL IN
EQUAL" end
#puts NumbersSet.new([1,2]).belong(2)
if aNumberSet<NumbersSet.new([1,2]) then raise "**MSG** FAIL IN SUBSET" end

unless NumbersSet.new([1,2])<aNumberSet then raise '**MSG** FAIL IN SUBSET'
end

puts aNumberSet.to_v().inspect

equalNumberSet = NumbersSet.new([1,2,3,4,9])
almostEqualNumberSet = NumbersSet.new([1,2,3,4,6])
highNumberSet = NumbersSet.new([11,12,13,14])

puts 'UNION BEGINS'
puts (aNumberSet+highNumberSet).to_v().inspect
puts (aNumberSet+almostEqualNumberSet).to_v().inspect
puts (aNumberSet+equalNumberSet).to_v().inspect

puts aNumberSet.inspect();

```

A.1.3 Source code of The Composite Pattern exercise in Ruby

Listing A.3: The Composite Pattern exercise in Ruby

```

class Component

end

class TextComponent < Component
  attr_writer :text
  def initialize( text = nil )
    @text=text
  end
  def toString
    return @text
  end
end

class NumberComponent < Component
  attr_writer :number
  def initialize( number = nil )
    @number=number
  end
  def toString
    return @number.to_s
  end
end

```

```

    end
end
class DateComponent < Component
  attr_writer :date
  def initialize( date = nil )
    @date=date
  end
  def toString
    return @date
  end
end
class CompositeComponent < Component
  def initialize
    @composite=Array.new()
  end
  def add(component)
    raise 'ERROR IT IS NOT A COMPONENT' unless component.kind_of?(
      Component)
    @composite.push(component)
  end
  def size
    return @composite.size
  end
  def modify(index,component)
    raise 'ERROR IT IS NOT A COMPONENT' unless component.kind_of?(
      Component)
    @composite[index]=component
  end
  def toString
    result=String.new()
    @composite.each{|item| result+=item.toString}
    return result
  end
end
end

p= CompositeComponent.new()
p.add(TextComponent.new('This is string. Next a number'))
p.add(NumberComponent.new(1))
p.add(TextComponent.new("\n"))
p.add(TextComponent.new("Date: "))
p.add(DateComponent.new("Today"))
p.add(TextComponent.new("\n"))
p.add(TextComponent.new("Yesterday:"))
p.add(DateComponent.new('yesterday'))
p.add(TextComponent.new("\n"))
d = CompositeComponent.new()
d.add(TextComponent.new('Testing composite'))
d.add(TextComponent.new("\n"))
d.add(p)
p=nil
d.add(TextComponent.new('End testing composite'))
print d.toString
d.modify(0,TextComponent.new('Beginning modified'))
print d.toString

```

A.2 Source code of programming exercises in Groovy

Exercise 1

Listing A.4: Source code of The Strings, Files and Regular Expressions exercise in Groovy

```
VALID_EMAIL=/^[_a-z0-9-]+(\.[_a-z0-9-]+)*@[a-z0-9-]+(\.[a-z0-9-]+)*(\. [a-z]{2,4})$/
inputn = 'input'
dictionaryn = 'dictionary'
outputn = 'output'

//create dictionary
dictionarymap = [:]
dictionary = new File(dictionaryn)
dictionary.eachLine {
    dictionarymap[it]=true;
}

//read input file and treat it

inputf = new File(inputn)
outputf = new FileWriter(outputn)

inputf.eachLine{
    // split in words
    outputf << it.replaceAll(/\s,:'?!']*)([^\\s,:'?!]*)(\\s,:'?!']*)/){
        if ( it[2] ==~ VALID_EMAIL){
            it[2] = '['+it[2]+']'
        }else{
            if ( dictionarymap[it[2]] ) {
                it[2] = '['+it[2]+']'
            }
        }

        it[1]+it[2]+it[3]
    }
    outputf << '\n'
}
outputf.write()
outputf.close()
```

Exercise 2

Listing A.5: Source code of The Numbers Set exercise in Groovy

```
class NumbersSet{
    def numbersMap = [:]
    NumbersSet (vector){
        vector.each{
            if (numbersMap[it]) throw new Exception('Elements repeated in the
                vector')
```

```

        numbersMap[it]=true
    }
}

def size(){
    numbersMap.size()
}

def isEmpty(){
    return (numbersMap.size() == 0)
}

def add(number){
    if (numbersMap[number]) throw new Exception('The element is already in
        the vector')
    numbersMap[number] = true
}

def belong(number){
    numbersMap[number]
}

def equal(other){
    numbersMap == other.numbersMap
}

def subset(other){
    if (other.isEmpty()) return true
    !other.numbersMap.any{key,value ->
        !numbersMap[key]
    }
}

def toVector(vector){
    numbersMap.each{ key, value ->
        vector << key
    }
    return vector
}

def union(other){
    def unionset = numbersMap.clone()
    other.numbersMap.each{ key, value ->
        unionset[key] = true
    }
    return unionset
}

String toString(){
    def s_vector = 'NumberSet: ['
    numbersMap.each{key,value ->
        s_vector += key+', '
    }

    s_vector += ']'
}
}

```

```

println 'hello world'

anumbersetempty = new NumbersSet()

anumberset = new NumbersSet([1,2,3])
//anumberset = new NumbersSet([1,2,3,3]) throws exception
assert anumberset.size() == 3

assert (anumbersetempty.isEmpty())

//anumberset.add(1) throws exception

anumberset.add(4)

assert anumberset.size() == 4

assert anumberset.belong(4)

assert !anumberset.belong(5)

assert anumberset.equal( new NumbersSet([2,4,1,3]))

assert !anumberset.equal( new NumbersSet([2,4]))

assert anumberset.subset( new NumbersSet())

assert anumberset.subset( new NumbersSet([2,4]))

assert !anumberset.subset( new NumbersSet([2,4,6]))

assert [1,2,3,4]==anumberset.toVector([])

println anumberset.union( new NumbersSet([5,6,7]))

println anumberset.union( new NumbersSet([1,2,3]))

println anumberset.union( new NumbersSet([3,4,7]))

println anumberset.union( new NumbersSet())

print anumberset

```

Exercise 3

Listing A.6: Source code of The Composite Pattern exercise in Groovy

```

class Component{

}

class TextComponent extends Component{
    def text

```



```

    TextComponent(textv){
        text=textv
    }

    String toString(){
        text
    }
}

class NumberComponent extends Component {
    def number

    NumberComponent(numberv){
        number=numberv
    }

    String toString(){
        number
    }
}

class DateComponent extends Component{
    def date

    DateComponent(datev){
        date=datev
    }

    String toString(){
        date
    }
}

class CompositeComponent extends Component{
    def components = []
    def add( Component component){
        components << component
    }

    def size(){
        components.size()
    }

    def modify(index, Component component){
        components[index]=component
    }

    String toString(){
        def string=""
        components.each{
            string += it
        }
        string
    }
}

```



```

dic[line] = True

# i will open an out put file to write data
outputf = open(noutput,'w')

#i will read each line of the input file
for line in inputf:
    #i will search for every word in the input file
    words = line.split()
    for word in words:
        word = word.strip(string.punctuation)
        regexword = r'\b' + re.escape(word) + r'\b'
        if word in dic: #i will look if it is an email or it is in the
            diccionario
            line = re.sub(regexword, "[" + word + "]",line ) #i will change this
                line accoridng to it
        elif EMAIL_REGEX.match(word):
            regexword = r'\b' + re.escape(word) + r'\b'
            line = re.sub(regexword, "{" + word + "}",line ) #

    outputf.write(line) #write the line in the output file

#i will close the file
inputf.close()
dicf.close()
outputf.close()

```

A.3.2 Source code of The Numbers Set exercise in Python

Listing A.8: The Numbers Set exercise in Python

```

class NumbersSet:
    the_set = set()

    def __init__(self,vector=None):
        if vector==None:
            self.the_set = set()
        else:
            self.the_set = set (vector)

    def size(self):
        return len(self.the_set)

    def isEmpty(self):
        return len(self.the_set) == 0

    def add(self,element):
        if element in self.the_set:
            raise NameError('element is in the set')
        else:
            self.the_set.add(element)

    def belong(self,element):

```

```

        return element in self.the_set

    def equal(self, otherSet):
        return self.the_set == otherSet.the_set

    def subset(self, otherSet):
        return self.the_set < otherSet.the_set

    def vector(self):
        return list(self.the_set)

    def union(self, otherSet):
        newNumberSet = NumbersSet()
        newNumberSet.the_set = self.the_set | otherSet.the_set
        return newNumberSet

print 'hello world'
emptyNumberSet = NumbersSet()

threeNumberSet = NumbersSet([1,2,3])

assert (emptyNumberSet.size() == 0)

assert (emptyNumberSet.isEmpty())

assert (threeNumberSet.size() == 3)

threeNumberSet.add(4)

#threeNumberSet.add(3) #uncommented must throw an exception
assert ( threeNumberSet.belong(3) )
assert ( threeNumberSet.belong(2) )
assert ( threeNumberSet.belong(1) )
assert ( not threeNumberSet.belong(5) )

#testing equal
equalNumberSet = NumbersSet([1,2,3,4])
emptyEqualNumberSet = NumbersSet()

assert ( threeNumberSet.equal(equalNumberSet) )
assert ( emptyEqualNumberSet.equal(emptyNumberSet) )

#testing subset
twoNumberSet = NumbersSet([1,2])

assert ( not threeNumberSet.subset(twoNumberSet) )
assert ( twoNumberSet.subset(threeNumberSet) )

#testing vector method
print twoNumberSet.vector()
print threeNumberSet.vector()
print emptyNumberSet.vector()

```

```

#testing union method
sixNumberSet = NumbersSet([5,6,7])
oneToSixNumberSet = NumbersSet([1,2,3,4,5,6,7])

print sixNumberSet.union(threeNumberSet).the_set
print oneToSixNumberSet.union(threeNumberSet).the_set

print threeNumberSet.the_set

```

A.3.3 Source code of The Composite Pattern exercise in Python

Listing A.9: The Composite Pattern exercise in Python

```

import copy

class Component:
    pass

class TextComponent(Component):

    def __init__(self, text):
        self.text = text

    def toString(self):
        return self.text

class NumberComponent(Component):

    def __init__(self, number):
        self.number = number

    def toString(self):
        return str(self.number)

class DateComponent(Component):

    def __init__(self, date):
        self.date = date

    def toString(self):
        return self.date

class CompositeComponent(Component):

    def __init__(self):
        self.listComponents = list()

    def add(self, component):
        if isinstance(component, Component):

```

```

        self.listComponents.append(copy.deepcopy(component))
    else:
        raise NameError("No component type")

    def size(self):
        return len(self.listComponents)

    def modify(self, index, component):
        if isinstance(component, Component):
            self.listComponents[index]=component
        else:
            raise NameError("No component type")

    def toString(self):
        string = ''
        for element in self.listComponents:
            string += element.toString()
        return string

p = CompositeComponent()
p.add(TextComponent("String. Number:"))
p.add(NumberComponent(1))
p.add(TextComponent("\n"))
p.add(TextComponent("Date:"))
p.add(DateComponent("02/02/02"))
p.add(TextComponent("\n"))
d = CompositeComponent()
d.add(TextComponent("Testing composite"))
d.add(TextComponent("\n"))
d.add(p)
del p
d.add(TextComponent("End testing composite"))

print d.toString()

print '-----'
d.modify(0,TextComponent("beginning modified"))

print d.toString()

```

A.4 Source code of programming exercises in PHP

A.4.1 Source code of The Strings, Files and Regular Expressions exercise in PHP

Listing A.10: The Strings, Files and Regular Expressions exercise in PHP

```

<?php
const NINPUT = 'input';
const NDIC = 'dictionary';
const NOUTPUT = 'output';
const EMAIL_REGEX = "/^[_a-z0-9-]+(\\.[_a-z0-9-]+)*@[a-z0-9-]+(\\.[a-z0-9-]+)
    * (\\.[a-z]{2,4})$/";

```

```

$finput = fopen(NINPUT, 'r');
$fdic = fopen(NDIC, 'r');
$foutput = fopen(NOUTPUT, 'w');

while(!feof($fdic)){ // create dictionary
    $word = fgets($fdic);
    $word = trim($word);
    $dic[$word] = True;
}

while(!feof($finput)){ // read the input file to look for words
    $line = fgets($finput); // read the a line from inputfile
    $splitline = preg_split("/\s+/", $line, -1, PREG_SPLIT_NO_EMPTY); //
        separate all the words , or characters separed by spaces
    foreach ($splitline as $word){ // for each word test

        if ( preg_match(EMAIL_REGEX,$word) ){ //is email ?
            $replace = '{'.$word.'}'; // write the replacement word
            $line = str_replace($word,$replace,$line); // write it on the
                line variable
        }else{ // is in the dictionary ?
            preg_match("/([\w-]+)/", $word, $matches);

            if (isset($dic[$matches[1]])){

                $replace = '['.$matches[1].']'; // write the replacement
                    word
                $regex = '/\b'.$matches[1].'\b/';

                $line = preg_replace($regex,$replace,$line); // write it on
                    the line variable
            }
        }
    };
    fwrite($foutput,$line); // write the modified line
}

fclose($finput); // close files
fclose($fdic);
fclose($foutput);

echo 'hello';

?>

```

A.4.2 Source code of The Numbers Set exercise in PHP

Listing A.11: The Numbers Set exercise in PHP

```

<?php

class NumberSet{
    private $arrayset;

    function __construct($vector = null){
        if ($vector == null){
            $this->arrayset = array();
        }else{
            $this->arrayset = array();
            foreach ($vector as $element) {

                if (key_exists($element,$this->arrayset)){
                    throw new Exception('Trying to initialize with a vector
                    with repeated elements');
                }
                $this->arrayset[$element]=True;
            }
        }
    }

    function size(){
        return count($this->arrayset);
    }

    function isEmpty(){
        return count($this->arrayset)==0;
    }

    function add($number){
        if (key_exists($number,$this->arrayset)){
            throw new Exception('Trying to add a number which is inside the
            vector');
        }else{
            $this->arrayset[$number] = True;
        }
    }

    function belong($number){
        return key_exists($number, $this->arrayset);
    }

    function equal(NumberSet $theother){
        return ($this->arrayset == $theother->arrayset );
    }

    function subset(NumberSet $theother){
        foreach (array_keys($theother->arrayset) as $number){

            if (!key_exists($number, $this->arrayset)){
                return False;
            }
        }
        return True;
    }
}

```



```

function vector(){
    return array_keys($this->arrayset);
}

function union($theother){
    $union = new NumberSet(array_keys($this->arrayset)); //init $union
    with this instance numbers
    foreach (array_keys($theother->arrayset) as $number){ //find which
        numbers which are in the $theother are not in the $union and add
        them
        if (!key_exists($number, $union->arrayset)){
            $union->arrayset[$number]=True;
        }
    }
    return $union;
}
}

echo "hello\n";

$aempty = new NumberSet();

var_dump($aempty);

$first = new NumberSet(array(1,2,3,4));
//~ $fail = new NumberSet(array(1,2,3,3,4)); // throws exception

var_dump($first);

assert($first->size() == 4);
assert($aempty->size() == 0);

assert($aempty->isEmpty());

$aempty->add(1);
assert($aempty->size() == 1);
var_dump($aempty);

$first->add(5);
assert($first->size() == 5);
var_dump($first);

//~ $first->add(3); // THROWS EXCEPTION

assert($first->belong(1));
assert($first->belong(4));
assert($aempty->belong(1));

assert(!$first->equal($aempty));

$equal = new NumberSet(array(5,1,4,3,2));

```

```

assert ($equal->equal($first));

$second = new NumberSet(array(5,6,7));

assert ($equal->subset($first));
assert ($first->subset($aempty));
assert (!$aempty->subset($first));
assert (!$first->subset($second));

var_dump($equal->vector());
var_dump($aempty->vector());

var_dump($first->union($aempty));
var_dump($first->union($second));
$third = new NumberSet(array(6,7,8));
var_dump($first->union($third));

var_dump($first->union(new NumberSet()));

echo 'THE END';

?>

```

A.4.3 Source code of The Composite Pattern exercise in PHP

Listing A.12: The Composite Pattern exercise in PHP

```

<?php

abstract class Component{
    abstract public function __toString();
}

class TextComponent extends Component{
    private $text;

    public function __toString(){
        return $this->text;
    }

    function __construct($text){
        $this->text = $text;
    }
}

class NumberComponent extends Component{
    private $number;
    function __construct($number){
        $this->number = $number;
    }

    function __toString(){
        return $this->number;
    }
}

```

```

    }
}

class DateComponent extends Component{
    private $date;
    function __construct($date){
        $this->date = $date;
    }

    function __toString(){
        return $this->date;
    }
}

class CompositeComponent extends Component{
    private $list;

    function __toString(){
        $string = '';
        foreach ($this->list as $element){
            $string .= $element->__toString();
        }
        return $string;
    }

    function add(Component $element ){
        $this->list[] = clone $element;
    }

    function size(){
        return $this->list.count();
    }

    function modify($index, Component $element){
        $this->list[$index] = clone $element;
    }
}

$p = new CompositeComponent();

$p->add( new TextComponent("String. Number:"));
$p->add( new NumberComponent(1));
$p->add( new TextComponent("\n"));
$p->add( new TextComponent("Date:"));
$p->add( new DateComponent("7/7/12"));
$p->add( new TextComponent("\n"));
$d = new CompositeComponent();
$d->add( new TextComponent("Testing composite"));
$d->add( new TextComponent("\n"));
$d->add($p);
unset ($p);
$d->add( new TextComponent("End testing composite"));

echo $d;

echo " new test\n";

```

```
$d->modify(0, new TextComponent("Beginning modified"));  
echo $d;
```

```
?>
```

Appendix B

Examples of implementation source code with different web frameworks

B.1 Code Generated vs Modified code in Rails and Grails

B.1.1 Rails generated code and adaptation

Listing B.1: Generated Rails post model

```
class Post < ActiveRecord::Base
  belongs_to :blog
  attr_accessible :text, :title
end
```

Listing B.2: Adapted Rails post model

```
class Post < ActiveRecord::Base
  attr_accessible :body, :title
  belongs_to :blog
  has_many :comments, :dependent => :destroy
  validates :title, :presence => true, :length => { :maximum =>150 }
  before_save do |post|
    post.permalink = post.title.parameterize
  end
  def self.search(search)
    if search
      search = '%' + search + '%'
      where('(title LIKE ?) OR (body LIKE ?)', search, search)
    else
      find(:all)
    end
  end
  def to_param
    self.permalink
  end
end
```

Listing B.3: Generated Rails post controller

```

class PostsController < ApplicationController
  # GET /posts
  # GET /posts.json
  def index
    @posts = Post.all

    respond_to do |format|
      format.html # index.html.erb
      format.json { render :json => @posts }
    end
  end

  # GET /posts/1
  # GET /posts/1.json
  def show
    @post = Post.find(params[:id])

    respond_to do |format|
      format.html # show.html.erb
      format.json { render :json => @post }
    end
  end

  # GET /posts/new
  # GET /posts/new.json
  def new
    @post = Post.new

    respond_to do |format|
      format.html # new.html.erb
      format.json { render :json => @post }
    end
  end

  # GET /posts/1/edit
  def edit
    @post = Post.find(params[:id])
  end

  # POST /posts
  # POST /posts.json
  def create
    @post = Post.new(params[:post])

    respond_to do |format|
      if @post.save
        format.html { redirect_to @post, :notice => 'Post was successfully
          created.' }
        format.json { render :json => @post, :status => :created, :location
          => @post }
      else
        format.html { render :action => "new" }
        format.json { render :json => @post.errors, :status => :
          unprocessable_entity }
      end
    end
  end
end

```

```

    end
  end

  # PUT /posts/1
  # PUT /posts/1.json
  def update
    @post = Post.find(params[:id])

    respond_to do |format|
      if @post.update_attributes(params[:post])
        format.html { redirect_to @post, :notice => 'Post was successfully
          updated.' }
        format.json { head :no_content }
      else
        format.html { render :action => "edit" }
        format.json { render :json => @post.errors, :status => :
          unprocessable_entity }
      end
    end
  end

  # DELETE /posts/1
  # DELETE /posts/1.json
  def destroy
    @post = Post.find(params[:id])
    @post.destroy

    respond_to do |format|
      format.html { redirect_to posts_url }
      format.json { head :no_content }
    end
  end
end

```

Listing B.4: Adapted Rails post controller

```

class PostsController < ApplicationController
  before_filter :signed_in_user, :only => [:new, :create, :edit, :update, :
    destroy, :myindex]
  before_filter :correct_user, :only => [:edit, :update, :destroy]
  def show
    @post = Post.find_by_permalink(params[:post_permalink])
  end
  def new
    @post = Post.new()
  end
  def index
    @page_count = Post.search(params[:search]).count/3
    @page_count += 1 unless Post.search(params[:search]).count%3 == 0
    @posts = Post.order('created_at DESC').limit(3).offset(params[:offset])
      .search(params[:search])
    if signed_in?
      @user = User.find(current_user.id)
      render 'posts/signed_user_frontpage'
    else

```

```

        render 'posts/index'
      end
    end
  end
  def user_name_index
    @user = User.find_by_name(params[:user_name])
    @blog = @user.blog
    if @user
      @page_count = @blog.posts.count/3
      @posts = @blog.posts
      render 'index'
    else
      render :inline =>
        "<h1> Error: that user does not exists </h1>"
    end
  end
  def create
    @post = Post.new(params[:post])
    @post.blog_id = current_user.blog.id
    if @post.save
      flash[:success] = "Your post has been published!"
      redirect_to post_path(@post.blog.user.name,@post.permlink)
    else
      render "new"
    end
  end
  def edit
    @post = Post.find_by_permalink(params[:post_permalink])
  end
  def update
    @post = Post.find_by_permalink(params[:post_permalink])
    if @post.update_attributes(params[:post])
      flash[:success] = "Post updated"
      redirect_to post_path(@post.blog.user.name,@post.permlink)
    else
      render 'edit'
    end
  end
  def destroy
    @post = Post.find_by_permalink(params[:post_permalink])
    @post.destroy
    redirect_to :back
  end
  private
  def correct_user
    post = Post.find_by_permalink(params[:post_permalink])
    user = User.find_by_name(params[:user_name])
    unless current_user?(user)
      flash[:notice] = "You are not allowed to do that"
      redirect_to(root_path)
    end
  end
end
end

```

Listing B.5: Generated Rails show template


```

<p id="notice"><%= notice %></p>

<p>
  <b>Title:</b>
  <%= @post.title %>
</p>

<p>
  <b>Text:</b>
  <%= @post.text %>
</p>

<p>
  <b>Blog:</b>
  <%= @post.blog %>
</p>

<%= link_to 'Edit', edit_post_path(@post) %> |
<%= link_to 'Back', posts_path %>

```

Listing B.6: Adapted Rails show template

```

<%= render @post %>
<%= render @post.comments %>
<h2>Add a comment:</h2>
<% unless defined?(@comment) then @comment = @post.comments.build end %>
<%= render :partial => "comments/form",
  :locals => {:comment => @comment, :post => @post} %>

```

B.1.2 Grails generated code and adaptation

Listing B.7: Adapted Grails post model

```

package com.myblog

class Post {

  String title
  String permalink
  String body
  Date dateCreated
  Date lastUpdated

  static belongsTo = [blog: Blog]
  static hasMany = [comments: Comment]

  static constraints = {
    title blank: false
    body blank: false
  }

  static mapping = {

```

```

        comments sort: 'dateCreated'
    }

    def beforeValidate() {
        permalink = title.replaceAll(/\s/, '-')
    }
}

```

Listing B.8: Generated Grails post controller

```

package com.myblog

import org.springframework.dao.DataIntegrityViolationException

class PostController {

    static allowedMethods = [save: "POST", update: "POST", delete: "POST"]

    def index() {
        redirect(action: "list", params: params)
    }

    def list(Integer max) {
        params.max = Math.min(max ?: 10, 100)
        [postInstanceList: Post.list(params), postInstanceTotal: Post.count
        ()]
    }

    def create() {
        [postInstance: new Post(params)]
    }

    def save() {
        def postInstance = new Post(params)
        if (!postInstance.save(flush: true)) {
            render(view: "create", model: [postInstance: postInstance])
            return
        }

        flash.message = message(code: 'default.created.message', args: [
            message(code: 'post.label', default: 'Post'), postInstance.id])
        redirect(action: "show", id: postInstance.id)
    }

    def show(Long id) {
        def postInstance = Post.get(id)
        if (!postInstance) {
            flash.message = message(code: 'default.not.found.message', args
            : [message(code: 'post.label', default: 'Post'), id])
            redirect(action: "list")
            return
        }

        [postInstance: postInstance]
    }
}

```

```

def edit(Long id) {
  def postInstance = Post.get(id)
  if (!postInstance) {
    flash.message = message(code: 'default.not.found.message', args
      : [message(code: 'post.label', default: 'Post'), id])
    redirect(action: "list")
    return
  }

  [postInstance: postInstance]
}

def update(Long id, Long version) {
  def postInstance = Post.get(id)
  if (!postInstance) {
    flash.message = message(code: 'default.not.found.message', args
      : [message(code: 'post.label', default: 'Post'), id])
    redirect(action: "list")
    return
  }

  if (version != null) {
    if (postInstance.version > version) {
      postInstance.errors.rejectValue("version", "default.
        optimistic.locking.failure",
        [message(code: 'post.label', default: 'Post')] as
        Object[],
        "Another user has updated this Post while you
        were editing")
      render(view: "edit", model: [postInstance: postInstance])
      return
    }
  }

  postInstance.properties = params

  if (!postInstance.save(flush: true)) {
    render(view: "edit", model: [postInstance: postInstance])
    return
  }

  flash.message = message(code: 'default.updated.message', args: [
    message(code: 'post.label', default: 'Post'), postInstance.id])
  redirect(action: "show", id: postInstance.id)
}

def delete(Long id) {
  def postInstance = Post.get(id)
  if (!postInstance) {
    flash.message = message(code: 'default.not.found.message', args
      : [message(code: 'post.label', default: 'Post'), id])
    redirect(action: "list")
    return
  }
}

```

```

    try {
        postInstance.delete(flush: true)
        flash.message = message(code: 'default.deleted.message', args:
            [message(code: 'post.label', default: 'Post'), id])
        redirect(action: "list")
    }
    catch (DataIntegrityViolationException e) {
        flash.message = message(code: 'default.not.deleted.message',
            args: [message(code: 'post.label', default: 'Post'), id])
        redirect(action: "show", id: id)
    }
}
}

```

Listing B.9: Adapted Grails post controller

```

package com.myblog

import org.springframework.dao.DataIntegrityViolationException
import org.apache.shiro.SecurityUtils

@Mixin(AuthUtils)
class PostController {

    static allowedMethods = [save: "POST", update: "POST", delete: "POST"]

    def index() {
        redirect(action: "list", params: params)
    }

    def list(Integer max) {
        def postList
        def postTotal
        def hisPostList
        hisPostList=[]
        params.max = Math.min(max ?: 3, 6)

        if ( params.searchquery == null) {
            postList = Post.list(params)
            postTotal = Post.count()
        }
        else{
            postList = Post.findAllByTitleLikeOrBodyLike('%'+params.searchquery+'%','%'+params.searchquery+'%',params)
            postTotal = Post.findAllByTitleLikeOrBodyLike('%'+params.searchquery+'%','%'+params.searchquery+'%').size()
        }

        if (SecurityUtils.subject.authenticated){
            def user = currentUser()
            hisPostList = user.blog.posts
        }

        [postList: postList, postTotal: postTotal, params: params, hisPostList:
            hisPostList ]
    }
}

```

```

}

def create() {

    [post: new Post(params)]
}

def save() {
    def post = new Post(params)
    post.blog = currentUser().blog
    if (!post.save(flush: true)) {
        render(view: "create", model: [post: post])
        return
    }

    flash.message = message(code: 'default.created.message', args: [
        message(code: 'post.label', default: 'Post'), post.id])
    redirect(mapping: "post", params: [username: post.blog.user.
        username, permalink: post.permalink])
}

def show() {

    def post = Post.findByPermalink(params.permalink)
    if (!post) {
        flash.message = "Post not found"
        redirect(action: "list")
        return
    }

    def newComment = new Comment()
    newComment.post = post
    [post: post, newComment: newComment, comments: post.comments]
}

def edit(Long id) {
    def post = Post.get(id)
    if (!post) {
        flash.message = message(code: 'default.not.found.message', args
            : [message(code: 'post.label', default: 'Post'), id])
        redirect(action: "list")
        return
    }

    [post: post]
}

def update(Long id, Long version) {
    def post = Post.get(id)
    if (!post) {
        flash.message = message(code: 'default.not.found.message', args
            : [message(code: 'post.label', default: 'Post'), id])
        redirect(action: "list")
        return
    }

    if (version != null) {

```

```

        if (post.version > version) {
            post.errors.rejectValue("version", "default.optimistic.
                locking.failure",
                [message(code: 'post.label', default: 'Post')] as
                    Object[],
                "Another user has updated this Post while you
                    were editing")
            render(view: "edit", model: [post: post])
            return
        }
    }

    post.properties = params

    if (!post.save(flush: true)) {
        render(view: "edit", model: [post: post])
        return
    }

    flash.message = message(code: 'default.updated.message', args: [
        message(code: 'post.label', default: 'Post'), post.id])
    redirect(action: "show", id: post.id)
}

def delete(Long id) {
    def post = Post.get(id)
    if (!post) {
        flash.message = message(code: 'default.not.found.message', args
            : [message(code: 'post.label', default: 'Post'), id])
        redirect(action: "list")
        return
    }

    try {
        post.delete(flush: true)
        flash.message = message(code: 'default.deleted.message', args:
            [message(code: 'post.label', default: 'Post'), id])
        redirect(action: "list")
    }
    catch (DataIntegrityViolationException e) {
        flash.message = message(code: 'default.not.deleted.message',
            args: [message(code: 'post.label', default: 'Post'), id])
        redirect(action: "show", id: id)
    }
}
}
}

```

Listing B.10: Generated Grails post show

```

<%@ page import="com.myblog.Post" %>
<!DOCTYPE html>
<html>
    <head>
        <meta name="layout" content="main">
    </head>
    <body>

```

```

<g:set var="entityName" value="${message(code: 'post.label', default: '
    Post')}" />
<title><g:message code="default.show.label" args="[entityName]" /></
    title>
</head>
<body>
<a href="#show-post" class="skip" tabindex="-1"><g:message code="
    default.link.skip.label" default="Skip to content&hellip;" /></a>
<div class="nav" role="navigation">
    <ul>
        <li><a class="home" href="${createLink(uri: '/')}"><g:message code=
            "default.home.label" /></a></li>
        <li><g:link class="list" action="list"><g:message code="default.
            list.label" args="[entityName]" /></g:link></li>
        <li><g:link class="create" action="create"><g:message code="default
            .new.label" args="[entityName]" /></g:link></li>
    </ul>
</div>
<div id="show-post" class="content scaffold-show" role="main">
    <h1><g:message code="default.show.label" args="[entityName]" /></h1>
    <g:if test="${flash.message}">
    <div class="message" role="status">${flash.message}</div>
    </g:if>
    <ol class="property-list post">

        <g:if test="${postInstance?.title}">
        <li class="fieldcontain">
            <span id="title-label" class="property-label"><g:message code="
                post.title.label" default="Title" /></span>

            <span class="property-value" aria-labelledby="title-label"><g:
                fieldValue bean="${postInstance}" field="title"/></span>

        </li>
        </g:if>

        <g:if test="${postInstance?.body}">
        <li class="fieldcontain">
            <span id="body-label" class="property-label"><g:message code="
                post.body.label" default="Body" /></span>

            <span class="property-value" aria-labelledby="body-label"><g:
                fieldValue bean="${postInstance}" field="body"/></span>

        </li>
        </g:if>

        <g:if test="${postInstance?.dateCreated}">
        <li class="fieldcontain">
            <span id="dateCreated-label" class="property-label"><g:message
                code="post.dateCreated.label" default="Date Created" /></span>

            <span class="property-value" aria-labelledby="dateCreated-label
                "><g:formatDate date="${postInstance?.dateCreated}" /></span>
        </li>
        </g:if>
    </ol>

```

```

</li>
</g:if>

<g:if test="${postInstance?.lastUpdated}">
<li class="fieldcontain">
  <span id="lastUpdated-label" class="property-label"><g:message
    code="post.lastUpdated.label" default="Last Updated" /></span>

    <span class="property-value" aria-labelledby="lastUpdated-label"
      "><g:formatDate date="${postInstance?.lastUpdated}" /></span>
    >

</li>
</g:if>

<g:if test="${postInstance?.permalink}">
<li class="fieldcontain">
  <span id="permalink-label" class="property-label"><g:message code
    ="post.permalink.label" default="Permalink" /></span>

    <span class="property-value" aria-labelledby="permalink-label"
      "><g:fieldValue bean="${postInstance}" field="permalink"/></span>

</li>
</g:if>

</ol>
<g:form>
  <fieldset class="buttons">
    <g:hiddenField name="id" value="${postInstance?.id}" />
    <g:link class="edit" action="edit" id="${postInstance?.id}"><g:
      message code="default.button.edit.label" default="Edit" /></g:
      link>
    <g:actionSubmit class="delete" action="delete" value="${message(
      code: 'default.button.delete.label', default: 'Delete')}"
      onclick="return confirm('${message(code: 'default.button.
        delete.confirm.message', default: 'Are you sure?')}');" />
  </fieldset>
</g:form>
</div>
</body>
</html>

```

Listing B.11: Adapted Grails post show

```

<%@ page import="com.myblog.Post" %>
<!DOCTYPE html>
<html>
  <head>
    <meta name="layout" content="main">
    <g:set var="entityName" value="${message(code: 'post.label', default: '
      Post')}" />

```



```

<title><g:message code="default.show.label" args="[entityName]" /></title>
</head>
<body>
<a href="#show-post" class="skip" tabindex="-1"><g:message code="default.link.skip.label" default="Skip to content&hellip;" /></a>
<div class="nav" role="navigation">
<ul>
<li><a class="home" href="{createLink(uri: '/')}"><g:message code="default.home.label" /></a></li>
<li><g:link class="list" action="list"><g:message code="default.list.label" args="[entityName]" /></g:link></li>
<li><g:link class="create" action="create"><g:message code="default.new.label" args="[entityName]" /></g:link></li>
</ul>
</div>
<div id="show-post" class="content scaffold-show" role="main">
<h1><g:message code="default.show.label" args="[entityName]" /></h1>
<g:if test="{flash.message}">
<div class="message" role="status">${flash.message}</div>
</g:if>
<ol class="property-list post">

<g:if test="{postInstance?.title}">
<li class="fieldcontain">
<span id="title-label" class="property-label"><g:message code="post.title.label" default="Title" /></span>

<span class="property-value" aria-labelledby="title-label"><g:fieldValue bean="{postInstance}" field="title"/></span>

</li>
</g:if>

<g:if test="{postInstance?.body}">
<li class="fieldcontain">
<span id="body-label" class="property-label"><g:message code="post.body.label" default="Body" /></span>

<span class="property-value" aria-labelledby="body-label"><g:fieldValue bean="{postInstance}" field="body"/></span>

</li>
</g:if>

<g:if test="{postInstance?.dateCreated}">
<li class="fieldcontain">
<span id="dateCreated-label" class="property-label"><g:message code="post.dateCreated.label" default="Date Created" /></span>

<span class="property-value" aria-labelledby="dateCreated-label"><g:formatDate date="{postInstance?.dateCreated}" /></span>
>

</li>
</g:if>

```

```

<g:if test="${postInstance?.lastUpdated}">
<li class="fieldcontain">
  <span id="lastUpdated-label" class="property-label"><g:message
    code="post.lastUpdated.label" default="Last Updated" /></span>

    <span class="property-value" aria-labelledby="lastUpdated-label"
      "><g:formatDate date="${postInstance?.lastUpdated}" /></span>
    >

</li>
</g:if>

<g:if test="${postInstance?.permalink}">
<li class="fieldcontain">
  <span id="permalink-label" class="property-label"><g:message code
    ="post.permalink.label" default="Permalink" /></span>

    <span class="property-value" aria-labelledby="permalink-label"
      "><g:fieldValue bean="${postInstance}" field="permalink"/></span>

</li>
</g:if>

</ol>
<g:form>
  <fieldset class="buttons">
    <g:hiddenField name="id" value="${postInstance?.id}" />
    <g:link class="edit" action="edit" id="${postInstance?.id}"><g:
      message code="default.button.edit.label" default="Edit" /></g:
      link>
    <g:actionSubmit class="delete" action="delete" value="${message(
      code: 'default.button.delete.label', default: 'Delete')}"
      onclick="return confirm('${message(code: 'default.button.
        delete.confirm.message', default: 'Are you sure?')}');" />
  </fieldset>
</g:form>
</div>
</body>
</html>

```

B.2 Source Code of Post in Django: controllers, views and templates

Listing B.12: Python post model

```

from django.db import models
from django.contrib.auth.models import User
from django.core.urlresolvers import reverse
from django.template.defaultfilters import slugify

from django.dispatch import receiver
from django.db.models.signals import pre_delete

```

```

# Create your models here.

class Post(models.Model):

    blog = models.ForeignKey(Blog, editable=False)
    title = models.CharField(max_length=100)
    slug = models.CharField(max_length=200, editable=False, unique=True)
    body = models.TextField()
    created_at = models.DateTimeField(auto_now_add = True)

    def get_absolute_url(self):
        return reverse('post',
                        kwargs={'username': self.blog.user.username,
                                'slug': self.slug})

    def save(self):
        if not self.id:
            self.slug = slugify(self.title)
        return super(Post, self).save()

    class Meta:
        ordering = ["-created_at"]

```

Listing B.13: Python post controller

```

from django.contrib.auth.decorators import login_required
from django.utils.decorators import method_decorator
from django.core.urlresolvers import reverse, reverse_lazy

from blog.models import Blog, Post, Comment
from django.contrib.auth.models import User
from django.views.generic.edit import CreateView, UpdateView, DeleteView
from django.views.generic.detail import DetailView
from django.views.generic.edit import FormMixin

from django.contrib import messages

from django.core.exceptions import PermissionDenied

from django.forms import models as model_forms

from blog.forms import RegisteredUserCreateCommentForm
# Create your views here.
class PostDetail(DetailView, FormMixin):
    """
    Post view with a form to comment on a post
    """
    model = Post
    form_class = model_forms.modelform_factory(Comment)

    def get_form_class(self):
        """
        Returns the form class to use in this view
        """
        if self.request.user.is_authenticated():

```

```

        return RegisteredUserCreateCommentForm
    else:
        return self.form_class

def get_success_url(self):
    return reverse('post',
                   kwargs={'username': self.object.blog.user.username,
                           'slug': self.object.slug})

def get_context_data(self, **kwargs):
    context = super(PostDetail, self).get_context_data(**kwargs)
    form_class = self.get_form_class()
    context['form'] = self.get_form(form_class)
    return context

def post(self, request, *args, **kwargs):
    self.object = self.get_object()
    form_class = self.get_form_class()
    form = self.get_form(form_class)
    if form.is_valid():
        return self.form_valid(form)
    else:
        return self.form_invalid(form)

def form_valid(self, form):
    """
    Relates the comment with the post.
    """
    form.instance.post = self.object

    if self.request.user.is_authenticated():
        form.instance.user = self.request.user.username
        form.instance.website = self.request.user.userprofile.website
        form.instance.email = self.request.user.email
        form.instance.reg_user = self.request.user

    form.save()

    messages.success(self.request, "Your comment has been sucessfully
                               created")

    return super(PostDetail, self).form_valid(form)

def get_object(self):
    """
    Returns the post asked
    """
    slug = self.kwargs.get(self.slug_url_kwarg, None)
    username = self.kwargs.get('username', None)

    return User.objects.get(username=username).blog.post_set.get(slug=
        slug)

class PostCreate(CreateView):
    model = Post

```

```

@method_decorator(login_required)
def dispatch(self, *args, **kwargs):
    return super(PostCreate, self).dispatch(*args, **kwargs)

def form_valid(self, form):
    """
    This method is called when valid form data has been POSTed.
    The post has to be linked to the user's blog
    """

    form.instance.blog = self.request.user.blog

    return super(PostCreate, self).form_valid(form)

class PostUpdate(UpdateView):

    def get_object(self):
        """
        Returns the user's post if the user is the owner
        """
        slug = self.kwargs.get(self.slug_url_kwarg, None)
        username = self.kwargs.get('username', None)

        user = User.objects.get(username=username)

        if user == self.request.user:
            return User.objects.get(username=username).blog.post_set.get(
                slug=slug)
        else:
            raise PermissionDenied

    @method_decorator(login_required)
    def dispatch(self, *args, **kwargs):
        return super(PostUpdate, self).dispatch(*args, **kwargs)

class PostDelete>DeleteView):
    success_url = '/'

    def get_object(self):
        """
        Return the user's post if the user is the owner. This is Code
        reptition. It is the same in PostUpdate, consider refactoring.
        """
        slug = self.kwargs.get(self.slug_url_kwarg, None)
        username = self.kwargs.get('username', None)

        user = User.objects.get(username=username)

        if user == self.request.user:
            return User.objects.get(username=username).blog.post_set.get(
                slug=slug)
        else:
            raise PermissionDenied

    def delete(self, request, *args, **kwargs):
        """

```

```

        Insert the message and call the super method
        """
        messages.success(self.request, "Your post has been sucessfully
            deleted")
        return super(PostDelete, self).delete(request, *args, **kwargs)

    @method_decorator(login_required)
    def dispatch(self, *args, **kwargs):
        return super(PostDelete, self).dispatch(*args, **kwargs)

```

Listing B.14: Python post view

```

{% extends "base.html" %}

{% block content %}

    {% load blog_extras %}

    {% show_post post %}

    <p> You can leave a comment</p>
    <form method="post" action="{% url 'post' post.blog.user post.slug %}"
    >
        {% csrf_token %}

        {{ form.as_p }}

        <input type="submit" value="Create a comment" />
    </form>
    <h3> Comments </h3>
    {% for comment in post.comment_set.all %}
        {% if comment.reg_user %}
            <h4> {{ comment.reg_user.username }} </h4>
            <h4> {{ comment.reg_user.userprofile.website }} </h4>
            <h4> {{ comment.reg_user.email }} </h4>
        {% else %}
            <h4> {{ comment.user }} </h4>
            <h4> {{ comment.website }} </h4>
            <h4> {{ comment.email }} </h4>
        {% endif %}
        <p> {{ comment.body }} </p>
        {% if user.is_authenticated %}
            <form method="post" action="{% url 'comment_delete' post.blog.
                user post.slug comment.id %}" >
                {% csrf_token %}
                <input type="submit" value="Delete Comment" >
            </form>
        {% endif %}
    {% endfor %}
{% endblock %}

```

B.3 Source code of Post in CodeIgniter: models, controllers and views

Listing B.15: PHP post model

```
<?php if (!defined('BASEPATH')) exit('No direct script access allowed');

/**
 * Post
 *
 * This model represents post data. It operates the following tables:
 * - post data
 */
class MPost extends CI_Model
{
    public $rules = array(
        array(
            'field' => 'title',
            'label' => 'title',
            'rules' => 'trim|required|xss_clean|max_length[255]'
        ),
        array(
            'field' => 'body',
            'label' => 'Body',
            'rules' => 'trim|required|xss_clean'
        )
    );

    private $table_name = 'post'; // blog table name

    function __construct()
    {
        parent::__construct();
        $this->load->database();
    }
    /**
     * Create new blog record
     *
     * @param $data array
     * @return array
     */
    function create($data)
    {
        $data['permalink'] = url_title($data['title']);
        if ($this->db->insert($this->table_name, $data)) {
            return True;
        }
        return False;
    }

    function get_post_by_id($id)
    {
        $query = $this->db->get_where(
            $this->table_name,
            array('id' => $id)
        );
        if ($query->num_rows() == 1) return $query->row_array();
    }
}
```

```

        return NULL;
    }

    function get_post($permalink)
    {
        $query = $this->db->get_where(
            $this->table_name,
            array('permalink' => $permalink)
        );
        if ($query->num_rows() == 1) return $query->row_array();
        return NULL;
    }

    function total_posts(){
        return $this->db->count_all($this->table_name);
    }

    function get_posts($per_page, $offset=0)
    {
        $this->db->limit($per_page, $offset);
        $this->db->order_by('posted_on', 'desc');
        $query = $this->db->get($this->table_name);

        if ($query->num_rows() > 0) return $query->result_array();
        return NULL;
    }

    function get_posts_by_blog_id($blog_id)
    {
        $this->db->order_by('posted_on', 'desc');
        $query = $this->db->get_where(
            $this->table_name,
            array('blog_id' => $blog_id)
        );
        if ($query->num_rows() > 0) return $query->result_array();
        return NULL;
    }

    function get_post_author_by_blog_id($blog_id)
    {
        $this->load->model('mblog');

        $blog = $this->mblog->get_blog_by_id($blog_id);
        $user = $this->users->get_user_by_id($blog['user_id']);

        return $user['username'];
    }

    function update($id, $data){
        $this->db->where('id', $id);
        return $this->db->update($this->table_name, $data);
    }

    function delete($id)
    {

```



```

        $this->db->where('id', $id);
        $this->db->delete($this->table_name);
        if ($this->db->affected_rows() > 0) {
            return TRUE;
        }
        return FALSE;
    }

    function get_total_posts_which_contains($text)
    {
        $this->db->or_like('title', $text);
        $this->db->or_like('body', $text);
        $this->db->from($this->table_name);
        return $this->db->count_all_results();
    }

    function get_posts_which_contains($text,$per_page,$offset=0)
    {
        $this->db->limit($per_page, $offset);
        $this->db->order_by('posted_on','desc');

        $this->db->or_like('title', $text);
        $this->db->or_like('body', $text);

        $query = $this->db->get($this->table_name);

        if ($query->num_rows() > 0) return $query->result_array();
        return NULL;
    }

    function delete_posts_from_blog_id($blog_id)
    {
        $this->db->where('blog_id', $blog_id);
        $this->db->delete($this->table_name);
        if ($this->db->affected_rows() > 0) {
            return TRUE;
        }

        return FALSE;
    }

}

/* End of file users.php */
/* Location: ./application/models/auth/users.php */

```

Listing B.16: PHP post controller

```

<?php if (!defined('BASEPATH')) exit('No direct script access allowed');

class Post extends CI_Controller
{

```

```

function __construct()
{
    parent::__construct();

    $this->load->helper(array('form', 'url', 'array', 'html', 'post'));
    $this->load->library('form_validation');
    $this->load->library('tank_auth');
        $this->load->model('mpost');
    $this->load->model('mblog');
    $this->load->model('mcomment');
}

function create()
{
    $this->tank_auth->logged_in_or_redirect();
    $data['title'] = 'Create a Post';

    $this->form_validation->set_rules($this->mpost->rules); //set
        rules

    if ($this->form_validation->run())
    {

        $user_id = $this->tank_auth->get_user_id();//always to the crrent
            user
        $blog = $this->mblog->get_blog_by_user_id($user_id); //get
            current blog

        $post = elements(array('title', 'body'), $this->input->post());
        $post['blog_id'] = $blog['id'];

        if ($this->mpost->create($post))
        {
            $this->session->set_flashdata('message', 'Post successfully
                created');
            //TODO REDIRECT TO POST
            redirect(''. $this->tank_auth->get_username());

        }else{ //some error

        }

    }

    $this->load->view('templates/header', $data);
    $this->load->view('post/create', $data);
}

function index()
{
    $data['title'] = 'My blog';

    $this->load->library('pagination');
    $config['base_url'] = 'http://localhost/index.php?' ; //had to do this,
        no other way
    $config['per_page'] = 2;
    $config['page_query_string'] = TRUE;

```

```

$offset = $this->input->get('per_page');

$this->form_validation->set_rules(array(
    array(
        'field' => 'search',
        'label' => 'search',
        'rules' => 'trim|xss_clean|max_length[1000]'
    )));

if ($search = $this->input->get('search'))
{
    $data['search'] = $search;
    $data['posts'] = $this->mpost->get_posts_which_contains($search,
        $config['per_page'], $offset);
    $config['total_rows'] = $this->mpost->get_total_posts_which_contains(
        $search);
    $config['base_url'] .= 'search='.$search;
} else {
    $data['posts'] = $this->mpost->get_posts($config['per_page'], $offset);
    $config['total_rows'] = $this->mpost->total_posts();
}

$this->pagination->initialize($config);

$data['pagination'] = $this->pagination->create_links();

// In case user is logged in , his posts must be loaded
if ($this->tank_auth->is_logged_in()) {
    $this->load->model('mblog');
    $user = $this->tank_auth->current_user();
    $blog = $this->mblog->get_blog_by_user_id($user['id']);

    $data['his_posts'] = $this->mpost->get_posts_by_blog_id($blog['id']);
}

$this->load->view('templates/header', $data);
$this->load->view('post/index', $data);
}

function show($permalink)
{
    if ( $post = $this->mpost->get_post($permalink) )
    {
        $data['post'] = elements(array('title', 'body', 'posted_on', 'permalink', 'blog_id', 'id'), $post);
        $data['comments'] =
            $this->mcomment->get_comments_by_post_id($post['id']);

        $comment = array();
        if ($this->tank_auth->is_logged_in() AND
            $_SERVER['REQUEST_METHOD'] == "POST")

```

```

{
    $user = $this->tank_auth->current_user();
    $userprofile = $this->tank_auth->current_profile();
    $comment['user_id'] = $user['id'];
    $_POST['email'] = $user['email'];
    if (isset($userprofile['website'])) {
        $_POST['website'] = $userprofile['website'];
    }
    $_POST['username'] = $user['username'];
}

$this->form_validation->set_rules($this->mcomment->rules); //set
rules

if ($this->form_validation->run()) {

    $comment += elements(
        array('username', 'website', 'email', 'body'),
        $this->input->post()
    );
    $comment['post_id'] = $post['id'];

    if ($this->mcomment->create($comment))
    {
        $this->session->set_flashdata('message', 'Comment successfully
        created');

        redirect($this->uri->uri_string());

    }else{ //some error

    }

}

    $this->load->view('templates/header', $data);
    $this->load->view('post/show', $data);
}else{//does not exists
    show_404();
}

}

function update($permalink)
{

    $this->tank_auth->logged_in_or_redirect();

    if ( $post = $this->mcomment->get_post($permalink))
    {
        $user = $this->tank_auth->current_user();
        if ( $this->mcomment->get_post_author_by_blog_id($post['blog_id'])
            != $user['username'] )
            show_error( "Not authorized" ); //not authorized

        $data = elements(array('title', 'body', 'posted_on', 'permalink'),
            $post);
    }
}

```

```

        $this->form_validation->set_rules($this->mpost->rules);    //set
        rules

    if ($this->form_validation->run())
    {
        $new_values = elements(
        array('title', 'body'),
        $this->input->post());

        if ($this->mpost->update($post['id'],$new_values))
        {
            $this->session->set_flashdata('message', 'Post
            successfully updated');
            redirect('/post/update/'.$permalink);

        }else{ //some error
        }

    }

    $this->load->view('templates/header', $data);
    $this->load->view('post/update', $data);
}else{//does not exists
    show_404();
}

}

function delete($permalink)
{
    $this->tank_auth->logged_in_or_redirect();
    if ( $post = $this->mpost->get_post($permalink))
    {
        $user = $this->tank_auth->current_user();
        if ( $this->mpost->get_post_author_by_blog_id($post['blog_id'])
        != $user['username'] )
            show_error( "Not authorized" ); //not authorized

        if ($this->mpost->delete($post['id']))
        {
            $this->session->set_flashdata('message', 'Post successfully
            deleted');
            redirect('/'.$user['username']);
        }else{
            $this->session->set_flashdata('message', 'There was some
            error trying to delete it, please contact an
            administrator');
            redirect('/');
        }
    }else{
        show_404();
    }
}
}

```

```
}
```

Listing B.17: PHP post view

```
<?php

$username = array(
    'name' => 'username',
    'id'   => 'username',
    'value' => set_value('username'),
    'maxlength' => 255,
    'size'  => 30
);

$website = array(
    'name' => 'website',
    'id'   => 'website',
    'value' => set_value('website'),
    'size'  => 30
);

$email = array(
    'name' => 'email',
    'id'   => 'email',
    'value' => set_value('email'),
    'size'  => 30
);

$body = array(
    'name' => 'body',
    'id'   => 'body',
    'value' => set_value('body'),
    'rows'  => 10,
    'cols'  => 39,
);

?>

<? echo show_post($post) ?>

<?php echo form_open($this->uri->uri_string()); ?>
<h3> New comment </h3>
<table>
<? if (!$this->tank_auth->is_logged_in()) { ?>
    <tr>
        <td><?php echo form_label('username', $username['id']); ?></td>
        <td><?php echo form_input($username); ?></td>
        <td style="color: red;"><?php echo form_error($username['name']); ?><?php echo isset($errors[$username['name']])?$errors[$username['name']]:''; ?></td>
    </tr>

    <tr>
        <td><?php echo form_label('website', $website['id']); ?></td>
```

```

        <td><?php echo form_input($website); ?></td>
        <td style="color: red;"><?php echo form_error($website['name']); ?><?php echo isset($errors[$website['name']])?$errors[$website['name']]:''; ?></td>
    </tr>

    <tr>
        <td><?php echo form_label('email', $email['id']); ?></td>
        <td><?php echo form_input($email); ?></td>
        <td style="color: red;"><?php echo form_error($email['name']); ?><?php echo isset($errors[$email['name']])?$errors[$email['name']]:''; ?></td>
    </tr>
<? }?>
<tr>
    <td><?php echo form_label('body', $body['id']); ?></td>
    <td><?php echo form_textarea($body); ?></td>
    <td style="color: red;"><?php echo form_error($body['name']); ?><?php echo isset($errors[$body['name']])?$errors[$body['name']]:''; ?></td>
    <td><?php echo form_submit('create comment', 'Create comment'); ?>
    <td><?php echo form_close(); ?>
</tr>
</table>

<?php echo form_submit('create comment', 'Create comment'); ?>
<?php echo form_close(); ?>

<? if (isset($comments)) foreach ($comments as $comment){ ?>

    <? if (isset($comment['user_id'])) { ?>
        <? $comment_user =
            $this->users->get_user_by_id($comment['user_id']);
            $comment_userprofile =
            $this->users->get_user_profile($comment['user_id']);
        ?>
        <p> user : <b> <?=$comment_user['username'] ?> </b> </p>
        <p> website : <b> <?=$comment_userprofile['website'] ?> </b> </p>
        <p>email : <b> <?=$comment_user['email'] ?> </b> </p>

    <? }else{ ?>
        <p>user : <b> <?=$comment['username'] ?> </b> </p>
        <p>website : <b> <?=$comment['website'] ?> </b> </p>
        <p>email : <b> <?=$comment['email'] ?> </b> </p>
    <? } ?>

    <p> <?=$comment['body'] ?> </p>
    <? $current_user = $this->tank_auth->current_user(); ?>
    <? $author = $this->mpost->get_post_author_by_blog_id($post['blog_id']);
    ?>
    <? if ( $current_user['username'] == $author ) { //show button options
        only to the owners of the post ?>

        <?=$form_open('comment/delete/'.$comment['id']); ?>
        <?=$form_submit('delete', 'Delete comment'); ?>
    }
}

```

```

    <?= form_close(); ?>
  <? } ?>
<? } ?>

```

B.4 Search posts implementation in different web frameworks

Listing B.18: Grails search post implementation

```

#Controller

class PostController {
  .....
  def list(Integer max) {
    def postList
    def postTotal
    def hisPostList
    hisPostList=[]
    params.max = Math.min(max ?: 3, 6)

    if ( params.searchquery == null) {
      postList = Post.list(params)
      postTotal = Post.count()
    }
    else{
      postList = Post.findAllByTitleLikeOrBodyLike('%'+params.searchquery+'
        %', '%'+params.searchquery+'%', params)
      postTotal = Post.findAllByTitleLikeOrBodyLike('%'+params.searchquery+'
        %', '%'+params.searchquery+'%').size()
    }

    if (SecurityUtils.subject.authenticated){
      def user = currentUser()
      hisPostList = user.blog.posts
    }

    [postList: postList, postTotal: postTotal, params: params, hisPostList:
      hisPostList ]
  }
  .....
}

# Template file

<div id="list-post" class="content scaffold-list" role="main">

  <g:if test="${flash.message}">
    <div class="message" role="status">${flash.message}</div>
  </g:if>

  <g:render template="/blog/post" var="post" collection="${postList}" />

```



```

<div class="pagination">
    <g:paginate total="{{postTotal}}" action="list" params="{{params}}"/>
</div>
</div>

```

Listing B.19: Django search post implementation

```

#--- DJANGO CONTROLLER----- CALLED VIEWS IN DJANGO VOCABULARY --- #
# Create your views here.
# Following class is using the template blog/templates/blog/post_list.html
class PostList(ListView):
    paginate_by = 2 #this variable is used for pagination
    def get_queryset(self):

        if 'search' in self.request.GET:
            objects = Post.objects.filter(
                Q(title__icontains=self.request.GET['search']) | Q(
                    body__icontains=self.request.GET['search'])
            )
        else:
            objects = Post.objects.all()

        return objects

#Template

{% load blog_extras %}
{% for post in page_obj %}
    {% show_post post %}
{% endfor %}

<div class="pagination">
    <span class="step-links">
        {% if page_obj.has_previous %}
            <a href="?page={{ page_obj.previous_page_number }}">previous</a>
        >
        {% endif %}

        <span class="current">
            Page {{ page_obj.number }} of {{ page_obj.paginator.num_pages
            }}.
        </span>

        {% if page_obj.has_next %}
            <a href="?page={{ page_obj.next_page_number }}">next</a>
        {% endif %}
    </span>
</div>

```


Bibliography

- [1] J. I. Fernández-Villamor, L. Díaz, and C. A. Iglesias, “A Comparison Model for Agile Web Frameworks,” 2008.
- [2] M. Raible. (2012). Comparing JVM frameworks. Accessed: 2012-09-30, [Online]. Available: <http://www.parleys.com/#st=5&id=3084>.
- [3] P. Thomas. (Sep. 2009). Perfbench. Accessed: 2013-01-14, [Online]. Available: <http://ptrthomas.wordpress.com/2009/09/14/perfbench-update-tapestry-5-and-grails/>.
- [4] Stijn Van den Enden, Ward Vijfeijken, Guy Veraghtert. (2011). World Wide Wait. Accessed: 2013-02-3, [Online]. Available: <http://www.parleys.com/#id=2942&st=5>.
- [5] K. Wähner. (2010). Categorization and Comparison of Popular Web Frameworks in the Java / JVM Environment. Accessed: 2013-01-02, [Online]. Available: <http://java.dzone.com/articles/categorization-web-frameworks>.
- [6] W. W. H. Tony C Shan, “Taxonomy of Java Web Application Frameworks,” 2006.
- [7] W3Techs. (2012). Technology usage statistics. Accessed: 2012-12-18, [Online]. Available: <http://w3techs.com/>.
- [8] BuiltWith. (2013). Accessed: 2013-01-8, [Online]. Available: <http://builtwith.com/>.
- [9] S. Davis. (2011). The Great Web Framework Shootout. Accessed: 2013-01-17, [Online]. Available: <http://blog.curiasolutions.com/the-great-web-framework-shootout/>.
- [10] P. M. Jones. (2011). Benchmarking Web Applications and Frameworks. Accessed: 2013-01-08, [Online]. Available: <http://www.slideshare.net/pmjones88/benchmarking-applications-and-frameworks-9759357>.
- [11] J. Dev. (2013). Rails, Wicket, Grails, Play, Tapestry, Lift, JSP, Context. Accessed: 2013-01-13, [Online]. Available: <http://www.jtict.com/blog/rails-wicket-grails-play-lift-jsp/>.
- [12] Ruby on Rails Community. (2013). Ruby on Rails guide. Accessed: 2013-09-14, [Online]. Available: <http://guides.rubyonrails.org/>.
- [13] Django. (2013 v1.5). The Django Book. Accessed: 2013-09-25, [Online]. Available: <http://djangoproject.com>.
- [14] Ellislab. (2013). CodeIgniter User Guide. Accessed: 2013-07-24, [Online]. Available: <http://ellislab.com/codeigniter/user-guide/toc.html>.
- [15] Alioth.debian.org. (2013). The Computer Language Benchmarks Game. Accessed: 2013-02-08, [Online]. Available: <http://benchmarksgame.alioth.debian.org/>.
- [16] M. Hartl. (2013). Ruby on Rails Tutorial. Accessed: 2013-04-04, [Online]. Available: <http://ruby.railstutorial.org/ruby-on-rails-tutorial-book?version=3.2>.

- [17] Stackoverflow. (2013). How should I choose an authentication library for CodeIgniter? Accessed: 2013-05-14, [Online]. Available: <http://stackoverflow.com/questions/346980/how-should-i-choose-an-authentication-library-for-codeigniter>.

List of Figures

3.1	Comparison model. Source [1]	6
3.2	Framework evaluation results. Source [1]	6
3.3	Comparison frameworks matrix. Source [2]	8
3.4	Website types. Source [5]	8
3.5	Frameworks complexity and website types. Source [5]	9
8.1	Diffusion diagram.	21
8.2	Header logo of the blog for the project diffusion.	22
8.3	Square logo of the blog for the project diffusion.	22
9.1	Usage of server-side programming languages for websites. Source [7]	26
9.2	Usage of CMS for websites. Source [7]	26
9.3	Usage of server-side programming languages for websites not using CMS.	27
9.4	Server-side programming languages market position report. Source [7]	27
9.5	Historical trends in the usage of server-side programming languages for websites. Source [7]	28
9.6	Historical trends in the usage of CMS for websites. Source [7] on 25 Jan 2013	28
9.7	Web frameworks usage. Adapted from [8] consulted on 1 March 2013	29
9.8	Web frameworks growth. Adapted from [8] consulted on 1 March 2013	29
9.9	Mean page load time with java web frameworks . Source [4]	31
9.10	Cost of scale with different java web frameworks. Source [4]	31
9.11	Request per second with different frameworks. Source [4]	32
9.12	Relative request per second with different frameworks. Source [10]	33
9.13	Relative request per second with different frameworks. Source [11]	34
9.14	Main frameworks map.	35
9.15	Main frameworks table. Modified from Wikipedia	36
9.16	Matt Raible's Matrix weighed according to priorities	38
10.1	Exercises practicing component	41
10.2	Use cases - User	45
10.3	Use cases - Visitor	46
10.4	Data Model	46
10.5	Visitor's view of the front page	47
10.6	Visitor's view of a blog	48
10.7	Visitor's view of a post	49
10.8	Visitor's view of a register screen	50
10.9	Visitor's view of login screen	51
10.10	User's view of creating a blog	52
10.11	User's view of the front page	53
10.12	User's view of a blog	54

10.13	User's view of a post	55
10.14	User's view for creating a new post	56
10.15	User's view for the setting screen	57
11.1	Data flow in CodeIgniter. Source [14]	94
12.1	Performance comparison of programming languages. Source [15]	105
12.2	Python vs Ruby. Source [15]	105
12.3	Developing times with Rails, Grails, Django and CodeIgniter	108
14.1	Visits to the blog.	120

List of Tables

3.1	Addition of all the grades in the results of A Comparison Model for Agile Web Frameworks	6
7.1	The project plan	17
11.1	Hours needed for developing Ruby programming exercises	64
11.2	Developing hours for the website with Rails	67
11.3	Source code files of MyBlog on Ruby on Rails with counted lines	68
11.4	Hours needed for developing Groovy programming exercises	72
11.5	Developing hours for the website with Grails	74
11.6	Files of source code of MyBlog on Grails with counted lines	75
11.7	Hours needed for developing Python programming exercises	80
11.8	Developing hours for the website with Django	85
11.9	Source code files of MyBlog on Django with counted lines	86
11.10	Hours needed for developing PHP programming exercises	92
11.11	Developing hours for the website with CodeIgniter	97
11.12	Source code files of MyBlog on CodeIgniter with counted lines	98
12.1	Hours required to develop programming exercises with programming languages	101
12.2	Source code lines in programming exercises	102
12.3	Comparing readability of languages	104
12.4	Comparing features for readability in programming languages	104
12.5	Developing times with Rails, Grails, Django and CodeIgniter	107
12.6	Comparing steps in development	110
12.7	Comparison of lines of code of implementations with Rails, Grails, Django and CodeIgniter	113
14.1	Top 10 referrals in the blog	120

Listings

11.1	hello world example	62
11.2	Class example	62
11.3	An iterator example	64
11.4	Iterator with arguments example	64
11.5	Model example. Source [12]	65
11.6	Controller example. Source [12]	66
11.7	View example. Source [12]	66
11.8	Hello world example in Groovy	70
11.9	Class example in Groovy	70
11.10	List example in Groovy	71
11.11	Map example in Groovy	71
11.12	Closure example in Groovy	71
11.13	Closure example in Groovy	71
11.14	Iteration over a Map in Groovy	71
11.15	Grails model example: A book	73
11.16	Grails controller example: A book	73
11.17	Grails view example: A list of books	74
11.18	Hello world example in Python	78
11.19	Class example in Python	79
11.20	List comprehension example in Python	79
11.21	Command for creating a project in Django	82
11.22	Basic project skeleton in Django	82
11.23	Command for creating an app in Django	82
11.24	Basic app skeleton in Django	82
11.25	A model.py file in Django	83
11.26	A views.py file in Django	83
11.27	A template file in Django	83
11.28	A url file in Django	84
11.29	Class-based view example in Django	84
11.30	Syntax and semantic example in PHP	90
11.31	Foreach syntaxes in PHP	91
11.32	Example of routing rule in CodeIgniter	95
11.33	Example of controller in CodeIgniter	95
11.34	Example of model class in CodeIgniter	95
11.35	Example of view template in CodeIgniter	96
A.1	The Strings, Files and Regular Expressions exercise in Ruby	123
A.2	The Numbers Set exercise in Ruby	124
A.3	The Composite Pattern exercise in Ruby	126
A.4	Source code of The Strings, Files and Regular Expressions exercise in Groovy	128

A.5	Source code of The Numbers Set exercise in Groovy	128
A.6	Source code of The Composite Pattern exercise in Groovy	130
A.7	The Strings, Files and Regular Expressions exercise in Python	132
A.8	The Numbers Set exercise in Python	133
A.9	The Composite Pattern exercise in Python	135
A.10	The Strings, Files and Regular Expressions exercise in PHP	136
A.11	The Numbers Set exercise in PHP	137
A.12	The Composite Pattern exercise in PHP	140
B.1	Generated Rails post model	143
B.2	Adapted Rails post model	143
B.3	Generated Rails post controller	143
B.4	Adapted Rails post controller	145
B.5	Generated Rails show template	146
B.6	Adapted Rails show template	147
B.7	Adapted Grails post model	147
B.8	Generated Grails post controller	148
B.9	Adapted Grails post controller	150
B.10	Generated Grails post show	152
B.11	Adapted Grails post show	154
B.12	Python post model	156
B.13	Python post controller	157
B.14	Python post view	160
B.15	PHP post model	161
B.16	PHP post controller	163
B.17	PHP post view	168
B.18	Grails search post implementation	170
B.19	Django search post implementation	171