The Comparison of the Web Application Development Frameworks

J. Rogowski
Institute of Information Technology,
Lodz University of Technology,
Branch of the University of Lodz in Tomaszow Mazowiecki,
jan.rogowski@uni.lodz.pl

Порівняння Каркасів для Створення Веб-Додатків

Я. Роговський
Інститут інформаційних технологій,
Технологічний університет м. Лодзь,
Університет в Лодзі, Філія Томашів-Мазовецький
jan.rogowski@p.lodz.pl

***Abstract*—The main goal of this work is to evaluate the web application development frameworks in context of productivity, powerful features, security, flexibility scale, performance and size of the community. The chosen technologies are Ruby on Rails written in Ruby, Django written in Python, and Grails written in Groovy.**

***Анотація*—Основною метою даної роботи є оцінка можливостей каркасів для створення веб-додатків в контексті продуктивності, потужних функцій, безпеки, гнучкості та розміру спільноти. Обрані технології то Ruby on Rails створений в Ruby, Django створений в Python і Grails створений в Groovy.**

Keywords—web programming; frameworks comparison

Ключові слова—веб-програмування; порівняння каркасів веб-додатків

I. INTRODUCTION

The career path of basic designers of static websites is pretty straightforward. Problems really begin for so-called back-end programmers. They have to choose from many web technologies available on the market, and what is worse, there is currently no recommendation system for the best decision. What is pretty obvious nowadays, however, is that nobody creates web services without using frameworks. It would be like reinventing the wheel because each and every framework helps with the most common things we meet in everyday web application.

Usually, developers choose the most favourite frameworks by coincidence, for example, the company where they are hired, using the same technology for years, old acquaintance recommendation, a program of study focused on one of the technologies, and so on. To be excellent in a particular field requires spending many weeks of effort, and humans inherently do not like doing something which is not enjoyable or does not see promising results fairly quickly. That is why, once chosen, a specific technology most often becomes a favourite. Every framework website advertises themselves as the best, with a lot of possibilities, so the problem is how to choose a framework which we would like to work within over the next few years. What is crucial and what is worthy of our attention?

The main goal of this work is to evaluate a few of the most promising and so-called cutting-edge technology frameworks and find out which ones support the criteria in place for present-day services. The technologies chosen for appraising are: Ruby on Rails [1] written in Ruby, Django [2] writen in Python, and Grails [3] written in Groovy.

II. WEB FRAMEWORKS

Web applications realize almost the same activities, and it does not make sense to write similar code every time. That is why frameworks provide a predefined structure and all needed mechanisms. Frameworks can be seen as a set of ready pre-written code or bunch of miscellaneous libraries with one general purpose – working in the Internet based on the client - serve architecture. The difference between framework and a set of libraries is their focusing range. Libraries often solve a narrow scope of a problem, whilst frameworks give a broader span of functionalities. Framework should aid web developers in their work. Except the code, frameworks provide the design patterns and principles. Thanks to that, developers are able to implement more unified, nicer and more reusable code, which leads to better quality of application. Nowadays, so-called 'full stack' frameworks are the most popular. They form a connected software stack, that is useful for every aspect of web development. These frameworks may be considered as an extension of the programming language.

The list of benefits for using web frameworks is long, and what is more - nowadays there are various available web frameworks. Most of them are free to use and have an open

source. Anyone can use some part of web framework or even contribute to framework development, like propose improvements, review a code or find some bugs and security vulnerabilities. Service *java-source.net* presents almost 70 different open-source java web frameworks, along with short descriptions and links to them. For the purposes of this work there were selected only three web frameworks.

A. Ruby on Rails

Ruby on Rails is built in Ruby (open source, dynamic, interpreted, object-oriented programming language, created in Japan by Yukihiro Matsumoto) under MIT License. Ruby on Rails uses the Model-View-Controller architecture where models map tables in a database, views are interpreted and converted to HTML at the run-time, and controllers are the server-side component for responding to external requests from the web server. Sometimes people say that RoR is not only a framework, but also a way of thinking about web application. It can be true, because nowadays exist few great frameworks, which officially mention about inspiration of Rails. One of them is Grails. Convention over configuration is the strongest point in Ruby on Rails and along with DRY (Don't Repeat Yourself) are the key concepts of Ruby on Rails.

B. Django

Django is a framework based on Python that was written in 2003. Python is open source, dynamic, interpreted, object-oriented programming language, which implementation was started in December 1989 in Netherlands. The framework was developed to meet the needs of a fast-paced online newsroom environment. Django as well as RoR follows Model-View-Controller but with a significant difference. Django models define the data of the web application, views handle requests and make a response for them, and templates are used for rendering response. So, in the world of Django MVC does not work as an acronym, and is better to call it Model-Template-View. The view describes the data that gets presented to the user (which data is presented, not how is displayed) and is the Python callback function for an appropriate URL.

C. Grails

Grails is a web framework, for the Java platform built in Groovy. Groovy is object-oriented programming language for the Java platform, released on January 2, 2007 in England. Groovy is dynamically compiled to Java Virtual Machine bytecode. First name of Grails was 'Groovy on Rails'. In March 2006, that name was changed with regard to a request by David Heinemeier Hansson, the founder of the Ruby on Rails. This situation clearly shows the influence of RoR. However, what distinguishes Grails from the above frameworks, is a fact that it takes a set of other frameworks literally. Grails bundles Spring, Hibernate, H2, Tomcat and remove the complexity of it, and thanks to that, using them is simple and provide functionality out of the box. Grails should allow to smoothly integrate with any other library running on JVM, which provides powerful features of the enormous world of Java tools.

III. Frameworks evaluation

To achieve the stated goals, three web applications have been created in the above-specified technologies. They have the same database scheme, sample dataset and front-end interface. In addition, in order to perform the evaluation of the frameworks in an effective way, the process is split into three phases.

A comparison between frameworks is presented, showing the features, helpers, and methods in a few examples.

The results of the benchmarks for the most popular actions in web application are presented. All results of benchmarks have been counted via the application written in Java. This application uses HTTP requests to get access to the tested applications and trigger examined actions. Each test is executed many times in order to reject outlier results.

During the process of web application frameworks evaluation the following features have been evaluated: internationalization, URL mapping, Cross-Site Scripting attack defense, Cross-Site Request Forgery attack defense and SQL Injection attack defense.

Each framework provides an effective mechanism against the most popular vectors of attack, like SQL Injection, Cross Site Scripting or Cross-Site Request Forgery. These types of attacks are in the top of ten most critical web application security risks. All of the three frameworks bring the same approach. Security mechanisms against SQL Injection and XSS are turned on by default. The developer is fully conscious when he is doing an action which is possibly vulnerable. In case of CSRF, only Grails needs some extra action to do. Django and RoR do it again by default. For internationalization, Ruby on Rails came off the best, providing the most powerful features out of the box.

IV. Benchmarks

This section presents the real data of performance comparison of executing fundamental tasks such as JSON serialization, database access, and server-side template composition. All tests were done several times to reject incidental cases, on the same machine and with the identical algorithm. The comparison does not take into account server stress, multithreading, number of simultaneous requests or any other server features.

Each experiment provides specified REST API and has a stated JSON response. Due to these facts it was possible to execute all tests for each framework with the same input and expected formatted results. Another thing common to all tests is returning time in milliseconds. For the purpose of this work three almost identical web applications have been written. Models and relation between them are the part of a real financial application.

A. Persisting data in a database

This test is called simple insert. The function which is responsible for handling requests, takes the value of parameter *n* from the query, then it obtains time from a system clock, and opens transaction *n* times, creates a sample user object, saves it and closes the session. Finally, the algorithm obtains system time again, and is able to calculate passed time needed for creation of *n* user objects. The last necessary thing is to prepare specified JSON response. Experiment started from n = 10 to n

= 498 with step 2, from n = 500 to n = 1000 with step 5, where every step was repeated 3 times.

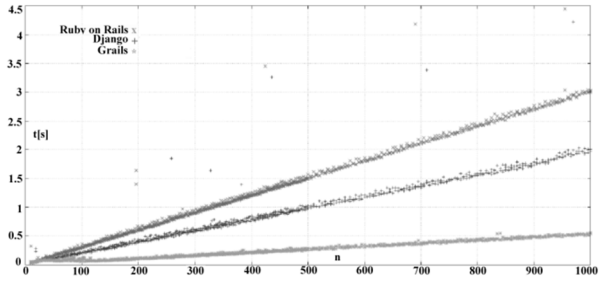

Fig. 1. Simple entity persisting results (time versus number of entities)

It is easy to notice that the dependency is linear (see Fig. 1). Ruby on Rails has the worst time needed to persist object in database. Django has a bit better result while Grails has the best score.

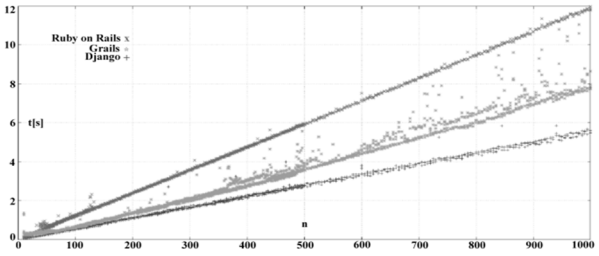

Fig. 2. Complex entities persisting results (time versus number of entities)

In the second experiment called complex insert, frameworks had to persist more complex objects, because this time the user had two kinds of relations. Each user entity has one of its project and three of its rights. This time Django had the best score, which came as a surprise, because Grails comes second in this competition. The longest time to persist connecting object was needed for Ruby on Rails. Making persisting harder did not change linear character of dependency (see Fig. 2), but just increased the time needed to insert all entities in the database.

B. *Reading from a database*

Reading data from a database is also the most needed task for a web application. In the beginning, the function takes the value of parameter *n* from the URL query, and prepares the data set. The method counts whether there is enough rows in the database and eventually inserts the desired number of rows. From this point, the time is 'started', the function gets the limited number of objects from the database using query set. Then the time is 'stopped', the elapsed time is calculated, and the response in JSON is prepared. This time Django has the worst times needed to read and map entities. Rails took second place but with small loss to the Grails framework (see Fig. 3).

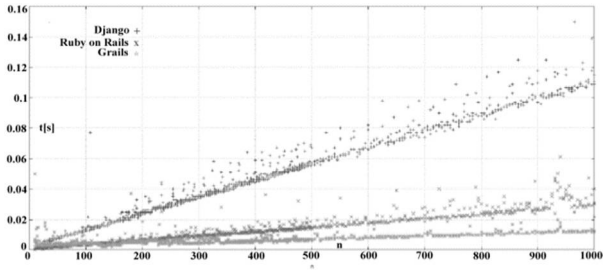

Fig. 3. Reading entities results (time versus number of entities)

C. *Serializing objects to JSON*

Sometimes web applications return JSON instead of HTML view. It takes place when the final client is not a user's browser but another application or service, for example, Facebook API which integrates other services with it.

In this experiment, like in the previous one, it is needed to have a given number of project entities in the database. After getting the value of parameter *n* from the query, the function optionally prepares missed rows. Next, when the objects are downloaded from the database, the time starts to be counted. The idea of this experiment is to evaluate the time needed to serialize objects to JSON.

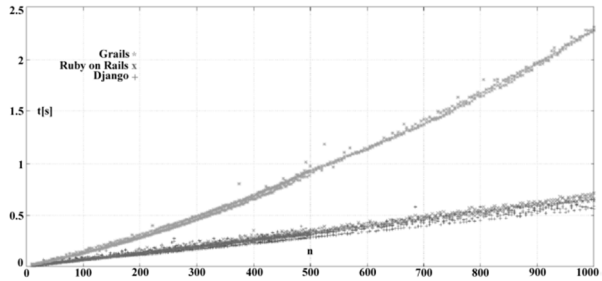

Fig. 4. Serialization results (time versus number of entities)

The result of serialization experiment, when it is needed to serialize more complicated object, for example, list of *Project* entities with two deeper levels of relations, is presented in Fig. 4. The plot indicates that this time Grails is the worst framework for serializing complex objects to JSON. The best time was achieved by Django and was only a little better than RoR.

D. *Handling request*

MVC frameworks function more or less in the same way. The request from the client goes to the server which forwards it to the *Dispatcher*. *Dispatcher* is a primary object which is responsible for dispatching requests to the appropriate controller. Then, the controller action could use all available framework features to do the job, and return some response. This experiment tested the average time needed to take care of client requests, and dispatching the requests to the controller.

Contrary to the other experiments, this one needs *startTime* parameters. Initially, after stepping into the controller action, a current time is obtained. After that it is possible to count the time, which passes between sending the request and stepping into the appropriate controller.

TABLE I. HANDLING REQUEST TIMES

Framework	Time
Ruby on Rails	13.53 ms
Django	1.8 ms
Grails	1.06 ms

The results are presented in Table 1. The time unit is presented in milliseconds, and this is the average value from one hundred tests. The difference between Grails and Django is minimal for the benefit of the first one. The average request handle time for Ruby on Rails is about ten times larger.

E. Sending response

Handling request experiments tested the time needed for dispatching the request to the appropriate controller. This test counts the time passed between the execution of the last instruction of the controller and the moment when the response is received by the final client. The research application has to count the passed time.

TABLE II. SENDING RESPONSE TIMES

Framework	Time
Ruby on Rails	0.54 ms
Django	1.05 ms
Grails	1.1 ms

The results of this test are presented in Table 2. Ruby on Rails is the fastest framework in this comparison in preparing response. Django and Grails have twice worse results, although one millisecond is still a very small value, almost unnoticeable.

F. Rendering view

Rendering experiment has to get a list of project entities from the database, map them to the object and pass to the template engine, which renders HTML view. The web application usually returns in response the HTML content ready to be displayed via the browser, and this test assesses the performance of the template engines in three frameworks.

First, the algorithm checks if the database has the required number of projects. If it does not, than the missing entities are inserted before counting the time. Next, the required number of projects is downloaded from the database, and after that the time starts to be counted. When the function returns HTML content, the end time is received and elapsed time is calculated. Now the function has only to prepare JSON response.

The experiment was done more than one thousand times with a different number of rendered rows. Again, the results show linear dependency (see Fig. 5). Django has the worst time, with Ruby on Rails three times and Grails almost ten times quicker.

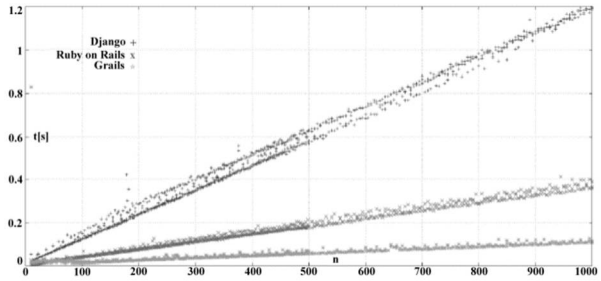

Fig. 5. Rendering template results (time versus number of entities)

V. CONCLUSION

Concluding the results of the benchmarks, the best results are obtained by Grails, which merely confirms a speed advantage of compiled language in comparison with interpreted language. Just in one test Grails scored the worst time, and it happened in serializing objects to JSON with two-level depth relations. Comparing Django and Ruby on Rails, they are competitive with each other. Hence, it would be fair to determine the tie. In the size of the community category, a job market, an amount of available books and resources, Ruby on Rails wins, while Grails has the worst score. Presented results are reflected in development process, because the least problems and the biggest help in documentation and the Internet was for RoR. All examined frameworks provide security features and support developers in writing non-vulnerable applications.

The conclusion is that the choice of the web framework is of secondary importance. In other words, any given framework is not the ultimate blocker in the development path. The most important thing in the decision process is to take into account programming language preferences, the size of the community and a job market.

REFERENCES

[1] Ruby on Rails. The website of ruby on rails framework, March 2017, URL http://rubyonrails.org/.

[2] Django. The website of django framework, March 2017, URL https://www.djangoproject.com/.

[3] Grails. The website of grails framework, March 2017, URL https://grails.org/.

[4] D. Heinemeier Hansson, S. Ruby, D. Thomas, "Agile Web Development with Rails 4", The Pragmatic Programmers, LLC, 2013.

[5] V. Kushner, O. Fernandez, K. Faustino, "The Rails 4 Way", Addison-Wesley, 2014.

[6] W. Chun, J. Forcier, P. Bissex, "Python Web Development with Django", Pearson Education Inc., 2009.

[7] B. Klein, D. Klein "Grails 2: A Quick-Start Guide", The Pragmatic Programmers, LLC, 2013.