Comparing Performance of Plain PHP and Four of Its Popular Frameworks

Author: Jone Samra
Supervisor: Johan Hagelbäck
Examiner: Sabri Pllana
Semester: VT 2015
Subject: Computer Science
Abstract
The objective of this study is to evaluate performance of four popular PHP frameworks Laravel, Symfony, CodeIgniter and Phalcon together with the plain PHP. The decision of making this study was based on the fact that there is a lack of comparison tests between the most popular PHP frameworks. Visiting the official websites of these frameworks, the first thing to notice is the slogans that have been made by the core teams. The majority of these slogans contain quality attributes like speed and high performance. As a developer looking for performance in your next project, choosing the right PHP framework by these slogans is not that easy. Therefore, the performance of these frameworks is put to the test. And to do that, three experiments are conducted in which five functionally equivalent PHP applications are developed and used as targets. One version of these applications represents the plain PHP and the other four represent the four mentioned frameworks. The experiments are conducted in two sessions. The first session deals with the execution time and the stack trace measurements while the second one is covering the measurement of the memory usage consumption. The result outcome of these experiments has been analyzed and interpreted in order to expose the performance of the targeted frameworks. The experiment results prove that the targeted frameworks perform differently compared with each other and the PHP stack. Plain PHP and Phalcon are performing well while the other three frameworks have both mediocre and low performance.

Keywords: PHP, framework, Symfony, Laravel, CodeIgniter, Phalcon, performance, comparison.
Preface
I want to thank the Linnaeus University and its Department of Computer Science for the knowledge that I have been illuminated with. Another thanks goes to the Gang of Four; my lovely wife and children for the good times. Finally a tribute dedicated to my mother, brothers and my deceased father.
Glossary

**Completely Automated Public Turing test to tell Computers and Humans Apart (CAPTCHA)**
It is a recognition technique that is used to tell if the computer is interacting with a human or just another computer.

**Content Management System (CMS)**
A computer application with features for publishing and modifying content by one or several users of the application.

**Convention Over configuration (CoC)**
It is a software design paradigm with the purpose of reducing or eliminating the configuration phase during a software development project.

**Create, Read, Update and Delete (CRUD)**
In computer programming, CRUD are the operations executed against the persistence storage in order to read, add and modify data.

**Cross Site Request Forgery (CSRF)**
It is a common security attack performed on web applications. The attacker executes malicious requests to the web application in order to expose or manipulate data in the application.

**Cascading Style Sheets (CSS)**
It is a language that is used to style and format the HTML-markup.

**Do not Repeat Yourself (DRY)**
It is a principle in software engineering, which is about the inhibitory of repetition in a software development project.

**Keep it Stupid, Simple (KISS)**
Is another principle in software engineering, which ideates the simplicity in a software development process.

**Model-View-Controller (MVC)**
It is a widely used design pattern in web development. The aim of the pattern is to separate the business logic (model) from the presentation (View) and handle the execution flow in the controller.

**Object Relational Mapping (ORM)**
It is an abstraction technique, in which the data that is used in a computer program is mapped to its persistence storage.

**Representational State Transfer (RESTful)**
It is a software architectural style, often used in web development. By implementing the style, the data is handled as a resource between the client and server. To read, write, update and delete the resource, the HTTP’s GET, POST, PUT and DELETE requests are used.

**Cross site scripting (XSS)**
It is a security attack performed on a web application. The aim of XSS is to force the user to execute malicious code without her knowledge.
List of Figures

Figure 2.1: The Xdebug configuration section inside php.ini that has been used in the study.................................................................8
Figure 2.2: Example of a create-entry view in the targeted applications.....11
Figure 2.3: Example of a view-entry view in the targeted applications. .....11
Figure 2.4: Example of an update-entry view in the targeted applications..12
Figure 2.5: Example of a rendered index view when an entry is deleted. ...12
Figure 2.6: Illustration of the usage of Webgrind tool.............................13
List of Tables

Table 3.1: Time execution measurement results..................................................14
Table 3.2: Memory usage measurement results..................................................15
Table 3.3: The number of called functions in each CRUD action.......................16
Contents

1. Introduction
   1.1 The language of PHP
   1.2 The rise of PHP frameworks
      1.2.1 Symfony
      1.2.2 CodeIgniter
      1.2.3 Laravel
      1.2.4 Phalcon
   1.3 Previous research
   1.4 Problem definition
   1.5 Purpose and research questions / hypothesis
   1.6 Limitations
   1.7 Target group
   1.8 Overview of the report

2 Method
   2.1 Hardware specification
   2.2 Software specification
   2.3 Five versions of the web application
      2.3.1 Plain PHP version
      2.3.2 Laravel version
      2.3.3 Symfony version
      2.3.4 CodeIgniter version
      2.3.5 Phalcon version
   2.4 Functional equivalence in the versions
   2.5 Measuring the performance
      2.5.1 Execution time comparison experiment
      2.5.2 Measuring the stack trace
      2.5.3 Measuring the memory usage
   2.6 The usage of Xdebug and Webgrind
   2.7 Reliability

3. Result and analysis
   3.1 The result of the execution time experiment
   3.2 The result of memory usage experiment
   3.3 The results of the stack trace experiment

4. Discussion
   4.1 Problem solving/results

5. Conclusion
5.2 Further research .......................................................... 18

References ............................................................................ 20
1. Introduction

PHP: Hypertext Preprocessor is well known scripting language often associated with web development even though it has other areas of usages. According to w3techs.com PHP is the most common used scripting languages on the Internet with 82% coverage [1]. Many frameworks based on PHP popped up during the last decade. Frameworks like CodeIgniter, Symfony, Phalcon and Laravel are widely used and according to sitepoint.com they are four of the most promising frameworks in 2014 [2].

The first thing to recognize when visiting the webpages of these frameworks is the promises, which the people behind will guarantee you. When visiting Laravel’s website the first thing you see are slogans like beautiful code, rapidity and speed [3]. Phalcon’s Team claims that their framework is the fastest [4]. Symfony’s website title says “High performance PHP framework for web development” [5]. CodeIgniter’s team states that their framework is “powerful with a very small print” [6].

A developer aiming to develop a project where PHP is required as the development language and high performance is one of the highly prioritized requirements in the requirements specification, how would the decisions be made on whether choosing the plain PHP stack or a PHP framework in the development? In the case of using the framework, which one of these above mentioned would be suitable bearing in mind the performance requirements? To answer these questions this study has been conducted in which five functionally equivalent blogging web applications have been developed and subjected to an experiment in which the performance of each of the mentioned frameworks is measured and evaluated.

The first web application will be developed in plain PHP using the MVC design pattern [7]. The other four are developed in the PHP frameworks Laravel, Phalcon, Symfony and CodeIgniter. After implementing the applications, performance metrics are measured on all these versions of the web application. These measurements will consist of (1) Execution time of CRUD-functionality [8][9], (2) Memory usage for each of them [10], (3) Tracing the number of the called functions in each CRUD-action [11]. The results from the experiments are analyzed and interpreted in order to become the basis of the decisions and conclusions of this study.

1.1 The language of PHP

PHP has been the dominating development language for creating dynamic websites in the last decades. It started out in 1995 and was originally called
Personal Home Page Tools (PHP Tools) by its creator Rasmus Lerdorf [12]. One of the main goals with PHP at that time was to make it easier to render dynamic content on a webpage. In the year 1998, when the PHP 3 was released, the language was renamed to PHP: Hypertext Preprocesor due to the fact that the language got more sophisticated and it is not only for hobbyists anymore [13]. Today, Content Management Systems (CMSs) like WordPress, Joomla, Drupal or services like Facebook are all developed using the PHP language [14].

One of the main reasons for PHP popularity is because it is open source. Another reason would be the easy deployment and integration of PHP on any operating system and the fact that PHP will run relatively fast on all operating systems. Furthermore, the easy and C like syntax would attract many developers working with web development and also the huge built-in and external development libraries available for use by third party developers [15].

And to not forget to mention the drawbacks, one thing anybody would recognize when reviewing the PHP documentation is the mixing of programming paradigms in the development libraries, some of the libraries are written using the procedural concept, others use the object-oriented design [16]. This behavior can also be found when reviewing source code of any randomly selected PHP application not developed using a framework.

Issues like security has been a subject of concern in the PHP community since the beginning. Generally, the security vulnerabilities in web applications of any kind are still a big issue [17]. PHP provides the developers with many built-in libraries containing concrete implementations regarding the validation and sanitization of any kinds of input [18][19]. These are the most easy security issues to handle, but when dealing with security vulnerabilities like Cross-site Scripting (XSS) [20], Cross-site request forgery (CSRF) [20], Session and cookie hijacking [21], PHP does not provide any concrete fixes to deal with these kinds of attacks. The developer has to make her own implementation or use third party libraries to deal with these kinds of issues.

1.2 The rise of PHP frameworks

Generally, a PHP framework is an implementation layer upon PHP and it encapsulates some architectural patterns, principles and other implementations solving different problems. Example of architectural designs used is the Model-View-Controller (MVC) for which the purpose is to separate a webpage (View) from the business logic (Model). Another common architectural pattern is the Active Record pattern, which is simply an abstraction technique in which implemented objects are mapped to their source of persistence [22].
Common software development principles and paradigms that a PHP framework may encapsulate are the Don’t Repeat Yourself (DRY), Keep It Simple, Stupid (KISS) and Convention over Configuration (CoC) [23][24][25]. By applying these principles and paradigms, a developer is spending less or no time working on configuration, reusability of the code is obvious and the implementation structure is well preserved.

Examples of other implementations that a framework may include are CSRF-protection, Session- and Cookie hijacking and even authentication and authorization components [26]. Below is a brief description of the frameworks evaluated in this study:

1.2.1 Symfony

Symfony is one of the early PHP frameworks. Its creator Fabien Potencier released the first version in 2005 [27]. At the time of writing the report the stable version available for downloading is 2.3 and is built upon or requires PHP version 5.4 or higher.

A list of the most significant features that the framework characterizes:

- The ability to install the framework using a standalone tool.
- Command tool for generating and scaffolding the implementation.
- Separations of Concerns, which means that implementation files are organized in sections in the file structure.
- Built-in Object Relational Mapping (ORM) implementation.
- The framework embraces the usage of the MVC-design pattern.
- Built-in convention for view routing.
- Built-in implementations for sanitization, validation and security vulnerability protection.
- Components for user authentication and authorization.

1.2.2 CodeIgniter

CodeIgniter is also one of the early PHP frameworks. The first version was released in the beginning of 2006 [28]. The man behind CodeIgniter is Rick Ellis, rock musician turned programmer. The reason for developing CodeIgniter is because Rick Ellis was disappointed with that time’s available PHP frameworks. He was particularly disappointed with issues like terrible documentation, high learning curve, complicated deployment and the usage of terminal commands [28].

There is no installer available for installing the stable version of CodeIgniter, which is 2.2.1 at the time of this study. A compressed package containing the file structure of the framework is available and it is the only thing a developer would need to start using the framework. MVC-pattern and C-like Object orientation are recognizable techniques in the framework when
using it at the first time. However, one would miss the ability to generate or scaffold code using built-in tools in the framework.

1.2.3 Laravel

Laravel was first released in June 9, 2011 with its version 1 beta. Now four years later, the latest stable version is version 5.0.23 at the time of this study. According to Laravel’s creator Taylor Otwell, the reason for creation the framework was the lack of some essential functionality, like user authentication in the CodeIgniter framework [29].

Note that there are lots of similarities between Laravel and Symfony. In fact, all the characterizing future of Symfony listed in 1.2.1 may apply to Laravel as well.

1.2.4 Phalcon

Phalcon is one of the newest PHP frameworks that have been released so far. The first release was in 2012 and now, three years later the recent and stable version is Phalcon 1.3.4 [30]. The man behind Phalcon is Andres Gutierrez. What distinguishes Phalcon from the other frameworks is the fact that Phalcon is written as a PHP extension in the C language [30]. The extension is loaded into memory once and its API is exposed to the developer.

Regarding the characterizing features in the framework, it is certain that Phalcon shares similarities with both Symfony and Laravel in terms of how the framework is built and structured.

1.3 Previous research

Lancor and Katha used a similar approach to compare web applications written in plain PHP and two frameworks CodeIgniter and CakePHP [31]. The authors used both a qualitative and quantitative method to determine which one of these frameworks was best suited to be used in a student project in a school course. The quantitative analysis focused on three areas which are (1) Coding effort, where they count the lines of code for the three versions of the web application they wrote, (2) Performance, in which the authors measure the execution time and lines of codes of functionality like login/logout, session management and form handling implemented in the web applications and (3) Security, in which the authors use a penetration tool aimed to expose SQL-injection vulnerabilities.

The qualitative analysis was based on some quality measures, which the authors defined as ease of installation, some built-in functionalities like CAPTCHA support, learning curves, developer friendliness etc. The authors classified these quality measures in three categories High, Medium and Low.

Håkan Nylén has made another approach where he is comparing performance between the same frameworks CodeIgniter and CakePHP [32]. Nylén has performed a literature review in which he is answering the
questions regarding the web performance evaluations. What kinds of these evaluations exist and how they are performed. Nylén is also trying to find answers for what factors may impact the web performance. Furthermore, Nylén is trying to find out to what extent the open source PHP frameworks have been evaluated.

The conclusions Nylén draws from the literature review are:

- Web performance evaluations are made using load testing.
- Request per milliseconds are used as measurement units in the evaluation.
- The factors that impact the evaluation are the Server (Hardware), the network and wrong configuration.
- There is no previous research to be found addressing evaluation of the open source PHP frameworks.

As a final step in the study’s research method, Nylén is conducting a load testing experiment in which the two frameworks are targeted. The results of the experiment show that CakePHP is performing better than CodeIgniter.

1.4 Problem definition

In this study an experiment is conducted in which five functionally equivalent web applications representing a simple blog are developed. The first web application is developed using plain PHP and the MVC design pattern. The four other applications are developed in the frameworks Laravel, Symfony, CodeIgniter and Falcon.

In the experiments, execution time and the consumed memory usage when executing the CREATE, READ, UPDATE and DELETE (CRUD) implementation in each web application of the experiment will be measured. The tools and libraries to be used are Xdebug [33], which has built-in functionality for PHP profiling and a tool called Webgrind [34], which will be used for the visualization of the generated data. In case of measuring the memory usage, PHP provides a built-in function (memory_get_usage) for measuring the memory usage in the script files [10].

The stack trace in which the number of the functions that has been called will also be measured. This task is done in the same session as when the execution time is being measured. The reason for that is to see if the depth of the stack trace affects the execution time and the memory usage.

1.5 Purpose and research questions / hypothesis

The study aims to answer the following research questions:

- RQ1: How does execution time and memory usage differ for a simple blog web application written in plain PHP and the following frameworks Laravel, Phalcon, Symfony and CodeIgniter?
• RQ2: In the case of frameworks, which one of these four offers the best performance in terms of execution time and memory usage?

1.6 Limitations

We have defined the following limitations and boundaries for this study:

• The experiment will be conducted on one environment, which consists of a desktop computer running on local network.
• The versions of PHP frameworks used in the experiment will be the latest and stable ones published on their respective webpages at the time of this study.
• Since the aim of this study is to measure and compare performance of PHP, no other languages like Javascript or frameworks or styling techniques like CSS will be used or injected in the code.

1.7 Target group

The potential target groups for this study are developers, software architects and decision makers working with PHP. The result of this study may bring more insight for this group regarding the efforts of choosing the right technique for each development project in their jobs.

1.8 Overview of the report

This report begins with an introduction covering the purpose and motivation of the study. This is followed by the Methodology chapter, in which an experiment is constructed and explained. Next we will outline the experiment and present the results of it. The following chapter concludes the study by a discussion in which results and research questions are presented. Finally, conclusions that have been made in this study and suggestions for further research are outlined.
2 Method

To measure execution time, memory usage and the number of called functions an experiment is conducted. The experiment includes a simple blog web application for which performance metrics are measured and evaluated. This simple blog web application is written first in plain PHP using the MVC-pattern. Afterwards four functionally equivalent blog applications are written, one in each framework. The frameworks to be used in the experiment are Laravel, Symfony, CodeIgniter and Phalcon.

2.1 Hardware specification

In this experiment a Mac Mini personal computer with the following specification is used:

- Intel core i5 2.3GHz processor.
- SATA hard drive with 500GB capacity.
- 8GB 1333MHz DDR3 Memory.
- Intel HD Graphics 3000 shared 512MB memory.

2.2 Software specification

The software, tools, programming languages, libraries and frameworks used in this experiment are:

- OS X Yosemite x64 version 10.10.2.
- PHP 5.5.10 is the version used in the development of the five functionally equivalent blog applications.
- Laravel version 5.0.23.
- Symfony version 2.3.
- CodeIgniter version 2.2.1.
- Phalcon 1.3.4 is installed as a PHP-extension.
- Apache version 2.2.25 as Web server.
- MySQL server version 5.5.34 used as backend for data persistence.
- Xdebug version 2.2.3 is installed as a PHP-extension.
- Webgrind v 1.0, which is the web application that parse and humanize the profiling data generated by the Xdebug.

As a last step, the PHP had to be configured to enable the Xdebug profiling functionality. This is done by adding the following lines to the php.ini file:
This configuration will be enough to engage the profiling functionality of Xdebug, which includes the measurement of function calls and the execution time of these functions. The generated data is dumped into a file located in the /tmp folder. The profiling file is generated every time a HTTP request is made to the Apache server running on the local development environment. The Webgrind tool is used to parse and visualize the data contained in the generated file.

2.3 Five versions of the web application

As mentioned earlier, five functionally equivalent blog web applications had to be developed. The first is a plain PHP version, and the other four are developed using the frameworks Larave, Symfony, CodeIgniter and Phalcon. Each version is described in more detail below.

2.3.1 Plain PHP version

The development started by defining the blog model for the application. To keep it simple, the whole application consists of only one model representing a blog entry containing the following five attributes:

- Id: which holds the unique identifier for the blog entry.
- Title: representing the title of the blog entry.
- Body: which will contain the text content of the blog entry.
- Created at: is the attribute holding the date and time value for when the entry is created.
- Updated at: is also a date and time attribute for when the entry was last updated.

As a database backend MySQL is used. A database containing one table representing the blog entry model was created. To communicate with that database, a data persistence class was written to read and write to the

Figure 2.1: The Xdebug configuration section inside php.ini that has been used in the study.

```ini
[xdebug]
zend_extension="/path/to/xdebug.so"
xdebug.profiler_enable=1
xdebug.profiler_output_name=xdebug.out.%t
xdebug.profiler_output_dir=/tmp
xdebug.show_exception_trace=0
xdebug.trace_format = 1
xdebug.auto_trace = 1
```
database. The controller class was developed in a RESTful way [35], having action methods like index, view, create, update, save and delete. The views are generated using a static class, which is injected in the controller. Classes for input validation and sanitization were also implemented and as a security vulnerability protection, a simple CSRF class was also implemented. Finally, a class taking care of auto loading required classes was also implemented in the application.

2.3.2 Laravel version

When the first blog application has been developed, the implementation can be used as an inspiration source in order to develop the other framework versions. This is the approach used in this study.

The Laravel installer tool has been used to generate an empty Laravel project. To create the blog entry model, a table representing that entry has been created in the MySQL. Out of that table, PHP model and RESTful controller classes have been generated using the scaffolding tools provided by the framework. The views have been manually created and to write them, the template engine provided by the framework was used. And exactly as it has been done in the plain PHP version, there are implementation invoked for handling the input validation, sanitization and security.

2.3.3 Symfony version

An empty Symfony project has been generated using the installer tool downloaded from the framework’s website. To start working in Symfony, a bundle inside the newly created project had to be created. Once that is done, the configuration file was modified, by adding database connectivity parameters to it. And to create our blog entry implementation, a built-in tool in Symfony was used to scaffold and generate the model, controller and views. A couple of code lines have been added to handle the input validation & sanitization. CSRF protection was enabled by default in the framework.

2.3.4 CodeIgniter version

CodeIgniter is the only framework among the frameworks used in this experiment that does not have an installer. To use it, the whole framework bundle has to be installed. Once the configuration file is set up, the models, views and controllers have to be created manually. The convention is to create a PHP file and place it in the controller’s folder and the same is for views and models. Creating models in CodeIgniter does not work the same as Laravel or Symfony. Models require a little bit of coding to get them up and working. Input validation and sanitization were easy to set up and the CSRF protection as well.
2.3.5 Phalcon version

As in Lavarel and Symfony, Phalcon comes with an installer to create an empty project. There is also an external generator tool provided by the core team, by which the code generation is made easy for the developer. A model representing the blog entry and a RESTful controller taking care of the request actions and view, were easily generated. The blog entry model was modified to handle the input validation and sanitization. The views containing submission forms were also modified to enable the CSRF protection.

2.4 Functional equivalence in the versions

Below is a summary of the experiences from developing the five blog applications and the functional equivalence in them:

- The design pattern used in the development is MVC in which the model is identical in all five applications and that the model is backed up by an identical database table in the server.
- The controller class is functionally identical, designed in a RESTful way where actions like index, view, update, create etc. are handled by a similar implementation.
- The execution flow inside all five applications is identical. When a blog entry is created using the edit view, the user is directed to the view-blog-entry view and when the delete-blog-entry button is pressed, the user is redirected to the index view etc.
- No other languages or frameworks than HTML and PHP are used in the development. The code was cleaned from any generated CSS or linked JavaScript libraries.
- All five applications have implementations for CSRF protection, validation and sanitization of the input data.
- The input data used when creating and updating the blog entry is fixed, consisting of a “lorem ipsum” string. The length of the string is 11 characters for the entry title and 445 characters for the entry body. When updating the entry blog, the same input is used but with three extra characters “\s01”, appended to it.
- The whole experiment is executed on the same computer using the same system, software and libraries.

2.5 Measuring the performance

Five blog applications, one in plain PHP and four in different frameworks, have been implemented and are ready to be used. The three experiments that will be conducted using these applications are the following:
2.5.1 Execution time comparison experiment
The execution time is the time it takes for a processor to perform or execute a certain operation. The aim here is to choose a few of these operations in each application and perform execution time measurements. In an MVC application a good spot to perform the measurements on is the controller actions. Four of these actions have been chosen as measurement targets:

2.5.1.1 Create action
The measurement starts in the create-blog-entry view when the user has pressed the create-blog-entry button after the form has been filled in. The input data is posted to the controller. The action inside the controller checks the CSRF-token and then process the input data by validating and sanitizing to later create a new record in the database. Once done, the newly created blog entry is presented in view-blog-entry view and the measurement is stopped.

![Create new blog entry](image)

Figure 2.2: Example of a create-entry view in the targeted applications.

2.5.1.2 View action
The measurement starts when the link responsible to show the blog entry is clicked and ends when the controller views the blog entry. This happens in the index page where only one blog entry is created. Once the view is rendered, the measurement is stopped.

![Lorem ipsum](image)

Figure 2.3: Example of a view-entry view in the targeted applications.
2.5.1.3 Update action
The measurement starts when the updating input data is posted from the update-blog-entry view using a submission button. The measurement ends when the updated blog entry is shown in a view-blog-entry view.

![Edit blog entry](image)

Figure 2.4: Example of an update-entry view in the targeted applications.

2.5.1.4 Delete action
The measurement starts when the delete button in a view-blog-entry view is pressed and ends after the index page has been rendered completely with the information confirming that the deletion was successful.

![The entry has been deleted. Create new entry](image)

Figure 2.5: Example of a rendered index view when an entry is deleted.

2.5.2 Measuring the stack trace
At the same time the execution time measurement is conducted, the number of called functions is also measured. This is done using Xdebug to trace the called functions. The purpose of this effort is to see if there is any connection between a high execution time or high memory consumption and how deep the stack trace is in the measured actions.

2.5.3 Measuring the memory usage
Once the measurement of execution time and stack trace is done, a new measurement session is started in which the memory usage in all the four actions mentioned is measured. The aim here is to measure the amount of memory every action is consuming. A simple method is embraced in which
the PHP function (memory_get_usage) is used. According to the documentation, the function returns currently allocated memory in a PHP script [10]. The bootstrapping PHP files in the five web applications have been chosen to execute this method.

To be precise, the initial memory usage is measured in the beginning of a bootstrapping file. When reaching the end of the file, the memory usage is measured again but this time the initial value is subtracted. The result of the measurement is shown in the browser.

2.6 The usage of Xdebug and Webgrind

To summarize the process of what to measure and how to measure Xdebug generates a data file containing information about the stack trace and execution time. For each tested action in the time execution experiment, there is a generated Xdebug file in the /tmp folder. The dumped file is opened in the Webgrind application where the values for execution time and number of called functions are presented and stored. This process is repeated for all the actions (see chapter 2.5.1) and in all the involved web applications in the experiment.

2.7 Reliability

To estimate consistency of the experiments result outcome, each measurement will be conducted ten times in a sequence. Ten times for each of the four actions repeated during all the tree experiments. As previously mentioned, the three experiments will be performed in two sessions. The first session covers the time execution and stack trace measurements. The second session covers the memory usage measurements. The reason for this is because the implementation of the targeted applications has to be modified in order to measure the memory usage as mentioned in (chapter 2.5.3). The results will be shown as mean values of the total number of times each action has been tested.
3. Result and analysis

In this chapter, the results from the conducted experiments are presented and analyzed. The presentation is outlined in a tabular form in which the horizontal columns represent the tested actions Create, Read, Update and Delete. The vertical columns represent the version of the blog web application containing the measured actions. For each tabular form there is a corresponding visual presentation in a form of a bar graph.

3.1 The result of the execution time experiment

The results show that the Symfony framework has the highest execution time in the Create, Read and Update actions. The second highest is Laravel with result values that are slightly less than Symfony. Symfony is faster than Laravel only when executing the Delete action but on the other hand Symfony is showing deviant performance when executing Update action. Both Symfony and Laravel stand in a class for themselves hitting between 500-1100 milliseconds in all actions. CodeIgniter comes third reaching an execution time slightly over 100 milliseconds in all actions. It should be noticed that the variation gap between CodeIgniter and the both frameworks Symfony and Laravel is broad. Phalcon comes in the fourth place with execution time between 28-45 milliseconds in all actions. The fastest application is the one written in the plain PHP hitting between 10-17 milliseconds in all actions. For detailed results see Table 3.1.

<table>
<thead>
<tr>
<th>Test Targets</th>
<th>Create</th>
<th>Read</th>
<th>Update</th>
<th>Delete</th>
</tr>
</thead>
<tbody>
<tr>
<td>Plain PHP (MVC)</td>
<td>17</td>
<td>13</td>
<td>17</td>
<td>10</td>
</tr>
<tr>
<td>Phalcon (1.3.4)</td>
<td>43</td>
<td>28</td>
<td>45</td>
<td>41</td>
</tr>
<tr>
<td>CodeIgniter (2.2)</td>
<td>117</td>
<td>101</td>
<td>121</td>
<td>105</td>
</tr>
<tr>
<td>Laravel (5.0.23)</td>
<td>785</td>
<td>752</td>
<td>815</td>
<td>726</td>
</tr>
<tr>
<td>Symfony (2.3)</td>
<td>787</td>
<td>812</td>
<td>1110</td>
<td>591</td>
</tr>
</tbody>
</table>

Figure 3.1: Visual representation of the time execution measurement results.
3.2 The result of memory usage experiment
The results of the memory usage experiment follow the same pattern as in the execution time experiment. The Symfony framework tops the list as the most memory-consuming framework. 14.51 Megabytes are consumed when executing the create action in Symfony. The second framework in the list is Laravel with 9.45 Megabytes executing the same Create action. CodeIgniter comes third with only 2.15 Megabytes. As in the previous experiment there is however a broad gap between CodeIgniter and the two frameworks once again. Phalcon is doing well with only 0.17 Megabytes memory consumption of the Create action followed by the plain PHP application with only 0.14 Megabytes. For details see Table 3.2.

<table>
<thead>
<tr>
<th>Test Targets</th>
<th>Create</th>
<th>Read</th>
<th>Update</th>
<th>Delete</th>
</tr>
</thead>
<tbody>
<tr>
<td>Plain PHP (MVC)</td>
<td>0.14</td>
<td>0.13</td>
<td>0.14</td>
<td>0.13</td>
</tr>
<tr>
<td>Phalcon (1.3.4)</td>
<td>0.17</td>
<td>0.16</td>
<td>0.18</td>
<td>0.17</td>
</tr>
<tr>
<td>CodeIgniter (2.2)</td>
<td>2.14</td>
<td>2.13</td>
<td>2.14</td>
<td>2.13</td>
</tr>
<tr>
<td>Laravel (5.0.23)</td>
<td>9.45</td>
<td>8.94</td>
<td>9.53</td>
<td>8.67</td>
</tr>
<tr>
<td>Symfony (2.3)</td>
<td>14.51</td>
<td>14.51</td>
<td>15.68</td>
<td>11.93</td>
</tr>
</tbody>
</table>

3.3 The results of the stack trace experiment.
The Symfony framework has the deepest stack trace followed by Laravel. 1504 functions are called when the Create action is executed in Symfony. Laravel calls 1253 functions executing the same action. CodeIgniter comes third by calling 279 functions in the Create actions followed by Phalcon, which has almost the same number as CodeIgniter and is at the fourth place. The plain PHP takes advantages of only 50 functions when the Create action is executed, which makes it the lightest application of all. For details see Table 3.3.
Table 3.3: The number of called functions in each CRUD action.

<table>
<thead>
<tr>
<th>Test Targets</th>
<th>Create</th>
<th>Read</th>
<th>Update</th>
<th>Delete</th>
</tr>
</thead>
<tbody>
<tr>
<td>Plain PHP (MVC)</td>
<td>50</td>
<td>44</td>
<td>52</td>
<td>39</td>
</tr>
<tr>
<td>Phalcon (1.3.4)</td>
<td>225</td>
<td>202</td>
<td>256</td>
<td>236</td>
</tr>
<tr>
<td>CodeIgniter (2.2)</td>
<td>279</td>
<td>261</td>
<td>277</td>
<td>260</td>
</tr>
<tr>
<td>Laravel (5.0.23)</td>
<td>1253</td>
<td>1228</td>
<td>1310</td>
<td>1194</td>
</tr>
<tr>
<td>Symfony (2.3)</td>
<td>1504</td>
<td>1504</td>
<td>1967</td>
<td>1055</td>
</tr>
</tbody>
</table>

Figure 3.3: Visual representation of the called functions in each CRUD action.
4. Discussion

The presented results show clearly the gaps in performance between the different versions of the blog application. The size of the implementation of each version is also presented, and shows some variation as well.

4.1 Problem solving/results

The result analysis of the conducted experiments categorizes the four targeted PHP frameworks and the PHP language in the three categories heavy weight, middle weight and light weight. Symfony and Laravel falls absolutely in the heavy weight class since they are performing worst both in terms of execution time and memory consumption. These two frameworks have a much higher load on the server compared to the other frameworks.

The only framework representing the middle weight class in the analysis is CodeIgniter. CodeIgniter performs too well to be categorized as a heavy weight but it is still far away from being a light weight framework.

Phalcon is the only framework of the four frameworks that is performing very well in terms of execution time and memory consumption compared to plain PHP. The differences between Phalcon and the plain PHP are too small to be noticeable and either of them can be chosen for a development project where performance is important. What is interesting with Phalcon is that the stack trace measurement result. Its size is almost equal to CodeIgniter’s stack trace, but the execution time and memory usage of CodeIgniter is worse than Phalcon. When comparing Phalcon with the plain PHP application, the results of the execution time and memory usage measurements are almost equal. The difference between the both applications lies in the sizes of the stack traces in the both applications. The plain PHP application has significantly smaller stack trace compared to Phalcon yet they show almost equal results in performance.

The conclusions of the comparisons are:

• The choice of whether using a plain PHP or one of the PHP frameworks will have a significant impact on the performance of the application to be developed.

• Comparing two PHP frameworks having equivalence of implementation size and where one is built upon the traditional PHP stack (ex. CodeIgniter) and the other is not (ex. Phalcon) will not necessarily mean that they both perform similarly.
5. Conclusion

To answer the research questions defined below:

- **RQ1**: How does execution time and memory usage differ for a simple blog web application written in plain PHP and the following frameworks Laravel, Phalcon, Symfony and CodeIgniter?
- **RQ2**: In the case of frameworks, which one of these four offers the best performance in terms of execution time and memory usage?

It would initially be stated that the following text would be considered as an answer for both research questions. The reason for that is that the questions are conditional, where the second research question (RQ2) is depended on the outcome of the first research question (RQ1).

So therefore, the result analysis for the plain PHP and Phalcon shows that the differences between them are relatively small to plain PHP’s advantage. The plain PHP is only a dozens milliseconds faster and consumes a few bytes less memory than Phalcon. The results also show that the plain PHP application is having relatively shorter stack trace in its function calls compared to Phalcon. If the plain PHP application was larger or had a stack trace tree of the same size as the Phalcon version, it would mean that the plain PHP would not have been performing better than the Phalcon. That is because the growth of the stack trace in the plain PHP application would have caused a higher execution time and memory consumption as it has been proven in chapter 4.1. Plain PHP is mostly suited for smaller projects even if there are no obvious reasons to not use Phalcon, but Phalcon would be the preferred choice when developing larger projects.

The result analysis also showed that Phalcon is the only light weight framework among all the tested frameworks in this study. Followed by CodeIgniter, which in the terms of performance has showed mediocre values in this study. The remaining two frameworks Symfony and Laravel are the two heavy weight frameworks performing worst both in terms of execution time and memory usage. This is due to the fact that these frameworks have deep stack trace trees in their implementations.

5.2 Further research

The software and/or hardware specification in this study can easily be modified. By doing so, new experiments with the aim of evaluating the impacts of these modifications can be conducted. For instance, using PostgreSQL instead of MySQL, Nginx instead Apache or a SSD drive instead of SATA.

The study can also be extended by adding additional PHP frameworks or future releases of the same frameworks used in the study for the purpose of
evaluation.
References


