Why you should always use Ruby on Rails instead of PHP

We live in exciting times. With new web frameworks and architectures popping up left and right it is essential to analyse them carefully and find the most effective solutions. In this essay I will compare and contrast Ruby on Rails with PHP frameworks, going through different perspectives on why Rails is the better choice.

One of the solid points web development on Ruby has going for it is the lack of fragmentation. Rails has complete monopoly over the language with over 250,000 stack overflow questions (StackOverFlow, 2016) compared to the measly 4,636 questions tagged with Sinatra (StackOverflow, 2016), its competitor. This effort concentrates Ruby development into a single framework which boosts community support and gem development. On the other hand, the PHP frameworks can be compared to Android fragmentation (Amit, 2016) where developer time is spread thinly over a variety of frameworks and projects.

The cornerstone of Rails is the notion of Convention over configuration (Hansson, The Rails Doctrine, 2016). An excellent example of this is David Heinemeier Hansson's video (Hansson, Ruby on Rails demo, 2016) where he uses the Rails scaffold and generate methods to create a blog engine in 15 minutes. However, these decisions can clash with other engineering patterns and with the use of legacy databases. For example, ActiveRecord automatically names tables to plurals of class names, although this can be overcome with the following addition (with class for holding fixtures) (Rails, 2016):

```
class Product < ApplicationRecord
self.table_name = "my_table"
end
```

Therefore, Rails' automatic generation makes decisions for the developer, leading to less code being written and more work being done. After all, this is exactly what Rails was made for. It should also be noted that many PHP frameworks such as Laravel, Yii and CakePHP offer scaffolding as well (Renton, 2016).

Rails keeps up with modern trends. In the front-end department it implements Haml, AngularJS, Sass and Coffeescript to keep designers on their toes. Using most PHP frameworks developers implement tools separately. In contrast, Rails has a tried and tested full stack environment that provides all required functionality out of the box. This is perfect for beginners who can learn new technologies from the start, rather than sticking with older methods that are no longer in use. Rails also includes its own server, WEBrick, which makes local deployment just as easy as with PHP.

With ruby's OO style, Rails uses Object Relational Mapping with ActiveRecord to deal with databases. I love it since doodling about with SQL commands can be a pain – with ActiveRecord a table is simply a class and a row is an instance of that class. These objects can be used anywhere within the project, rendering complex queries and SQL-configs a memory of the past. High level abstraction can be a problem though, since new developers can rely too much on ORM and not understand what is actually going on behind the scenes.

Stefan Rudvin 51549217

One of the finest features of Rails is RubyGems. While some PHP frameworks have tried to emulate gems with varying success, none have quite reached the refinement of RubyGems. In march Damir Svrtan studied RubyGems statistics and found that gem 'downloads are skyrocketing' along with Rails downloads. So, even though the mainstream is of the opinion that JavaScript-based solutions are stealing the spotlight, Rails is still going strong. Gems really are a tremendous addition to rails with a low learning curve, considerable community support and painless management through Bundler (Reiz, 2016). Appreciable examples of Gems are covered by Leonard Teo (Teo, 2016), who recommends Devise for authentication, Paperclip for file uploads and Simple Form for form creation. All gems are available through bundle or RubyGems.org. PHP's answer to RubyGems Composer and Packagist. They rely on packagist.org to distribute packages, which is very similar to Bundle and RubyGems. However, the maturity of the Rails framework still gives it the edge for third party packages.

My next point is essentially a toss-up between preference. As Yukihiro Matsumoto says, Ruby was created to have the user in mind, not the computer. While the language is extremely readable (almost like pseudo-code in some cases), it has plenty of special syntax that can be frustrating to learn. Yet it has some features like nested classes which differentiate it from PHP and substantially advance its web development prowess. However, with Ruby's perks come its negative quirks as well. In his article "What are the Benefits of Ruby on Rails?", Krešimir Bojčić demonstrates how the built-in String class can be changed with a few lines of code:

```
"1".to_i
#=> 1
class String
  def to i
    raise 'foobar'
  end
end
"1".to_i
#=> RuntimeError: foobar
```

Krešimir explains that even if this can be used maliciously, innately changing default methods is not dangerous. He continues on to state "Ruby has knife-like edges. I like to think that grown-ups can handle knives just fine." This is the natural beauty of the language – customizable, powerful and user-centric. PHP on the other hand has its own shortcomings, such as core implementations and language design, but it makes up for them in ease of use and popularity.

Now there are a couple problems with Rails. First we have the ever-longing problem of scalability. Ever since Twitter left Rails in favour of Scala, the Rails name has been associated with bad scaling. In his article "Why I wouldn't use Rails for a new company", Jared Friedman details how Rails' performance results are low due to it using Ruby, the slowest mainstream programming language at the moment. Similarly, Joseph Montanez in his article "Ruby on Rails vs PHP Comparison" critiques Ruby's scalability, requests/second performance and the hardware toll of Rails. While these are valid remarks, the truth is even most advanced Rails developers will never even come close to the same performance problems Twitter experiences. While being terribly slow, Rails nor Ruby are rarely the bottleneck for a lacking user-experience. In essence, most of these problems are un-noticeable for the mainstream. Also, Rails features inbuilt caching (Rails, 2016) which helps alleviate some of the performance hiccups.

In conclusion, passing the steep learning curve, biting into Rails is worth it for cleaner, more elegant code through a mature and well-established framework. While Rails and Ruby have major pitfalls (such as falling popularity), it is still a solid choice for refined high-end web production.

Bibliography

Amit. (2016, 09 26). *PHP: Which framework?* Retrieved from http://blog.igeek.info/2013/which-framework/

Bojčić, K. (2016, 09 29). Retrieved from https://www.toptal.com/ruby-on-rails/after-two-decades-of-programming-i-use-rails

Friedman, J. (2016, 09 29). Retrieved from https://blog.jaredfriedman.com/2015/09/15/why-i-wouldnt-use-rails-for-a-new-company/

Hansson, D. H. (2016, 09 28). Retrieved from The Rails Doctrine: http://rubyonrails.org/doctrine/#convention-over-configuration

Hansson, D. H. (2016, 09 28). *Ruby on Rails demo*. Retrieved from https://www.youtube.com/watch?v=Gzj723LkRJY

Montanez, J. (2016, 09 30). Retrieved from http://www.comentum.com/ruby-on-rails-vs-php-comparison.html

Rails. (2016, 09 27). Retrieved from http://guides.rubyonrails.org/active_record_basics.html

Rails. (2016, 09 27). Retrieved from http://guides.rubyonrails.org/caching_with_rails.html

Rails. (2016, 09 28). Retrieved from http://guides.rubyonrails.org/active_record_basics.html

Reiz, R. (2016, 09 27). Retrieved from https://blog.versioneye.com/2014/01/15/which-programming-language-has-the-best-package-manager/

Renton, J. (2016, 09 30). Retrieved from https://github.com/jrenton/laravel-scaffold

StackOverflow. (2016, 09 27). Retrieved from http://stackoverflow.com/tags/sinatra/info

StackOverFlow. (2016, 09 27). Retrieved from http://stackoverflow.com/tags/ruby-on-rails/info

Svrtan, D. (2016, 09 29). *Trends in Ruby - analyzing rubygems stats for 2015*. Retrieved from https://infinum.co/the-capsized-eight/articles/analyzing-rubygems-stats-v2015

Teo, L. (2016, 09 30). Retrieved from http://www.leonardteo.com/2012/07/ruby-on-rails-vs-php-the-good-the-bad/