# The *play!* Framework

Peter Kolbe (314201)
Berlin Institute of Technology (TU-Berlin)
June 22, 2012

***Abstract*—Web Development has grown tremendously during the last decade. Enabling the possibility to create and deploy web applications in a more convenient way, frameworks like Ruby On Rails, Grails, Django or CakePHP have arisen. The concept of rapid web development has not been available for the java world until Zenexity released its framework *play!*. This paper will demonstrate the key concepts of *play!* and compare it to the present state of the art in rapid web development.**

## I. Introduction

During the last decade, not only the traffic but also the opportunities of the Internet have grown tremendously, as shown in `Figure 1`. In addition, smart phones have enabled users to surf the web even more. This broad base of potential customers has become a critical part of today's business world, which is known for its explosive increase in new products and therefore its demand for ever-shortening development times. [1]

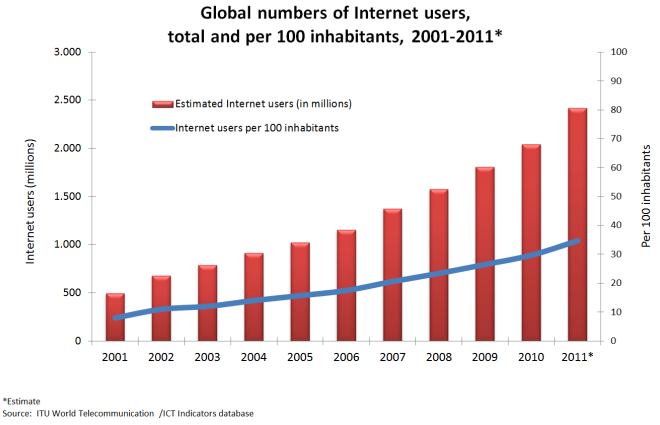


Fig. 1. Development of the world-wide Internet usage in the last decade, available at http://www.itu.int/ITU-D/ict/statistics

As a consequence, an approach close to rapid prototyping[1] is becoming more and more common. Rapid wWeb development is the concept, which is much more appropriate for switching requirements and short development times. But since GeoCities has become obsolete and the web has been upgraded to its second edition, websites have become almost full fledged applications being displayed by a browser that acts almost like an operating system itself.

[1]Concept that focuses on the production of fully functional prototype, which consists of "[...] parts that are close to or are in the final shape. [...] the time to produce any part [...] will be fast and can be in a matter of hours." [2]

Therefore, applications are nowadays increasingly complex on the one, since software and web engineering have become more and more entwined. In addition, general-purpose languages have been designed to be greatly generic and highly configurable.

Concerning Java for example, the servlet specification is a powerful standard, but it also means having a lot of configuration and deployment overhead. In contrast to the typical heavyweight frameworks, Ruby on Rails, Django or CakePHP follow the lightweight *convention over configuration*[2] approach and focus on HTTP instead of integrating it as just another module.

On the other side, the Java Virtual Machine provides high performance and reliability, as well as libraries for each case. Furthermore Java is a programming language that nearly every developer has been in touch with and which is commonly used in many companies. That is why there is a need for a lightweight Java-based web framework as ***play!***.[4]

Already in 2007, the French web development company Zenexity[3] started ***play!*** as an internal project, which became an open source project only two years later. [5] Since its release, the framework has evolved enormously:

Already in version *1.1* (2010), the used HTTP server changed from Apache Mina to JBoss Netty, the framework was extended by HTTPS, OAuth, Glassfish and Scala support. In 2011, with the release of ***play!*** *1.2*, Apache Ivy has been integrated as an dependency management mechanism, the database has been exchanged and many new features like WebSockets have been implemented. Since March 2012, ***play!*** *2.0* is available and has been integrated into the *Typesafe Stack*[4], because of its increased stabiltiy and scalability. [6]

## II. Play!

In sum, ***play!*** can be considered as the Java antagonist to the existing rapid web development frameworks like Ruby On Rails, Django or CakePHP. In this chapter, ***play!*** 's key concepts and features of ***play!*** will be explained, as well as its structure and test integration.

[2]An approach, that aims for the reduction of development complexity by avoiding compulsory configuration down to the very last detail[3]

[3]http://www.zenexity.com/

[4]"The Typesafe Stack is a modern software platform that makes it easy for developers to build scalable software applications in Java and Scala. It combines the Scala programming language, Akka middleware, Play web framework, and robust developer tools in a simple package that integrates seamlessly with existing Java infrastructure" http://www.typesafe.com/

### A. Key Concepts

Basically, ***play!*** comes *Out of the Box*, which means, the framework is a full stack where all necessary components and APIs are integrated already. In addition, it follows the *fix and reload* approach, which means developers do not suffer from delays caused by recompiling or redeploying, changes are directly accessible after a single page reload. Since the build system has changed from Python scripts to the *simple build tool*[5], live compiling and reloading is the basis for this strong rapid web development feature. In addition, the new build system provides the possibility to customize and adapt the build and deployment in much more detail though ***play!*** 2.0 "[...] comes with a preconfigured build script that will just work for most users." [5]
The ***play!*** framework has always been RESTful[6], which promises a much higher scalability and reliability.
A big difference to older versions of ***play!*** and other frameworks is the treatment of requests: Any request is potentially handled as being long-lived due to the underlying web server JBoss Netty[7]. To deal with the concurrency and to provide the possibility of creating highly-distributed systems, ***play!*** leverages the middleware *Akka*. This enables developers to tap the full potential of asynchronous programming and to avoid the overhead of general Java EE frameworks. [5]

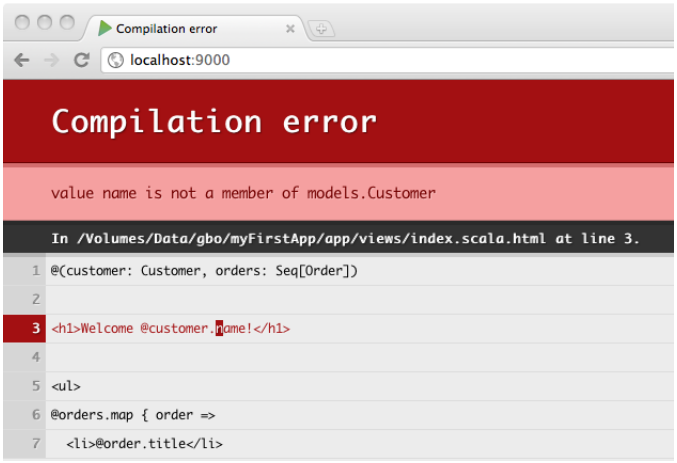


Fig. 2. Error Detection in ***play!*** [5]

Running in the JVM[8], ***play!*** applications benefit from being written in a statically-typed language. Furthermore the Scala-based template engine[9] and the routing system are also fully type-checked. This contributes to faster error detection, which is supported by the way in which errors are displayed: The browser even highlights mistakes in templates with an exact error message and the line number, as shown in `Figure 2`. Concerning database management, ***play!*** has evolved as well and is following the approach that the developer should be able to decide which database suits best for his specific use case, as there is no generic solution. Therefore a couple of database access libraries like Ebean, JPA, or Anorm are already integrated, as well as a CRUD module to keep up the bundle character of ***play!*** . But developers are also given the opportunity to be able to make use of any kind of data store technology.
Another important feature of ***play!*** is the integrated unit testing support for JUnit and Selenium[10], which benefit from the powerful build system and thus allow much faster debugging. [5]

### B. Architecture

As several other web frameworks, ***play!*** is based on the *MVC pattern*[11], because it implies a much better degree of flexibility, clarity and maintainability. This has a direct influence on the layout of an application, as shown in `Figure 3`.

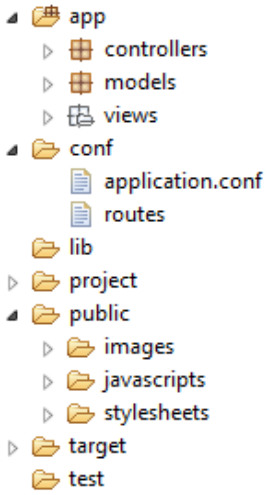


Fig. 3. ***play!*** 's Code Structure

The *app/* directory contains *controllers*, being implemented in Java or Scala, *models*, represented as database entities, and *views* which are Scala templates.
In *conf/*, the main configuration file and route definitions are pooled. In *lib/*, JAR files can be put to be added manually to the application's classpath. To customize sbt, changes can be applied to the build script, which is situated in *project/*. The generated code can be checked out in *target* and unit *tests* are of course stored in the so called folder. [5]
How these components interact with each other and how ***play!*** handles HTTP requests is shown in `Figure 4`.

[5]Minimally intrusive build tool for Scala projects, http://www.scala-sbt.org/
[6]REpresentational State Transfer, http://www.oracle.com/technetwork/articles/javase/index-137171.html
[7]http://www.jboss.org/netty
[8]Java Virtual Machine, http://docs.oracle.com/javase/specs/jvms/se7/html/index.html
[9]In previous versions, Groovy was the template engine
[10]Portable software testing framework with a record/playback function and other features, http://www.seleniumhq.org/
[11]Illustration of the MVC pattern in the web context: http://msdn.microsoft.com/en-us/library/ff649643.aspx

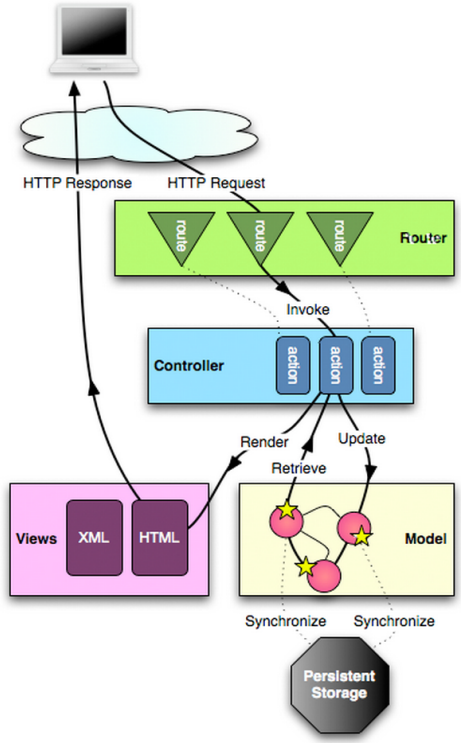


Fig. 4. ***play!*** 's Architecture [5]

The starting point for *HTTP requests* is an application's *routes* file, which maps the incoming information, comprising HTTP method and URL, to certain actions. These are calls of Java/Scala methods which are described in the *controllers*. Conflicts between competing routes are solved by prioritizing the one being firstly declared. This is also important for the reverse routing mechanism, which allows the definition of an application's URI patterns in the same file, which is thereby a single point of access for refactoring. [5]
After a request has been routed to a method call in a controller, the latter processes the additional parameters of a request. Furthermore it eventually reads and applies changes to persistent database elements, which are defined in *models*. Finally, the controller calls a render method and optionally passes several parameters, according to the signature of the addressed Scala template. Moreover, these *views* are organized in a modular way, which means that they can be built upon each other. As the result of processing the templates, HTML code is generated, which is returned as a *HTML response* to the client. [4] [7]

## III. State Of The Art

In the last chapter, a promising Java solution for rapid web development has been presented. But this kind of frameworks do exist already since several years. Today, there is a long list of them, but concerning the scope of this paper, only the most common representative of a certain language is presented. To have a better overview, `Figure 5` illustrates how much these frameworks are used in professional practice and subsequently a brief summary of each framework will contribute to a better overview of today's rapid web development world.

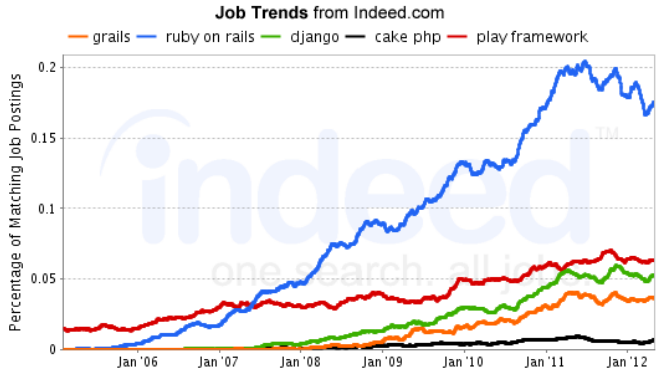


Fig. 5. Job trends for the considered frameworks, by http://www.indeed.com

The following frameworks have qualified for the ongoing comparison:

- Ruby On Rails - based on Ruby[12]
- Django - based on Python[13]
- Grails - based on Groovy [14]
- CakePHP - based on PHP [15]

### A. Ruby On Rails

Ruby On Rails went open source in 2004, the first release version came out one year later. At the moment, 3.2 is the lastest version. [8] As all the mentioned frameworks, it is a full stack, which follows the *convention over configuration* and the *DRY*[16] approach. Furthermore a Scaffolding mechanism is implemented to facilitate CRUD support, but this is also a common feature in rapid web development frameworks. [8] One of the most visited pages built in Rails is *yellow-pages.com*, which has been migrated from Java to Rails in 2008. Saving more than 100 000 lines of Java code, the new framework succeeded in optimizing the whole page[17]. [10] Other popular websites using Rails are *Groupon*[18] and *twitter*. [8]

### B. Django

Though Django has been open souce since 2005, the first release came out in 2008. At the moment version 1.4 is the latest, being available since march. Django was named after the famous jazz guitarist Django Reinhardt. [11] In comparison to other frameworks, there are several content management systems (CMS) built in Django, e.g. *Django*

[12]http://www.ruby-lang.org/
[13]http://www.python.org/
[14]http://groovy.codehaus.org/
[15]http://www.php.net/
[16]"Don't Repeat Yourself" which means avoiding as much redundancy as possible [9]
[17]Reduced code from 125 000 to 20 000 lines, http://brianburridge.com/2008/06/08/yellowpagescom-on-rails/
[18]http://www.groupon.com

*CMS*[19] or *Mezzanine*[20]. Examples for popular pages being built on Django are Facebook's newest acquisition *Instagram*[21] or *Mozilla*'s website[22]. [11]

### C. Grails

Groovy in a dynamically typed script language running in the JVM, which was created to transport Ruby's concepts to the Java world. [12] This is why the framework was given the name *Groovy on Rails* in the first place. But becoming more popular in 2006, as response to the request of Ruby On Rails founder David Heinemeier Hansson, the name was changed to *Grails*[23]. As the other frameworks, Grails follows the MVC paradigm and uses its own template system for views.
The company behind Groovy and Grails was G2One, which was acquired by SpringSoure in 2008, which again was acquired by VMWare in 2009. [13] A popular example for Grails is the website of the British broadcasting company *sky*[24]. [14]

### D. CakePHP

CakePHP makes use of the already mentioned typical approaches, which are followed by rapid web application frameworks. The project started in 2005, when Ruby On Rails became popular. The idea was to integrate Rails' most important concepts into a PHP framework. At the moment, it is available in version 2.1.3, which came out in May. A popular website using CakePHP is for example *Teamspeak*[25]. [15] [10]

## IV. Conclusion

Summarizing the current state of rapid web development, it becomes quite clear, that all of the mentioned representatives have gradually evolved by adapting the best concepts from each other, whereas Ruby On Rails was probably the first mover. Since all of them but ***play!*** are based on dynamically typed languages, an advantage of ***play!*** is its error detection, which enables much faster developing and testing.
In addition, a solution like ***play!*** has the advantage of using a well-known general purpose language, which has a much broader user basis than the much *younger* script languages. Starting to work with a rapid web development framework and learning a new programming language is much more effort than just getting used to some new design patterns.
Therefore CakePHP is a good entry point for a lot of companies that have employed PHP developers yet. Rails makes this up with its large community which provides a lot of samples, documentation and forums as support for developers. In addition, the community provides great feedback for the Rails team to improve the framework. To conclude, rapid web development will probably increase even more in the next years and with the upcoming HTML5, there will be a lot of new possibilities. To manage them in an efficient way will be the task for the next generation of the mentioned frameworks or new concepts.

[19] https://www.django-cms.org/
[20] http://mezzanine.jupo.org/
[21] http://www.Instagram.com
[22] http://www.mozilla.org
[23] Forum post by the Grails Project Lead, http://grails.1312388.n4.nabble.com/Groovy-on-Rails-is-no-more-kind-of-td1313422.html
[24] http://www.sky.com
[25] http://www.teamspeak.org

## References


[1] F. Maurer and S. Martel, "Extreme programming. rapid development for web-based applications," *Internet Computing, IEEE*, vol. 6, no. 1, pp. 86 –90, jan/feb 2002.
[2] C. Chua, K. Leong, and C. Lim, *Rapid Prototyping: Principles and Applications*. World Scientific, 2010. [Online]. Available: http://books.google.de/books?id=4OYcyiDUpsQC
[3] J. Miller, "Patterns in practice. convention over configuration," *MSDN Magazine*, Feb 2009.
[4] A. Reelsen, *Play Framework Cookbook*, ser. Community experience distilled. PACKT PUB, 2011. [Online]. Available: http://books.google.de/books?id=9Zf5bhIbH5MC
[5] "Introducing play 2.0," http://www.playframework.org/, last visited: 12.06.2012.
[6] "Typesafe stack 2.0: Wir sehen java und scala harmonisch miteinander koexistieren," http://it-republik.de/jaxenter/artikel/Webanwendungen-mit-dem-Java-Framework-Play!-2813.html, Mar 2012.
[7] W. Ellis, *Play!: Introducing the Play Framework*. Wayne Ellis, 2010. [Online]. Available: http://books.google.de/books?id=4AzAXwAACAAJ
[8] "Web development that doesn't hurt," http://www.rubyonrails.org/, last visited: 12.06.2012.
[9] K. Eilebrecht and G. Starke, *Patterns kompakt: Entwurfsmuster für effektive Software-Entwicklung*, ser. Kompakt-Reihe. Spektrum Akademischer Verlag, 2010. [Online]. Available: http://books.google.de/books?id=KBbiCYea9mUC
[10] J. Plekhanova, "Evaluating web development frameworks: Django, ruby on rails and cakephp," http://ibit.temple.edu/wp-content/uploads/2011/03/IBITWebframeworks.pdf, Sep 2009, last visited: 12.06.2012.
[11] "The web framework for perfectionists with deadlines." https://www.djangoproject.com/, last visited: 12.06.2012.
[12] "A dynamic language for the java platform," http://groovy.codehaus.org/, last visited: 12.06.2012.
[13] "Vmware acquiring springsource which acquired g2one," http://colinharrington.net/blog/2009/08/vmware-acquiring-springsource-which-acquired-g2one/, Aug 2009, last visited: 12.06.2012.
[14] "The search is over." http://www.grails.org/, last visited: 12.06.2012.
[15] "Cakephp makes building web applications simpler, faster and require less code." http://www.cakephp.org/, last visited: 12.06.2012.
[16] T. Haberkern, "Webanwendungen mit dem java-framework play!" http://it-republik.de/jaxenter/artikel/Webanwendungen-mit-dem-Java-Framework-Play!-2813.html, Jan 2010.
[17] J. S. Linowes, "Evaluating web development frameworks: Rails and django," https://s3.amazonaws.com/vaporbase/uploads/File/rails-vs-django.pdf, 2007, last visited: 12.06.2012.