|  |
| --- |
| National College of Ireland |
| Dissertation Title |
| [Document subtitle] |

|  |
| --- |
| Derek Caprani  [Date] |

# Abstract

The exponential growth of internet users over the past decade, following on from the emergence of web based applications for the masses, specifically Social Media, has led Web Developers and Technology Developers on a quest for ever more efficient ways to enable many millions of users to access the various services available to them concurrently with little or no delay or latency in responding to their various different requests. When the internet first began to become popular, websites were static and, with the exception of returning the complete contents of a webpage in response to a link being clicked, these sites didn’t interact with the user and the issues of concurrency and latency weren’t really an issue. However by stark contrast, present day users expect instant responses to their inputs which places huge demands on the underlying technologies if these expectations are to be met.

Whilst threading offered a solution to this problem of concurrency for a time, the quest for an efficient and robust non-blocking, asynchronous technology became the holy grail of sorts for web developers and the arrival of Node.js in 2009 appeared to be the answer to everyone’s problems, offering request and response times that couldn’t be matched by existing frameworks such as Ruby on Rails or PHP.

However in recent years, ReactPHP, written in PHP and EventMachine (which predates Node.js), written in Ruby have shown that non-blocking, event driven, asynchronous I/O can be achieved with these older technologies. This thesis will research, analyse and compare these technologies to establish which of the three offers the best solution to the concurrency/non-blocking problem.

# Introduction

Is comparing a non-blocking technology with a blocking technology a fair and accurate comparison to make? For example comparing journey times between two points on a map using secondary roads going through towns as one test and using a three lane motorway as the other, will illustrate that the Motorway will offer the faster the faster journey time, however this is not comparing like with like. Similarly, when comparing Node.js which is using an asynchronous, non-blocking event loop with PHP, which is using threads to allow concurrency but still relying on synchronous requests, will illustrate that non-blocking, asynchronous technology is faster than synchronous, blocking technology. With the arrival of event driven technologies such as EventMachine and ReactPHP to the Ruby and PHP developer communities respectively, these technologies can now offer request/response times that compare favourably with those achieved using Node.js (Igvita, 2008) (Sturgeon, 2013). In this paper it is planned to compare these technologies not only on the basis of speed, as this isn’t the only consideration when choosing a technology, but also in the areas of memory footprint and CPU consumption in order that a developer can make an informed choice when deciding which to use.

Whilst all three technologies achieve asynchronous behaviour via a continuous event loop and involve callback functions which are triggered once the submitted request has been completed, EventMachine and ReactPHP both rely on the Reactor Pattern to manage the complex business of ensuring that the responses being returned correspond with their respective requests. Node.js on the other hand leverages the power of the Google V8 JavaScript engine which is at its core and uses a library called libev to run its event loop (Dahl, 2011)

The purpose of the research is to establish the effectiveness and efficiency of Node.js as a development tool/environment not only in terms of its now well documented performance, an example of which is published by Lie et al. in their 2014 paper for the International Conference on Computational Science and Engineering (Lie, et al., 2014), but also in terms of the ease of development, including the full range of development tools and frameworks that are available in other languages, including scaffolding, database connection/communication and de-bugging. Notwithstanding Ionnas K. et al’s conclusion that end-to-end JavaScript is a viable option for building modern web apps (Chaniotis, et al., 2014), Netflix, another high profile internet goliath, discovered that their Node based application was eating up CPU resources while developing hourly endpoint latency increases of 10ms (Xiao, 2014), indicating that it's not all plain sailing for applications running on Node.js and leads me to the following research question:-

***What factors should a developer consider when deciding to build a web application in Node.js and at what point are the obvious performance benefits of Node outweighed by the practicalities, impracticalities and potentially costly and unforeseen pitfalls involved with the implementation of this, still relatively new, technology?***

# Acknowledgements

# List of Figures

# List of Tables

# Definitions, Acronyms and Abbreviations

Contents

[Abstract 1](#_Toc434138833)

[Introduction 1](#_Toc434138834)

[Acknowledgements 3](#_Toc434138835)

[List of Figures 3](#_Toc434138836)

[List of Tables 4](#_Toc434138837)

[Definitions, Acronyms and Abbreviations 5](#_Toc434138838)

[Literature Review 8](#_Toc434138839)

[Node.js 8](#_Toc434138840)

[Conclusion 8](#_Toc434138841)

[Ruby 8](#_Toc434138842)

[Ruby Frameworks 9](#_Toc434138843)

[Ruby on Rails 10](#_Toc434138844)

[Conclusion 10](#_Toc434138845)

[PHP 10](#_Toc434138846)

[PHP Frameworks 11](#_Toc434138847)

[Conclusion 11](#_Toc434138848)

[Technology Comparisons 11](#_Toc434138849)

[Learn 12](#_Toc434138850)

[Getting Started 12](#_Toc434138851)

[Footprint 12](#_Toc434138852)

[Scale 13](#_Toc434138853)

[Threading/Threads 13](#_Toc434138854)

[Non-Blocking Event-Driven I/0 13](#_Toc434138855)

[Memory Footprint 14](#_Toc434138856)

[Databases 14](#_Toc434138857)

[Popularity 14](#_Toc434138858)

[Contributing Research/Investigations 14](#_Toc434138859)

[Background 15](#_Toc434138860)

[Architecture 16](#_Toc434138861)

[Server Architecture 16](#_Toc434138862)

[Testing and Evaluation 17](#_Toc434138863)

[Testing Strategy 17](#_Toc434138864)

[Test Machine 17](#_Toc434138865)

[Conclusion and Future Work 17](#_Toc434138866)

[Conclusion 17](#_Toc434138867)

[Future Work 17](#_Toc434138868)

[Bibliography 17](#_Toc434138869)

# Literature Review

## Node.js

Node.js of itself is relatively new in the world of Web Technologies, “*It’s a platform built on Chrome’s JavaScript runtime”* (Node.js, 2015)*,* and was launched by its creator Ryan Dahl to an unsuspecting JavaScript Conference in Berlin in November 2009 (Dahl, 2011). By Dahl’s own admission in his 2011 video presentation to meet up group in Phoenix, Node is still very much in its infancy and is far from perfect. But in addressing some of Node’s shortcomings, Dahl is keen to emphasise that there are certain things that Node will never address if it’s to remain true to its core “non-blocking” philosophy (Dahl, 2011). Within a relatively short space of time Node was adopted by some major players in the greater e-commerce community and now boasts goliaths Walmart, PayPal, Netflix, LinkedIn and New York Times amongst its ever increasing list of users (Joyent/Node, 2015). Chaniotis et al., after extensive testing, conclude that despite some shortcomings, JavaScript via the Node platform is a viable option for building modern web apps (Chaniotis, et al., 2014). Lie et al. concur with Chaniotis et al’s findings, highlighting that Node’s bias is toward IO-intensive scenarios as opposed to Compute-intensive ones (Lie, et al., 2014).

Outside of the academic world, whilst as the list above would suggest, Node’s rise in popularity could arguably be described as meteoric. PayPal clearly are impressed by its performance, stating that during the research phase before fully adapting Node, an application with the equivalent functionality of a corresponding Java application could be built in almost half the time, with a third less lines of code and only 60% of the files necessary for the Java app (Harrell, 2013). However all has not been entirely plain sailing for Node in industry, with Netflix experiencing problems where hourly latency increases of 10ms with some endpoints were eating up CPU resources (Xiao, 2014).

One of the more fundamental problems with developing in Node is its poor level of debugging and exception handling facilities due to the fact that, by its single threaded nature, it’s always destroying the stack. Dahl has indicated that the Node development community are very much aware of this and it is in the pipe line, although no firm date has been set for this particular feature (Dahl, 2011).

### Conclusion

## Ruby

As a programming language Ruby is regarded in very high esteem by many programmers, not least among which is Ryan Dahl, of Node.js fame, who states in a video address, that “*Ruby is a beautiful language*” and that he prefers it to JavaScript (Dahl, 2011). Matsumoto himself describes Ruby as *“simple in appearance, but is very complex inside, just like our human body”* (Ruby.org, 2015). Geer notes that the Ruby language combines the clean, structured qualities of languages such as Java with the speed and convenience of scripting languages, specifically PHP (Geer, 2006).

### Ruby Frameworks

#### Rails

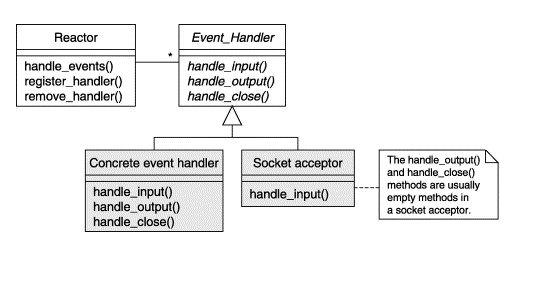
Heinemeier Hansson’s, and by default Rails’, approach to their framework is very rigidly “*Convention over Configuration*”, so much so that Heinemeier Hansson clearly states on his own website that, whilst he’s open to well informed criticism, he and his team, with few exceptions, know what’s best (Heineneier Hansson, 2012). However, as alluded to by Geer when quoting Boeing software developer Curt Hibbs (Geer, 2006), the Rails framework enables developers to achieve far more in less time, than could be achieved when working with combinations of other languages and technologies which have to be configured in order to fit together to achieve the same result.

#### Sinatra

Sinatra, created by Blake Mizerany in 2007 (Jones, 2012) is effectively a lighter version of Rails, following an almost identical structure (Hagerty, 2014). However, where Rails’ corner stone is “Convention over Configuration”, forcing developers to follow its rigid structure, Sinatra offers developers the ability to configure as required (Skillcrush, 2015) to build web applications which eliminate the unwanted and unnecessary folders and files that are often surplus to requirements in smaller projects built with rails.

#### EventMachine

EventMachine is a library which, when added to the Ruby environment as a gem, offers event driven non-blocking I/O to the Ruby developer (McCune, 2011). As specified on the EventMachine github repository README file, event-driven I/O is dealt with using the Reactor Pattern (EventMachine, 2015). However whilst a certain level of non-blocking asynchronous I/O is achieved using this pattern, there are limits to its capabilities, namely a default limit on concurrent requests of 20 (Igvita, 2008), also the Reactor Pattern effectively gives an illusion of asynchronous behaviour whilst in the background delivering requests concurrently to a service handler, demultiplexing them and distributing them synchronously to their various relevant handlers.

Figure 1 Class Diagram of Reactor Pattern (Chein-Chih & I. -Chen, 2006)

### Ruby on Rails

This pairing of a well-structured framework with a well written language in the form of Ruby on Rails makes for a very powerful web development tool which is in effect greater than the sum of its parts. However An et al. (An, et al., 2009) note that, not unlike Node.js, Ruby on Rails isn’t without its shortcomings when it comes to debugging. Specifically dynamic typing, meaning that type errors are difficult to identify due to their remaining latent until runtime. And although An et al.’s paper was written in 2009, Voelker’s June 29th 2014 Blog would suggest that this is still a live issue within the Ruby on Rails development community (Voelker, 2014).

### Conclusion

## PHP

By far the older of the three languages being explored, PHP’s origins date back to 1994 when Rasmus Lerdorf created his “*Personal Home Page Tools”* in the form of several C based CGI (Common Gateway Interface) binaries (PHP.net, 2015). Originally referred to as PHP Tools, the language has had several name changes, ranging through FI, Construction Kit and PHP/FI before finally settling on PHP with the release of PHP 3.0 in June 1998 (PHP.net, 2015). In its current version at the time of writing (Aug. 22nd 2015), PHP 5 accounts for 81.2% of all websites whose server-side languages are known (W3TECHS, 2015). That being said, with the explosion of network traffic and users’ apparently insatiable appetite for more and more interactive functionality placing greater and greater demands on servers, PHP is beginning to show its limitations, particularly in the area of high concurrency I/O intensive functions (Lie, et al., 2014).

### PHP Frameworks

As with Ruby and Node.js, where there are several Frameworks that can be used in association with the chosen scripting language, PHP is no different in this regard but the fact that it has been around for many more years means that the choice of Frameworks is far greater.

#### Prado

Prado (PHP Rapid Application Object-oriented) was created in 2004 by Qiang Xue and has been through many iterations since then, at its core is event-driven programming ***more to be found on this framework***

#### Yii

The Yii (Yes It Is) framework was created in 2008 by Qiang Xue (yiiframework.com, 2015) who, having worked on previous PHP framework projects, developed Yii in response to needs he identified over many years. Yii offers developers benefits of what Xue saw as the best qualities of other frameworks, including Rails’ “Convention over Configuration”. Equally Yii offers event-driven programming, a feature of the Prado framework and is something Dahl hit upon when creating Node.js a year later.

### Conclusion

## Technology Comparisons

Having corresponded via e-mail with Jim Brikman, previously a senior developer at LinkedIn, in relation to comparing Node.js with other frameworks, he suggested that technologies can’t be compared on any one single point but must be measured against each other according to a number of different criteria and offered the following list as a suggestion (Brikman, 2015):

1. **Learn**: getting started, ramp up, overall learning curve.
2. **Develop**: routing, templates, i18n, forms, json, xml, data store access, real time web.
3. **Test**: unit tests, functional tests, integration tests, test coverage.
4. **Secure**: CSRF, XSS, code injection, headers, authentication, security advisories.
5. **Build**: compile, run tests, preprocess static content (sass/less/CoffeScript), package.
6. **Deploy**: hosting, monitoring, configuration.
7. **Debug**: step by step debugger, profilers, logging,
8. **Scale**: throughput, latency, concurrency.
9. **Maintain**: code reuse, stability, maturity, type safety, IDEs.
10. **Share**: open source activity, mailing lists, popularity, plugins, commercial support, jobs.

Whilst this is undoubtedly a very comprehensive set of comparison parameters, given the timescale available it will be necessary to limit the scope of this project to no more than two of the above headings (with some minor modifications), plus one more. These will be as follows:

1. **Learn**: Getting started, available resources, tutorials etc.
2. **Footprint**: Disc space occupied by base code plus dependencies, CPU consumption.
3. **Scale**: Concurrency, throughput, latency.

### Learn

Node.js, Ruby, Ruby on Rails, and PHP all have excellent resources available, both online, in the form of video tutorials and in soft and hard copy. A quick search on YouTube using just “Node.js Tutorials”, “PHP Tutorials” and “Ruby Tutorials” returned 174,000 results for Node, 118,000 results for Ruby and 990,000 results for PHP (YouTube.com, 2015). Whilst these results are far from scientific and assuming that there’s a full spectrum of content ranging in quality from excellent to extremely poor, one could possibly conceivably progress from a complete novice to gain a reasonable understanding of any of the above technologies from YouTube alone. Separate and apart from self-directed learning opportunities, Node.js, PHP and Ruby have vibrant online communities consisting of user groups who communicate via blogs and meetups, where they share knowledge, with, as well as learn from, each other.

The lack of a good IDE for Node.js, as expressed by Kiran Prasad of LinkedIn (Norton & Coatta, 2014), is something which sets Node at a disadvantage when compared to both Ruby on Rails and PHP. Although there are several cloud based IDE’s such as Cloud9 and Koding, as well as native text editors such as SublimeText and WebStorm. However, whilst these offer varying degrees of excellence in terms of text editing and syntax colour coding, there’s still a conspicuous absence of comprehensive debugging tools to enable a developer to trace bugs to a definable source, as can be found in NetBeans for Java development and Visual Studio for .NET and C#. Yet despite this shortcoming, Prasad still favours Node.js over Ruby on Rails, based on its greater performance, smaller [memory] footprint and the significantly fewer lines of code required to achieve a desired result (Norton & Coatta, 2014).

### Getting Started

#### Node.js

As stated by Brikman, installing Node.js and getting up and running is relatively quick and easy by comparison with other technologies (Brikman, 2015).

#### Ruby EventMachine

### Footprint

To be established when testing and comparing applications.

Griffin et al point out that the Node.js model of making extensive use of callbacks to deal with concurrent requests demands a high percentage of CPU power, however this is deemed to be a justifiable trade off given the higher performance delivered (Griffin, et al., 2011).

### Scale

#### Concurrency

Prior to the arrival of Node.js in 2009, Paulson, in 2007, identified dynamic programming languages as growing in popularity, she also quotes Tim Bray, the then director of Web Technologies at Sun, as noting that the requirements of large software projects are beyond the capabilities of many dynamic languages but that “*some of their agility-related features are slowly percolating into enterprise languages such as Java*” (Paulson, 2007).

### Threading/Threads

Ruby has had the capacity to generate multithreaded code from the beginning and, as an indication of how well thought out the language is, will support multithreading across all platforms irrespective of whether the OS upon which it’s operating supports this (ruby-lang.org, 2015). Despite the fact that Purer (Purer, 2009) in 2009 states that PHP doesn’t support threads, PHP’s own documentation would suggest this to be incorrect with references to threads and multithreads appearing in their changelog dating back to Version 4.0 Beta 2 in Aug. ’99 (PHP, 2008). Node.js on the other hand eschews multithreading and runs on a single thread operating a continuous event loop, this effectively dispenses with the need for concurrent threads, thereby freeing up resources, enabling I/O intensive applications to process requests in near real-time (Node.js, 2015). Although this approach offers unmatched performance, it also represents a single point of failure, with one blocking function, in the absence of threading, having the potential to bring the whole operation to a halt until this particular process has completed (Brikman, 2014).

### Non-Blocking Event-Driven I/0

Node is the only technology, of the three being compared, which is built solely around the concept of event-driven non-blocking I/O (Dahl, 2011). The Ruby community offers a number of frameworks that can deal with concurrency using an event loop to give asynchronous non-blocking performance, EventMachine and Celluloid::IO being two examples (GitHub.com, 2011) whilst PHP enables its developers to produce event-driven non-blocking I/O functionality via ReactPHP (GitHub.com, 2012). Thruway is another PHP based non-blocking technology however this framework uses ReactPHP at its core and as such is simply an extension of ReactPHP (Dan, 2015). Both EventMachine and ReactPHP make use of the Reactor Design Pattern to achieve non-blocking, asynchronous I/O performance equivalent to that of Node.js. However whilst a certain level of non-blocking asynchronous I/O is achieved using this pattern, there are limits to its capabilities, namely a default limit on concurrent requests of 20 (Igvita, 2008), also the Reactor Pattern effectively gives an illusion of asynchronous behaviour whilst in the background delivering requests concurrently to a service handler, demultiplexing them and distributing them synchronously to their various relevant handlers.

### Memory Footprint

### Databases

### Popularity

TIOBE Software who, since 2001, have been maintaining an index of the popularity of A to Z spectrum of programming languages, ranging from little known languages such as ABAP, Boo, Yoric and Z shell to readily identifiable languages like Java, C#, Python and PHP, currently place PHP, JavaScript and Ruby at 7th, 9th and 13th respectively (TIOBE Software, 2015). On the other hand Carbonelles GitHub hosted ranking site, whose sorting criteria is based on how often a language tutorial is searched, ranks PHP at 2nd, JavaScript at 7th and Ruby at 10th (Carbonelle, 2015).

|  |  |
| --- | --- |
| Figure 2:-TIOBE Software Language Top 20 Aug 2015 | Figure 3:-PYPL Software Language Top 16 Aug 2015 |

Whilst neither source can be taken as definitive it’s noteworthy that, relative to each other, PHP, JavaScript and Ruby maintain the same order in both tables.

# Contributing Research/Investigations

# Background

# Architecture

## Server Architecture

# Testing and Evaluation

## Testing Strategy

The plan is to build two separate server applications with identical functionality, one using Node.js and the other using Ruby on Rails running EventMachine. Both applications will generate a WebSocketServer listening on a specific port. WebSockets will be used, as they are now an integral part of HTML 5 and not only offer greater speed of communication than HTTP but are also fully duplex, meaning they can send and receive using the same connection. This protocol fully compliments the asynchronous non-blocking technologies offered by both Node.js and EventMachine. Using a similar strategy to Griffin et al (Griffin, et al., 2011) the plan is to generate multiple user via a load generator and then connect these virtual users in incremental groups concurrently to the server application and then gradually scale up to a target of 10k users, with a random algorithm generating messages between the users. A number of load testing applications were explored including Apache JMeter, Tsung and HP’s LoadRunner, however whilst it is possible to implement websocket connections using these applications, Webload proved to be more suitable when running the kind of scenario envisaged.

A java application was created which combined the features of two sample applications, the first created a Client application which connected with a websocket server, sent an initial message and then instantly responded to any message received (Jiji\_Sasidharan, 2013), the second enabled a user defined amount of clients to connect to the websocket server and then randomly loop through these clients sending messages, which the server would then broadcast to the other connected clients (Webtide, 2011), whilst the second application appeared to do exactly what I was looking for, it proved to be difficult to implement due to unreliable dependencies, however I was able to achieve a similar result by modifying the first one.

### Test Machine

# Conclusion and Future Work

## Conclusion

## Future Work

Jansen (Jansen & TIOBE.com, 2015) made reference to the GWT (Google Web Toolkit) framework, which as the name would suggest is a Google project. GWT attempts to address the same problem as Node.js, namely offering web developers, from a non-programming background, the ability to write the front and back ends of an application in the same, easy to use, language (JavaScript) (GWT Project, 2015). At the time of writing, having already outlined the basis for the research, GWT was out of scope however, given that GWT and Node.js occupy a very similar space, it may be worth exploring their future progress in parallel to see which (if either) becomes the dominant development framework.

# Bibliography

An, J.-h., Chauduri, A. & Foster, J. S., 2009. *Static Typing for Ruby on Rails.* Washington, IEEE Computer Society.

Apache Foundation, 2015. *Apache JMeter - Apache JMeter tm.* [Online]   
Available at: http://jmeter.apache.org/  
[Accessed 23 October 2015].

Bachle, M. & Kirchberg, P., 2007. Ruby on Rails. *IEEE Software,* 24(6), pp. 105 - 108.

Brikman, J., 2015. *Comparing Web Technologies* [Interview] (25th September 2015).

Brikman, Y. (., 2014. *Node.js vs Play.* Tokyo, Scala Matsuri.

Carbonelle, P., 2015. *PYPL PopularitY of Programming Language.* [Online]   
Available at: http://pypl.github.io/PYPL.html  
[Accessed 28th August 2015].

Chaniotis, I. K., Kyriakou, K.-I. D. & Tselikas, N. D., 2014. Is Node.js a viable option for building modern web applications? A performance evaluation study. *Computing,* 2014(March), pp. 1-22.

Chein-Chih, H. & I. -Chen, W., 2006. An event-driven framework for inter-user communication applications. *Information and Software Technology,* 48(7), pp. 471-483.

Dahl, R., 2011. *Ryan Dahl - History of Node.js,* Phoenix: Phx Tag Soup.

Dan, D. w., 2015. *voryx/Thruway.* [Online]   
Available at: https://github.com/voryx/Thruway  
[Accessed 12th October 2015].

EventMachine, 2015. *eventmachine/README.md at master.* [Online]   
Available at: https://github.com/eventmachine/eventmachine/blob/master/README.md  
[Accessed 10th October 2105].

Geer, D., 2006. Will Software Developers Ride Ruby on Rails to Success?. *Computer,* Feb, 39(2), pp. 18 - 20.

GitHub.com, 2011. *eventmachine/eventmachine.* [Online]   
Available at: https://github.com/eventmachine/eventmachine  
[Accessed 12th October 2015].

GitHub.com, 2012. *reactphp/react.* [Online]   
Available at: https://github.com/reactphp/react  
[Accessed 12th October 2015].

Griffin, L., Keiran, R., de Leastar, E. & Bodvich, D., 2011. *Scaling Instant Messaging Communication Services,* Waterford: Waterford Institute of Technology.

GWT Project, 2015. *[GWT].* [Online]   
Available at: http://www.gwtproject.org/overview.html  
[Accessed 28th August 2015].

Hagerty, P. J., 2014. *Rails vs. Sinatra.* [Online]   
Available at: https://blog.engineyard.com/2014/rails-vs-sinatra  
[Accessed 23rd August 2015].

Harrell, J., 2013. *Node.js at PayPal.* [Online]   
Available at: http://www.paypal-engineering.com/2013/11/22/node-js-at-paypal/  
[Accessed 14th August 2015].

Heinemeier Hansson, D., 2015. *David Heinemeier Hansson (DHH).* [Online]   
Available at: http://david.heinemeierhansson.com/  
[Accessed 21st August 2015].

Heineneier Hansson, D., 2012. *Rails is omakase.* [Online]   
Available at: http://david.heinemeierhansson.com/2012/rails-is-omakase.html  
[Accessed 21st August 2015].

Igvita, I., 2008. *Ruby Eventmachine-The Speed Demon.* [Online]   
Available at: https://www.igvita.com/2008/05/27/ruby-eventmachine-the-speed-demon/  
[Accessed 28th September 2015].

Jansen, P. & TIOBE.com, 2015. *Programming Language Comparison.* Eindhoven: TIOBE.com.

Jones, D., 2012. *Rails or Sinatra: The Best of Both Worlds?.* [Online]   
Available at: http://www.sitepoint.com/rails-or-sinatra-the-best-of-both-worlds/  
[Accessed 23rd August 2015].

joyent/node, 2015. *Modules . joyent/node Wiki . GitHub.* [Online]   
Available at: https://github.com/joyent/node/wiki/modules#frameworks  
[Accessed 22nd August 2015].

Joyent/Node, 2015. *Projects, Applications and Companies Using Node.* [Online]   
Available at: https://github.com/joyent/node/wiki/Projects,-Applications,-and-Companies-Using-Node  
[Accessed 20th August 2015].

Joyent, 2014. *Error Handling in Nodejs.* [Online]   
Available at: https://www.joyent.com/blog/best-practices-for-error-handling-in-node-js  
[Accessed 15th August 2015].

Kar, S., 2015. *Comparing Node.js Frameworks: Express, Hapi, Loopbak, Sailsjs and Meteor,* s.l.: StrongLoop.

LangPop.com, 2013. *Programming Language Popularity.* [Online]   
Available at: http://langpop.com/  
[Accessed 27th August 2015].

Lie, K., Ma, Y. & Tan, Z., 2014. *Performance Comparison and Evaluation of Web Development Technologies in PHP, Python and Node.js.* s.l., IEEE.

McCarthy, J., 2013. *Introduction to Rails.* Dublin: s.n.

McCune, R. R., 2011. *Node.js Paradigms and Benchmarks,* Notre Dame: Notre Dame University.

Niclausse, N., 2013. *Tsung.* [Online]   
Available at: http://tsung.erlang-projects.org/index.en.html  
[Accessed 23 October 2015].

Node.js, 2015. *Node.js.* [Online]   
Available at: https://nodejs.org/  
[Accessed 20th August 2015].

Norton, K. & Coatta, T., 2014. Node at LinkedIn: the pursuit of thinner, lighter, faster. *Communications of the ACM*, February, pp. 44 - 51.

opensourcetesting.org, 2015. *Open source performance testing tools.* [Online]   
Available at: http://www.opensourcetesting.org/performance.php  
[Accessed 23 October 2015].

Paulson, L. D., 2007. Developers Shift to Dynamic Programming Languages. *Computer,* 40(2), pp. 12-15.

PHP.net, 2015. *PHP: History of PHP - Manual.* [Online]   
Available at: http://php.net/manual/en/history.php.php  
[Accessed 22nd August 2015].

PHP, 2008. *PHP: PHP 4 Changelog.* [Online]   
Available at: http://php.net/ChangeLog-4.php#4.4.9  
[Accessed 28th August 2015].

Purer, K., 2009. *PHP vs. Python vs. Ruby - The web scripting language shootout.* Vienna, s.n.

Pylot, 2105. *Pylot | Open Source Web Performance Tool.* [Online]   
Available at: http://www.pylot.org/  
[Accessed 23 October 2105].

Ruby.org, 2015. *About Ruby.* [Online]   
Available at: https://www.ruby-lang.org/en/about/  
[Accessed 21st August 2015].

ruby-lang.org, 2015. *About Ruby.* [Online]   
Available at: https://www.ruby-lang.org/en/about/  
[Accessed 28th August 2015].

Skillcrush, 2015. *Sinatra Cheatsheet.* [Online]   
Available at: http://4b93n32qwvjj3ddn5w3yhffoas6.wpengine.netdna-cdn.com/wp-content/uploads/2015/02/SinatraCheatsheet.pdf  
[Accessed 23rd August 2015].

Stella, L., Jarzabek, S. & Wadhwa, B., 2008. *A comparative study of maintainability of web applications on J2EE, .NET and Ruby on Rails.* Beijing, 10th International Simposium on Web Site Evolution.

StrongLoop, 2015. *Comparing Node.js Frameworks: Express, Hapi, LoopBack, Sailsjs and Meteor,* s.l.: LoopBack.

Sturgeon, P., 2013. *Benchmarking Codswallop: NodeJS v PHP.* [Online]   
Available at: https://philsturgeon.uk/php/2013/11/12/benchmarking-codswallop-nodejs-v-php/  
[Accessed 7th September 2015].

TIOBE Software, 2015. *TIOBE Software: Tiobe Index.* [Online]   
Available at: http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html  
[Accessed 26th August 2015].

Viswanathan, V., 2009. Rapid Web Application Development: A Ruby on Rails Tutorial. *Software, IEEE,* 25(6), pp. 98 - 106.

Voelker, A., 2014. *Sick of Ruby, dynamic typing, side effects and basically object-oriented programming.* [Online]   
Available at: https://blog.abevoelker.com/sick-of-ruby-dynamic-typing-side-effects-object-oriented-programming/  
[Accessed 21st August 2015].

W3TECHS, 2015. *Usage Statistics and Market Share of PHP for Websites, August 2015.* [Online]   
Available at: http://w3techs.com/technologies/details/pl-php/all/all  
[Accessed 22nd August 2015].

Webtide, 2011. *Websocket Example: Server, Client and LoadTest.* [Online]   
Available at: https://webtide.com/websocket-example-server-client-and-loadtest/  
[Accessed 28 October 2015].

Xiao, Y., 2014. *The Netflix Tech Blog.* [Online]   
Available at: http://techblog.netflix.com/2014/11/nodejs-in-flames.html  
[Accessed 14th August 2015].

yiiframework.com, 2015. *About Yii | Yii PHP Framework.* [Online]   
Available at: http://www.yiiframework.com/about/  
[Accessed 23rd August 2015].

YouTube.com, 2015. *YouTube.com.* [Online]   
Available at: http://youtube.com  
[Accessed 13th October 2015].